The XIS Generative Programming Techniques

Alberto Rodrigues da Silva, Gonçalo Lemos, Tiago Matias, Marco Costa

alberto.silva@acm.org, goncalo.lemos@fordesi.pt,
tiago@millennium.net.dhis.org, marco.costa@tagus.ist.utl.pt
INESC & IST (Technical University of Lisbon)
Rua Alves Redol, nº 9, 1000-029 LISBOA, PORTUGAL

Abstract. *XIS[1] is a R&D project which main mission is to analyze, develop and evaluate mechanisms and tools to produce information systems from a more abstract, high-level, efficient and productive way than it is done currently. XIS project is influenced by MDA reference model, and is mainly based on three principles: namely high-level models specification; generative programming techniques; and it is component-based architecture-centric. XIS is not a conceptual research plan, it is a working on project with concrete results and produced systems. In this paper we detail the generative programming techniques used in the XIS project as well as the discussions and main decisions tackled on. Finally, we present the main conclusions, the relationship between XIS and MDA, and the work that will be handled in the near future.*

1. Introduction

The vision of the XIS approach is that the emphasis of software development projects should be stressed in the project and design activities, and consequently the effort in production activities, like programming, should be minimized and performed as automatically as possible [3,w2]. Broadly, as suggested in the Figure 1, XIS approach receives system requirements (e.g., functional, non-functional and development requirements) as its main *input*, and produces a set of artifacts (e.g., source code, configuration scripts or data scripts) as its main *output*.
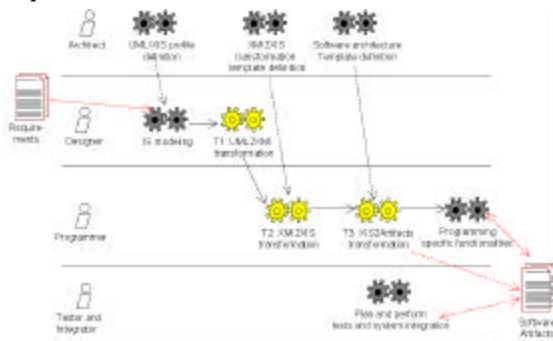

Figure 1. Key elements of the XIS approach.

---

[1] XIS is a name, not an acronym.

XIS approach is strongly based on three principles, namely high-level models specification, generative programming techniques; and it is component-based architecture-centric. An introductory vision and discussion of the XIS' more relevant principles were made previously in another paper [3]. In this paper we detail the generative programming techniques used in the XIS project as well as the main decisions tackled on.

This paper is structured around five sections. Section 1 describes the context and the motivation of the XIS project. Section 2 is an overview of the XIS project and its main elements, namely the XIS approach, the XIS platform, the XIS/UML profile and the XIS/XML specification language. Sections 3 and 4 discuss with some detail two best practices adopted by the XIS approach: it is based on based on models specification (Section 3) and it is based on generative programming techniques (Section 4). Finally, Section 5 concludes the paper and discusses the work that should be handled in the near future.

2. Overview of XIS' Generative Programming Techniques

One relevant principle adopted by the XIS project involves the use of generative programming techniques [4,w1,w4,w5]. This best practice is naturally interconnected with the two others (i.e., based on models specification and component-based architecture-centric), functioning as a "glue" between them [3]. Generative programming techniques automatically transform system specifications into a set of final software artifacts taken into account a given architecture [5,6,7].

An interesting point of the XIS approach is the theoretical possibility to produce code in different programming languages, according a set of software architectures and frameworks, as it is suggested by the MDA philosophy [w6]. That issue will be researched and evaluated in future work.

As illustrated previously in Fig. 1, the XIS automatic generative process consists in the consecutive application of three transformations, which can be formalized by the next expression:

```
IS = T3(T2(T1(XIS/UML-Model, XIS/UML-Profile),
XIS/XSL), SoftArch)
```
Or separately by:

```
XIS/UML-Model      (defined through a CASE-Tool and based on the
XIS/UML profile)

XMI-Model = T1(XIS/UML-Model, XIS/UML-Profile)

XIS/XML-Model = T2(XMI-Model, XIS/XSL)

IS (FinalSystem) = T3(XIS/XML-Model, SoftArch)
```
Where:

- **XIS/UML-Model** corresponds to the original model specified according to XIS/UML profile.
- **T1** is the first transformation. T1 transforms the XIS/UML-Model into the equivalent model in XMI format. XMI parsers, such as XMIToolkit [w3], available nowadays by the majority of modern CASE tools, support naturally T1 transformation.
- **T2** is the second transformation. T2 transforms the model specified in XMI format into an equivalent model in the XIS/XML format. This second transformation is convenient because the XMI models are too large, complex and hard to manipulate and to read or understand by human beings as well as by computer tools (see discussion of Section 4). T2 is performed by a XSLT parser and based on a developed XIS/XSL script.
- **T3** is the third transformation. T3 automatically produces multiple artifacts from the models specified in XIS/XML regarding a given software architecture (**SoftArch**). The key element of the T3 transformation is a generic code generator, called XISGenerator, which provides some interesting properties such as generality, modularity and flexibility and support for different software architectures.
- **IS** is the final system and consequently it consists of a suite of multiple and integrated artifacts such as source code, configuration, SQL scripts, HTML, image and documentation files.

3. First Generative Programming Technique: T1

T1 is a model-to-model transformation. It transforms the XIS/UML-Model, including the used stereotypes and tag values defined by the XIS/UML Profile, into the equivalent model in XMI format. After T1 the visual diagram models are translated into a textual description of the domain specification.

The model in the XMI format is tool independent, and can be exchanged over the Internet in a standardized way, thus bringing consistency and compatibility to applications created in collaborative environments. The defined XMI-Model combines the benefits of the web-based XML standard for defining, validating, and sharing document formats on the web with the benefits of the object-oriented UML, which provides application developers with a common language for specifying, visualizing, constructing, and documenting objects and business models.
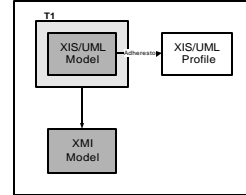

*Fig. 2. **The T1 transformation.***

This is the simplest artifact transformation in the process. Despite this, to support the explanation of the next transformations, we introduce an example of a XIS/UML-Model. The purpose of this very simple application, MyAddressBook, is to manage a group of contact names and e-mails as suggested in the following figures.
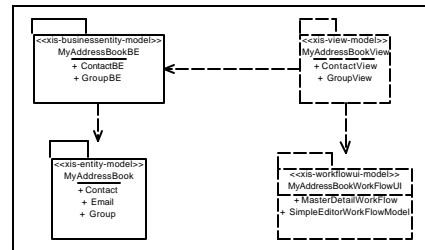

*Fig. 3. **MyAddressBook's XIS/UML Model – Package View.***

There are three domain entities, Contact, Email and Group, classified with the stereotype <<xis-entity>>. These are grouped in two business entities ContactBE and GroupBE. The first is composed by the Contact entity as "master", Email as "detail" and Group as "lookup". The second is only composed by the Group entity.
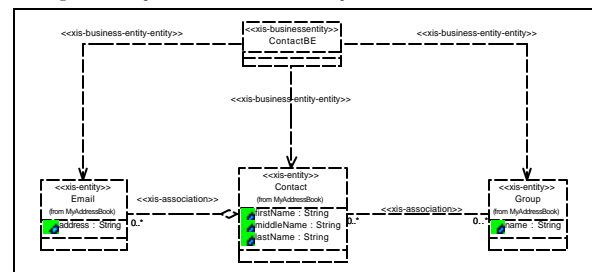

*Fig. 4. **MyAddressBook 's Business Entity Model.***

We defined two different user interface workflows to support editing processes: (1) a master-detail workflow for the ContactBE management, and (2) an usual CRUD (Create-Read-Update-Delete) editor, for GroupBE. An aspect of the workflow-ui modeling is depicted in the Fig. 5.
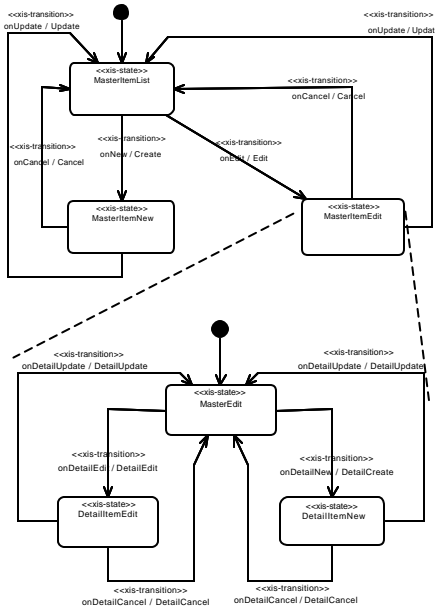
*Fig. 5. **An aspect of the MyAddressBook's Workflow-UI Model***

4. Second Generative Programming Technique: T2

T2 is another model-to-model transformation. **T2** transforms the model specified in XMI format into an equivalent model according to the XIS/XML format.

```
<UML:CompositeState    xmi.id="G.38"    name="MasterEdit"    visibility
="public">
  <UML:CompositeState.subvertex>
    <UML:SimpleState    xmi.id="G.57"    name="DetailItemEdit"    visibility
="public"/>
      <UML:SimpleState    xmi.id="G.62"    name="DetailItemNew"    visibility
="public"/>
      <UML:SimpleState    xmi.id="G.67"    name="MasterItemEdit"    visibility
="public"/>
  </UML:CompositeState.subvertex>
</UML:CompositeState>
<UML:SimpleState    xmi.id="G.43"    name="MasterItemList"    visibility
="public"/>
<UML:CompositeState    xmi.id="G.48"    name="MasterNew"    visibility
="public"/>
      <UML:CompositeState.subvertex>
        <UML:SimpleState xmi.id="G.57"    name="DetailItemEdit" visibility
="public"/>
        <UML:SimpleState    xmi.id="G.62"    name="DetailItemNew"
visibility="public"/>
        <UML:SimpleState    xmi.id="G.67"    name="MasterItemNew"
visibility="public"/>
      </UML:CompositeState.subvertex>
</UML:CompositeState>
```

*Fig. 6. **Example of the XMI-Model – the Master-Detail editor workflow states (adapted)***

This second transformation is convenient because the XMI models are too large, complex and hard to manipulate and to read or understand by human beings as well as by computer tools (as is very well understandable from Figs. 6 and 7). Besides that, the

XMI model contains a lot of information that won't be used when generating code. **T2** is performed by a XSLT parser and based on a developed XIS/XSL script that filters out the needed subset of the information contained in the model. Hence, T2 can be eventually classified following the TEMPLATES+FILTERING generator pattern [w4].

```
<UML:Transition xmi.id="G.39" name="" visibility="public">
    <UML:Transition.source>
        <Behavioral_Elements.State_Machines.StateVertex    xmi.idref
="G.38"/>
    </UML:Transition.source>
    <UML:Transition.target>
        <Behavioral_Elements.State_Machines.StateVertex    xmi.idref
="G.43"/>
    </UML:Transition.target>
    <UML:Transition.effect>
        <UML:ActionSequence    xmi.id="XX.37"    name=""    visibility
="public">
            <UML:ActionSequence.action>
                <UML:UninterpretedAction    xmi.id="XX.36"    name
="Cancel" visibility="public"/>
            </UML:ActionSequence.action>
        </UML:ActionSequence>
    </UML:Transition.effect>
</UML:Transition>
```

*Fig. 7. **Example of the XMI-Model – an editor workflow state transition (adapted).***

The XIS/XML-Model, obtained after **T2,** is a textual description, just like the XMI-Model used as this transformation input. It is equally based on UML elements, but it is filtered only with the relevant ones, that will be used for the code generation. Even in the selected elements there is an attribute selection.

```
<xis-state-list>
    <xis-state name="MasterItemList" kind="Initial"/>
    <xis-state name="MasterEditMasterItemEdit"/>
    <xis-state name="MasterEditDetailItemNew"/>
    <xis-state name="MasterEditDetailItemEdit"/>
    <xis-state name="MasterNewMasterItemNew"/>
    <xis-state name="MasterNewDetailItemNew"/>
    <xis-state name="MasterNewDetailItemEdit"/>
</xis-state-list>
```

*Fig. 8. **Example of the XIS/XML-Model – the Master-Detail editor workflow states.***

```
<xis-action-list>
    <xis-action name="Cancel">
        <xis-map-list>
            <xis-map source="MasterItemEdit" target="MasterItemList"/>
            <xis-map source="MasterItemNew" target="MasterItemLst"/>
        </xis-map-list>
    </xis-action>
</xis-action-list>
```

*Fig. 9. **Example of the XIS/XML-Model – an editor of the workflow state transition.***

In this transformation, common UML information, as well as those provided following the `XIS/UML-Profile` (i.e, specific stereotypes and tag-values), are conveniently filtered and added to the elements of the `XIS/XML-Model`. Figs. 8 and 9 give some simple examples of the `XIS/XML` generated code.

The `XIS/XML-Model` is much more compact than the `XMI-Model` (as easily understandable if you compare Figs. 6 and 7 against Figs. 8 and 9) and its elements are instances of the `XIS/UML-Profile`, which will simplify the model manipulation when performing the final T3 transformation.
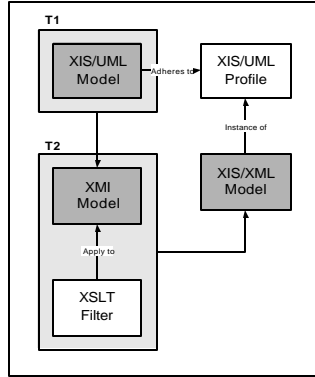

*Fig. 10. **The T1 and T2 transformations.***

5. Third Generative Programming Technique: T3

T3 is a model-to-code transformation. **T3** produces automatically multiple artifacts from the models specified in XIS/XML and taking into account a given software architecture (**SoftArch**). A generic code generator, called XISGenerator is the key pivot of the T3 transformation. T3 can be better recognized according to the TEMPLATES+METAMODEL generator pattern [w4].
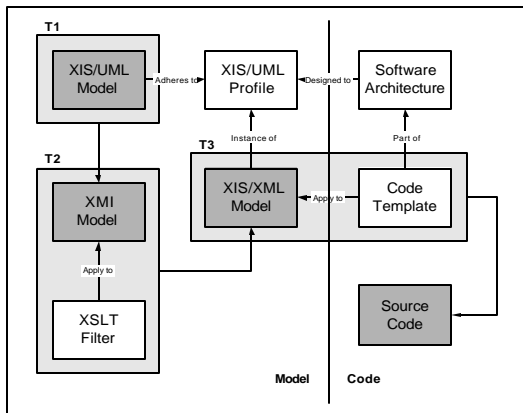

*Fig. 11. **The T1, T2 and T3 transformations.***

The target software architecture is defined by a set of artifacts called templates. These artifacts are reasonable small and self-contained modules that work on the XIS/XML-model's elements. The templates are defined as engines that write different artifacts from instances of the XIS meta-model elements. The XIS meta-model acts as a contract between the architecture design, the template production and the captured domain model, enabling these activities to be conviniently separated and independently executed.

When the XISGenerator is executed each template is fed with a XIS document and produces source code or other artifacts, such as deployed scripts, SQL code, or documentation. For example, consider the following snippet of XIS/XML code produced by **T2** and fed to **T3** as suggested in Fig. 12.

```
<xis-entity name="Contact">
    <xis-entity-attribute-list>
        <xis-entity-attribute name="firstName" type="String" size="20"
isPkey="false"
            allowNull="false"/>
        <xis-entity-attribute name="middleName" type="String" size="20"
            isPkey="false" allowNull="false"/>
        <xis-entity-attribute name="lastName" type="String" size="20"
isPkey="false"
            allowNull="false"/>
    </xis-entity-attribute-list>
</xis-entity>
(…)
<xis-busines-entity name="ContactBE">
    <xis-business-entity-entity-list>
        <xis-business-entity-entity entity="Contact" role="Master" index="0"
            alias="Contact"/>
        <xis-business-entity-entity entity="Email" role="Detail" index="1"
            alias="Emails"/>
        <xis-business-entity-entity entity="Group" role="LookUp" index="0"
            alias="Group"/>
    </xis- business-entity-entity-list>
</xis- business-entity>
(…)
<xis-view name="ContactView" model=" ContactBE "
    workflow="MasterDetailEditor "/>
```

*Fig. 12. **XIS/XML code produced by** T2 **transformation.***

This document is then passed to the templates that define a chosen architecture. A typical template might be as shown in Fig. 13.

```
public void generate(XisDoc doc)
{
 foreach (XisView view in doc.XisViewList) {
   createFile(view.modelName, ".java");
   writeln ("public class "+view.modelName+" {");
   XisBusinessEntity model = new XisBusinessEntity(doc, view.modelName);
   XisEntity masterEntity = model.getmasterEntity();
   foreach (XisAttribute attr in masterEntity.xisAttributeList)        {
       writeln("private "+attr.type+" "+attr.name+";");     }
   foreach (XisAttribute attr in masterEntity.xisAttributeList)        {
       writeln("public "+attr.type+" get"+attr.name+"() {");
       writeln("return (this."+attr.name+");");
       writeln("}");
       writeln("public void set"+attr.name+"("+attr.type+" "+attr.name+") {");
       writeln("this."+attr.name+" = "+attr.name+";");
       writeln("}");
       }
   writeln("}");
}}
```

*Fig. 13. **The template's piece of code used by T3 transformation.***

For this particular example, the output would be source code defining a class called `ContactForm`. Below is a snippet of the generated code.

```java
public class ContactForm {
    private String firstName;
    private String middleName;
    private String lastName;
    public String getfirstName () {
    return (this. firstName);
    }
    public void setfirstName (String firstName) {
      this.firstName = firstName;
    }
    public String getmiddleName () {
        return (this.middleName);
    }
(...)
```

*Fig. 14. **Java source code produced by T3 transformation**.*

With trivial manipulation of a DOM object containing the XIS document, the art of writing XIS's templates can be a straightforward task.

6. Generative Programming Applied

6.1 Best Practices

Iterative and incremental approach

First of all, a generative approach should be iterative and incremental. A well-known problem of classic generative approaches is the difficulty to delimit their action because they usually adopt "blind" or "brut-force" strategies: if there is a new requirement or feature, "all" the involved artifacts are regenerated without any criteria. For relatively small systems the "brut-force" approach can be feasible, however, an increasing number of systems, tend to be large and complex with a disparate number of shared components. Consequently, in those systems, classic generative approaches become promptly impractical. In order to cope with these issues a clever approach is recommended based on the iterative and incremental model. In simple and practical terms, the programmers or architects define their own specific generative processes, which are a set of specific generative steps. Thus, for each generative step, the architects specify the artifacts that would be generated.

Shared Repository Support

Another good practice is to make the models and programmers' or teams' specific generative processes persistant in a repository. With shared repository models and processes they can be integrated in a better and deeper way. It is possible to define lists of components based on model subsets that will be associated with generative steps in order to refine a generative domain and improve control of iteration's impact on the application development. Using such a repository also offers other good practices like dependency management, meta-model extensibility, generation execution history, and so on.

Integration of Generated with Non-Generated Code

A problem that may arise is the integration of generated with non-generated code. It could not be easy to understand the generated code. Not as much because of code layout, format and style, since any generator can write well-formatted code, but with the dealing of the low-level details of the generated code. However, in most cases, it is possible to use it like a black box framework, which is only used and not read by the programmer.

A few approaches exist in OO scenarios to successfully integrate generated and non-generated code. Generated code can simply call non-generated code contained in libraries, which is an important best practice, since it leads to generating few code as possible and rely on pre-implemented components that are called from the generated code. In another approach, a non-generated framework can call generated parts. To make this approach more practicable, such framework could be implemented against abstract classes or interfaces which the generated code implements.

Another possible approach is to subclass non-generated classes in generated code. These base classes can contain useful generic methods that allow the generated code to be more restricted and compact. The base classes can also contain abstract methods that will be implemented by the generated subclasses. Well-known design patterns like FACTORIES and TEMPLATE METHOD [w4] can be applied to achieve the goals described above.

6.2 Benefits and Pitfalls

On the topic of architecture "dissemination" the benefits of code generation are twofold. First, developers save a considerable amount of time and effort writing code allowing them to focus more on designing and developing business logic. Second, the developers' preference of generated over custom "hand written" code guarantees the architect that the design solutions used respect the devised architecture for the application. This leads to good design homogeneity throughout the application, which has several advantages, such as to allow developer interchange between different teams and projects. However, it is advised to write code generation templates with very high quality levels. Architects cannot neglect template testing before developers start using them to generate production code. Although the correctness of a code template may be hard to prove, once it's guaranteed, all the generated code can be considered tested at development time.

5

One pitfall is to consider the need for reverse engineering in the transformations that compose the XIS process, especially in T3. Questions like "What if I change the code? My changes shouldn't be reflected in the model or templates?" frequently arise. This generative approach doesn't use what is known as implementation (or platform specific) models, preferring more compact and abstract design models. The implementation models offer a poor overview since the actual important information is masked by the flood of details and the way from the analysis model to the implementation is very long with no well defined milestones in the middle. Usually, for cost reasons, design changes are made in the source code, using reverse engineering to update the implementation model, ignoring unclear effects in more abstract interim issues. Doing this, by the end of the project, there is no technical documentation that is more abstract than the source code itself. Using code templates, the design phase ends much earlier and could be started when the meta-model is defined, not by the analysis phase ending, promoting a clearer role model with corresponding responsibilities in the development process. The implementation cost is also considerably reduced because a large portion of the code is done fully automatically. The code templates together, representing the architecture design could be reused for different architectures with shared features, improving even more the development processes.

7. Conclusions and Future Work

The vision subjacent to the XIS project is a revisitation of existing expectations that in the past did not have so much success: *the ideia that the building of information systems would be performed almost automatically starting from high-level and platform independent specifications.* Meaning in the end, that classic tasks like programming should be performed almost automatically or at least not so manually intensive as it is done nowadays and in general, with all the well-known costs and problems [1,2,w6].

The proposed approach starts to be known increasingly by the expression "software development from models and component-based architectures" and we believe *this time* would be possible to succed in that vision atending the maturity of software enginneering and its respectives technologies. This general approach is being progressively discussed and promoted in different context such as the OMG and in particular its MDA (*Model Driven Architecture*) workgroups [w6]. MDA is also increasingly analysed and promoted by the main software companies. However, there are some subtle differences between the MDA and the XIS approach that were analyzed accordingly in [3].

Finally, **XIS is not a conceptual research plan, it is a working on project with concrete results and produced systems**. (The interested reader is invited to contact the author or visit the project web site [w2] for more information).

References

[1]   G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.

[2]   P. Kruchten. *The Rational Unified Process*, 2nd edition. Addison Wesley, 2000.

[3]   A. Rodrigues Silva. *The XIS Approach and Principles*. Proceedings of the 29th Euromicro Conference. (Euromicro Conference'2003). IEEE Computer Society.

[4]   J. Craig Cleaveland. *Program Generators with XML and JAVA.* Prentice-Hall, 2001.

[5]   C. Hofmeister, R. Nord, D. Soni. *Applied Software Architecture.* Addison Wesley, 1999.

[6]   *The Software Patterns Series*. Addison Wesley, 1996-2002.

[7]   M. Juric, et al. *J2EE Design Patterns Applied.* Wrox Press, 2002.

Electronic-Web References

[w1]   Generative Programming Group, http://www.generative-programming.org

[w2]   INESC-ID's Information System Group. *The XIS Project.* http://berlin.inesc-id.pt/projects/xis/

[w3]   IBM AlphaWorks, *XMIToolkit.* http://www.alphaworks.ibm.com/tech/xmitoolkit/

[w4]   Markus Voelter, *A Collection of Patterns for Program Generation.* http://www.voelter.de/publications/index.html

[w5]   K. Czarnecki et al., *Beyond Objects: Generative Programming,* ECOOP'97 Workshop on Aspect-Oriented Programming. http://citeseer.nj.nec.com/czarnecki97beyond.html

[w6]   OMG. *Model Driven Architecture.* http://www.omg.org/mda/