Generating XML Based Specifications of Information Systems

Miro Govedarica[1], Ivan Luković[1], and Pavle Mogin[2]

[1] Faculty of Technical Sciences, Trg D. Obradovića 6, 21000 Novi Sad, Serbia and Montenegro
miro@uns.ns.ac.yu, ivan@uns.ns.ac.yu

[2] Victoria University of Wellington School of Mathematical and Computing Sciences, Wellington, New Zeland
pmogin@mcs.vuw.ac.nz

Abstract. This paper outlines a methodology for designing information systems based on XML. The methodology uses XML DTDs to define the design standards, and the structure and constraints of the design specifications. The result of the design process is a set of valid XML documents that are the specifications of transaction programs and applications of the information system. At the start of a design process, the methodology uses a CASE tool to map user requirements into initial XML specifications. Final design specifications are produced by a sequence of XSL transformations of the initial XML specifications. A key feature of the methodology is that it produces a platform independent design of an information system. To enable an early feedback from users, the methodology uses further XSL transformations that produce an executable prototype of the information system in the Java programming environment.

1. Introduction

The design of a database schema and the corresponding database applications is an important task in the development of an information system (IS). The quality of their design specifications has a great influence on the overall development cost and the exploitation performance of the IS. The development of applications alone adds considerably to the overall development costs of an IS. Namely, each application generally consists of several transaction programs, and developing transaction programs is a time consuming task, usually performed by a number of designers and programmers.

The problems of high development costs and long development time of an IS are resolved, or at least tempered today by applying an appropriate CASE tool. That CASE tool is supposed to support an automated database schema design, and defining and generating application prototypes. There are commercial CASE tools available on the market that partially or fully support application design and generating.

There are also commercially available common software development methodologies on the market that incorporate using specific CASE tools. Examples are *Oracle Custom Development Method* (CDM) [24] with *Oracle Project Management Method* (PJM) [25], and *Rational Unified Process* (RUP) [14], which is based on using *Unified Modeling Language* (UML) [32]. These methodologies define a number of abstract concepts, specification types, and procedures that should be applied in the software development. The structure and the content of these specifications are usually well defined, but they need to adhere to a particular document format, such as MS Word DOC format, or to the particular repository structure of a CASE tool, which is supposed to be used in the software development process. Consequently, the opportunity to exchange such structured specifications between different repositories, or to extend their definitions is often very limited.

The other aspect of the same problem is that the today's CASE tools very often support generating software solutions that are fully dependent on a specific run-time environment. In other words:

- The structure of specifications of transaction programs and applications is tightly coupled to a repository structure and a code generator, and
- A code generator is tightly coupled to the chosen run-time environment.

Thus, selection of a particular runt-time environment is highly influenced by the chosen CASE tool, and vice versa. This mutual relationship may have negative consequences, particularly when a reengineering of the IS, or just its migration to a new software platform is needed. If, for example, there is a need to migrate IS applications to a new software platform, which is based on the completely new IT concepts, the run-time environment will change. We may suppose with a high certainty that the current CASE tool will not support the new platform. Consequently, it should be replaced by a new one that suits the new platform. However, changing the CASE will cause a complex problem of transforming and restructuring IS specifications that must be exported from the old repository and imported into the new one. Consequently, it may increase the overall costs and the development time of the project.

The main goal of the paper is to present a methodology for designing specifications of transaction programs and applications of an IS that are independent of a run-time environment. To achieve that goal, we:

- Define initial formal specifications of transaction programs and applications using a CASE tool, at the start of the methodology. These specifications are formal, since they are created using a notation with a precise syntax and meaning.
- Express the initial formal specifications as XML documents. These XML documents, which we call *XML specifications*, are valid with regard to DTDs that define the structure and constraints of the software specifications.
- Apply a sequence of XSL transformations onto initial XML specifications to produce final XML specifications.

Using XML to express initial and final specifications makes them independent from both the repository of a chosen CASE tool and any run-time environment. In this way, project managers will have more freedom in making decisions regarding the selection of the run-time environment and the CASE tool.

In the design of the specifications of transaction programs, we use a common model of the user interface (UI), and express its specification as a valid XML document. The common UI model enables producing software with a uniform logic and functionality. So, introducing and outlining this common UI model, is another goal of the paper.

Finally, we also use XML specifications for automated generating executable prototypes of transaction programs and applications. Hence, outlining the process of generating prototypes, which is based on using XSL transformations and Java programming environment is the final goal of the paper.

Apart from Introduction and Conclusion, the paper consists of seven sections. Section 2 briefly discusses related works. Section 3 is a short overview of our methodology.

Section 4 outlines developing software specifications using IIS*Case. IIS*Case is a laboratory CASE tool that we used in designing: (i) conceptual specifications, and (ii) implementation specifications of an IS, and in transforming implementation specifications into appropriate initial XML specifications.

Section 5 discusses the UI design and the following two fundamental UI concepts: the UI presentation form and UI functioning logic. There, we introduce a common UI model and identify several characteristic UI form types that may be used in transaction programs. The model is specified using *User Interface Markup Language* (UIML) [33].

The structures of initial XML specifications of applications and transaction programs are considered in Section 6. The following three main components of a transaction program specification are discussed: (i) the data presentation form, (ii) the subschema as an abstract definition of that part of a database schema, which is used by the program, and (iii) the specification of data processing logic.

Section 7 outlines the process of producing final XML specifications of applications and transaction programs. The process is based on an automated transforming the corresponding XML specifications. Transforming is done by merging initial XML specifications with the UIML based UI specification. The transforming rules are built into a number of XSL [5] documents.

Section 8 gives an overview of the process of generating IS application prototypes. There, we discuss the automated generation of software components based on the interpretation of XML specifications of applications and transaction programs. We adapted *Java Render by Harmonia Incorporation®* [31, 33] as the programming and run-time environment.

2. Related Work

This section briefly discusses the works, related to specifying applications, transaction programs and UI models.

Bernstein [1] considers the problem of exchanging information models (i.e. types and structures of software specifications) and proposes the standardization of information models as a solution of the problem. The author states, "It appears that the best approach is to have the standard information model that include core functions that most tool vendors and customers can agree to, and to have tool vendors rely on information model extensibility to add the key features that allow them to differentiate their tools." In the paper, we propose a different approach. We introduce the concept of XML specifications of an IS that are independent of both a CASE tool and a run-time environment. We discuss the roles of ZML and UML in the realization of standard information models in the subsequent paragraphs.

Z specification language is a formal language based on the ISO Z Standard (2002). Generally, it may be used as a standard language for formal specifying transaction programs and applications. There is an XML representation of Z language, called ZML [34, 37]. The advantages of using ZML are as follows. (i) Particular tools do not need to parse Z, directly. It may be done by one tool, and the results may be used by many other tools. (ii) There are many analysis and transformation tools for XML, such as the XSLT language that makes it easy to transform XML files into other formats [37]. The motives and the expected benefits of introducing ZML are the same as those, proposed in this paper. However, we consider Z as being a too general specification language that should be additionally amended for the purpose of IS design in two ways:

– It should be extended by specific abstract concepts, intended for specifying transaction programs and applications in a more declarative way, and
– The design of such Z based specifications should be supported by a visually oriented tool, regarding the fact that it is hard to expect that an IS designer would be able to understand and use mathematically oriented syntax rules of Z language.

In recent years, the efforts to define standard specification techniques go trough UML, as an object-oriented specification language, issued by Object Management Group (OMG). *Meta-Object Facility* (MOF) [4, 19] is also an OMG standard, used as a universal meta-metamodel to describe arbitrary metamodels, such as UML itself. In order to support exchanging concrete models (i.e. software specifications) between the different repositories supporting UML and MOF, *XML Metadata Interchange* (XMI) language [4, 36] is introduced. XMI is an XML representation of MOF. A particular UML model may be transformed into an XML document that conforms to XMI. In [4] XMI is considered "as the most promising model interchange format solution. It fulfills most of the requirements that a good model interchange format should". This allows UML CASE tools or repositories from different vendors to use XMI to exchange UML models [13]. In this way, the selection of a CASE tool may not be tightly related to the selection of a run-time environment.

The goal of *Model Driven Architecture* (MDA), issued by OMG, is automatic generation of program code from particular UML models. In [12] it is stated that "although formal UML models, expressed using Executable UML, often prove appropriate as MDA's platform-independent models, a major drawback of this approach is that these models are too verbose, since Executable UML is kept as general as possible so that it can be used for a wide variety of different domains". Moreover, in [6] the authors state that, "unfortunately, the standard UML metamodel is inadequate for maintaining the consistency between a design model and the corresponding program code. This is mainly because the

UML metamodel considers the whole method body as implementation specific". We agree with the authors of [12] that a possible solution of this problem is to make an extension of the UML metamodel and define domain-specific abstractions.

The main ideas, proposed in this paper and in [12] are similar and focused on specific domains. We consider developing XML specifications of an IS and generating executable IS applications. In [12], the authors consider, as an example, developing database-intensive applications with application logic executed on a database server. In both cases, the specifications should be independent of any run-time environment. The rendering rules, defining the mapping specification components into platform-specific components must be defined to enable automatic generating executable applications.

Many research works have been devoted to the methods, techniques and tools for modeling UI [2, 8, 18, 22, 23, 27, 30]. An open question is how to design and specify a common UI model, which will be suitable for applying in all software applications of an IS. One of the desired features of such common model is independency. The independency here denotes the fact that the UI model should be independent of:

– Any run-time environment, and
– Any formalisms, notations or structuring rules used to design IS specifications.

In this way, in the case of a migration of the current IS applications to a new software platform, an independent UI model will remain unchanged.

Markup languages may also be used for creating UI. Two of their advantages are that they are declarative and extensible. The extensibility denotes the fact that they provide mechanisms to introduce new abstract concepts at meta level. Thus, they may be suitable for designing common UI models, i.e. to provide abstract concepts that will make such models independent [26]. The ability to provide the run-time independence was the reason we adopted XML to describe our common UI.

3. An Outline of The Methodology

The international standards in software engineering, such as ISO 9001 [10] with ISO 9000-3 [9] proposals, and particularly ISO 12207 [11] outline a common software development methodology, but they do not specify a common structure of design specifications for software products, at all. Consequently, a development methodology of a particular project should be defined in the phase of project planning. Also, the structure and the content of design speci-fications must be established by interim standards of a software development project, as a part of the proposed methodology.

The methodology that we propose in the paper is based on a combination of the lifecycle and the prototype approach. In this section, we discuss the most important activities of the methodology and place them in the framework of the lifecycle methodology. As the most important activities, the Fig. 1 depicts:

developing specifications of applications and transaction programs, and generating application prototypes.

In addition, we consider that the following activities should take place in the phase of project planning:

– Defining a common UI model that should specify the structure of presentation forms and functioning logic,
– Defining the structure of specifications of transaction programs and applications,
– Defining the design process of program and application specifications and providing the appropriate software tools, and
– Providing tools for automatic generating application prototypes.

We consider the design of an IS as two parallel groups of activities: (i) the data base design and (ii) the design of transaction programs and applications. According to the life cycle methodology, these activities are performed in the analysis and design phase [20].

According to our approach, conceptual design of an IS is performed in the analysis phase. The main goal of that phase is to transform user requirements into the formal specifications of an IS that are independent of any DBMS or a programming paradigm. Thus, conceptual design of: a database schema, a transaction program, and an application results in a corresponding *formal specification* (see Fig. 1). We propose that the design of these specifications must be performed using an appropriate CASE tool. In this way, the designed specifications are stored in the CASE repository. Consequently, the structures of the aforementioned specifications are defined by the structure of the repository schema and each designed specification conforms to it.

Implementation design of an IS is performed in the design phase. The main goal of that phase is to transform the formal specifications into the equivalent implementation specifications. These specifications are independent of any DBMS and run-time environment. We propose an automatic transforming the conceptual specifications into appropriate implementation specifications (see Fig. 1). In this way, an *application specification* is transformed into the *Initial XML Specification of an Application* (IXSA). An XML DTD named *MenuDTD* [7] defines its structure. A *program specification* is transformed into the *Initial XML Specification of a Program* (IXSP). The UIML DTD and an XML DTD named *SubschemaDTD* [7] define its structure. The initial XML specifications are combined with the *XML Based UI Model* (XBUM) and transformed into the *Final XML Specification of an Application* (FXSA) (see Fig. 1). The XBUM is a UIML document. It represents the common UI model of an IS, which is independent of any run-time environment.

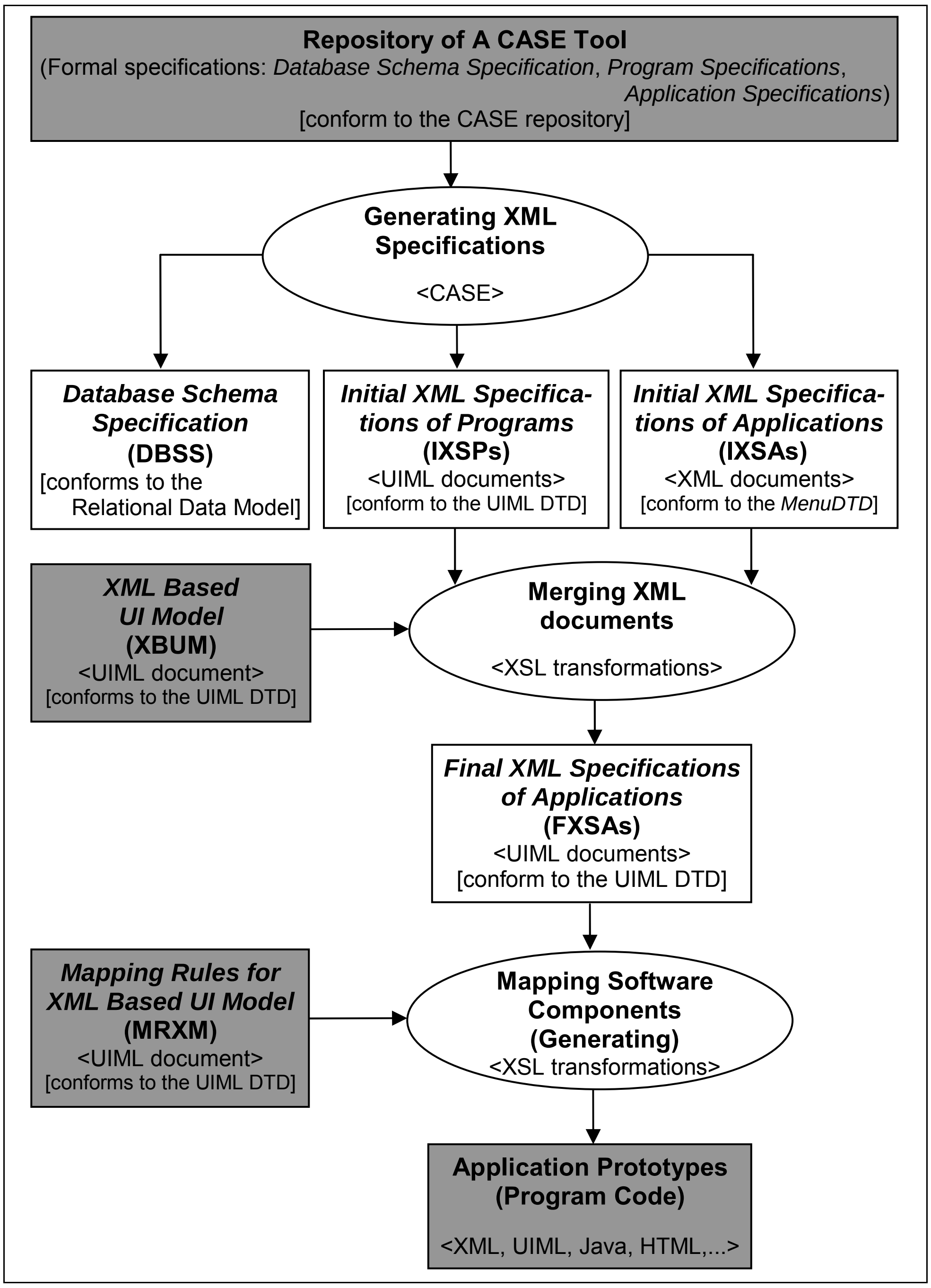

Repository of A CASE Tool
(Formal specifications: *Database Schema Specification*, *Program Specifications*, *Application Specifications*)
[conform to the CASE repository]
Generating XML Specifications
<CASE>
***Database Schema Specification* (DBSS)**
[conforms to the Relational Data Model]
***Initial XML Specifications of Programs* (IXSPs)**
<UIML documents>
[conform to the UIML DTD]
***Initial XML Specifications of Applications* (IXSAs)**
<XML documents>
[conform to the *MenuDTD*]
***XML Based UI Model* (XBUM)**
<UIML document>
[conforms to the UIML DTD]
Merging XML documents
<XSL transformations>
***Final XML Specifications of Applications* (FXSAs)**
<UIML documents>
[conform to the UIML DTD]
***Mapping Rules for XML Based UI Model* (MRXM)**
<UIML document>
[conforms to the UIML DTD]
Mapping Software Components (Generating)
<XSL transformations>
Application Prototypes (Program Code)
<XML, UIML, Java, HTML,...>

Fig. 1. An overview of the process of developing IS specifications and application prototypes

According to the life cycle methodology, programming applications and transaction programs is performed in the build phase [20]. In our approach, we perform programming by an automatic transforming the final XML specifications into executable application prototypes. Prototypes are also specified by means of XML, but they are platform dependent (see Fig. 1). These prototypes may be interpreted by a Java render. A UIML document that we call *Mapping Rules for XML based UI Model* (MRXM) defines how the components of the FXSA will map into the appropriate Java specific run-time components.

4. Developing Software Specifications Using IIS*Case

Using a CASE tool in the development of an IS is a common practice today. We also consider using a CASE tool in applying our methodology being justified due to the following. First, we stress that XML design specifications are usually quite long, even much longer than it would be the corresponding code, written in a particular programming language. Therefore, it is hard to expect that even well-trained and experienced designers would be able to make error-free XML specifications by hand in a short time interval. As the second alternative, suppose designers will use a common tool (like XML Spy) for designing and validating XML documents. Such tool may assist them only to produce formally valid XML specifications, since it is not intended to help them in conceptual design and semantic analysis of design specifications. Thus, there is a little guarantee that such XML specifications will be semantically correct.

In the application of the methodology, we decided to use a CASE tool, named IIS*Case [7, 20, 21, 28]. It is a laboratory software product from the class of integrated CASE tools. IIS*Case is aimed at:

- The conceptual design of a database schema, transaction programs and applications,
- Automated generating relational database schema, satisfying at least the third normal form (3NF), and
- Automated generating application prototypes.

IIS*Case works as a client application over its own repository at the server side. The repository must be implemented as a relational database.

4.1. Conceptual Design of IS Specifications

One of the main motives for developing IIS*Case was to overcome the complexity of identifying, formalizing and specifying database constraints in the database design. To achieve that goal we introduced the form type as the sole concept at the conceptual level. Thus, the conceptual design of an IS is based on the concept of the form type. The form type is a generalization of screen or print forms, used to specify UI. These user forms are mainly derived form the corresponding business documents, which bear information about attributes and constraint in a real system.

The form type is an abstraction very similar to the concept of the object class in the object-oriented approach. It provides formalisms for expressing:

- The structure over the set of attributes of a transaction program,
- The behavioral characteristics of a transaction program,
- Data constraints of various types, and
- The data presentation form layout. [16, 17, 20, 21, 28, 29].

IIS*Case supports creating: (i) a set of form types, and (ii) a number of hierarchical structures over the set of form types. In this way, we design specifications of a database, transaction programs and applications, at the conceptual level.

Each form type represents the *formal specification of a program* at the conceptual level. The structure over the set of attributes and the data constraints represent a database view. A data presentation form represents the layout of the screen or print form of a transaction program. Behavioral characteristics define the operations, i.e. data processing logic that a transaction program may perform over the database. They may include predefined ("standard") database operations: data retrieval, inserting, deleting and updating, and non-standard (specific) operations. The standard operations are specified declaratively.

The structure over the set of attributes, data constraints and behavioral characteristics of a form type represent an *external schema*. The set of all external schemas, created by IIS*Case, represents the *formal specification of a database schema* at the conceptual level.

Each hierarchical structure over a set of form types represents the *formal specification of an application*. It specifies a menu system that provides calls to the appropriate transaction programs. It should include:

- The specification of the menu structure with menu items, and
- The specification of behavioral and visual properties of each menu item.

Behavioral property of a menu item specifies the association of the item with a transaction program, represented by the form type, or with another menu. The visual property of a menu item defines the appearance of the item in the UI.

At the start of the design process, we use IIS*Case to map user requirements into the following conceptual specifications:

- *Conceptual database schema specification*, defined by the set of all external schemas,
- *Conceptual program specifications*, each defined by a form type, and
- *Application specifications*, each defined by a hierarchical structure over the union of a set of form types and a set of menus.

4.2. Implementation Design of IS Specifications

In the process of implementation database design, IIS*Case transforms each external schema into a *subschema*. A subschema is expressed using concepts of the relational data model. Roughly speaking, a subschema consists of a set of relation schemes, a set of interrelation constraints, and a predefined behavior. Each relation scheme of a subschema is a view definition over a base relation scheme. The set of interrelation constraints of a subschema must be stricter or equal to the projection of database interrelation constraints onto the set of the subschema relation schemes. The predefined behavior declaratively specifies the set of basic database operations (such as select, insert, update and delete) that may be performed by a transaction program using the subschema.

The relational database schema of an IS is obtained by the integration of the subschemas. We do not subject to integration the predefined behavior of a subschema. More details about the relationship between an external schema and a subschema, and the integration process may be found in [15, 16, 17, 20, 28]. For the purpose of the paper, it is important that IIS*Case supports generating:

– The implementation specification of a database schema, and
– A set of subschema specifications.

Recall that a conceptual program specification consists of an external schema specification and the specification of data presentation form. A corresponding *implementation program specification* consists of a *subschema specification* and the conceptual *specification of data presentation form,* since we only transform each external schema into a subschema.

The next step of the proposed methodology is to map the formal specifications into the equivalent XML representations. We call them initial XML specifications. IIS*Case supports generating initial XML specifications (see Fig. 2). In this way, it automatically transforms each subschema specification into the *XML Specification of a Subschema* (XSS), each specification of data presentation form into the *Initial XML Specification of data presentation Form* (IXSF), and each application specification into the *Initial XML Specification of an Application* (IXSA). Details of these transformations are presented in [7].

The methodology uses XML DTDs to define the design standards, and the structure and constraints of the XML design specifications. The result of the whole design process is a set of valid XML documents that represent the specifications of transaction programs and applications. For that purpose, we developed a specific XML DTD, named *SubschemaDTD*, which models the abstract structure of a subschema. Each XSS document, generated by IIS*Case, is valid with respect to the *SubschemaDTD*. Section 6.2 of the paper is devoted to XSS. A complete specification of the *SubschemaDTD* may be found in [7, 16]. It complies with the SQL92 standard [3, 20].

In order to define the structure of an IXSF by an XML DTD document, we adopted the UIML DTD [33]. Each IXSF, generated by IIS*Case, is valid with respect to the UIML DTD. Section 6.2 is also devoted to IXSF.

We also developed a specific XML DTD, named *MenuDTD*, which models the common structure of a menu system. It is defined using the concepts of UIML. Each IXSA, generated by IIS*Case, is valid with respect to the *MenuDTD*. Section 6.1 is devoted to IXSA. A complete specification of the *MenuDTD* may be found in [7].

The subsequent sections are devoted to: designing a common UI model (Section 5), initial XML specifications (Section 6), and final XML specifications (Section 7), and generating executable application prototypes (Section 8).

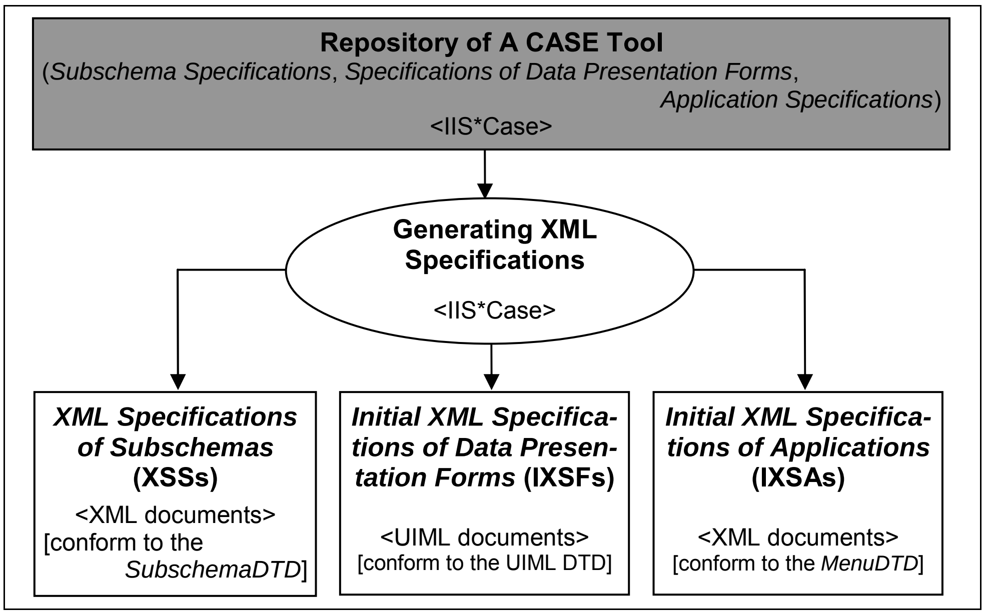

Repository of A CASE Tool
(Subschema Specifications, Specifications of Data Presentation Forms, Application Specifications)
<IIS*Case>
Generating XML Specifications
<IIS*Case>
XML Specifications of Subschemas (XSSs)
<XML documents>
[conform to the SubschemaDTD]
Initial XML Specifications of Data Presentation Forms (IXSFs)
<UIML documents>
[conform to the UIML DTD]
Initial XML Specifications of Applications (IXSAs)
<XML documents>
[conform to the MenuDTD]

Fig. 2. Generating XML specifications from the IIS*CASE repository

5. XML Specification of The User Interface

Adapting a common software development methodology to the given project goals is usually done in the planning phase of the lifecycle methodology. Since International standards in software engineering do not explicitly impose any general UI model for software products, we advocate that specifying a project specific UI model of IS applications should be a compulsory task in the same phase. Although project specific, this UI model should also be *common* to all applications of the IS. Consequently, we call it the *common UI model*.

With respect to the goals of our methodology, we assume that the common UI model satisfies the following conditions:

- It may be applied as a template for developing UI of all IS applications,
- It is fully independent of any CASE tool, and
- It is fully independent of any run-time environment.

It should be noted that the independence of any CASE tool implies that both conceptual specifications stored in the repository of a CASE tool, and the output XML implementation specifications of Fig. 2 should not contain details specifying common UI characteristics of the IS. The generated initial XML specifications will be combined with the common UI model later in the process. In this way, final XML specifications that are obtained from initial XML specifications, will inherit common UI characteristics from the common UI model.

We have chosen XML as a declarative markup language for specifying the common UI model, because it allows formal specifying a common UI model, which is fully independent of a run-time environment. The other often used techniques like specific programming environments, or descriptive textual documents, are either not run-time independent, or the resulting specifications are not formal.

The first task of designing an XML based common UI model is to establish a specific XML DTD or XML Schema, which will define the concepts that are needed to represent specifications of the content and functionality of the UI model. Next, a UI model should be specified as a valid XML document, i.e. an instance of that XML DTD or XML Schema. We call that instance *XML Based UI Model* (XBUM) and it is the XML document that should be combined with the initial XML specifications of Fig. 2.

There are several markup languages aimed at specifying UI, which are based on XML technology. *User Interface Markup Language* (UIML) [31, 33] is a language for specifying UI in a device-independent manner. Creating a UI model using UIML is performed by developing a UIML document, which must conform to UIML DTD or UIML Schema document. UIML provides some abstract concepts that support defining components of a UI model. It also supports defining common visual properties (the presentation style) of UI components. These visual properties are independent of a specific run-time environment.

In our application of the methodology, we adopted the UIML DTD [33] as the DTD defining concepts of the UI model. Using the UIML DTD, we developed the XBUM [7], as its valid instance.

The XBUM specifies: (a) the content, and (b) the embedded behavior (i.e. functionality) of the UI.

The content covers the specifications of: (a1) the structure of UI components and (a2) presentation rules of UI elements. The structure of UI components defines the set of basic UI elements and the structuring rules for building the complex UI components. The specification of the presentation rules defines common visual properties of the UI elements that are independent of a run-time environment.

The embedded UI behavior specifies associations of UI elements, i.e. buttons, menus and menu items, with: (b1) predefined (common) functions, and (b2) additional (specific) functions. The predefined functions are associated with screen or print forms, aimed at presenting data or performing standard database select and update operations. The specific functions are associated with the UI elements of only those transaction programs that perform some specific actions or business rules.

Our XBUM contains the following UI components, representing the form templates:

– The authorization form template that is aimed at generating the authorization form of an application, supporting the authorization of a particular user,
– The title form template that is aimed at generating the first form of an application, appearing after the authorization is successfully done. It enables tabular representation and browsing of database data,

– The query qualification form template that is used to generate forms for defining query selection criteria,
– The data presentation form template that is used to generate read-only forms for presenting just one record of database data,
– The insert form template, aimed at generating forms for inserting in the database one record of data at a time,
– The update form template, aimed at generating forms for modifying one record of database data at a time,
– The delete form template, aimed at generating forms for deleting one record of database data at a time,
– The message form template, aimed at generating forms for presenting the messages to the user, and
– The termination form template, aimed at generating the form notifying that the execution of an application is terminated.

The XBUM did not specify any run-time specific presentation rules or visual properties of UI components. On the other hand, each common UI model should be interpreted in a run-time environment.

Using a markup language to specify a platform independent, common UI model, enables its interpreting in various runt-time environments. One of the advantages of using markup languages is that they are supported by so-called renders that are able to interpret the appropriate markup specifications in a specific run-time environment.

In Section 8 we consider the mapping of the UI components defined in XBUM into the components of a specific run-time environment and outline the way of their interpreting by a specific Java render.

6. Initial XML Specifications of IS

One of the aims of implementation design is to produce initial XML specifications that may be automatically transformed into executable transaction programs and applications.

6.1. Initial XML Specification of An Application (IXSA)

An IXSA provides for specifying the menu system of an application. It is an XML document that is valid with respect to the *MenuDTD* [7]. The *MenuDTD* defines common concepts for specifying a menu system. It enables specifying a tree structure of menu items in a recursive form. The basic concept is *menu item*. Each menu item is a node of a tree structure.

We distinguish the following types of menu items:
– *Main menu* that is the root of a menu tree,
– *Submenu* that is a non-leaf node in a menu tree,
– *Leaf item* that is a leaf node in a menu tree, and

– *Context menu* that is a separate menu, appearing in the context of a screen form (usually invoked by right mouse button click on the form).

For each menu item, behavioral and visual properties are defined.

Behavioral properties of a menu item specify:

– The association of the item with one of the following: a menu, submenu, context menu, or a transaction program, and
– A logical condition that must be satisfied in order to activate the item.

The main visual property of a menu item is the label (title) of the item, which will appear in the UI. Since the *MenuDTD* preserves the independence of the IXSA from the XBUM, there are no overlapping definitions of concepts in the *MenuDTD* and the XBUM. Thus, there is no need to define common visual properties of several menu items in the context of an IXSA. They should be defined in the XBUM, only.

6.2. Initial XML Specification of A Program (IXSP)

According to our approach, each program specification includes:

– A subschema, and
– A data presentation form.

Thus, an IXSP consists of: (i) an XSS, and (ii) an IXSF (see Fig. 2).

An XSS is an XML document that is valid with respect to the *SubschemaDTD* [7, 16]. The *SubschemaDTD* defines: (a) a data structure at schema level of abstraction, with data constraints embedded, and (b) characteristics of the predefined behavior of a transaction program that will use the subschema. The subschema and it's embedded behavior are discussed in Section 4.2 of the paper. Accordingly, an XSS defines the abstract database structure and the predefined data processing logic of a transaction program are specified. The logical design of a subschema is discussed in [16, 17, 28].

Apart from the predefined data processing logic, a program specification may include a definition of specific data processing logic, which defines the specific functionality of a transaction program. It may be expressed in a procedural form. We propose using some formal language ("pseudocode") for this purpose. If it is used, a specific DTD may be provided as a specification of the formal language itself. One of the examples of such a specification language is the *XML Expression Language* (XEXPR) [35]. It would be a matter of further research to provide for defining specific data processing logic and the automatic transforming such specifications into *initial XML specifications of specific data processing logic*.

An IXSF is a UIML document [7]. It specifies a form type. A user will use form type to communicate with a transaction program and create form type instances. The form type will inherit the properties of the appropriate form type templates in the process of generating final XML program specifications. This process will be outlined in Section 7. We recall that each form template is defined in the XBUM and it has an embedded content and behavior (see Section 5). The IXSF defines:

– The content, i.e. the components and the structure of the form type,
– The properties of the specific behavior of the form type components, and
– The specific visual properties of the form type components.

The UIML DTD defines types of components of a form type and the rules for their structuring. A form type component is a UI element, such as: data item, group (i.e. panel, or block) of data items, scrollbar, toolbar, button, etc. There are form type components that must be associated with the appropriate components of a subschema. Such associations are also expressed by this UIML specification. They model the relationship between user form components and a database schema. For example, if a data item of a form type is associated with an attribute of a subschema, then it will hold the database values of that attribute. If a form type component is associated with an appropriate subschema component, then it will inherit its predefined functionality. For example, if a panel of items is associated with a relation scheme of the subschema, then it will be used to support database operations associated with this relation scheme.

Specific visual and behavioral properties are those ones that are not covered by the XBUM. Specific behavioral properties may define some additional data processing or validating procedures that should be performed only within a given form type. For example, it may be a formula for local calculating or validating the value of a form type item that does not correspond to any subschema attribute.

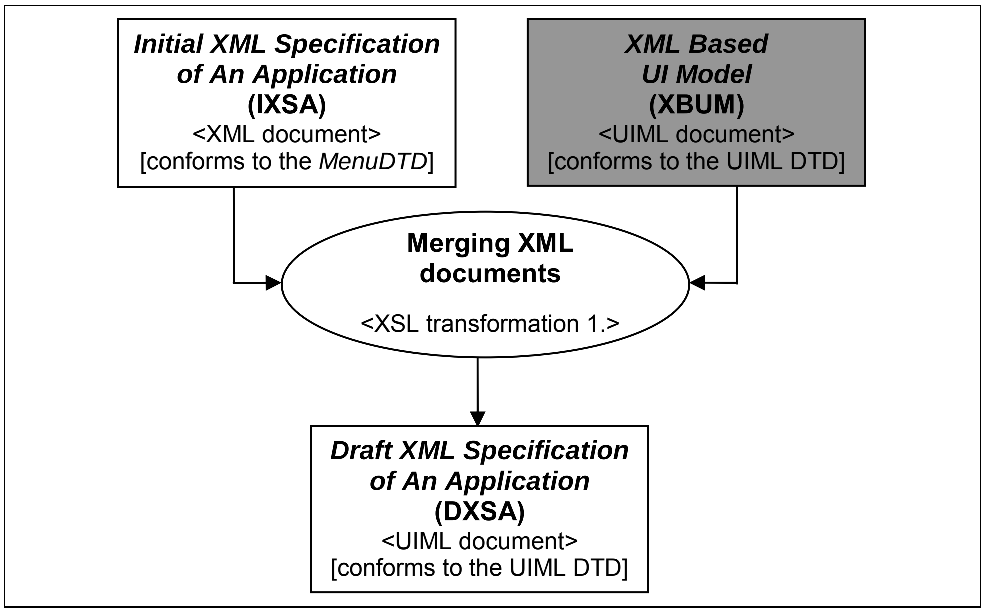
7. Design of Final XML Specifications

This section outlines the process of creating final XML application specifications. The inputs in the process are: (i) an IXSA, (ii) a set of IXSPs and (iii) the XBUM. The output is a *Final XML Specification of an Application* (FXSA), which is generated by applying a sequence of XSL transformations. The overall process has two phases that are shown in Fig. 3 and Fig. 4, respectively.

The first phase generates the first version of a *Draft XML Specification of an Application* (DXSA) by merging appropriate XML and UIML input documents (Fig. 3). The result is a UIML document. The *XSL transformation 1* merges an IXSA and the XBUM and produces a DXSA. The *DXSA* inherits:

– All UI components with the embedded visual and behavioral properties from the XBUM, and
– The menu structure, behavioral and specific visual properties of menu items from the IXSA.

After merging, the DXSA contains the whole specification of the XBUM. A complete definition of the *XSL transformation 1* may be found in [7].



Initial XML Specification of An Application (IXSA)
<XML document>
[conforms to the MenuDTD]
XML Based UI Model (XBUM)
<UIML document>
[conforms to the UIML DTD]
Merging XML documents
<XSL transformation 1.>
Draft XML Specification of An Application (DXSA)
<UIML document>
[conforms to the UIML DTD]

Fig. 3. Producing the draft XML specification of an application by means of an XSL transformation

Fig. 4 depicts the next phase of the process. It consists of two steps that are iteratively applied onto each IXSP at the input, until a FXSA is produced.

In the first step, the *XSL transformation 2* merges the current version of a DXSA with the IXSF of the current program specification. The resulting UIML document is a new version of the DXSA. It inherits all form templates from the current version of the DXSA. The *XSL transformation 2* maps these templates into concrete form types. The new version of the DXSA inherits the content and the structure of these form types, and the behavioral and specific visual properties of the form type components from the user form type, specified by the IXSF.

After the first step, the new version of the DXSA becomes the current one. It will be one of the inputs into the *XSL transformation 3*. In the second step, the *XSL transformation 3* merges the current version of a DXSA with the XSS of the current program specification. In this way, a new version of the DXSA is obtained. The new DXSA is a UIML document that unifies concepts of data presentation form and a subschema.

The XSS augments the functionality of the new DXSA with the predefined data processing logic. Accordingly, the new DXSA inherits from the previous version of the DXSA only those form types that are necessary to support the basic database operations, allowed by the subschema. In addition, the new DXSA inherits from the XSS definitions of relation schemes and data constraints that the transaction program should validate.

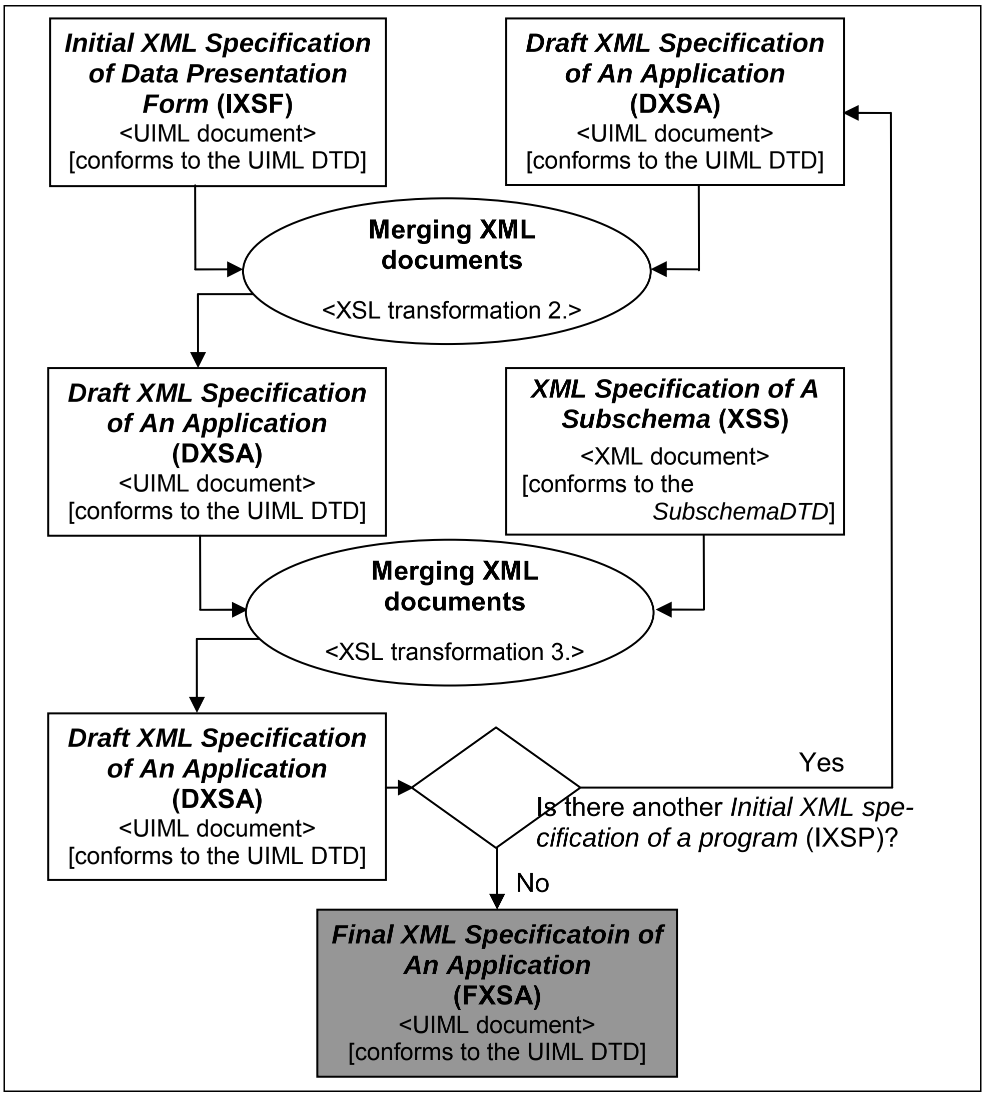

Initial XML Specification of Data Presentation Form (IXSF)
<UIML document>
[conforms to the UIML DTD]
Draft XML Specification of An Application (DXSA)
<UIML document>
[conforms to the UIML DTD]
Merging XML documents
<XSL transformation 2.>
Draft XML Specification of An Application (DXSA)
<UIML document>
[conforms to the UIML DTD]
XML Specification of A Subschema (XSS)
<XML document>
[conforms to the SubschemaDTD]
Merging XML documents
<XSL transformation 3.>
Draft XML Specification of An Application (DXSA)
<UIML document>
[conforms to the UIML DTD]
Yes
Is there another Initial XML specification of a program (IXSP)?
No
Final XML Specificatoin of An Application (FXSA)
<UIML document>
[conforms to the UIML DTD]

Fig. 4. Producing the final XML specification of an application by means of XSL transformations

After the *XSL transformation 3,* the new DXSA contains information about components of a form type, components of a subschema, and their relationships. This enables the corresponding transaction program to validate database constraints and to warn the user about their violation earlier, than if the constraints were validated only at the level of a DBMS. Thus, we make the UI of a transaction program "more reactive".

The next (third) step of our approach would be to create a new XSL transformation, which would support transforming a DXSA with the predefined functionality of transaction programs into a DXSA with a full (predefined and specific) functionality of transaction programs. A DXSA with full functionality may inherit the specific functionality from an *initial XML specification of specific data processing logic* mentioned in Section 6.2. It will be the subject of a further research to specify such an XSL transformation.

After the application of the *XSL transformation 3* on an IXSP, the new version of a DXSA either becomes the current one and a new IXSP will augment it in the next iteration, or becomes a FXSA. A FXSA unifies all structural, functioning and visual characteristics of an IXSA, a set of IXSPs, and the XBUM, and it is fully independent of any run-time environment.

Complete definitions of XSL transformations from Fig. 3 and Fig. 4 may be found in [7].

8. Generating Application Prototypes

One of the main goals of our methodology is to enable a direct interpreting of XML specifications as program prototypes in a chosen run-time environment.

Generally, there are three techniques of transforming design specifications into the executable software applications:

– The manual coding,
– The automatic coding, i.e. full generating, and
– A combination of the previous two techniques.

With respect to the proclaimed goal of the methodology, the technique of full code generating is the only appropriate one. However, applying this technique assumes that the input program specifications must be formal and semantically rich enough to express all necessary details concerning the functionality and UI characteristics of the generated software. We assume here that the formal specifications of programs and applications, and their XML representations satisfy these conditions. Thus, the input in the process of generating an executable application prototype is a *Final XML Specification of an Application* (FXSA). The output will be an XML based application prototype, intended for direct interpreting under a specific run-time environment.

Renders are run-time environments that are able to transform software components of one (source) class into software components of another (executable) class and then interpret them under a given run-time engine. Since our source software components are specified by means of UIML (i.e. XML), there are Java renders, which are able to transform them into the appropriate Java software components that may be executed under *Java Virtual Machine* (JVM) engine. In the application of the methodology, we chose *Java Render by Harmonia Incorporation®* [31, 33] as a programming and run-time environment and adapted it [7] for interpreting generated application prototypes. The main characteristic of that render is that it is able to interpret UIML specifications under JVM engine.

Since the FXSA is platform independent, rendering generally requires:

- Defining mapping rules for transforming components of a FXSA into the appropriate run-time components, and
- Establishing an XSL transformation that will map a FXSA into an XML specification that is interpretable by the rendering software.

In this way, the XSL transformation will play the role of a generator of the complete program code.

If the UIML model of a UI is independent of a run-time environment and we intend to render the UI, we must specify mapping between UI components and run-time software components. UIML provides the syntax for defining mapping between UI components and the appropriate Java software components.

Java Render by Harmonia incorporates only mapping rules and Java software components that enable transforming some of the UIML components into the provided Java software components. These components implement only the visual characteristics of UI components. We used them to enable visual interpreting of the XBUM under JVM. [7]

On the other hand, *Java Render by Harmonia* provides neither Java software components nor the appropriate mapping rules that may support the communication with a DBMS. Therefore, *Java Render by Harmonia* does not support embedded behavioral characteristics of XBUM. Thus, we had to extend it by adding new Java software components intended for supporting the embedded behavioral characteristics of the XBUM. [7]

We extended *Java Render by Harmonia* with the new Java software components, intended for:

- Supporting the screen form fields that should contain and present database data,
- Communication with a DBMS by means of the JDBC protocol and SQL, and
- Validating constraints locally within a transaction program.

Additionally, we defined the mapping rules that enable transforming behavioral characteristics of the XBUM into the appropriate Java software components. The *Mapping Rules for XML based UI Model* (MRXM) is an UIML document. It supplements the XBUM with mapping rules that enable interpreting the XBUM under *Java Render by Harmonia*.

Fig. 5 depicts the process of generating and rendering an *XML application prototype*. The MRXM augments the FXSA by mapping UI components into the appropriate Java software components. The XSL transformation of Fig. 5 is the generator of an *XML application prototype*. An *XML application prototype* is a UIML document, which is adapted for interpreting by *Java Render by Harmonia* under JVM.

It should be noted that the XBUM itself might be merged with MRXM by the XSL transformation. Thus, it would be directly transformed into a stand-alone *XML application prototype*, just like a FXSA. In this way, we enable visual testing the UI model during its design.

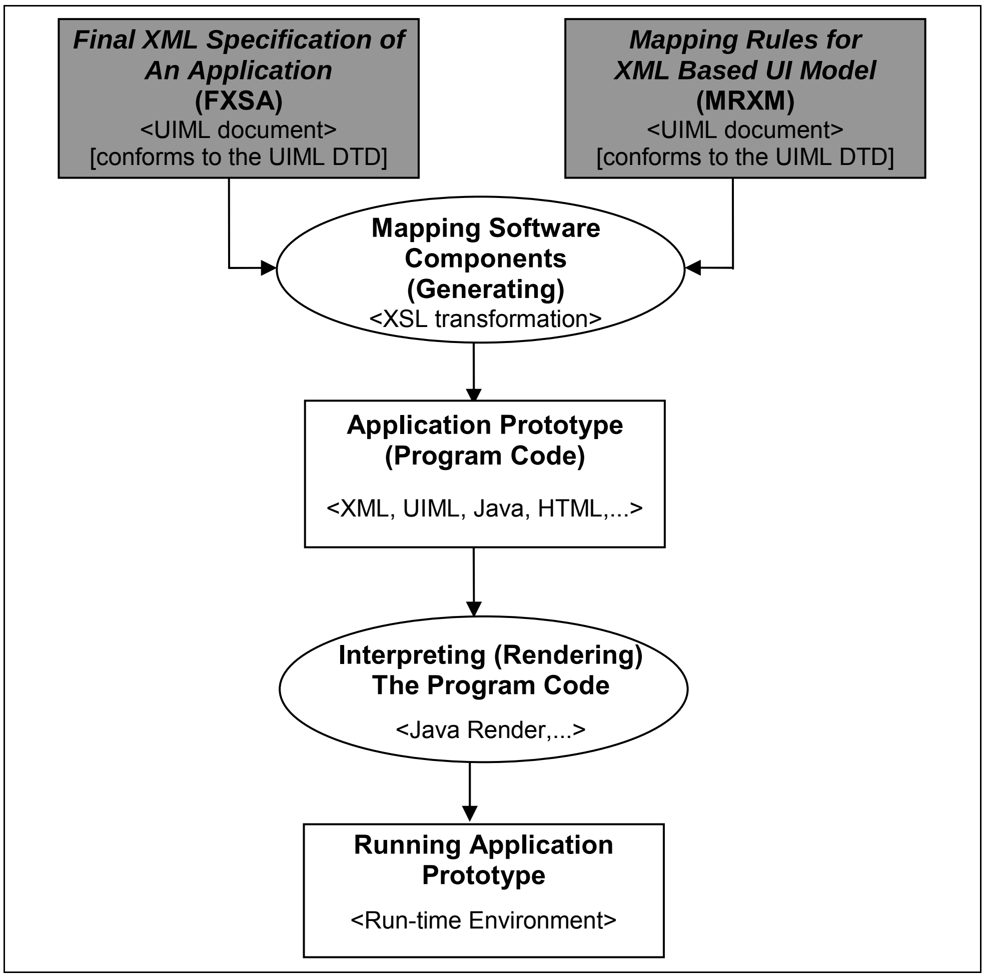

Final XML Specification of An Application (FXSA)
<UIML document> [conforms to the UIML DTD]
Mapping Rules for XML Based UI Model (MRXM)
<UIML document> [conforms to the UIML DTD]
Mapping Software Components (Generating)
<XSL transformation>
Application Prototype (Program Code)
<XML, UIML, Java, HTML,...>
Interpreting (Rendering) The Program Code
<Java Render,...>
Running Application Prototype
<Run-time Environment>

Fig. 5. The process of generating an application prototype

To conclude this section, we stress that the amending the FXSA for interpreting in a specific run-time environment must include:

– Programming software components that will implement the XBUM,
– Designing an appropriate UIML document, which specifies the mapping rules for XBUM, and
– Programming an appropriate XSL transformation that will generate executable application prototypes.

More details concerning transforming FXSAs into the executable application prototypes and applying *Java Render by Harmonia* for interpreting generated prototypes may be found in [7].

9. Conclusion

An approach to formal specifying and automatic generating application prototypes of an IS is presented in the paper. The approach is based on XML, as a language for expressing specifications of IS applications and the applications themselves as the executable software components. The concept of software design and generating, presented in the paper, is built into a specific CASE tool and practically verified.

By applying this approach, designers will be able to generate quickly almost fully functional and highly standardized application prototypes during the design of an IS. Thus, they may use such application prototypes to communicate with the end users, in order to identify early and precisely all user requirements, data structures, business rules and constraints that must be covered by the IS.

One of the advantages of this approach is that the XML design specifications and the UI model are independent of any run-time environment and the repository structure of any CASE tool. In order to achieve this independence, it is necessary to develop the appropriate:

– Software drivers that will generate initial XML specifications from the repository of the chosen CASE tool (see Fig. 2), and
– XSL transformations that will generate software components from the final XML specifications of applications and programs for a specific run-time environment (see Fig. 5).

In this way, reengineering or migrating an IS to a new IT platform does not impose any change to our initial XML specifications, XML based UI model and the XSL transformations for producing final XML specifications.

Our future research will focus on the following two extensions of our methodology:

– Generating fully functional applications by including the definition of specific data processing logic into the structure of program specifications, and
– Transforming definitions of our formal specifications to conform UML meta-model. We believe that this research will also initiate extending the UML meta-model by adding some specific abstractions.

10. References

1. Bernstein, P. A.: Repositories and Object-Oriented Databases. In Proceedings of the 7th Conference on Database Systems for Business, Technology and Web (BTW), Ulm, Germany. Lecture Notes in Computer Science, Springer-Verlag, Berlin, 34-46. (1997)
2. Boswell, D., King, B., Oeschger, I., Collins, P., Murphy, E.: Creating Applications with Mozilla. O'Reilly, USA. (2002)
3. Date, C. J.: A Guide to the SQL Standard. Addison-Wesley-Publishing-Company, USA. (1994)
4. Denis, S. G., Schauer, R., Keller, K. R.: Selecting a model interchange format: The SPOOL Case Study. In Proceedings of the 33rd Hawaii International Conference on System Sciences, Maui, Hawaii, 1-10. (2000)

5. Extensible Markup Language (XML) and Extensible Stylesheet Language Family (XSL). W3C World Wide Web Consortium. [Online]. Available: http://www.w3.org/. (current: November, 2003)
6. Gorp, P. V., Stenten, H., Mens, T., Demeyer, S: Towards automating source-consistent UML Refactorings. In Proceedings of the 6th International Conference on Unified Modeling Language (UIML), San Francisco, USA. Lecture Notes in Computer Science, Springer-Verlag, 144-159. (2003)
7. Govedarica, M.: An Automated Development of Information System Application Prototypes. Ph.D. Thesis. University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Yugoslavia. (2001)
8. Hartson, H. R., Hix, D.: Human-Computer Interface Development: Concepts and Systems for Its Management. ACM Computing Surveys, Vol. 21, No. 1, 5-92. (1989)
9. ISO 9000-3:1997, Quality management and quality assurance standards -- Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software. International Organization for Standardization (ISO). [Online]. Available: http://www.iso.org/. (current: December, 2003)
10. ISO 9001:2000, Quality management systems – Requirements. International Organization for Standardization (ISO). [Online]. Available: http://www.iso.org/. (current: December, 2003)
11. ISO/IEC 12207:1995, Information technology -- Software life cycle processes. International Organization for Standardization (ISO). [Online]. Available: http://www.iso.org/. (current: December, 2003)
12. Kovse, J., Härder, T.: DSL-DIA - An Environment for Domain-Specific Languages for Database-Intensive Applications. In Proceedings of the 9th International Conference on Object Oriented Information Systems (OOIS), Geneva, Switzerland. Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, 304-310. (2003)
13. Kovse, J., Härder, T.: Generic XMI-Based UML Model Transformations. In Proceedings of the 8th International Conference on Object Oriented Information Systems (OOIS), Montpellier, France. Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, 192-198. (2002)
14. Kruchten, P.: The Rational Unified Process: An Introduction (2nd Edition). Addison-Wesley Pub. Co. (2000)
15. Luković, I., Mogin, P.: On The Role of Subschema as A Component of The Implementation Program Specification. In Proceedings of the 6th Symposium on Computer Science and Information Technologies YUINFO, Kopaonik, Yugoslavia, CD ROM. (2000)
16. Luković, I., Mogin, P., Govedarica, M., Ristić, S.: The Structure of A Subschema and Its XML Specification. In Proceedings of the 13th International Conference on Information and Intelligent Systems, Varaždin, Croatia, 45-56. (2002)
17. Luković, I., Ristić, S., Mogin, P.: A Methodology of A Database Schema Design Using The Subschemas. In Proceedings of IEEE International Conference on Computational Cybernetics, Siofok, Hungary, CD ROM. (2003)
18. Luo, P., Szekely, P., Neches, R.: Management of Interface Design in Humanoid. In Proceedings of INTERCHI '93, Amsterdam, The Netherlands, 107-114. (1993)
19. Meta-Object Facility (MOF), V.1.4. Object Management Group, Inc (OMG). [Online]. Available: http://www.omg.org/technology/documents/formal/mof.htm (current: December, 2003)
20. Mogin, P., Luković, I., Govedarica, M.: The Principles of Database Design. University of Novi Sad and MP Stylos, Novi Sad, Serbia and Montenegro. (2000)

21. Mogin, P., Luković, I., Karadžić, Ž.: Relational Database Schema Design and Application Generating Using IIS*CASE Tool. In Proceedings of International Conference on Technical Informatics, Timisoara, Romania, Vol. 5, 49-58. (1994)
22. Myers, B. A.: User Interface Software Tools. ACM Transactions on Computer-Human Interaction, Vol. 2, No. 1, 64-103. (1995)
23. Myers, B. A., Ferrency, A., McDaniel, R., Miller, R. C., Doane, P., Mickish, A., Klimovitski, A.: The Amulet V2.0 Reference Manual. Technical Report CMU-CS-95-166-R1. Carnegie Mellon University, Computer Science Department. (1996). [Online]. Available: http://www.cs.cmu.edu/~amulet (current: November, 2003)
24. ORACLE Custom Development Method Handbook, Rel. 1.0. ORACLE Corporation. (1996)
25. ORACLE Project Management Method Handbook, Rel. 1.0. ORACLE Corporation. (1996)
26. Phanouriou, C.: UIML: A Device-Independent User Interface Markup Language. PhD Thesis. Virginia Polytechnic Institute and State University, Blacksburg, Virginia. (2000)
27. Phanouriou, C., Abrams, M.: Transforming Command-Line Driven Systems to Web Applications. Computer Networks and ISDN Systems, Vol. 29, 1497-1505. (1997)
28. Ristić, S.: A Research of Subschema Consolidation Problem. Ph.D. Thesis. University of Novi Sad, Faculty of Economics, Subotica, Serbia and Montenegro. (2003)
29. Ristić, S., Mogin, P., Luković, I.: Specifying Database Updates Using A Subschema. In Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems, Assiut – Luxor, Egypt, 203-212. (2003)
30. Szekely, P., Luo, P., Neches, R.: Beyond Interface Builders: Model-Based Interface Tools. Human Factors in Computing Systems, Proceedings of INTERCHI '93, Amsterdam, The Netherlands, 383-390. (1993)
31. UIML Presentations and Examples. [Online]. Available: http://www.harmonia.com/resources/ (current: November, 2003)
32. Unified Modeling Language (UML), V.1.5. Object Management Group, Inc (OMG). [Online]. Available: http://www.omg.org/technology/documents/formal/uml.htm (current: December, 2003)
33. User Interface Markup Language (UIML), Draft Specification, Version 2.0. Harmonia Inc. (2000). [Online]. Available: http://www.uiml.org/specs/index.htm (current: November, 2003)
34. Utting, M., Toyn, I., Sun, J., Martin, A., Dong, J. S., Daley, N., Currie, D.: ZML: XML Support for Standard Z. In Proceedings of the 3th International Conference of B and Z Users, Turku, Finland. Lecture Notes in Computer Science, Springer-Verlag Heidelberg, 437-456. (2003)
35. XEXPR – A Scripting Language for XML. Copyright ©2000 eBusiness Technologies, Inc. (2000). [Online]. Available: http://www.w3.org/TR/2000/NOTE-xexpr-20001121/ (current: November, 2003)
36. XML Metadata Interchange (XMI). Object Management Group, Inc (OMG). [Online]. Available: http://www.omg.org/technology/documents/formal/xmi.htm (current: December, 2003)
37. ZML: An XML markup for the Z specification language. The Community Z Tools project. [Online]. Available: http://czt.sourceforge.net/zml/. (current: December, 2003)

Miro Govedarica graduated at the Faculty of Civil Engineering in Sarajevo in 1987, where he received a B.Sc. degree. He finished his M.Sc. studies at the University of Novi Sad, Faculty of Technical Science in 1998, and he completed his Ph.D. thesis at the same University in 2001. Currently, he holds the position of an Assistant Professor at the Faculty of Technical Sciences, where he lectures in Computer Science and Informatics courses. His research interests are in the area of Information System Design, Geo-Information Systems and Object Oriented Software Engineering. He published extensively and he is the author or co-author of one book and 40 papers.

Ivan Luković received his B.Sc. degree in Informatics from the Faculty of Military and Technical Sciences in Zagreb in 1990. He completed his M.Sc. at the University of Belgrade, Faculty of Electrical Engineering in 1993, and his Ph.D. at the University of Novi Sad, Faculty of Technical Sciences in 1996. Currently, he works as an Associate Professor at the Faculty of Technical Sciences at the University of Novi Sad, where he lectures in several Computer Science and Informatics courses. His research interests are related to Information System Design and Database Systems. He is the author or co-author of over 40 papers and 4 books in the area.

Pavle Mogin received his B.Eng.(Honours) degree from the University of Belgrade, Faculty of Electrical Engineering in 1964. He completed his Ph.D. at the University of Nis in 1974. Currently, he holds the position of a Senior Lecturer at the University of Wellington, Faculty of Science, where he lectures Computer Science courses. His research interests are in the area of Database Systems. He is the author or co-author of six books and over 80 papers in the area.