International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue-17
December 2017

# A Component Model for Model Transformations

K. M .Shazmeen Banu & P.Viswanatha Reddy
1 M.Tech Student (SED), Sir Vishveshwaraiah Institute Of Science & Technology Hyderabad
Email: shazmeentaj542@gmail.com
2Sr. Assistant Professor, Department of CSE, Sir Vishveshwaraiah Institute of Science & Technology Hyderabad

***ABSTRACT***

*Model-driven engineering promotes an active use of models to conduct the software development process. In this way models are used to specify, simulate, verify, test and generate code for the final systems. Model transformations are key enablers for this approach, being used to manipulate instance models of a certain modelling language. However, while other development paradigms make available techniques to increase productivity through reutilization, there are few proposals for the reuse of model transformations across different modelling languages. As a result, transformations have to be developed from scratch even if other similar ones exist. In this paper, we propose a technique for the flexible reutilization of model transformations. Our proposal is based on generic programming for the definition and instantiation of transformation templates, and on component-based development for the encapsulation and composition of transformations. We have designed a component model for model transformations, supported by an implementation currently targeting the Atlas Transformation Language (ATL). To evaluate its reusability potential, we report on a generic transformation component to analyse workflow models through their transformation into Petri nets, which we have reused for eight workflow languages, including UML Activity Diagrams, YAWL and two versions of BPMN.*

## INTRODUCTION:

MODEL transformations are programs that take one or more models as input, and produce a number of output models. The aim of transformations is automating model manipulation when possible, while reducing the number of errors in this manipulation. This technology is key in model-driven engineering (MDE) [1], where it is used to implement model refactoring, model refinements, model synchronization mechanisms, and translators of models into other formalisms for analysis, among other tasks. The increasing adoption of MDE is leading to the construction of model transformations of raising complexity. However, building new transformations from scratch is costly, error prone, and requires specialized skills. Hence, transformation developers would benefit from mechanisms enabling the construction of new transformations by reusing proven, existing ones, adapted to the particular problem to be solved. In current MDE practice, there is a proliferation of metamodel variants for the same languages. This is partially caused by the focus of MDE on domain-specific languages and due to simplifications or variations introduced in large meta-models (like UML class diagrams and BPMN-like process modelling languages) to make them fit for the project purpose. For instance, the ATL meta-model includes different meta-models for Petri nets meta-models describing conference organization systems, and six variations of the Java meta-model. This variety hampers reuse because transformations are developed for a particular meta-model and cannot be reused for other related ones. Appropriate mechanisms for transformation reutilization would alleviate this problem, and we claim that they are essential for the success of the MDE paradigm at industrial scale.

## LITURATURE SURVEY:

**Automaticallydiscoveringhidden transformation chaining constraints:-**In this work, we propose a framework that automatically discovers dialect-specific phonetic rules. These rules characterize when certain phonetic or acoustic transformations occur across dialects. To explicitly characterize these dialect-specific rules, we adapt the conventional hidden Markov model to handle insertion and deletion transformations. The proposed framework is able to convert pronunciation of one dialect to another using learned rules, recognize dialects using learned rules, retrieve dialect-specific regions, and refine linguistic rules. Potential applications of our proposed framework include computer-assisted language learning, sociolinguistics, and diagnosis tools for phonological disorders.

**Componentsandgenerativeprogramming:-**
Generative and component approaches are revolutionizing software development just as automation and componentization revolutionized manufacturing. Key technologies for automating program development are Generative Programming for program synthesis, Component Engineering for modularity. In this paper we show the contribution of

these two approaches in the implementation of a multi-target learning management system generator adaptable to different target runtime environment. This article introduces the basics of a new programming method of virtual learning environments. This approach is based on generative programming that integrates the user specifications (abstract models) and technologies desired in order to produce bricks software, then put them together to produce a solution adapted to area and users' needs. This idea is used at different levels in the design and implementation of our system LMSGENERATOR,

**Feature-based survey of model transformation approaches:-**Model transformations are touted to play a key role in Model Driven Development™. Although well-established standards for creating met models such as the Meta-Object Facility exist, there is currently no mature foundation for specifying transformations among models. We propose a framework for the classification of several existing and proposed model transformation approaches. The classification framework is given as a feature model that makes explicit the different design choices for model transformations. Based on our analysis of model transformation approaches, we propose a few major categories in which most approaches fit.

## SYSTEM ANALYSIS:

### EXISTING SYSTEM:-

Transformations are tight to concrete meta-models, and it is not possible to reuse them for semantically related meta-models lack of meta-information and missing repositories, which makes it difficult to search and find transformations challenging and limited specialization of existing transformations, which limits the adaptation of a transformation to unforeseen contexts insufficient integration support. Thus, the most used approach in MDE is code scavenging. Developers typically search for related transformations pick some rules and adapt them to the meta-models at hand.

**Advantages:-**we provide an alternative with three main advantages: there is no need to generate an intermediate instance model of the pivot, but the template gets adapted to the new meta-model, being more efficient in terms of performance and memory footprint, traceability between the source and target models of the transformation is automatic because there is no intermediate step, and there is support to bind meta-models and concepts, which is normally simpler than using a full-fledged transformation language.

**Proposed system:-**Once a component is created, tested and deployed, its evolution is similar to the case of regular transformations. However the components with a large degree of variability may profit from modularization techniques like the ones proposed in a related technique applicable to MDE and ATL is proposed in and product lines techniques applied to model transformations. our components include a version number to allow several versions of the same component to coexist. We have framed this component model into the four dimensions of reuse proposed by Krueger abstraction, specialization, selection and integration. We use them to present the details of our model in the rest of the paper but first we briefly introduce them with respect to the metamodel presented above.

**Disadvantage:-**reusable transformations are defined over concepts making them simpler to define; transformations can be adapted to specific metamodelsthroughbindings,promotingtheirreutilization in a black-box manner; bindings induce a transformation adaptation, which results in an efficient reutilization approach; (iv) transformations are encapsulated in generic components, which promotes compos ability of transformations; and

Components expose features, helping in the definition of transformation variants.

## SYSTEM SPECIFICATION:

### Hardware Requirements:-

- System : Pentium IV 2.4 GHz.
- Hard Disk : 40 GB.
- Floppy Drive : 1.44 Mb.
- Monitor : 15 VGA Colour.
- Mouse : Logitech.
- RAM : 256 Mb.

### Software Requirements:

- Operating system : Windows 7.
- Front End : Dot net
- Database : SQL SERVER2008
- Tools : Visual Studio 2010

## SYSTEM TESTING:-

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**Types of tests:-**

**Unit testing: -** Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**Integration testing: -** Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

**Functionaltest:-**Functionaltests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

**System Test: -** System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**White Box Testing:-**White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing:-**Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

## SYSTEM STUDY

**Feasibility study:-**The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

**Economical feasibility:-**This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

**Technical feasibility**: - This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high

demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**SOCIAL FEASIBILITY:-**The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

## MODULE DESCRIPTION:

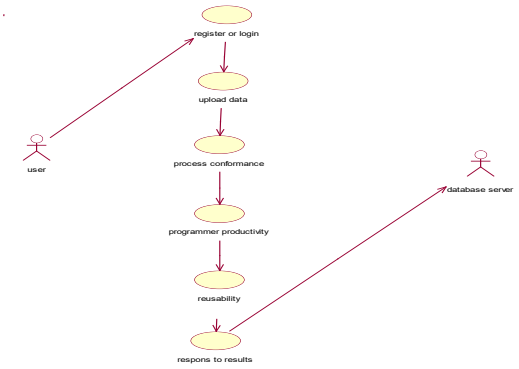
**Modeldrivenengineering:**Theaimoftransformations is automating model manipulation when possible, while reducing the number of errors in this manipulation. This technology is key in model-driven engineering (MDE) where it is used to implement model refactoring, model refinements, model synchronization mechanisms, and translators of models into other formalisms for analysis, among other tasks. The increasing adoption of MDE is leading to the construction of model transformations of raising complexity. However, building new transformations from scratch is costly, error prone, and requires specialized skills. Hence, transformation developers would benefit from mechanisms enabling the construction of new transformations by reusing proven, existing ones, adapted to the particular problem to be solved.

**Model transformation:-**we improve our binding DSL to consider more complex adaptations, and propose a component model for model transformations based on the notion of generic transformation. The model supports both simple and composite components, which are treated in a unified way by using concepts as their interfaces. Simple components encapsulate a transformation template and expose one or more concepts specifying the requirements that meta-models need to fulfil to apply the component. Components may also expose features which can be used to configure the behaviour of the transformation template, or to select between different executions paths in a composite component.

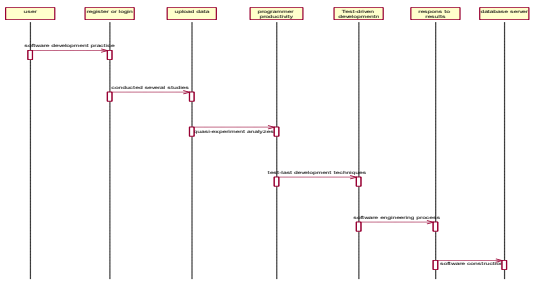
**Component-based development:-**We would also like to capitalize on existing meta-model evolution and differencing techniques, to semi-automatically derive a first version of the binding. We believe these techniques would help in transferring the approach into practice, for which initiatives for open component repositories (similar to the ATL zoo) are also necessary. Finally, it would be interesting to empirically evaluate how well developers are able to specify concepts and how well the text-based documentation, contracts, and tagged components allow other developers to identify reusable transformations.

## UML DIGRAMS:

**Use Case Diagram:-**

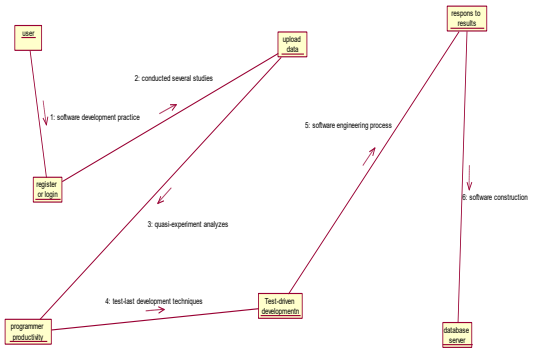



**Sequence Diagram:-**




**Collaboration Diagram:-**

International Journal of Research
Available at https://edupediapublications.org/journals
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 04 Issue-17
December 2017




## SOFTWARE ENVIRONMENT:

**Features OF. Net:-** Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate. There's no language barrier with .NET: there are numerous languages available to the developer including Managed C++, C#, Visual Basic and Java Script. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate. ".NET" is also the collective name given to various software components built upon the .NET platform. These will be both products (Visual Studio.NET and Windows.NET Server, for instance) and services (like Passport, .NET My Services, and so on).

**The .net framework:-**The .NET Framework has two main parts:

1. The Common Language Runtime (CLR).

2. A hierarchical set of class libraries.

The CLR is described as the "execution engine" of .NET. It provides the environment within which programs run.

| ASP.NET<br>XML WEB SERVICES | Windows Forms |
|---|---|
| Base Class Libraries | |
| Common Language Runtime | |
| Operating System | |

**.Net Framework**

**Languages supported by .net:-** The multi-language capability of the .NET Framework and Visual Studio .NET enables developers to use their existing programming skills to build all types of applications and XML Web services. The .NET framework supports new versions of Microsoft's old favourites Visual Basic and C++ (as VB.NET and Managed C++), but there are also a number of new additions to the family. Visual Basic .NET has been updated to include many new and improved language features that make it a powerful object-oriented programming language. These features include inheritance, interfaces, and overloading, among others. Visual Basic also now supports structured exception handling, custom attributes and also supports multi-threading. Visual Basic .NET is also CLS compliant, which means that any CLS-compliant language can use the classes, objects, and components you create in Visual Basic .NET. Managed Extensions for C++ and attributed programming are just some of the enhancements made to the C++ language. Managed Extensions simplify the task of migrating existing C++ applications to the new .NET Framework's# is Microsoft's new language. It's a C-style language that is essentially "C++ for Rapid Application Development". Unlike other languages, its specification is just the grammar of the language. It has no standard library of its own, and instead has been designed with the intention of using the .NET libraries as its own. Microsoft Visual J# .NET provides the easiest transition for Java-language developers into the world of XML Web Services and dramatically improves the interoperability of Java-language programs with existing software written in a variety of other programming languages. Active State has created Visual Perl and Visual Python, which enable .NET-aware applications to be built in either Perl or Python. Both products can be integrated into the Visual Studio .NET environment. Visual Perl includes support for Active State's Perl Dev Kit.

Other languages for which .NET compilers are available include

- FORTRAN
- COBOL
- EIFFEL

**Features of SQL-SERVERT**:-he OLAP Services feature available in SQL Server version 7.0 is now called SQL Server 2000 Analysis Services. The term OLAP Services has been replaced with the term Analysis Services. Analysis Services also includes a new data mining component. The Repository component available in SQL Server version 7.0 is now called Microsoft SQL Server 2000 Meta Data Services. References to the component now use the term Meta Data Services. The term repository is used only in reference to the repository engine within Meta Data Services-SERVER database consist of six type of objects, they are,

- 1. TABLE
- 2. QUERY
- 3. FORM
- 4. REPORT
- 5. MACRO

## CONCLUSION

We have presented a novel reutilization approach for model transformations based on the definition of generic transformation templates over concepts, which can be bound to different meta-models. Transformation templates are encapsulated into components, which can be configured through features, and composed to form composite components. We have developed a tool that supports our approach. Moreover, we have evaluated our proposal with respect to flexibility of reuse and concept abstraction power using several realistic scenarios.

## REFERENCES:

[1] M. Brambilla, J. Cabot, and M. Wilmer, Model-Driven SoftwareEngineering in Practice, series Synthesis Lectures on Software Engineering. an Rafael, CA, USA: Morgan & Claypool Publishers,2012.

[2] A. Kusel, J. Sch€onb€ock, M. Wilmer, G. Keppel, W. Retschitzegger, and W. Schwinger, "Reuse in model-to-model transformation languages: Are we there yet?" Soft. Syst. Model., vol. 13, 2013.

[3] K.-K. Lau and Z. Wang, "Software component models," IEEETrans. Soft. Eng., vol. 33, no. 10, pp. 709–724, Oct. 2007.

[4] K. Saks. (2009). JSR 318: Enterprise java beans, version 3.1[Online]. Available: http://download.oracle.com/otndocs/jcp/ejb-3.1-mrel-evalu-oth-JSpec/

[5] R. van Immersing, F. van deer Linden, J. Kramer, and J. Magee, "The Koala component model for consumer electronics software," Computer, vol. 33, no. 3, pp. 78–85, Mar. 2000.

[6] J. Sanchez Cuadrado, E. Guerra, and J. de Lara, "Generic model transformations: Write once, reuse everywhere," in Proc. 4th Int. Conf.Theory Practice Model Transformations, 2011, pp. 62–77.

[7] J. Sanchez Cuadrado, E. Guerra, and J. de Lara, "Flexible modelto-model transformation templates: An application to ATL," JOT,vol. 11, no. 2, pp. 4:1–28, 2012.

[8] J. de Lara and E. Guerra, "From types to type requirements: Genericity for model-driven engineering," Soft. Syst. Model.,vol. 12, no. 3, pp. 453–474, 2013.

[9] D. Gregory, J. Java, J. G. Sick, B. Stroustrup, G. D. Reis, and A.Lumsdaine, "Concepts: Linguistic support for generic programming in C++," in Proc. 21st Innu. ACM SIGPLAN Conf. Object-Oriented Program. Syst., Languages, Appl., 2006, pp. 291–310.

[10] A. Stepanov and P. McJones, Elements of Programming. Reading, A, USA: Addison Wesley, 2009.

[11] R. Chenouard and F. Rouault, "Automatically discovering hidden transformation chaining constraints," in Proc. 12th Int. Conf. Model Driven Eng. Languages Syst., 2009, pp. 92–106.