DOI: 10.15439/2017F120
ISSN 2300-5963 ACSIS, Vol. 11

An Approach for Modeling Events in Information Systems

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

Ivan Luković, Vladimir Dimitrieski
University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia
Email: {ivan, dimitrieski}@uns.ac.rs

Verislav Đukić
Djukic Software GmbH, Nürnberg, Germany
Email: info@djukic-soft.com

***Abstract* — Contemporary tools aimed at information system (IS) development often use models to generate system implementation. Starting from an IS model, these tools commonly generate database implementation schema as well as code for generic CRUD operations of business applications. On the other hand, at the level of platform-independent models (PIMs) there is a lack of support for specification of more complex functionalities associated with events. In this paper, we present an approach aimed at specification of events at the level of PIMs. We introduce new concepts to describe context in which an event may occur, while we use our IIS*CFuncLang language to define event business logic. We also developed adequate transformations to generate executable program code from these specifications.**

I. Introduction

SIGNIFICANT efforts have been invested into the research of approaches and tools, aimed to completely, or partially, automate the IS development process. In these approaches, models play a key role in the IS development process [1]. Usually, a model is transformed into (i) a database implementation schema and (ii) program code of business applications performing simple CRUD (create, retrieve, update, and delete) operations over the generated database [2]. Beside these typical functionalities, business applications usually include more complex functionalities, i.e., business logic, that is specific for the application domain. For example, such application-specific functionalities include complex calculation and validation tasks, series of database operations triggered by an event, etc.

A similar classification of application functionalities may be found in [3], where authors classify application program code as: (i) generic; (ii) schematic; and (iii) individual. Generic and schematic program code is common for various applications domains and has patternable structure. Individual program code is specific for an application, and it is hard to generate it from such a model [3]. The approaches and tools aimed at the IS development, support modeling of typical functionalities as well as generation of program code for these functionalities. Unfortunately, modeling of application-specific functionalities associated with events, and the generation of appropriate program code often is not supported by these approaches and tools. Manual customization of generated program code is used for implementation of these functionalities. In order to formally specify business logic of application-specific functionalities at the level of platform-independent models (PIMs) we have developed a domain-specific language (DSL) named IIS*CFuncLang [2]. Research efforts presented in this paper are extension of our previous work devoted to a formal specification of application-specific functionalities ([4]).

In the paper we will present an approach for modeling application-specific functionalities associated with IS events. This approach is aimed at specifying IS events at the abstraction level of PIMs. An event specification consists of two parts: (i) business logic that has to be executed upon event occurrence, and (ii) context in which the event may occur. For the specification of business logic we use the IIS*CFuncLang language. In order to formally specify an event context we introduce a new PIM concept named *Event*. Using this concept a designer may specify event properties such as event source, an action that trigger the event execution, and level that the event is handled at. Also, we have developed algorithms aimed at transformation of event specifications into executable program code. In this way we generate complete program that implements application-specific functionalities associated with events.

II. Fundamentals

Commonly, the application-specific business logic is executed upon the occurrence of an event. Therefore, specification of events is an important part of an IS model. We analysed several approaches and tools aimed at IS modeling and code generation, and in the most cases, specification of events is only partially allowed at the PIM level. Business logic for an event is specified at the lower abstraction level by amending the generated program code. This approach may raise several concerns such as portability, operational maintenance, and synchronization between generated and hand-written program code [2]. Also, a developer must possess expertise in the target programing language and platform services. In our approach IS events are completely specified at the abstraction level of PIMs, and complete program code is generated using the

The research presented in this paper was partly supported by Ministry of Education, Science and Technological Development of Republic of Serbia, Grant III-44010.

transformation algorithms. In this way, generated program code does not require additional customization, which may help in overcoming aforementioned problems caused by amending generated program code. A designer does not need to be familiar with a target language nor target platform services. Also, it is easier to achieve portability since emergence of a new platform or a target language does not require customization of the generated program code. Only new transformation algorithms for the platform need to be implemented. Furthermore, usage of DSLs instead of manual writing of code in a general-purpose programing language (GPL) brings additional benefits as it is discussed in [5].

For the practical verification of our approach we have chosen the IIS*Case tool ([6]). Starting from PIMs, IIS*Case provides generation of a database implementation schema for various relational database management systems (RDBMSs) as well as executable business applications and transaction programs. However, until now this tool did not provide modeling of application-specific functionalities associated with events. IIS*Case is an open source tool, and the authors of the paper are actively involved in its development.

A. IIS*Case preliminaries

At the abstraction level of PIMs, IIS*Case currently provides conceptual modeling of database schemas and business applications of an IS. Starting from such PIM models as a source, a chain of transformations is performed so as to obtain executable program code of business applications and database SQL/DDL scripts for a selected target platform ([7], [8]).

The form type is a central concept for design and integration of database schemas in the IIS*Case tool. It generalizes document types, i.e. screen forms used for communication with an IS. Each form type is a named tree structure, whose nodes are called component types. In the transformation process a component type will be used as a starting point for generation of both screen forms and database tables. Analogously, attributes in component types are mainly used as a source for generation of columns in database tables, as well as input and output fields in screen forms. The component type and attribute are amended with new concepts in order to formally specify events. These new concepts are described in detail in Section 3

B. IIS*CFuncLang

The IIS*CFuncLang language is a textual DSL aimed at specification of an application-specific business logic [2]. In our approach we use this language to define business logic associated with events. In this section we present the main concepts of IIS*CFuncLang that are important for specification of events.

IIS*CFuncLang includes commands specific for the domain of database applications, such as commands for performing operations over database records, commands for updating properties of screen forms, etc. In addition to the commands from the concrete domain of business applications, the language includes concepts from GPLs, such as control-flow statements, variable and array declarations, various operators etc. In this way, when some application-specific functionality cannot be described with domain concepts a designer may use less abstract concepts from GPLs. In Listing 1, an example of an IIS*CFuncLang function is presented. The function checks if the input parameter is an empty string, and in that case reports an error and aborts the transaction.

```
FUNCTION ValidateName(In1 STRING)
RETURNS BOOLEAN
VAR
  i INT;
END_VAR
BEGIN
  IF (Len(In1) = 0) THEN
    i := ShowErrorMessage('Error!!!');
    signal(abort_trigger);
    RETURN FALSE;
  ELSE
    RETURN TRUE;
  ELSE;
END
```

Listing 1 An example of IIS*CFuncLang function

The IIS*CFuncLang execution semantics is based on the interpreter approach. The compiler transforms IIS*CFuncLang specifications into intermediate code. The intermediate code is similar to Java byte-code, and it is prepared for interpretation. Also we have developed transformation from IIS*CFuncLang specifications to SQL program code for database triggers. This approach is suitable when some application-specific functionality has to be implemented at the level of a RDBMS.

Each event is associated with one IIS*CFuncLang function. When event is handled at the level of business applications, then the compiler generates intermediate code for the function. The interpreter is embedded into the business application. Upon the event occurrence within system, business application starts interpreter that executes intermediate code for the event function. The interpreter returns control to the business application after code execution. If event is handled at the level of database server then appropriate database triggers are generated and deployed to the target server.

III. Events

Frequently, an application-specific functionality is executed when an event occurs within a system. In order to formally describe such a scenario, we introduce a concept named Event. Each event has the following attributes: (i) source, (ii) IIS*CFuncLang function defining business logic, (iii) event handling level, and (iv) type.

An event source may be exactly one instance of the following concepts: form type, component type, or attribute in a component type. Let's suppose that a form type or a component type is selected as an event source. Staring from a form type or a component type specification, the code generator creates screen forms and database tables. The event business logic will be executed when appropriate action is performed over the generated screen form (e.g., mouse clicked), or over the database table (e.g., a record is inserted).

There are three levels of event handling: (i) the database server level, (ii) the application server level, and (iii) the client application level. If an event is handled at the level of a database server, then a function associated with the event will be transformed into the PL/SQL program code of database triggers. Currently, we provide generation of program code aimed to be executed by Oracle RDBMS. A generation of program code for other RDBMSs is a matter of further research. If an event is handled at the level of an application server or a client application, then intermediate code will be generated as described in the previous section.

The event type determines the type of action that triggers the event. It includes typical software event types common for various programming environments, such as mouse events, keyboard events, and events over database tables and columns. The set of allowed event types has more that forty elements and it will be presented in detail in the rest of the section.

A. Events in Component Type Attributes

Each attribute in a component type may be associated with a set of events. There are events, such as *Mouse Clicked*, *Key Pressed*, and *Focus gained*, that are only handled at the client application level. They are activated when a user performs appropriate operations mouse over the screen form fields generated for the attribute that the event is associated to. Events such as *After Update Record* and *Before Update Record* are activated before and after the update operation is performed over the database column generated for the attribute to which the event is associated. These types of events are only handled at the database server level.

B. Events in Component Types

Component type specifications are used as a starting point for the generation of screen forms and database tables. We extended this concept so a designer may associate set of events to each component type. These events may be divided into two categories.

Events such as *After Update Record* and *Before Update Record* belong to the first category. They may be handled at the database server, client application or application server level. If event is handled at the database server level then the generated trigger will be activated when appropriate operation is performed over the database table generated for the component type to which the event is associated. When the event is handled at the client application level, then it will be activated when a user presses the *Save* button in the screen form generated for component type to which the event is associated.

The second category includes events related to the typical software events performed over screen forms, e.g., mouse clicked, focus gained, the *Save* button pressed, etc. These events are only handled at the client application level.

C. Events in Form Types

We amended the form type concept with list of events. These events are related to the screen forms generated for the form type. There are two events that belong to this category: *On Open Form*, and *On Close Form*. These types of events are only handled at the client application level. Events are activated when the screen form generated for the form type is opened or closed.

IV. Use Case

In this section we will describe a use case from the application subsystem for university administration that we have developed using IIS*Case. In order to specify the application subsystem, we defined two form types with the following component types: (i) DEPARTMENT(DeptId, DepName), and (ii) STUDENT(Sid, Name, DateOfBirth, Status). Beside typical functionalities, the user requirements also included the following: (i) department name must be non-empty string, and (ii) if a student status is changed to part-time, i.e., value of the *Status* attribute is set to 'PT', all statuses of his or her enrolments must be changed accordingly.

In order to realize these requirements, two new events are defined. The first one is associated with the *Department* component type, and it is activated before a new record is inserted into the database table generated for the component type. The second is defined for the *Status* attribute in the STUDENT component type, and it is activated before an update operation is performed over the database column generated for the *Status* attribute. Business logic of events is defined by the functions presented in Listing 1, and Listing 2. If the event is handled at the database server level, then PL/SQL program code will be generated. Generated code consists of two parts. The first part is a package implementation containing a function generated from the function defining business logic. The second part contains a database trigger. In the trigger header it is specified that is the trigger is activated before each update of a row, or before each update of the appropriate attribute. The trigger body is rather simple, including only invocation of the generated function from the package.

Let's assume that the event should be handled at the client application level. In this case, generated intermediate code and the interpreter are embedded into the generated business application. Also, the business application is extended in order to include an event handler listening the specified event, i.e., a record is updated when a user presses the *Save* button. When the event occurs within system the event handler invokes the interpreter to execute the intermediate code. The interpreter returns result and various execution statuses to the business application. Based on the result and statuses, the business application determines whether the operation will be aborted or committed. For example, IIS*CFuncLang provides *SIGNAL* command that informs the execution environment about specific state of the execution. If this command is executed with the *abort_trigger* argument, then the business application should abort the current operation.

```
FUNCTION UpdateStatuses(Sid INT, Status
String)
RETURNS INT
VAR
  RES INT;
END_VAR
BEGIN
IF Status == 'PT' THEN
  RES := Execute_NonQuery('update ENROLLMENT
set Status=\'PT\' where Sid=' ||
To_String(Sid));
  return RES;
END_IF;
END
```

Listing 2 An example of IIS*CFuncLang function

V. Related work

Nowadays, many commercial tools allow PIM modeling of database schemas and ISs. We analysed tools that also provide modeling of application-specific functionalities associated with events. Usually, these tools, such as *IntegraNova Modeler* ([8]) and *SOLoist* ([10]), provide only partial specification of events is at a level of PIMs, while business logic is implemented by customizing generated program code by means of a target language. Potential problems with this approach are already discussed in the Section 1, such as synchronization of generated and hand-written program code. However, we propose a specialized language and concepts to fully specify events at the level of PIMs. Also, we provide adequate transformations for generating a complete program that implement business logic for events. Such a generated program code does not require additional customization.

Executable UML (xUML) is an approach aimed at creating models detailed enough to enable generation of complete system implementation [11]. Object Management Group (OMG) introduced an action language in order to allow specification of system procedural behaviour using algorithmic concepts. This language includes concepts for specification of system events, such as event, signal, input and output pins etc. In our approach, we provide number of high-level commands from the domain of business applications, such as commands for aborting transactions, executing queries, and updating GUI properties.

VI. Conclusion

In this paper we presented an approach aimed at complete specification of IS events at the level of PIMs. During the research we have also identified several directions for future research. We plan to extend the set of allowed event types. For example, introduction of the *Value Change* event type will allow a designer to specify actions that will be executed after each change of input fields in generated screen forms. A future research will encompass the development of a new group of commands that will act as a query language over PIM concepts, such as form type and component type. Additionally, we intend to transform such commands into SQL program code aimed to be executed over various RDBMs.

References

[1] D.S. Frankel, "Model Driven Architecture: Applying MDA to Enterprise Computing", *Wiley Publishing Inc.*, 2003.

[2] A. Popović, V. Dimitrieski, I. Luković, V. Đukić, "A DSL for modeling application-specific functionalities of business applications", *Computer Languages, Systems & Structures (COMLAN)*, Elsevier Science Publishers B. V., DOI: 10.1016/j.cl.2015.03.003, 2015 .

[3] T. Stahl, M. Völter, "Model-Driven Software Development: technology, engineering, management", *John Wiley & Sons Inc*, Hoboken, USA, ISBN: 0-470-02570-0, 2006.

[4] I. Luković, A. Popović, J. Mostić, S. Ristić, "A Tool for Modeling Form Type Check Constraints and Complex Functionalities of Business Applications", *Computer Science and Information Systems (ComSIS)*, Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia and Montenegro, ISSN: 1820-0214, Vol. 7, No. 2, 2010, pp. 359-385.

[5] M. Mernik , J. Heering, M.A. Sloane, "When and How to Develop Domain-Specific Languages", *ACM Computing Surveys (CSUR)*, Association for Computing Machinery, USA, Vol. 37, No. 4, 316-344. 2005

[6] I. Luković, P. Mogin, J. Pavicević, S. Ristić, "An Approach to Developing Complex Database Schemas Using Form Types", *Software: Practice and Experience*, John Wiley & Sons Inc, Hoboken, USA, ISSN: 0038-0644, Published Online, May 29, 2007, DOI: 10.1002/spe.820

[7] S. Aleksić, I. Luković, P. Mogin, M. Govedarica, "A Generator of SQL Schema Specifications", *Computer Science and Information Systems (ComSIS)*, Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, ISSN: 1820-0214, DOI:10.2298/CSIS0702081A, Vol. 4, No. 2, 2007, pp. 79-98.

[8] I. Luković, V. Ivančević, M. Čeliković, S. Aleksić, "DSLs in Action with Model Based Approaches to Information System Development", in the book: *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, IGI Global, USA, 2013, ISBN: 978-1-4666-2092-6, DOI: 10.4018/978-1-4666-2092-6, pp. 502-532.

[9] IntegraNova Modeler, Available on: *http://www.integranova.com/*

[10] SOList4UML documentation, available at *http://www.soloist4uml.com/soloist-tutorial*

[11] Milićev D., Model-Driven Development with Executable UML, John Wiley and Sons, July 2009, ISBN 9780470481639