2013

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

AN EVALUATION OF MODEL DRIVEN ARCHITECTURE (MDA) TOOLS

Richard Alford

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2013

Approved by

Advisory Committee

_____	_____
Dr. Ron Vetter	Dr. Bryan Reinicke

_____
Dr. Devon Simmonds, Chair

Abstract

An Evaluation of Model Driven Architecture (MDA) Tools. Alford, Richard, 2013. Capstone Paper, University of North Carolina Wilmington.

The aim of this research is to evaluate and compare three software modeling and architecture tools within the context of their built-in support for Model Driven Architecture (MDA). MDA's primary goal is to develop standards based on the notion that modeling is a preferred base for developing and maintaining systems over platform specific source code. By promoting models to primary software artifacts MDA bridges the gap between design/analysis and the implementation of software systems. This research evaluates and compares IBM Rational Software Architect, Altova UModel, and Sparx System Enterprise Architect. The evaluation is done using a well defined metric suite and a procedures log template that guided the analysis and evaluation processes. The project addressed four research questions: (1) How effective are current tools in regards to MDD within the framework of MDA? (2) What new challenges are introduced by adopting MDD within the enterprise? (3) How easy/difficult is it to integrate the tools into an existing software development environment or toolset? and (4) What are the relative strengths and weakness of the selected MDD tools? The research results suggest that a progressive approach to implementing MDA should be taken within the enterprise. Additionally, tool selection should be based on environment, resource availability, project scope, and budget. Other inferences and results are presented and discussed.

List of Tables

Table 1: Initial Feature Set Required for Tool Consideration 20

Table 2: Definition of Model Transformation/Code Generation Metric 22

Table 3: Definition of Integration Metric 23

Table 4: Definition of Usability Metric 24

Table 5: Definition of Requirements Metric 24

Table 6: Procedure Log Template 25

Table 7: Specification of Testing Environment 28

Table 8: Results of Model/Transformation/Code Generation Capabilities 82

Table 9: Model Transformation/Code Generation Capabilities of MDA Tools – Range 0-2 (0 Lowest to 2 Highest) 84

Table 10: Results of Integration Capabilities of MDA Tools 85

Table 11: Integration Capabilities of MDA Tools – Range 0-2 (0 Lowest to 2 Highest) 86

Table 12: Results of Usability of MDA Tools 86

Table 13: Usability of MDA Tools – Range 0-2 (0 Lowest to 2 Highest) 87

Table 14: Results of Requirements of MDA Tools 87

Table 15: Requirements of MDA Tools – Range 1-3 (1 Lowest to 3 Highest) 88

List of Figures

Figure 1: Banking System Class Diagram 29

Figure 2: Banking System Sequence Diagram 30

Figure 3: Enterprise Architect version control setup 32

Figure 4: Enterprise Architect XMI Import 33

Figure 5: Enterprise Architect Project Browser 34

Figure 6: Enterprise Architect class PIM from XMI 35

Figure 7: Enterprise Architect sequence PIM from XMI 36

Figure 8: Enterprise Architect Version Control Check In 37

Figure 9: Enterprise Architect MDA Transform Screen 1 38

Figure 10: Enterprise Architect MDA Transformation Screen 2 39

Figure 11: Enterprise Architect class C# PSM from MDA Transformation 40

Figure 12: Enterprise Architect class PSM modification 41

Figure 13: Enterprise Architect Code Engineering Screen 1 42

Figure 14: Enterprise Architect Code Engineering Screen 2 43

Figure 15: Enterprise Architect C# Code Generation Results 44

Figure 16: Enterprise Architect C# Code Generation Compiler Error List 44

Figure 17: Enterprise Architect BrokerageAccount Source File 45

Figure 18: Enterprise Architect Synchronize Model from Code 46

Figure 19: Enterprise Architect Synchronize C# profile class PSM from Source Code 47

Figure 20: Enterprise Architect Export to XMI 48

Figure 21: Enterprise Architect Enterprise Java profile class PSM from MDA Transformation. 49

Figure 22: Enterprise Architect C# Code Generation Results 51

Figure 23: Enterprise Architect Synchronize Java profile class PSM from Source Code 52

Figure 24: Altova UModel version control setup menu 54

Figure 25: Altova UModel XMI Import messages 55

Figure 26: Altova UModel class PIM from XMI 56

Figure 27: Altova UModel sequence diagram stubs in Model Tree 57

Figure 28: Altova UModel Version Control Check In 58

Figure 29: Altova UModel MDA Transform Screen 1 59

Figure 30: Altova UModel Class C# PSM from Model Transformation 60

Figure 31: Altova UModel PSM to PIM Component Model Describing Transformation 61

Figure 32: Altova UModel PSM to Code Targeting C# 63

Figure 33: Altova UModel PSM to Code Targeting C# Synchronization Settings 64

Figure 34: Altova UModel Code Generation Smart Comments 65

Figure 35: Altova UModel Visual Studio Bridge with Real-Time Sync Code to Model 65

Figure 36: Altova UModel ProcessCheck Modification 66

Figure 37: Altova UModel Sequence PSM from Source Code 67

Figure 38: Altova UModel Sync Code from Sequence PSM modification 68

Figure 39: Altova UModel Class Java PSM from Model Transformation 69

Figure 40: Altova UModel Synchronize Java Class PSM from Source Code 71

Figure 41: Altova UModel Java Bank.ProcessCheck() Code 72

Figure 42: Altova UModel Generate Sequence Diagram from Code 73

Figure 43: Altova UModel Sequence PSM from Code 73

Figure 44: Altova UModel PSM Sequence Diagram Modification 74

Figure 45: Altova UModel Generate Code from Sequence Diagram 75

Figure 46: Altova UModel Sequence PSM Modification to Java Code 76

Figure 47: IBM Software Architect Subversive Installation 77

Figure 48: IBM Software Architect Subversive Install Error 78

Figure 49: IBM Software Architect XMI Import 78

Figure 50: IBM Software Architect XMI Import Attempt 2 79

Figure 51: IBM Software Architect PIM to C# PSM Transformation 80

Figure 52: IBM Software Architect PIM to C# PSM Transform Error 80

Figure 53: IBM Software Architect PIM to Java PSM 81

Figure 54: IBM Software Architect Generated Source Bank.java 82

Table of Contents

Abstract 1

List of Tables 2

List of Figures 3

Table of Contents 6

Chapter 1: Introduction 8

1.1 Proposed Solution 9

1.2 Outline of the Paper 10

Chapter 2: Review and Analysis 11

2.1 The Unified Modeling Language 11

2.2 The State of Model Driven Architecture 12

2.3 The Case for Model Driven Development with Model Driven Architecture 13

2.4 Challenges Facing Model Driven Development 14

2.5 Enterprise Tool support for MDA 15

2.5.1 Sparx System Enterprise Architect 15

2.5.2 IBM Rational Rose Architect 16

2.5.3 Altova UModel 17

Chapter 3: Methodology and Plan 19

3.1 Selected Tools 19

3.2 Evaluation Metrics 21

3.3 Evaluation Approach 24

3.3.1 Procedure Log Template 25

3.3.1 Evaluation Environment 27

3.3.2 Formal Requirement Analysis as Input Parameters 28

3.3.2.1 The Class Diagram 29

3.3.2.2 The Sequence Diagram 30

Chapter 4: Results 31

4.1 Evaluation of Enterprise Architect 31

4.2 Evaluation of Altova UModel 53

4.3 Evaluation of IBM Software Architect 77

4.4 Results from the Evaluations of Selected Tools 82

Chapter 5: Discussion and Lessons Learned 89

5.1 Project Rationale 89

5.2 Lessons Learned 89

5.3 Addressing the Research Questions 90

5.3.1 How effective are current tools in regards to MDD within the framework of MDA? 90

5.3.2 What new challenges are introduced by adopting MDD within the enterprise? 91

5.3.3 How easy/difficult is it to integrate the tools into an existing software development environment or toolset? 92

5.3.4 What are the relative strengths and weakness of the selected MDD tools? 92

5.4 Challenges Encountered 94

Chapter 6: Conclusion and Future Work 94

References 96

Chapter 1: Introduction

Model driven development (MDD) is a software development methodology that raises the level of abstraction for designing and developing systems by focusing on the creation of models which are systematically transformed into source code. Bran Selic, IBM Distinguished Engineer was quoted as saying "*Software has the rare property that it allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods*" [1]. Raising the level of abstraction in software engineering is critical to address the reality of growing and evolving complexity and pervasiveness of software [4]. IT environments are often characterized by frequently changing business requirements, varied platforms, and legacy integration concerns [2]. The complexity associated with these characterizations can burden organizations with costly and time consuming solutions [2].

The Object Management Group (OMG) [3] is an international organization whose mission is to develop enterprise integration standards that provide real-world value. Founded in 1989, the OMG has set standards for a variety of technologies including but not limited to the Unified Modeling Language (UML) and Model Driven Architecture (MDA). These modeling standards enable powerful visual design, execution and maintenance of software and other processes such as IT Systems Modeling and Business Process Management [3].

Over the past 7 years I have worked in the business software space across a variety of roles including programmer/analysis, enterprise software account manager, and senior level developer/architect. Based on my experiences over this time, I have noticed that one aspect of business software that remains fairly constant is the subject domain of the core business. User interfaces come and go, web service applications are constructed and torn down, database

conversions from iSeries DB2 to Distributed SQL Servers are implemented, but the core structural and behavior concepts that describe the business domain logic persists and grows with the demands of the business itself. The business domain often needs to be reflected in multiple platforms scattered across the enterprise. For example, in a typical multi-tiered rich internet application, the business domain is realized in at least three platforms; on the client as JavaScript objects, on the server as C# objects, and relationally as table entities. Moreover, even in larger enterprises, my personal experiences have found that formal requirements analysis through UML diagramming is shortsighted, leveraged for specific project demands and abandoned early in the software development lifecycle leaving the dated diagrams inconsistent with production software. I have seen a neglect of documentation lead to poor design choices by developers because they have an incomplete or inconsistent view of what the business software does or what it is intended to do. All too often I encounter organizations that lean on the "*Business Analyst Oracle*", a term I use to describe instances where business and system documentation is absent except for in the heads of a few seasoned business analysts. These experiences have driven me with a desire to not fall into the same pitfalls as my predecessors. There must be a way to express, preserve, update and reuse subject domain knowledge independent of software platforms, not as blueprints but as primary software artifacts. This is why I have chosen to research the area of Model Driven Architecture for my Capstone Project.

1.1 Proposed Solution

The adoption of new tools and processes within a pre-defined software development methodology should provide measurable value, should not hinder adaptability of the development process, and should meet the requirements of the development team. This research aims to evaluate the suitability of several current MDD tools used within the enterprise. In order

to measure the effectiveness of MDD tools the research defines a metric suite which is used to evaluate a set of industry leading MDD tools. This project aims to address the following research questions:

1. How effective are current tools in regards to MDD within the framework of MDA?
2. What new challenges are introduced by adopting MDD within the enterprise?
3. How easy/difficult is it to integrate the tools into an existing software development environment or toolset?
4. What are the relative strengths and weakness of the selected MDD tools?

1.2 Outline of the Paper

In the next chapter related work and technologies are discussed. Chapter 3 presents the research methodology and describes the MDD tools selected for evaluation. The tool evaluation results are presented in Chapter 4. Chapter 5 discusses the research results, presents lessons learned and challenges encountered, as well as describe how the research results provide answers to the research questions posed in the introduction. Chapter 6 concludes the project and presents future research opportunities.

Chapter 2: Review and Analysis

2.1 The Unified Modeling Language

Unified Modeling Language (UML) is a group of graphical notations, defined by a single meta-model, that are used to describe and design software systems [10]. The UML is an open standard controlled by the Object Management Group. The OMG first standardized UML in 1997 and since then it has gone through a series of revisions. In 2003 UML 2.0 was released to support new challenges that software modelers face including deeper support for MDA through a refined meta-model [10]. Martin Fowler, a founding contributor to the UML describes three common ways UML is used; UML as a sketch, UML as a blueprint, and UML as a programming language [9]. UML as a sketch includes scenarios where the developer creates brief "throwaway" diagrams to explain or gain insight into a particular system concept. UML as a blueprint goes further; it provides detailed specifications of a system with UML diagrams. Forward and reverse engineering may be used to keep UML blueprints in sync with code. UML as a programming language involves a direct path from UML models into executable code. In this scenario, every aspect of the system is modeled.

UML 2.2 describes 14 official diagram types which are each categorized as either structural or behavioral. Structural diagrams are used to model the static structure of a system. This is typically done by defining objects, attributes, operations and relationships. 7 of the 14 official diagram types of UML 2.0 are classified as structural diagrams. Behavioral diagrams focus on the dynamics of a system by modeling interactions between system objects and internal state changes. Two UML 2 diagram types that are often used to model structural and behavior in MDA are class and sequence diagrams respectively.

Classes are at the core of any object-oriented system; therefore, it is expected that the UML 2 Class Diagram is one of the most popular. The class diagram is a structural diagram that illustrates the objects that make up a system. The class diagram is made of classes which contain attributes and operations, it shows relationships between classes, and provides insight into high level object oriented design decisions such as abstraction, interfaces, and visibility.

The sequence diagram is a type of interaction diagram that captures how groups of objects collaborate in some behavior. Sequence diagrams are specifically used to capture the order of interactions between parts of your system. Sequence diagrams are composed of participants (parts of your system involved in the interaction), time, events, signals, and messages.

2.2 The State of Model Driven Architecture

MDA is a standards based approach to model driven development (MDD). MDA is an OMG initiative to develop standards based on the notion that modeling is a preferred base for developing and maintaining systems over platform specific source code. MDA based development typically goes through three steps; platform independent modeling (PIM), platform specific modeling (PSM), and code production.

At the base of MDA, PIMs are created to describe the application's business functionality and behavior in a platform/technology independent manner. Next, one or more PSMs are produced to capture platform specific constructs mapped to a particular implementation technology. PSMs typically weave functional requirements defined in PIMs with non-functional requirements such as runtime environments, database management systems, security, to name a few. Defining non-functional, platform specific details often involves marking up an instance of

the PIM. It is through this markup that a PIM can be transformed into a PSM. Markings are neither a part of the PIM or the PSM; this decoupling of markup ensures that both PIMs and PSMs can stand alone and be repurposed [4]. From PSMs source code and descriptors are produced through platform specific model transformations which can be deployed as a functional software system.

Transformations in MDA are realized due to a defined metamodel and xml interchange format (XMI) provided by the OMG. A metamodel is a model of the modeling language that formally specifies the concepts of a given model, for example, the OMG's UML metamodel captures the concepts of the UML. The UML metamodel is expressed using the OMG's MetaObject Facility (MOF). The OMG defines MOF as "The MetaObject Facility Specification is the foundation of OMG's industry-standard environment where models can be exported from one application, imported into another, transported across a network, stored in a repository and then retrieved, rendered into different formats, transformed, and used to generate application code [5]. Metamodels are fundamental to MDA whereby they specify the model language, provide communication about models, and provide a means to specify mapping functions to and from a given model.

2.3 The Case for Model Driven Development with Model Driven Architecture

Traditional software development methodologies isolate modeling tasks as analysis and design phase functions. During this phase use cases, class diagrams, activity diagrams and other UML models are generated by the developer to model the business problem and validate requirement analysis with the business group. Requirements change for a variety of reasons during the course of a software development lifecycle. Often changes introduced in later phases of software development are handled within the implementation itself leaving the models

produced during design and analysis outdated and inconsistent with the software. Fu, Hao, Bastani, and Yen point out that the gap between design/analysis and implementation as it relates to modeling can lead to further problems; notably maintenance and documentation. The models produced end up out of date leading to incomplete or misleading documentation referenced during the maintenance phase [6]. By promoting models to primary software artifacts MDD bridges the connection between design/analysis and the implementation of software systems. This promotion ensures model artifacts remain in-sync with platform specific implementations providing accurate documentation across the software development lifecycle.

2.4 Challenges Facing Model Driven Development

Model driven development faces a range of challenges as it relates to fully bridging the gap between high level model abstractions and the full transformation of those models into complete code of complex software systems. France and Rumpe recognize that these challenges are wide reaching and involve difficult interrelated social and technical problems. They label this challenging environment as “wicked”, a problem domain whose solutions are costly to develop and are consistently related with other problems [1].

In an industrial case study conducted in 2006 two primary MDD challenges were identified; cost to entry and expertise in modeling techniques. The study suggests a perception exists in the enterprise that access to model driven development tools capable of supporting the entire software development lifecycle were costly compared to traditional tools. Further, it suggests adoption of MDD requires a specialized skill set in modeling techniques [2]. This specialization is often perceived as difficult to source and/or costly to learn. A survey constructed by Teppola, Parviainen, and Takalo in 2009 that evaluated industry adoption of MDD confirms these challenges. Eleven MMD challenges identified from existing literature

were measured in the survey. The largest challenge by respondent count focused on the lack of modeling experts in organization [7].

Debugging is an important aspect of software development and a challenge to MDD tools. Most code-generator frameworks and generators completely ignore this issue. This ignorance forces the developer to conduct debugging at the assembly level [8]. The disconnected nature of model level development and code level debugging raises another concern surrounding expertise. In fact, multiple expertise are needed to fully resolve software bugs in cases where the MDD tool lacks debugging capabilities; a code level or platform specific expert for debug tracing, a model level or platform independent expert for debug resolution.

2.5 Enterprise Tool support for MDA

This project evaluates and compares a set of enterprise software modeling and architectural tools in context of their MDA capabilities. Chapter 3 provides details regarding how the selected tools were chosen for inclusion in this research.

2.5.1 Sparx System Enterprise Architect

Sparx Systems is an Australian based software company focused on providing high performance and scalable visual modeling tools for the planning, design, and construction of software intensive systems [11]. Sparx Systems is a Contributing Member of the OMG which reflects their commitment to developing tools based on the open standards of MDA [11]. Enterprise Architect is the company's flag ship product. It was first released in 2000, has evolved through ten major releases, and has over 300,000 registered users [11]. Enterprise Architect is a visual platform targeted for the design and construction of software systems, business process

modeling, and general purpose modeling [12]. Major features of Enterprise Architect 10 include [11]:

- Modeling composition and extension of the full UML 2.4 specification.
- Comprehensive requirements management.
- Test management including model based test execution.
- Documentation generation.
- Code engineering and reverse engineering with built-in MDA transformations with out of box support for over 10 programming languages.
- Extendable modeling and transformation capabilities with transform templates.

2.5.2 IBM Rational Rose Architect

Founded in 1911 IBM is a global information technology company with over a century of experience with business systems [13]. With a market capitalization of $240 billon they remain one of the largest Information Technology companies in the world. They operate five business segments: Global Technology Services, Global Business Services, Software, Systems/Technology, and Global Finance. Rational Rose Architect V8.5 is the realization of over two decades of product development and enhancements. The groundwork for Rational Rose was provided by Grady Booch, James Rumbaugh and Ivar Jacobson; the three engineers credited with creating what became UML [14]. Built on top of the popular open source integrated development environment, Eclipse, Rational Rose Architect encompasses three core components; Architect for design and development with UML support, Architect for Websphere Software an optimized tool for J2EE, SOA, and Websphere applications, and Architect Design Manager for management and collaboration of design information [15]. This research focuses on

the first of the three components, Rational Rose Architect. Major features of Rational Rose Architect include [15]:

- Modeling composition and extension of UML 2.2.
- Requirements integration.
- Support for the creation of Domain Specific Languages based on UML.
- Graphical mapping tools to develop model-to-model transformations.
- Built-in UML to code transformations for seven popular programming languages.
- A set of extensions focused on providing additional functionality such as simulations and platform specific deployments.

2.5.3 Altova UModel

Altova is a software company founded in 1992 based in Beverly, MA, USA and Vienna, Austria. Their flagship product is XMLSpy which is a XML editor and IDE targeted towards the development of XML-based and Web Service applications [16]. Altova currently maintains a portfolio of nine complimentary products all designed to assist developers with data management, software and application development, and data integration [17]. UModel 2013 is Altova's current tool for software modeling and application development. UModel was first released in 2005 [18]. Major features of UModel 2013 include [17]:

- Modeling composition and extension of the full UML 2.4 specification.
- Built-in code generator that creates Java, C#, and VB.Net code from UML models.
- Reverse engineering of existing code into UML models.
- Round trip engineering.
- Built-in support for MDA through transformations of PIM UML to PSM UML.

- Documentation generation.

Chapter 3: Methodology and Plan

The goal of this project is to evaluate and compare a set of enterprise scale software modeling and architectural tools in context of their MDA capabilities. The tools are evaluated based on their ability to support the development of a business domain class library targeting both Java and .NET environments. To accomplish this goal the research sets out to:

1. Research and select a set of modeling and architectural tools to be evaluated.
2. Create a metric suite used to evaluate the tools in the context of MDA capabilities and related features.
3. Define a subset of business domain formally using structural and behavioral UML diagrams. These diagrams serve as input parameters for the tool evaluations.
4. Construct a procedure log template to serve as a blueprint that outlines the chronological steps the tool evaluations should follow.
5. Evaluate the effectiveness of each tool in regards to the metric suite define using the procedure log template from 4) and diagrams from 3) as input parameters.
6. Interpret the results.
7. Make recommendations and reflection.

3.1 Selected Tools

Three enterprise grade software modeling and architectural tools are identified that meet the initial feature set requirements outlined in Table 1. They include IBM Rational Rose Architect, Altova UModel, and Sparx System Enterprise Architect. These tools are evaluated based the metric suite described in section 3.2.

This project focuses on evaluating the selected enterprise grade software modeling and architectural tools within the context of their built-in MDA capabilities. Several MDD tools are available other than the ones selected for this research. Tools selected for this study must meet a minimum set of features described below and outlined in Table 1.

Table 1: Initial Feature Set Required for Tool Consideration

	Definition
UML 2 Composition	This feature provided by the tool should provide a graphical interface for composition of UML 2 diagrams.
Model-to-Model Transformations	This feature provided by the tool should provide built-in model transformations between PIMs and PSMs.
Model-to-Code Transformations	This feature provided by the tool should provide built-in model transformations from PSMs to code.
PSM targets	This feature provided by the tool should be capable of targeting two popular platform specific profiles; JAVA and C#.
IDE Integration	This feature provided by the tool should be capable of integrating with popular Integrated Development Environments (IDE); Eclipse for Java development and Visual Studio for C# development.
XMI Support	This feature provided by the tool should provide import and export of UML diagrams via XMI standard.
Version Control	This feature provided by the tool should provide support for popular third party source/version control of model artifacts.

UML 2 composition involves the creation of UML 2 models through a graphical interface. The user should find it easy to create, modify, and organize UML 2 models within the tool. As presented in Chapter 2, model-to-model transformations are critical to implementing a MDA process. The selected tools should provide built-in model-to-model transformations from

PIMs to PSMs targeting popular platform specific profiles including Java and C#. Developers work inside IDEs and MDA does not prohibit developers from expressing and implementing certain aspects of a solution with code particularly in cases where the aspects are either difficult or impossible to model. The tools should support integration with the most popular Java and C# IDEs, Eclipse and Visual Studio. XML Metadata Interchange (XMI) is a fundamental component of MDA. It generally defines a XML standard for exchanging metadata information. More specifically in regards to common MDA scenarios; XMI is applied to UML models so that they can easily move from tool to tool. This is important as it ensures that models are not locked into a specific tool and/or vendor. MDA promotes models from blueprints and documentation to primary software development artifacts. Primary software artifacts should be controlled by a version control system to enable collaboration, manage merging, archive versions, and assist in recovery.

3.2 Evaluation Metrics

This research defines an evaluation metric suite composed of four evaluation metric aspects including model transformation/code generation, requirement features, usability, and integration. The metric suite is used to quantify the effectiveness of the selected tools in regards to their MDA capabilities in the enterprise.

The model transformation/code generation metric measures the effectiveness of each tool's built-in model transformation and code generation capabilities. Table 2 describes each model transformation/code generation feature included in this metric.

Table 2: Definition of Model Transformation/Code Generation Metric

	Definition
Structural PIM to/from PSM Transforms targeting C# Profile	The capability of the tool to transform the structural (class diagram) PIM of the subject domain into a new structural PSM targeting the C# Profile. Additionally, this metric item evaluates the tool's ability to sync the PIM from the generated PSM after modification of the PSM.
Structural PIM to/from PSM Transforms targeting JAVA Profile	The capability of the tool to transform the structural (class diagram) PIM of the subject domain into a new structural PSM targeting the JAVA Profile. Additionally, this metric item evaluates the tool's ability to sync the PIM from the generated PSM after modification of the PSM.
Structural C# Profile based PSM to/from Code Transforms targeting C# code	The capability of the tool to transform the generated structural (class diagram) PSM of the subject domain into C# source code. Additionally, this metric item evaluates the tool's ability to sync the PSM from the generated source code after a structural code modification.
Structural Java Profile based PSM to/from Code Transforms targeting JAVA code	The capability of the tool to transform the generated structural (class diagram) PSM of the subject domain into JAVA source code. Additionally, this metric item evaluates the tool's ability to sync the PSM from the generated source code after a structural code modification.
Behavioral PIM to/from PSM Transforms targeting C# Profile	The capability of the tool to transform the behavioral (sequence diagram) PIM of the subject domain into a new behavioral PSM targeting the C# Profile. Additionally, this metric item evaluates the tool's ability to sync the PIM from the generated PSM after modification of the PSM.
Behavioral PIM to/from PSM Transforms targeting JAVA Profile	The capability of the tool to transform the behavioral (sequence diagram) PIM of the subject domain into a new behavioral PSM targeting the JAVA Profile. Additionally, this metric item evaluates the tool's ability to sync the PIM from the generated PSM after modification of the PSM.
Behavioral C# Profile based PSM to/from Code Transforms targeting C# code	The capability of the tool to transform the generated behavioral (sequence diagram) PSM of the subject domain into C# source code. Additionally, this metric item evaluates the tool's ability to sync the PSM from the generated source code after a behavioral code modification.
Behavioral Java Profile based PSM to/from Code Transforms targeting JAVA code	The capability of the tool to transform the generated behavioral (sequence diagram) PSM of the subject domain into JAVA source code. Additionally, this metric item evaluates the tool's ability to sync the PSM from the generated source code after a behavioral code modification.

The integration metric is the next metric defined. This metric measures the effectiveness of each tool in regards to their integration capabilities. The integration metric measures three sub categories, IDE integration and XMI support, and version control. Table 3 defines each model transformation/code generation feature included in this metric.

Table 3: Definition of Integration Metric

	Definition
Eclipse Integration	Capability and level of integration of the tool with the Eclipse Development Environment.
Visual Studio Integration	Capability and level of integration of the tool with the Microsoft Visual Studio Development Environment.
Import of XMI Representation of UML Models	Capability of the tool to import structural and behavioral PIMs from an independent modeling tool via XMI
Export of XMI Representation of UML Models	Capability of the tool to export structural and behavioral PIMs to an independent modeling tool via XMI
Version Control Integration	Capability of the tool to integrate with version control systems. What systems can the tool integrate with? How does the tool handle typical version control operations such as check-in, check out, and roll back?

The third metric defined is usability. The usability metric measures aspects of the tool and supporting artifacts that are targeted towards ease of use. Documentation, support options, tutorials, and overall end user experience is captured within this metric. Table 4 defines the usability features included in this metric.

Table 4: Definition of Usability Metric

	Definition
Tutorials	Availability of tutorials related to MDA within each tool. Are the tutorials easy to follow?
General Documentation	How is the documentation provided to the user? Is documentation easy to read? Is the documentation concise?
End User Experience	Is the tool easy to use? Is it responsive? Does it crash or freeze?
Ease of installation	How easy is it to install the software? Are there prerequisites? Can the IDEs be installed after the tool installation or must they be installed as prerequisites?
Support Options	What support options are available? User forums, ticketing systems, support response times.

The final metric measures the general requirements of the tools. The requirements metric measures a variety of non-functional aspects of the tools including licensing options, pricing, and supported operating systems. Table 5 defines the requirements features included in this metric.

Table 5: Definition of Requirements Metric

	Definition
Licensing Options	Are there multiple versions of the tool? Can the tool be licensed a la carte? Are upgrades included in the licensing? Are there discounts for group licensing? Is the license annual or perpetual?
Pricing	How much does the software cost?
Supported Operating Systems	What operating systems can the tool run on?

3.3 Evaluation Approach

The selected modeling tools are evaluated based on their capabilities to support MDA within the enterprise. The tool evaluation approach is comprised of three related components; a

procedure log template, evaluation environment, and input parameters as formal requirement analysis.

3.3.1 Procedure Log Template

In order to ensure each tool is evaluated in a uniform manor a procedure log template is defined as Table 6. The procedure log template describes the general steps required for the evaluations. Steps within the procedure log template are mapped to specific metric suite items. This mapping provides a concrete relationship between the evaluation steps within each tool to the associated metric suite items. Any deviations from the procedure log template for a given tool evaluation are documented by the evaluator in the corresponding procedure log and discussed in the results. In addition, screenshots are appended to the steps in the tool evaluation's procedure log.

Table 6: Procedure Log Template

Evaluation Step	Maps to Metric Items
1. Setup version control settings in MDA Tool 2. Import XMI of PIM of Banking from MagicDraw XMI Export. 3. Commit to version control.	• Import of XMI Representation of UML Models • Version Control Integration
4. Retrieve PIMs from version control. 5. Generate C# PSM models from PIM of Banking.	• Structural PIM to/from PSM Transforms targeting C# Profile • Behavioral PIM to/from PSM Transforms targeting C# Profile
6. Modify PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires adding a new class to the class PSM named Bank. 7. Sync the class PIM from the newly updated PSM.	• Structural PIM to/from PSM Transforms targeting C# Profile
8. Modify PSM models to accommodate for a	• Behavioral PIM to/from PSM Transforms

Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires modifying the sequence PSM named Process Check. Add Ledger as a participant to the Process Check diagram. 9. Sync the sequence PIM from the newly updated PSM.	targeting C# Profile
10. Generate C# source code from PSMs. 11. Open C# source code project in Visual Studio or through integrated bridge provided by tool under evaluation. Create a new C# class BrokerageAccount inheriting from abstract class BankAccount. Add attributes and methods to the BrokerageAccount. 12. Sync the class PSM from the modified source code.	• Structural C# Profile based PSM to/from Code Transforms targeting C# code • Behavioral C# Profile based PSM to/from Code Transforms targeting C# code • Visual Studio Integration
13. Open C# source code project in Visual Studio or through integrated bridge provided by tool under evaluation. Modify Bank method ProcessCheck(), if balance < amount, then attempt to recover the difference from account holder's savings account before applying overdraft fees. 14. Sync the sequence PSM from the modified source code.	• Behavioral C# Profile based PSM to/from Code Transforms targeting C# code • Visual Studio Integration
15. Export the models to XMI and attempt to import into MagicDraw.	• Export of XMI Representation of UML Models.
16. Generate Java PSM models from PIM of Banking.	• Structural PIM to/from PSM Transforms targeting Java Profile • Behavioral PIM to/from PSM Transforms targeting Java Profile
17. Modify PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires adding a new class to the class PSM named Bank. Sync the class PIM from the newly updated Java PSM.	• Structural PIM to/from PSM Transforms targeting Java Profile • Behavioral PIM to/from PSM Transforms targeting Java Profile

18. Modify PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires modifying the sequence PSM named Process Check. Add Ledger as a participant to the Process Check diagram. 19. Sync the sequence PIM from the newly updated PSM.	• Behavioral PIM to/from PSM Transforms targeting Java Profile
20. Generate Java source code from PSMs. 21. Open Java source code project Eclipse or through integrated bridge provided by tool under evaluation. Create a new Java class BrokerageAccount inheriting from abstract class BankAccount. Add attributes and methods to the BrokerageAccount. 22. Sync the class PSM from the modified source code.	• Structural Java Profile based PSM to/from Code Transforms targeting Java code • Eclipse Integration
23. Open Java source code project in Eclipse or through integrated bridge provided by tool under evaluation. Modify Bank method ProcessCheck(), if balance < amount, then attempt to recover the difference from account holder's savings account before applying overdraft fees. 24. Sync the sequence PSM from the modified source code.	• Behavioral Java Profile based PSM to/from Code Transforms targeting Java code • Eclipse Integration

3.3.1 Evaluation Environment

Each tool is installed and evaluated within a cloned Windows 7 Professional virtual machine preloaded with Visual Studio, Eclipse, MagicDraw, and Subversion. Visual Studio and Eclipse are required to evaluate the tools effectiveness regarding integrating with popular IDEs. MagicDraw serves as the third party UML tool required to evaluate XMI import and export functionality with each tool. Table 7 provides further details regarding the specifications of the testing environment.

Table 7: Specification of Testing Environment

Manufacturer	Dell
Model	Optiplex 790
Operating System	Windows 7 Professional x64
Processor	Intel Core i5-2400 CPU @ 3.10GHz
RAM	8.00 GB
Virtualization Software	VMware Workstation 9.0.2
Allocated RAM to Virtual Machine	4.00 GB
Virtual Machine OS	Windows 7 Professional x86
Visual Studio Version	Visual Studio 2010 Professional
Eclipse Version	Juno Service Release 2
Subversion	CollabNet Cloud Offering
MagicDraw Version	MagicDraw UML 17.0.4

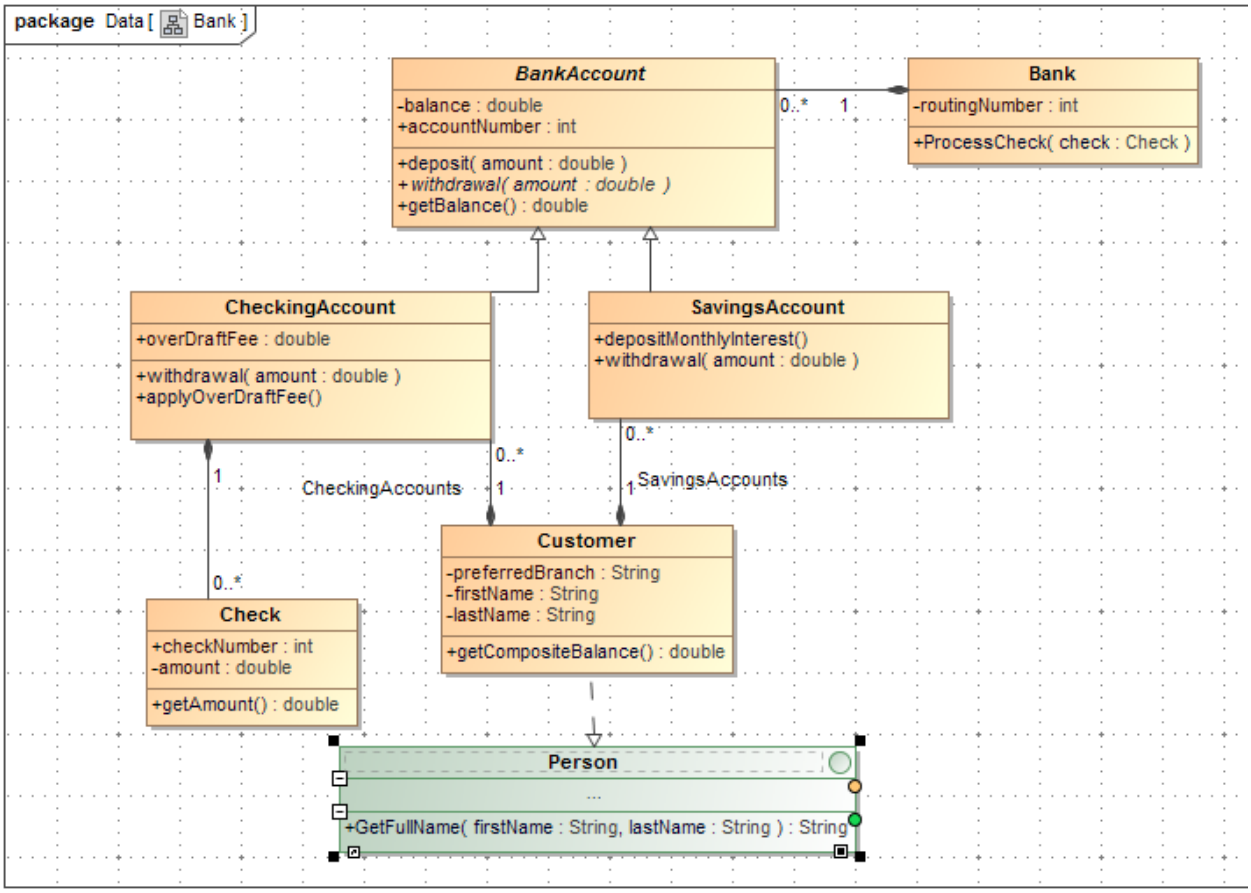
3.3.2 Formal Requirement Analysis as Input Parameters

In order to evaluate each tool in regards to their MDA capabilities, UML 2 class and sequence diagrams are composed and used as input parameters during the tool evaluation. In this evaluation the subject domain of a limited banking system is defined using UML class and sequence diagrams. The banking system domain was selected because of its familiarity among computer science and information technology peers as a generally recognized example subject domain.

3.3.2.1 The Class Diagram

Class diagrams are used to model and define the structural aspects of a domain. They can be composed using a variety object oriented concepts. The following object oriented concepts are applied within the class diagram input parameter that describes the structure of the banking system; visibility, attributes, operations, associations, generalizations, abstractions, and interfaces. These concepts are tested for transformation completeness within the procedure logs for each tool. Figure 1 shows the class diagram composed as the structural input parameter for the tool evaluations.

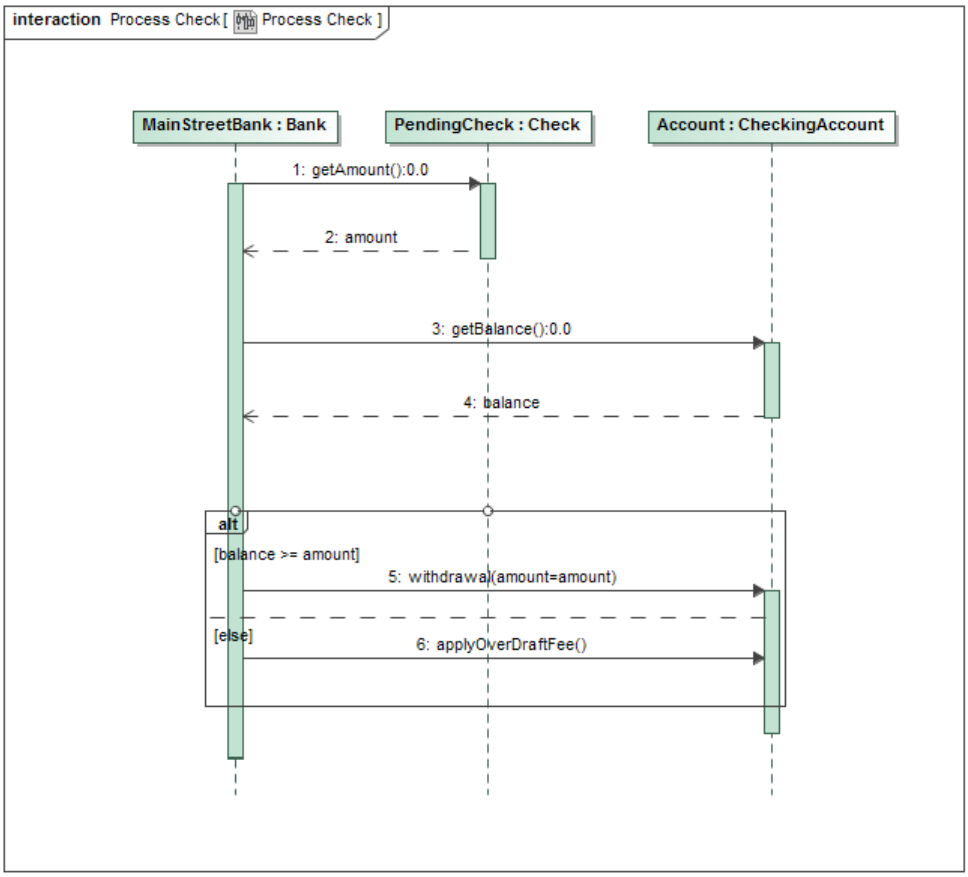
Figure 1: Banking System Class Diagram



3.3.2.2 The Sequence Diagram

Sequence diagrams are used to model and define the behavior of a system through ordered interactions often between multiple classes. In order to evaluate the tools effectiveness at transforming sequence diagrams to code the following parts are defined within the input sequence diagram; multiple participant classes, and events composed of synchronized messages with return types. These concepts are tested for transformation completeness within the procedure logs for each tool. Figure 2 shows the sequence diagram composed as the behavioral input parameter for the tool evaluations.

Figure 2: Banking System Sequence Diagram



Chapter 4: Results

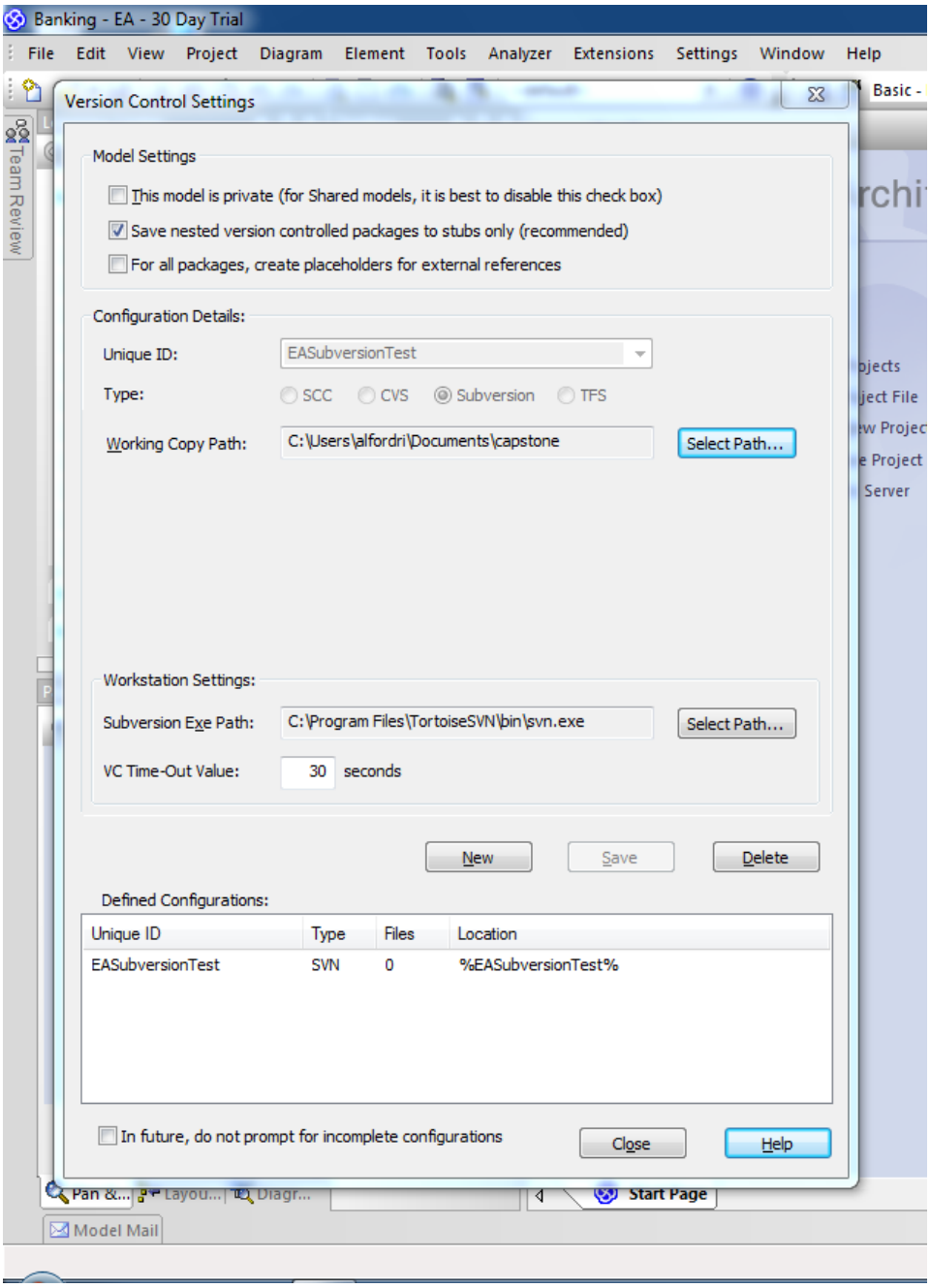
The goal of this project is to evaluate a set of software modeling and architectural tools within the context of their MDA capabilities. Each tool is evaluated following the defined procedure log template within its dedicated virtual environment using the provided input UML diagrams. Results are compiled for the defined metric suite for each of the selected tools based on their procedure log instances. The selected tools are evaluated and compared to each other.

4.1 Evaluation of Enterprise Architect

The evaluation of Enterprise Architect is documented by the evaluator in the following Procedure Log.

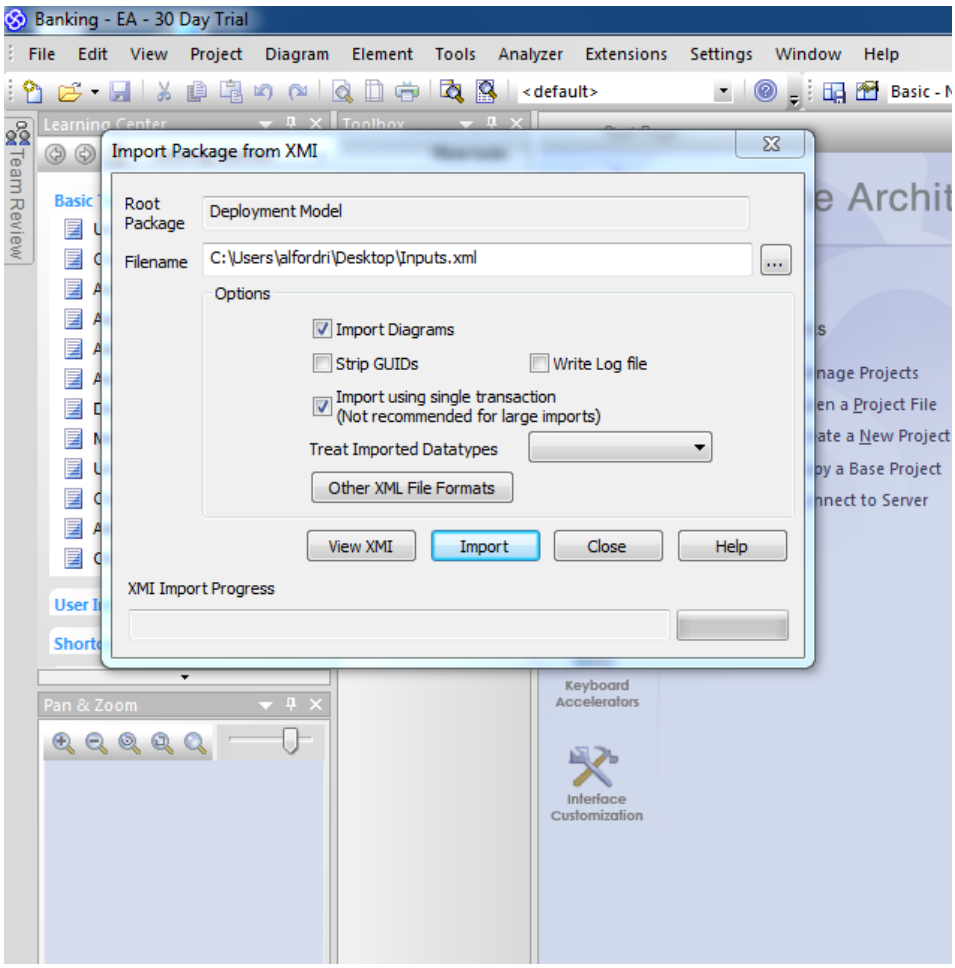
1. Setup version control settings in MDA Tool: Setting up and connecting to a subversion repository in Enterprise Architect was straightforward. A keyword search from Help Contents within the Enterprise Architect quickly illustrated the settings required to connect to an existing version control repository. The entire setup was completed via a single graphic interface form within the Enterprise Architect application as illustrated in Figure 3.

Figure 3: Enterprise Architect version control setup



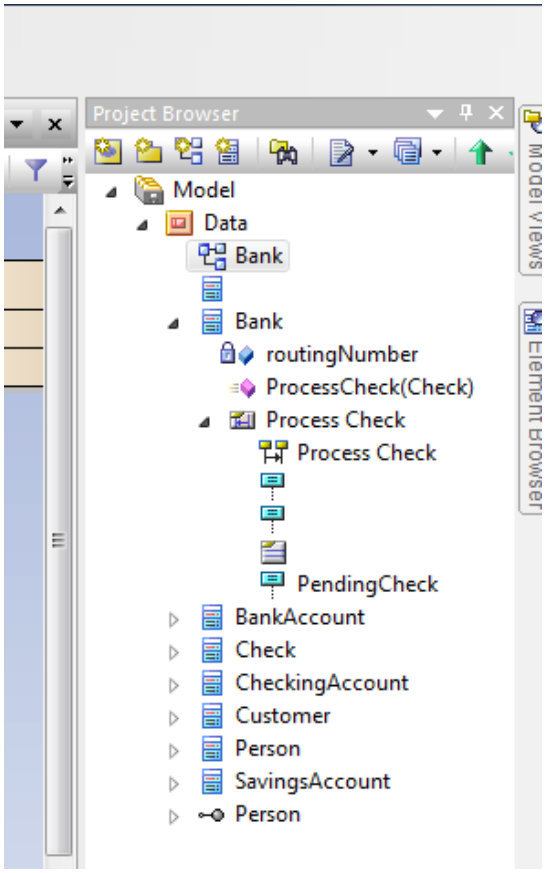
2. Import XMI of PIM of Banking from MagicDraw: Enterprise Architect provides a graphical interface for importing XMI into a new or existing project illustrated in Figure 4.

Figure 4: Enterprise Architect XMI Import



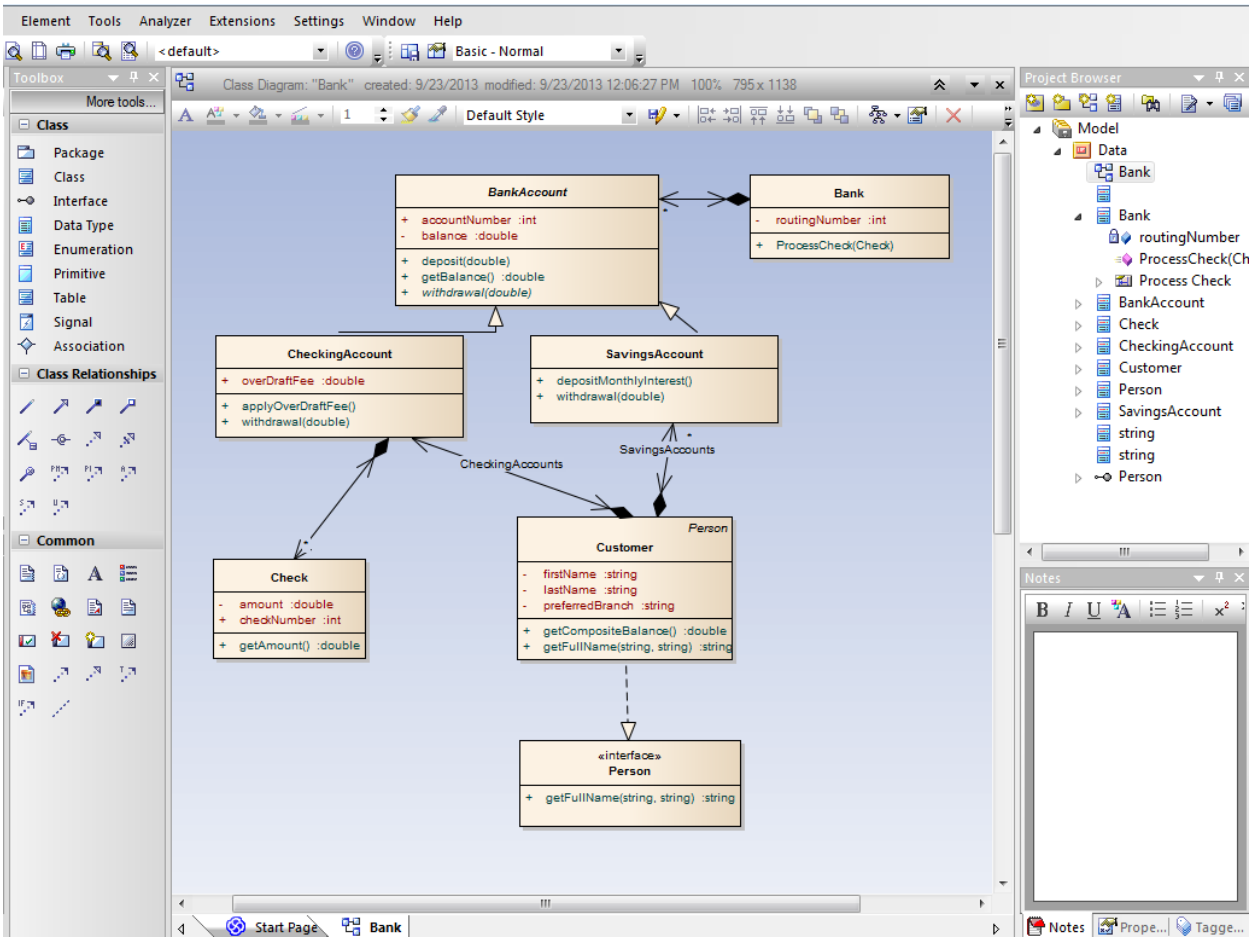
The import package progress completed with no reported errors. Figure 5 shows the models and UML elements added to the Project Brower pane.

Figure 5: Enterprise Architect Project Browser



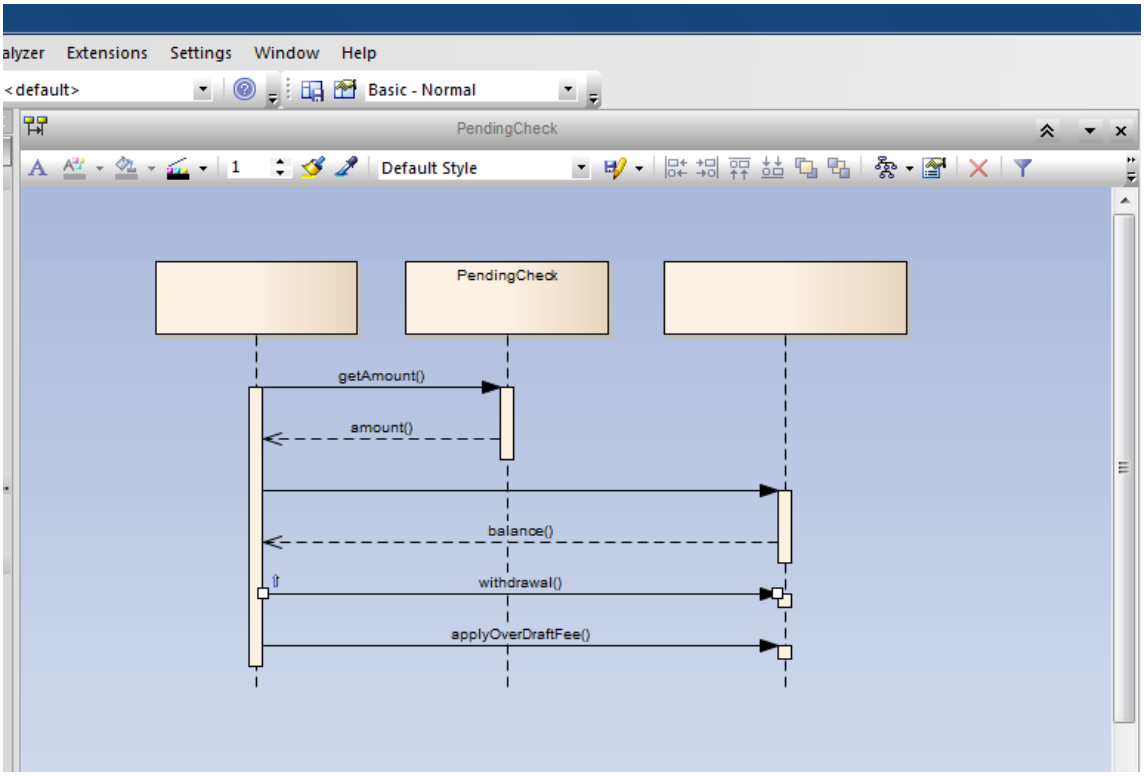
The results of the XMI import were mixed. The generated class diagram Bank, illustrated in Figure 6, successfully rendered and preserved the following object oriented concepts from the XMI import; objects, visibility, attributes, operations, generalizations, abstractions, and interfaces. It failed to preserve some of the objects' association multiplicities; therefore, these associations were manually updated within Enterprise Architect by the evaluator.

Figure 6: Enterprise Architect class PIM from XMI



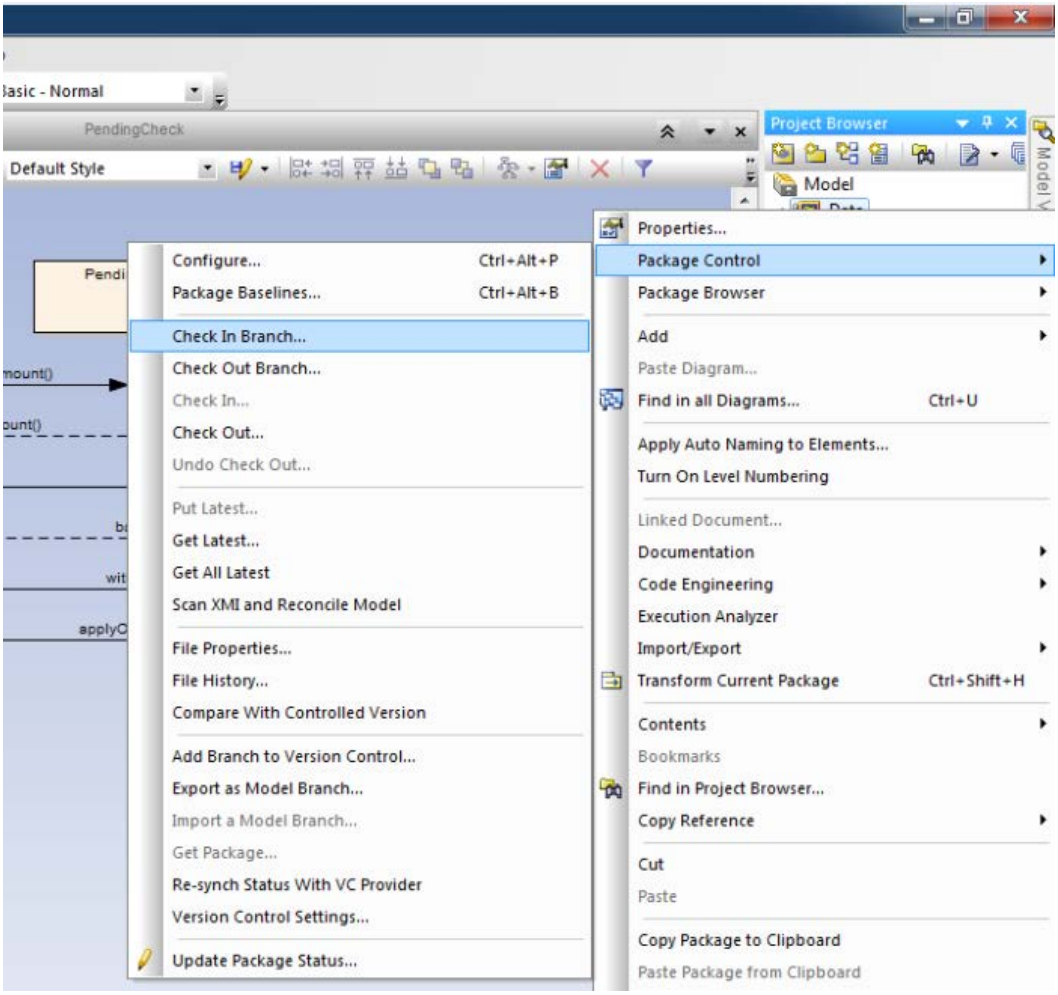
The sequence diagram import did not fare as well as the class diagram import. First, all of the participants defined in the input sequence diagram were not preserved in the generated sequence diagram. Second, the alt guard conditions defined in the input sequence diagram responsible for controlling the execution of `applyOverDraftFee()` were not preserved as a result of the XMI import. The generated sequence diagram is illustrated in Figure 7. The architect made manual updates to the generated sequence diagram to correct the issues reported.

Figure 7: Enterprise Architect sequence PIM from XMI



3. Commit to version control: Committing the generated models to version control within Enterprise Architect was successful. Version control commits with Enterprise Architect must be performed at the package level. This constraint hinders the ability of multiple members of the architect team wishing to work with different model elements within the same package at the same time. Figure 8 illustrates the drop down menu options available for version control within Enterprise Architect.

Figure 8: Enterprise Architect Version Control Check In

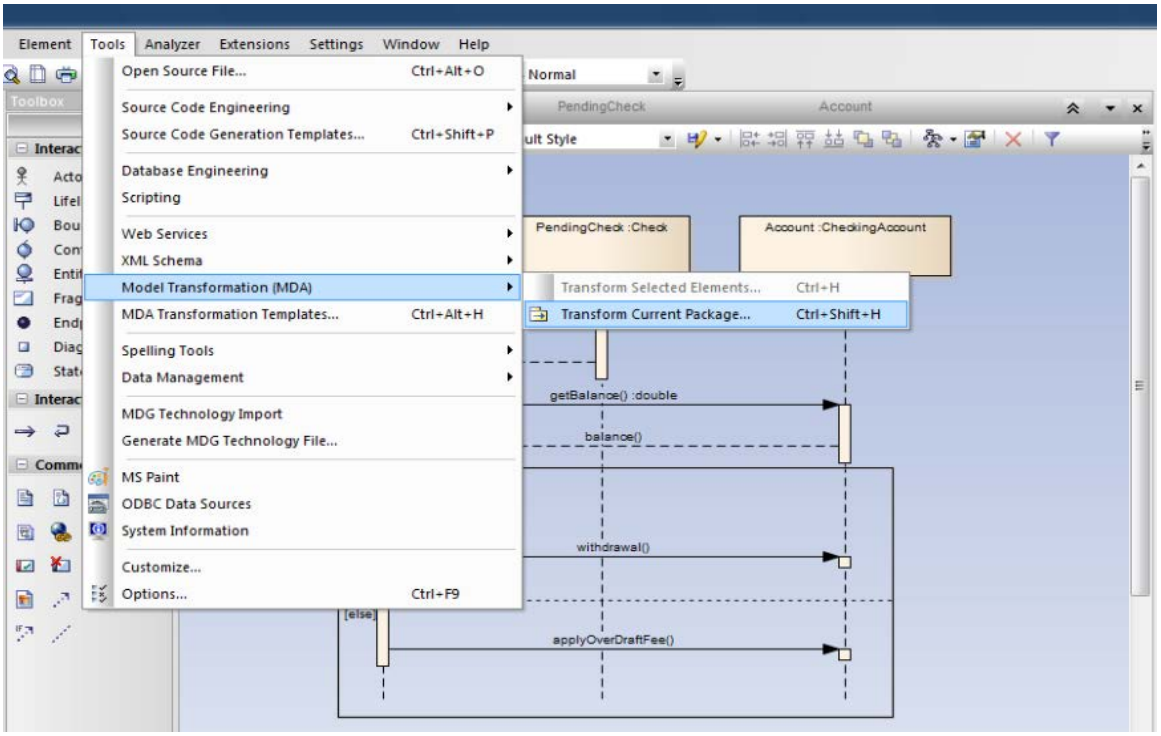


4. Retrieve PIMs from version control: Retrieving models from version control within Enterprise Architect was successful. Models are checked out within Enterprise Architect at the package level. As previously indicated, the limitation of package level source control hinders the ability of multiple members of the architect team wishing to work with different model elements within the same package at the same time.

5. Generate C# PSM models from PIM of Banking: Enterprise Architect provides a menu option for producing PSM models from PIMs via the Model Driven Architecture feature illustrated in Figure 9. First, the evaluator creates an empty package named CSharp within the

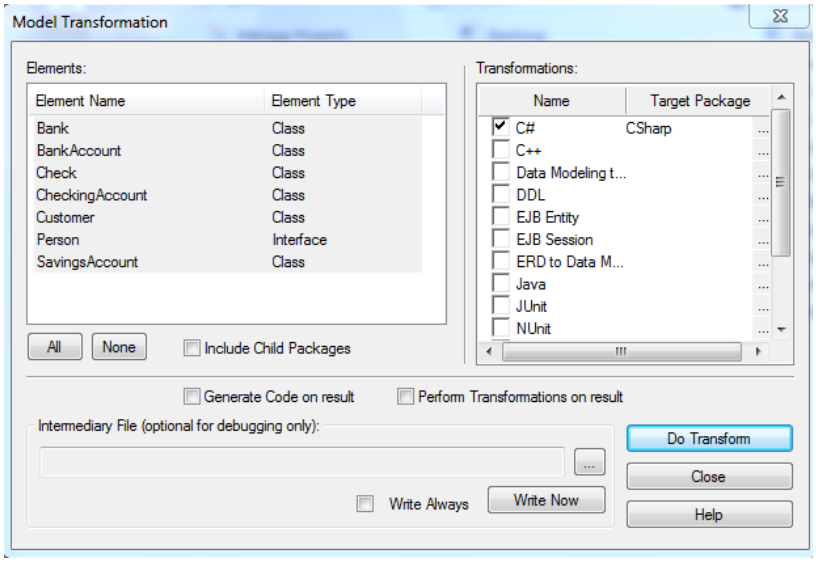
root package Model. The evaluator then browses to the Transform Current Package option illustrated in Figure 9.

Figure 9: Enterprise Architect MDA Transform Screen 1



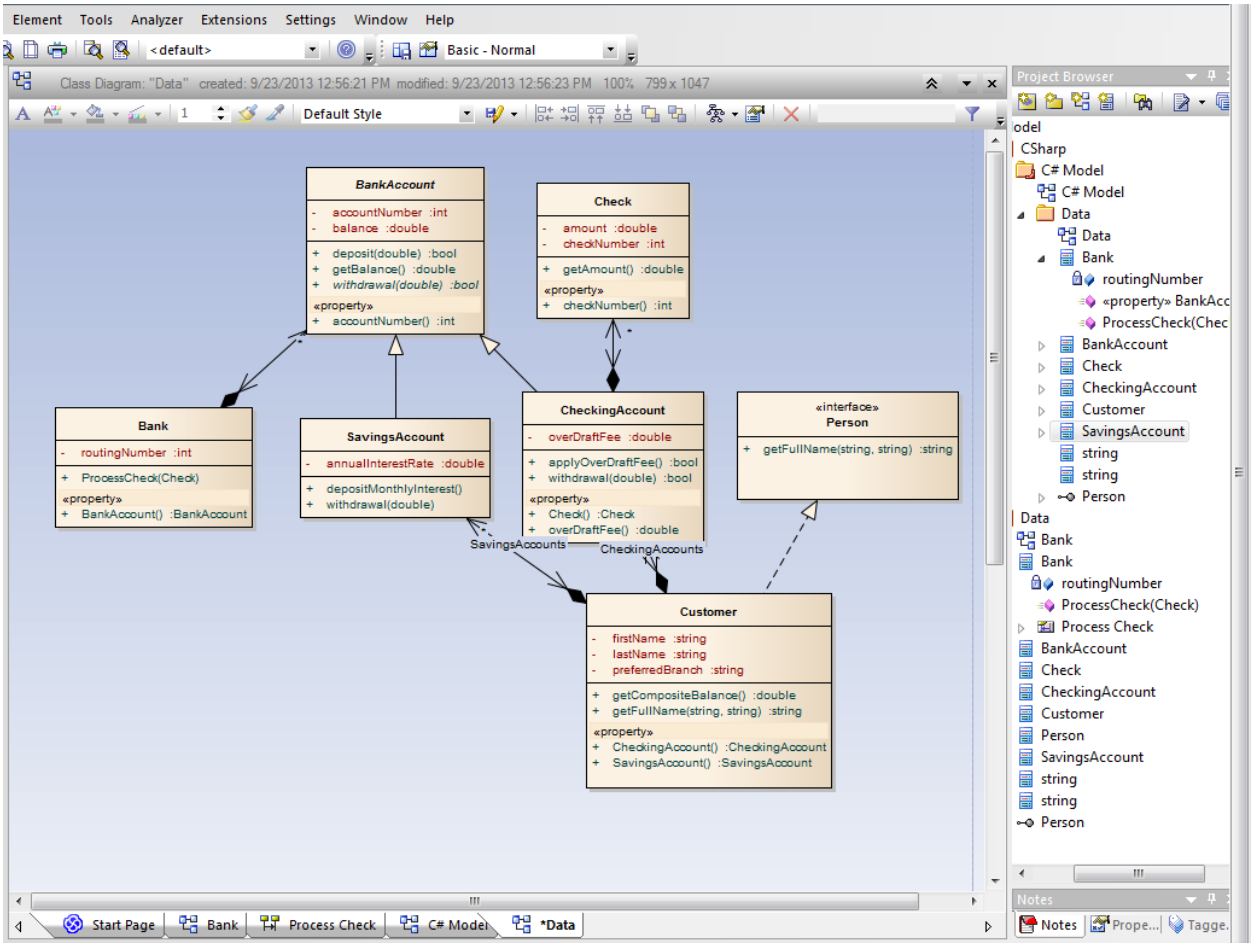
A pop up window illustrated in Figure 10 is provided by Enterprise Architect requesting information from the architect such as the PSM target profile and destination package.

Figure 10: Enterprise Architect MDA Transformation Screen 2



After executing the transform, the CSharp package is populated with the corresponding PSM elements targeting the C# profile. The class PSM is illustrated in Figure 11. After reviewing the generated class PSM, several issues are worth noting. First, for all associations defined in the class PIM, a getter operation is produced in the mapped class PSM. All the getter operations produced return a single instance of the associated class regardless of the defined multiplicity. Second, in an irregular manner, some of the primitive attributes defined in the class PIM were mapped to getter operations while others are not. The evaluator cannot find any straightforward cause for this inconsistency.

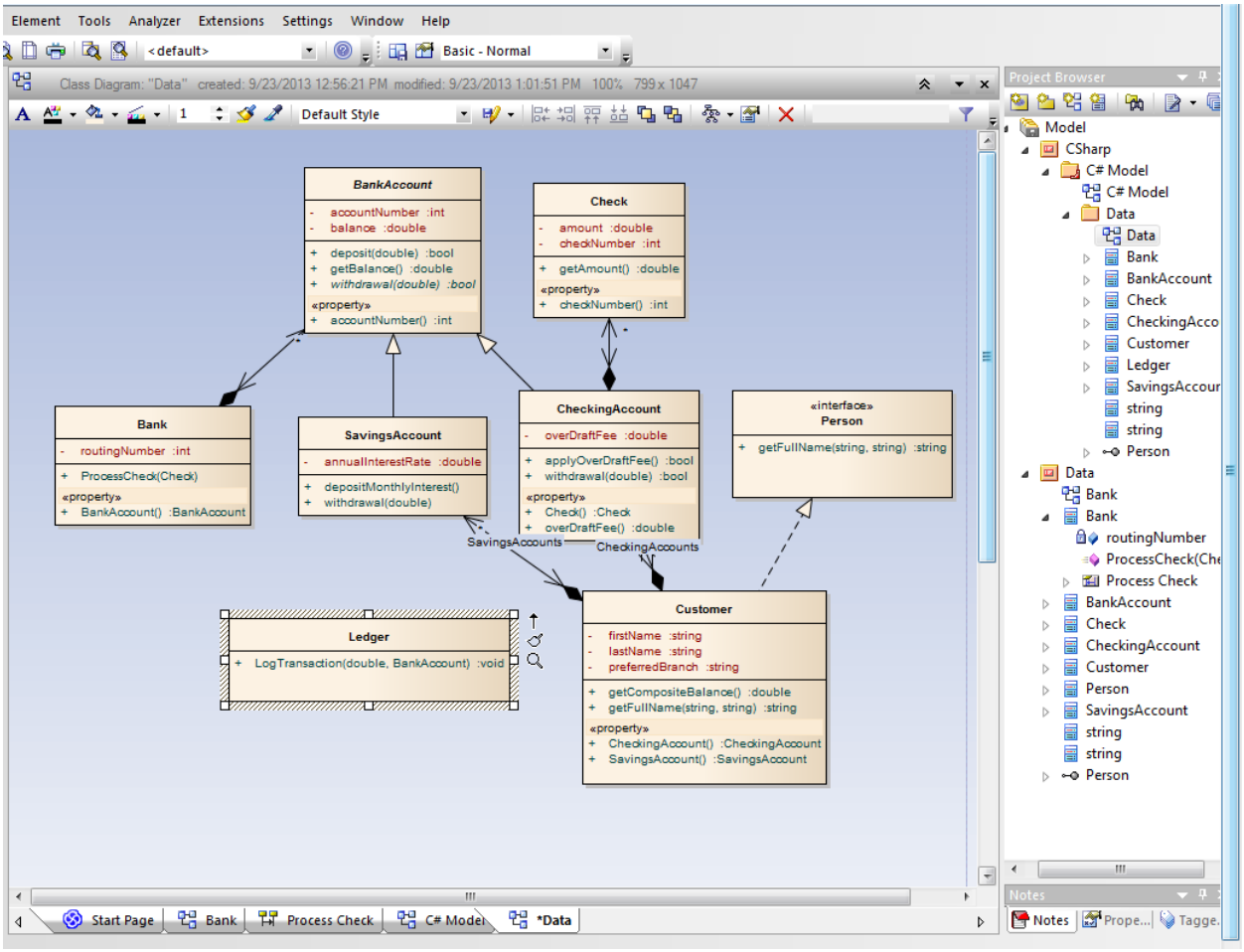
Figure 11: Enterprise Architect class C# PSM from MDA Transformation



The transformation did not produce a corresponding sequence PSM for the sequence PIM describing the Bank operation `ProcessCheck()`.

6. Modify PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires adding a new class to the class PSM: As illustrated in Figure 12, adding the Ledger class to the class PSM was straightforward and accomplished by utilizing the class modeling tools within Enterprise Architect.

Figure 12: Enterprise Architect class PSM modification



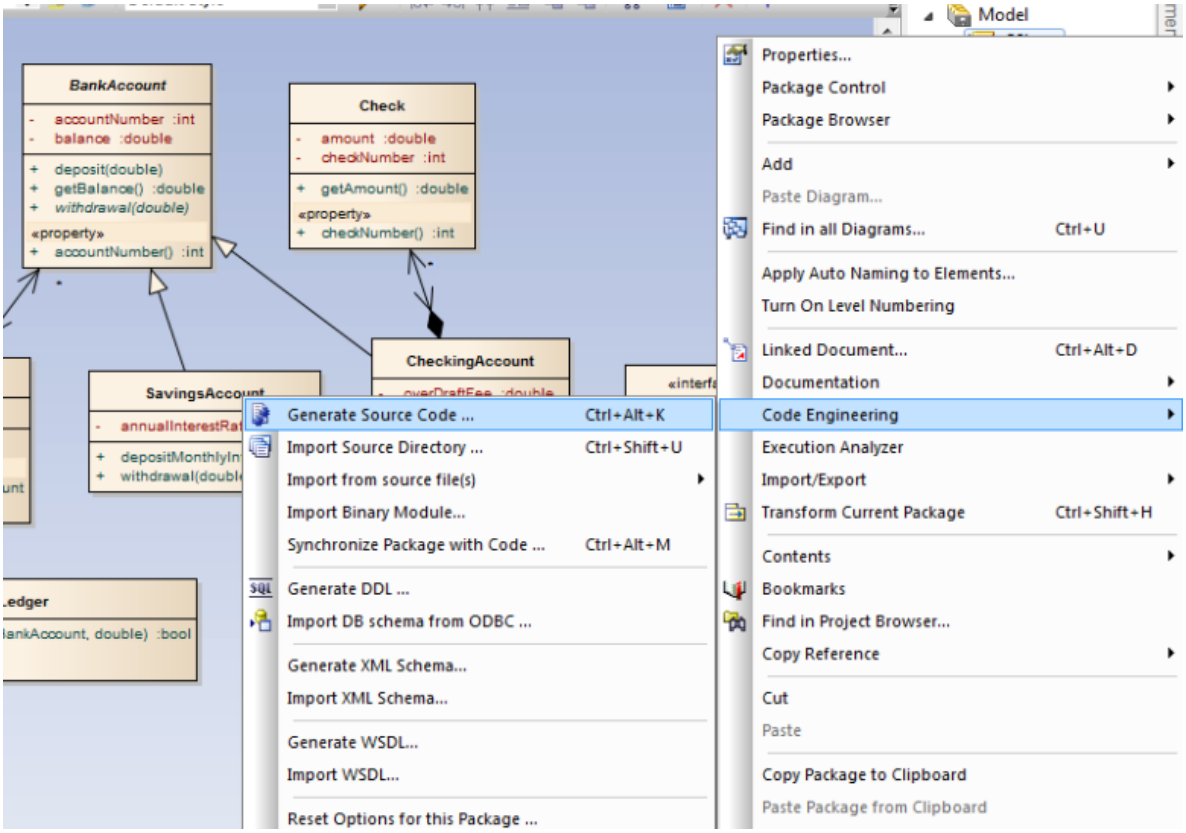
7. Sync the class PIM from the newly updated PSM: Enterprise Architect does not support syncing from PSMs to PIMs. It only provides forward transformations from PIMs to PSMs.

8. Modify PSM models to accommodate for a Ledger Class that logs withdrawals and deposits for a given BankAcount class. This step requires modifying the sequence PSM named Process Check. Add Ledger as a participant to the Process Check diagram: Enterprise Architect does not produce the required sequence PSM; therefore, this procedure step is not required.

9. Sync the sequence PIM from the newly updated PSM: Enterprise Architect does not support syncing from PSMs to PIMs. It only provides forward transformations from PIMs to PSMs.

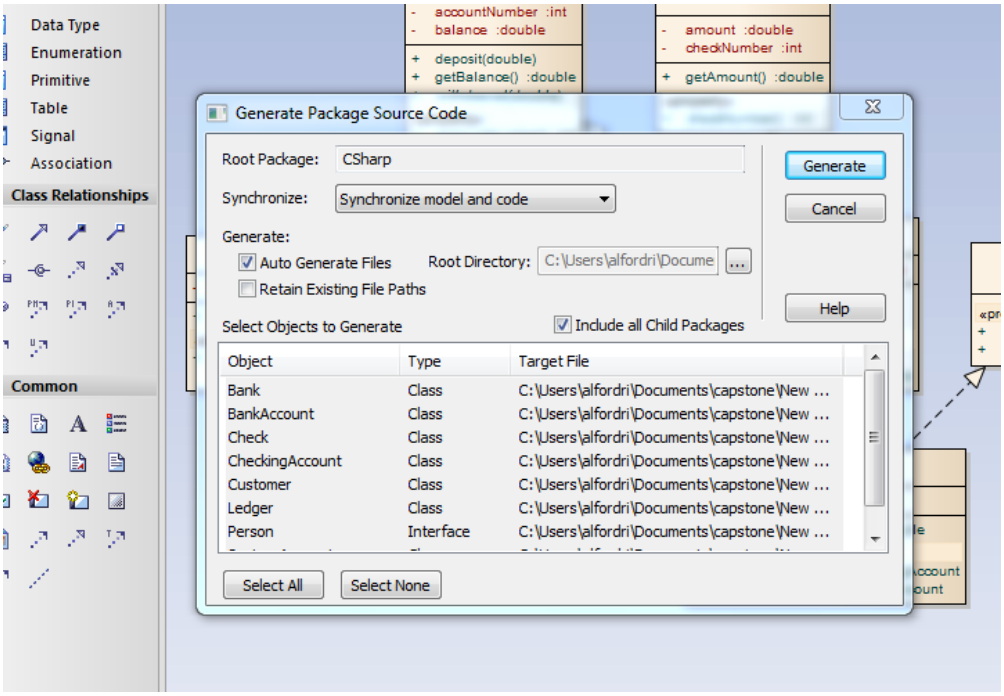
10. Generate C# source code from PSMs: Enterprise Architect provides a menu option under Code Engineering for source code generation as illustrated in Figure 13.

Figure 13: Enterprise Architect Code Engineering Screen 1



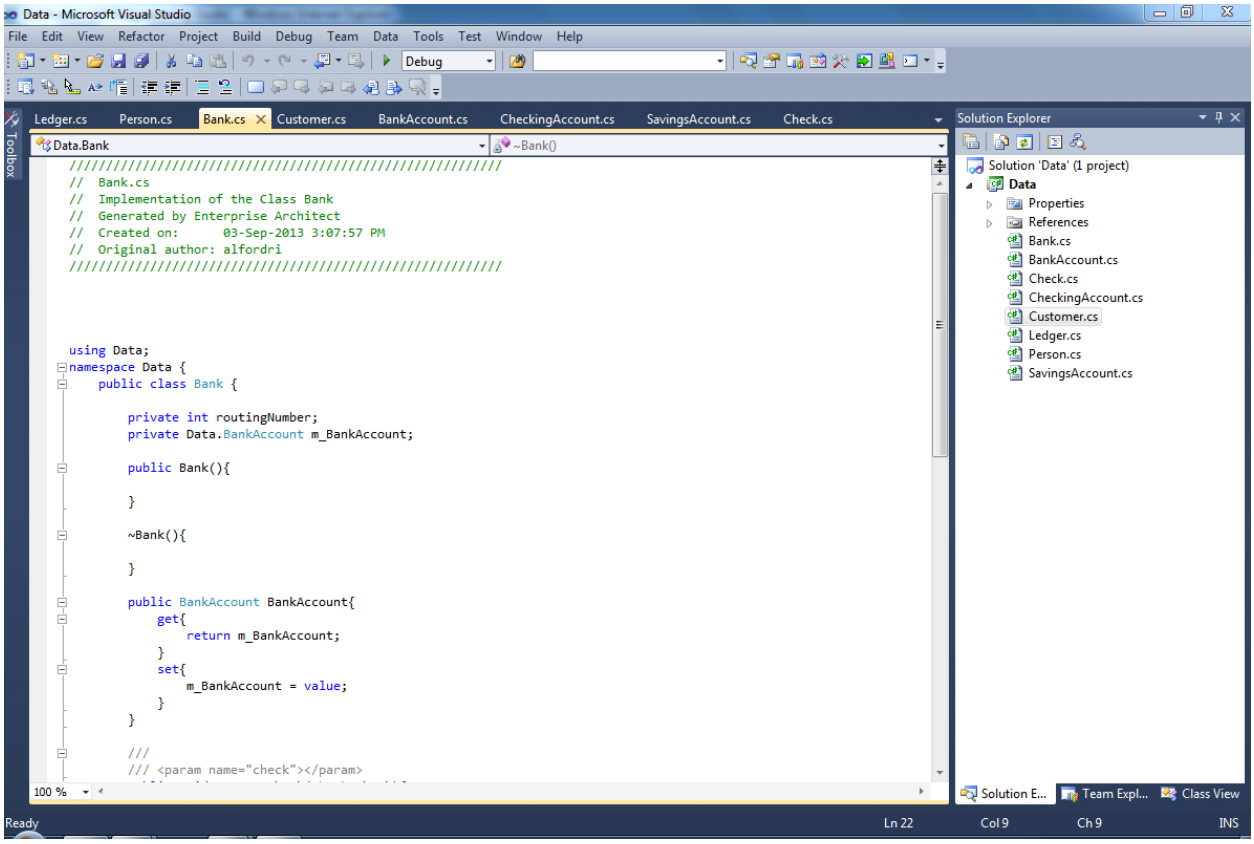
After selecting the Generate Source Code option, Enterprise Architect provided a form to specify details for the code generation including an option for synchronizing model and code as illustrated in Figure 14.

Figure 14: Enterprise Architect Code Engineering Screen 2



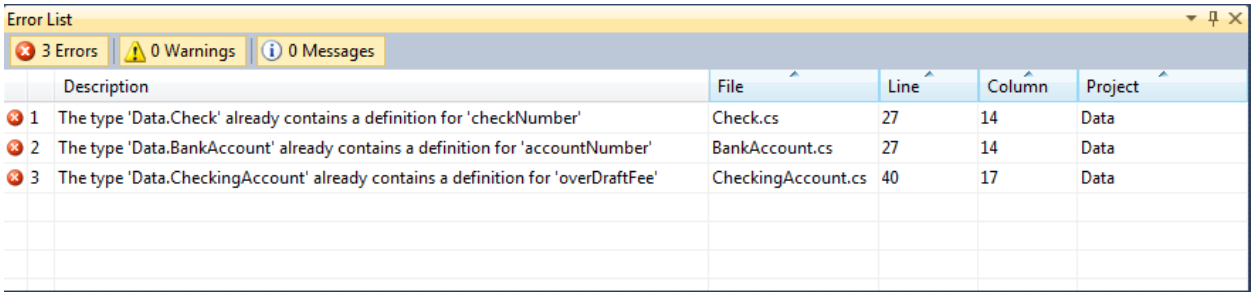
11. Open C# source code project in Visual Studio or through integrated bridge provided by tool under evaluation. Create a new C# class BrokerageAccount inheriting from abstract class BankAccount. Add attributes and methods to the BrokerageAccount: Enterprise Architect provides an integration bridge to Visual Studio available as an additional download. The bridge is named the Model Driven Generation (MDG) Link for Visual Studio. It is compatible with Visual Studio 2005, 2008, and 2010. The bridge can be licensed per seat or floating and the cost per license ranges from $89 to $145. Once installed, the generated source code from step 10 is merged into a new Visual Studio Project manually by the evaluator by creating a Visual Studio Solution and Project in the directories associated with Enterprise Architect's code generation project settings. The evaluator inspected the generated code from within Visual Studio. As illustrated in Figure 15, all classes and interfaces were created during the code generation process resulting from step 10.

Figure 15: Enterprise Architect C# Code Generation Results



The evaluator attempted to build the Visual Studio project composed of the generated class library. The compiler reported three instances of the same syntax error illustrated in Figure 16.

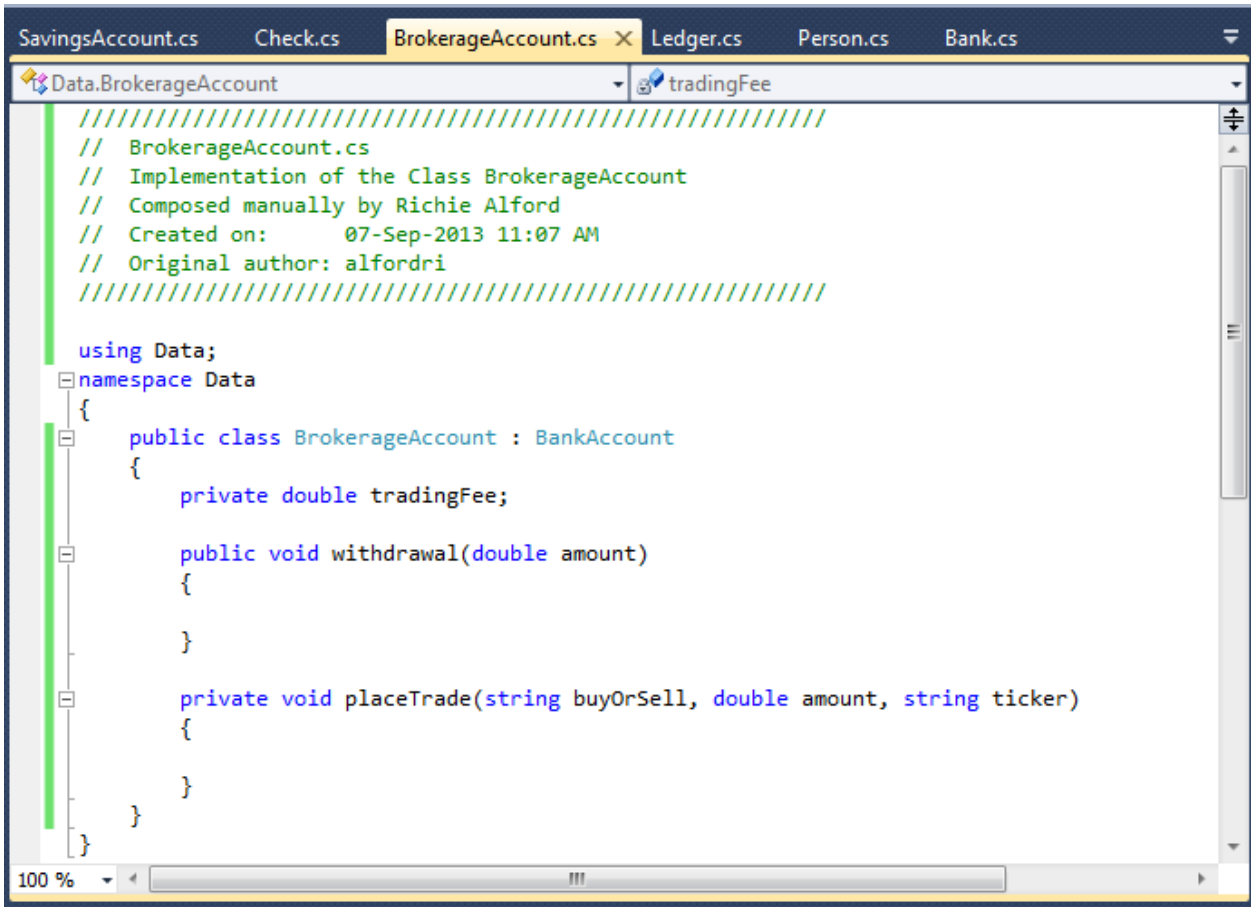
Figure 16: Enterprise Architect C# Code Generation Compiler Error List



Based on inspection from the evaluator, the syntax errors are related to a common pattern identified by the evaluator in step 5. The syntax errors are mapped to each instance in the class PSM where getter operations were created for primitive attributes. To test Enterprise Architect's

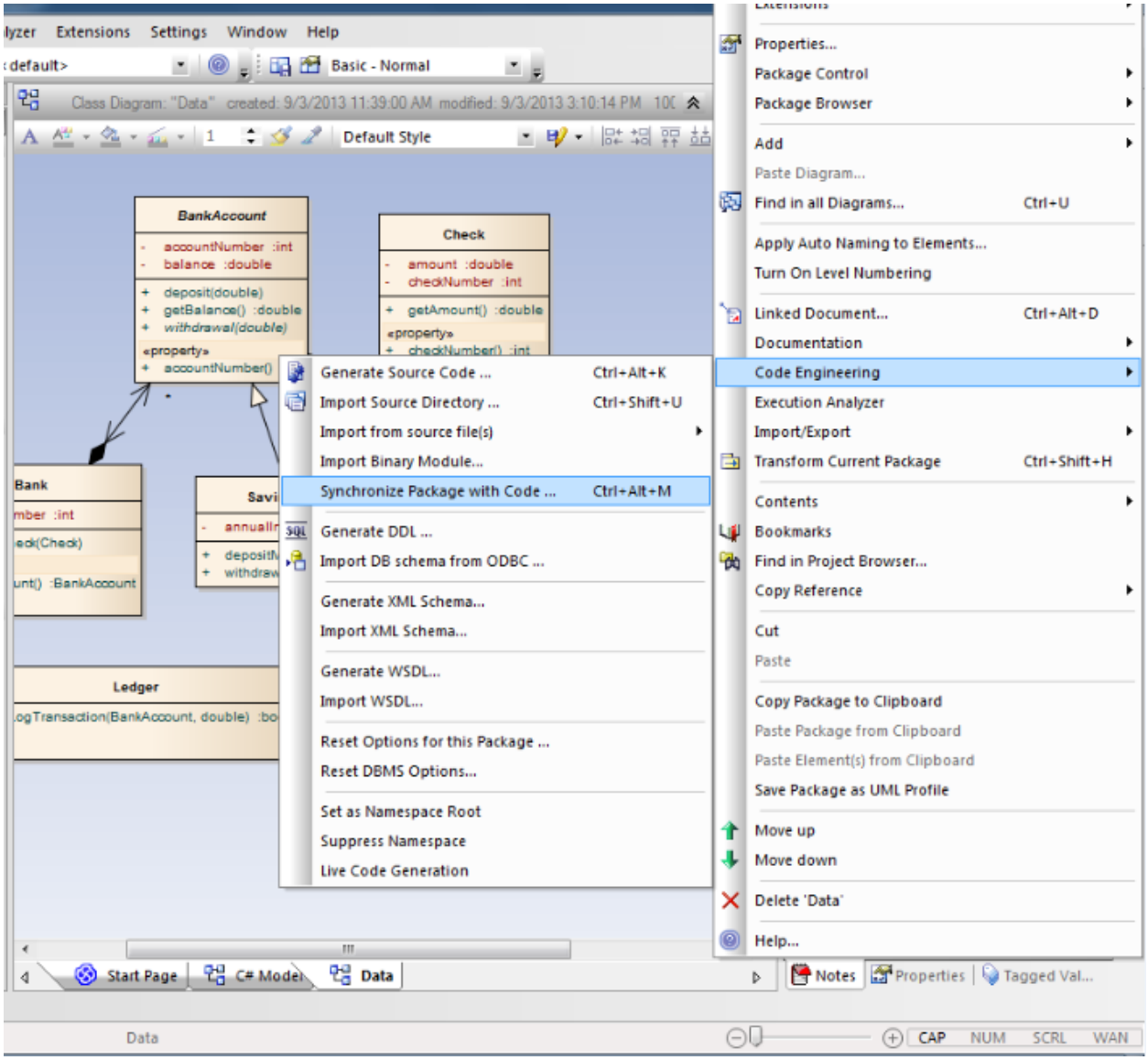
capabilities to sync PSMs from source code modifications the evaluator created a new class within the Visual Studio project named BrokerageAccount shown in Figure 17.

Figure 17: Enterprise Architect BrokerageAccount Source File



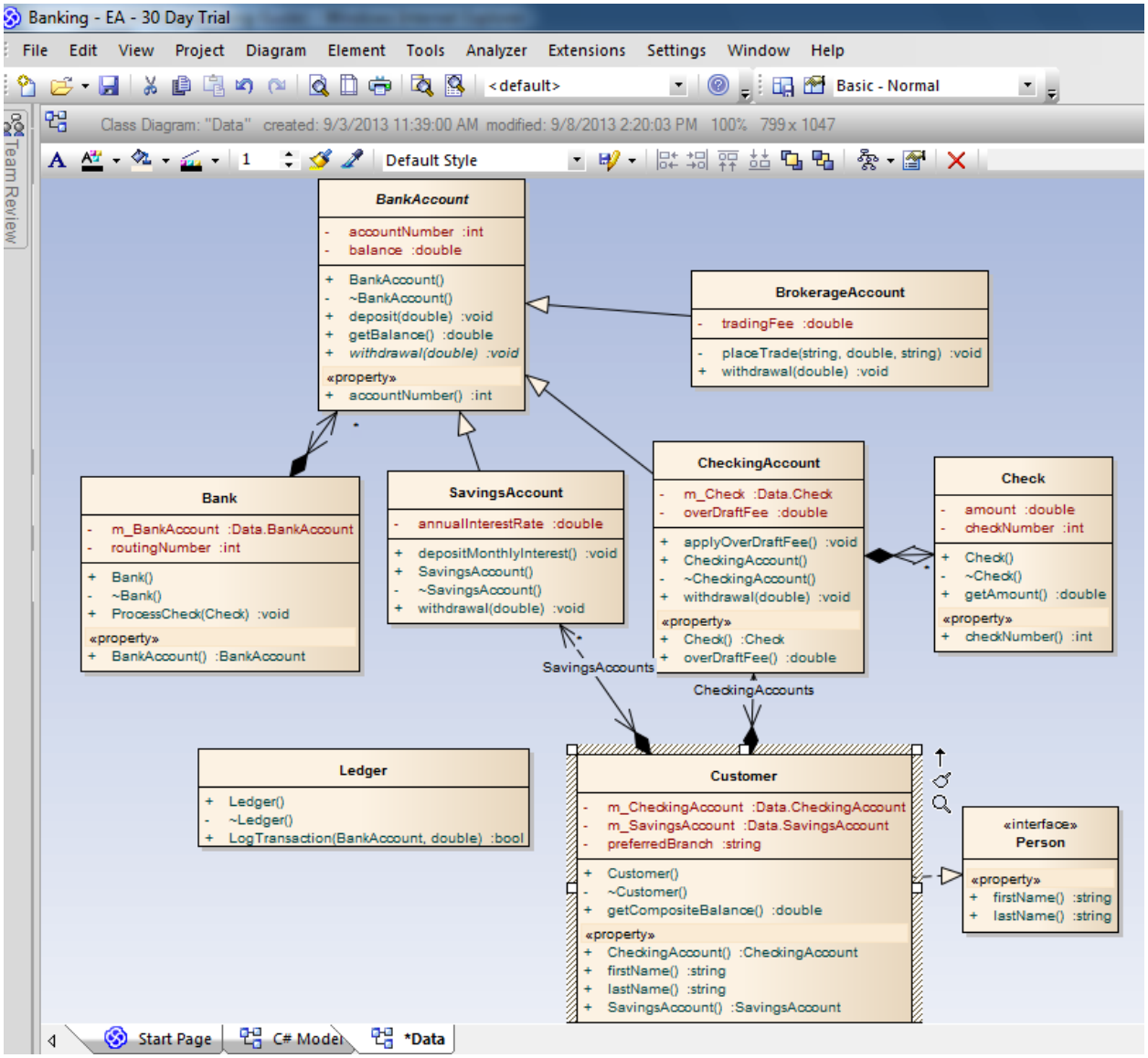
12. Sync the class PSM from the modified source code: The evaluator synced the PSM with the source code project by selecting Synchronize Package with Code from the Code Engineering menu option available within Enterprise Architect. This option is illustrated in Figure 18.

Figure 18: Enterprise Architect Synchronize Model from Code



The synchronization completed with no errors; however, the new BrokerageAccount class was not included in the synchronization. The existing PSM elements did sync with the corresponding auto-generated source files from step 10. The PSM elements were updated with the constructers, destructors, getter and setter methods, and attributes created based on associations. The evaluator was able to add the BrokerageAccount class to the PSM package by selecting the Import from source file(s) option shown in Figure 18. The modified PSM from code synchronization is displayed in Figure 19.

Figure 19: Enterprise Architect Synchronize C# profile class PSM from Source Code

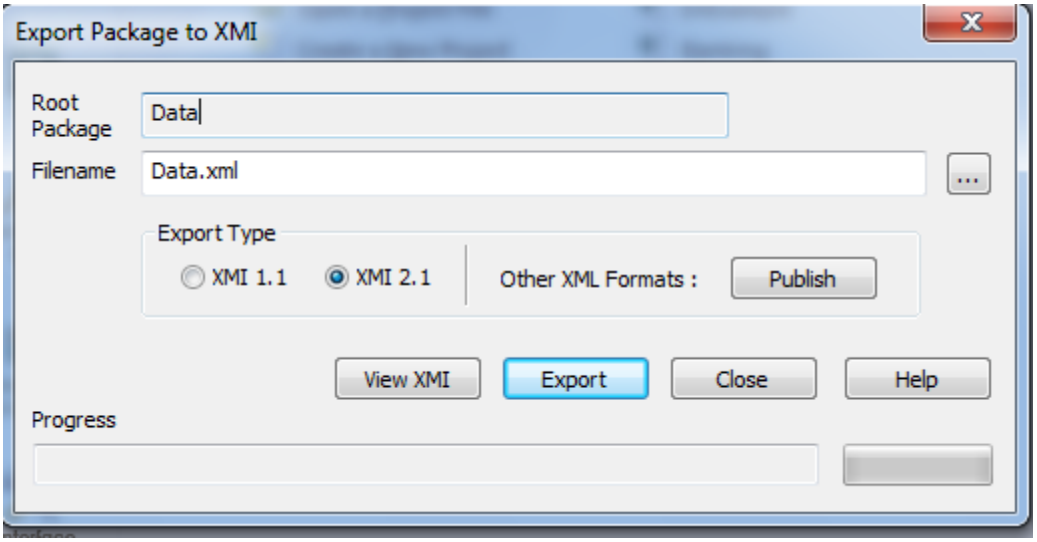


13. Open C# source code project in Visual Studio or through integrated bridge provided by tool under evaluation. Modify Bank method ProcessCheck(), if balance < amount, then attempt to recover the difference from account holder's savings account before applying overdraft fees: Although Enterprise Architect supports code generation from sequence models, the evaluator could not establish how to achieve this functionality; therefore, this step was not performed by the evaluator. The evaluator located the menu option for code generation from sequence diagram; however, Enterprise Architect had disabled it for the active project.

14. Sync the sequence PSM from the modified source code: Although Enterprise Architect supports code generation from sequence models, the evaluator could not establish how to achieve this functionality; therefore this step was not performed by the evaluator. The evaluator located the menu option for code generation from sequence diagram; however, Enterprise Architect disabled it.

15. Export the models to XMI and attempt to import into MagicDraw: Enterprise Architect provides a menu option to export model elements as XMI as illustrated in Figure 20. MagicDraw successfully imported both the class and sequence diagrams and related elements from the XMI export from Enterprise Architect.

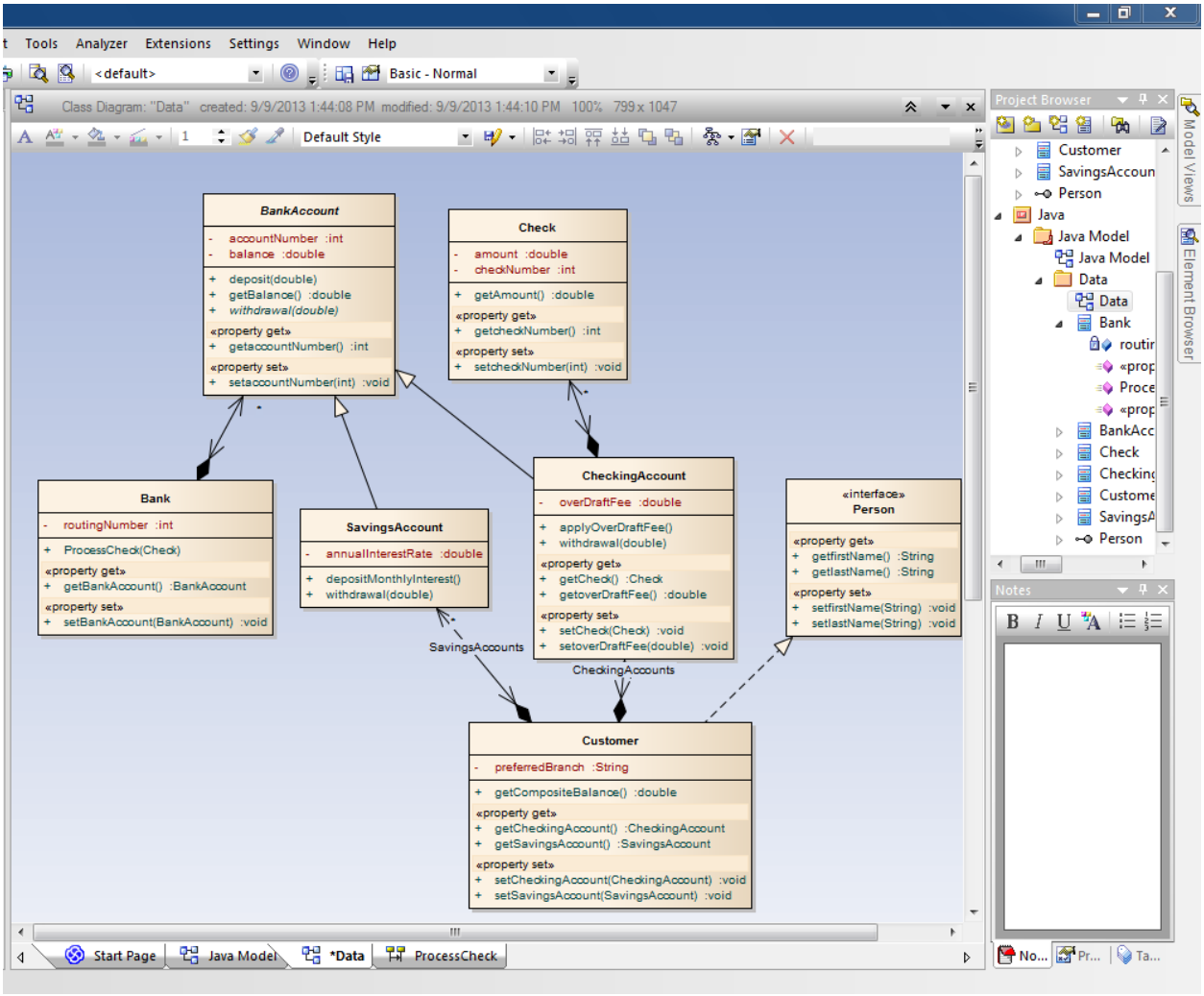
Figure 20: Enterprise Architect Export to XMI



16. Generate Java PSM models from PIM of Banking: The evaluator executed the same Enterprise Architect screens presented in Figures 9 and 10 from step 5; this time targeting Java. The results were similar to the PIM to PSM transformation targeting C#. In fact, the same discrepancies outlined in step 5 manifested themselves in the Java transform. The only difference is, rather than producing only getter properties for some of the attributes as described in step 5,

the Java PSM transformation also includes setter properties. The class PSM targeting Java is illustrated in Figure 21.

Figure 21: Enterprise Architect Enterprise Java profile class PSM from MDA Transformation



The transformation did not produce a corresponding sequence PSM from the sequence PIM describing the Bank operation `ProcessCheck()`.

17. Modify Java profile PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAccount class. This step requires adding a new class to the Java profile class PSM: Adding the Ledger class to the class PSM is

straightforward and is accomplished by utilizing the class modeling tools within Enterprise Architect.

18. Sync the Java Profile class PIM from the newly updated PSM: Enterprise Architect does not support syncing from PSMs to PIMs. It only provides forward transformations from PIMs to PSMs.

19. Modify Java profile PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires modifying the Java profile sequence PSM named Process Check. Add Ledger as a participant to the Process Check diagram: Enterprise Architect does not produce the required sequence PSM; therefore, this procedure step is not required.

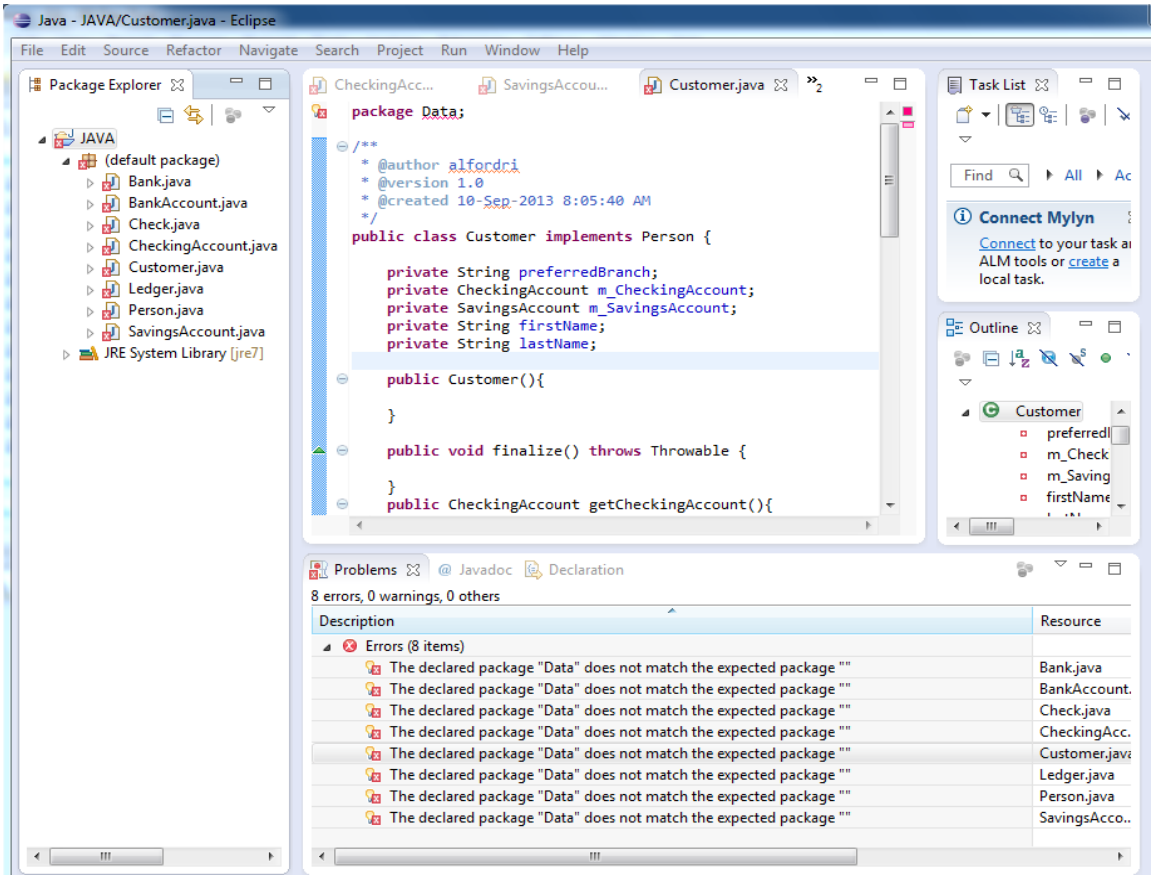
20. Sync the Java profile sequence PIM from the newly updated Java profile PSM: Enterprise Architect does not support syncing from PSMs to PIMs. It only provides forward transformations from PIMs to PSMs.

21. Generate Java source code from Java profile PSMs: Enterprise Architect provides a menu option under Code Engineering for source code generation as described in step 10. The Java PSM is selected as the source model and Java is selected as the target source code. Code generation completed without error.

22. Open Java source code project Eclipse or through integrated bridge provided by tool under evaluation. Create a new Java class BrokerageAccount inheriting from abstract class BankAccount. Add attributes and methods to the BrokerageAccount: Enterprise Architect provides an integration bridge to Eclipse available as an additional download. The bridge is named the Model Driven Generation (MDG) Link for Eclipse. The bridge can be

licensed per seat or floating and the cost per license ranges from $89 to $145. Once installed, the generated source code from step 10 is merged into a new Eclipse Project manually by the evaluator by creating an Eclipse Workspace and Java Project in the directories associated with Enterprise Architect's code generation project settings. The evaluator inspected the generated code from within Eclipse. As illustrated in Figure 22, all classes and interfaces were created during the code generation process resulting from step 21. The generated Java code manifested only one type of syntax error related to package name; however, this is a user error and can be resolved easily.

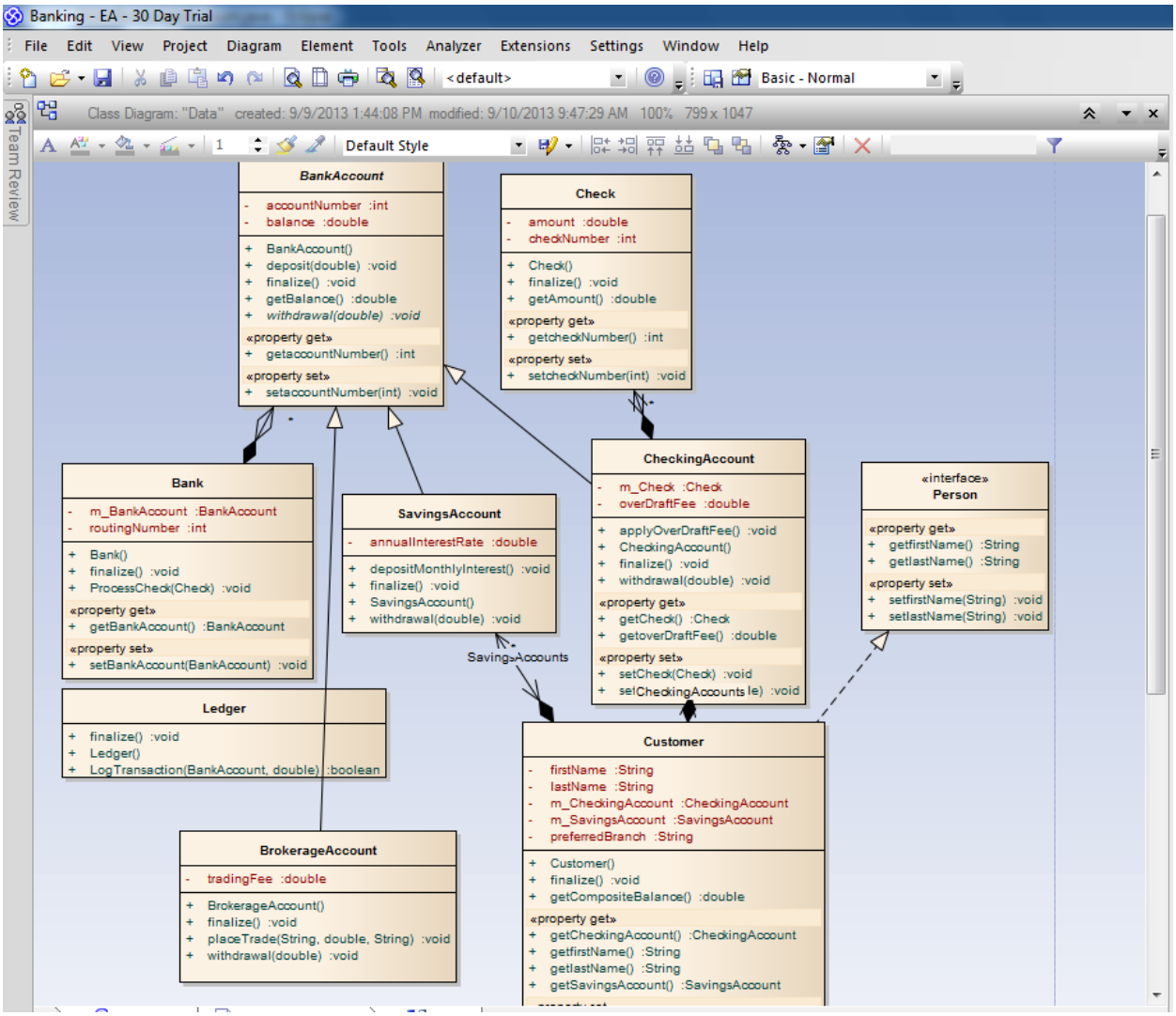
Figure 22: Enterprise Architect C# Code Generation Results



23. Sync the Java profile class PSM from the modified source code: The evaluator synced the PSM from code by selecting Synchronize package from source from the Code Engineering menu

option available within Enterprise Architect. This option is illustrated in Figure 18. The synchronization completed with no errors; however, the new Brokerage Account was not included in the synchronization. The PSM elements were updated with the constructers, destructors, getter and setter methods, and attributes created based on associations. The evaluator was able to easy add the BrokerageAccount class to the PSM package by simply selecting the Import from source file(s) option shown in Figure 18. The modified PSM from code synchronization is displayed in Figure 23.

Figure 23: Enterprise Architect Synchronize Java profile class PSM from Source Code



24. Open Java source code project in Eclipse or through integrated bridge provided by tool under evaluation. Modify Bank method ProcessCheck(), if balance < amount, then attempt to recover the difference from account holder's savings account before applying overdraft fees: Although Enterprise Architect supports code generation from sequence models, the evaluator could not establish how to achieve this functionality; therefore this step was not performed by the evaluator. The evaluator located the menu option for code generation from sequence diagram; however, it was disabled in Enterprise Architect.

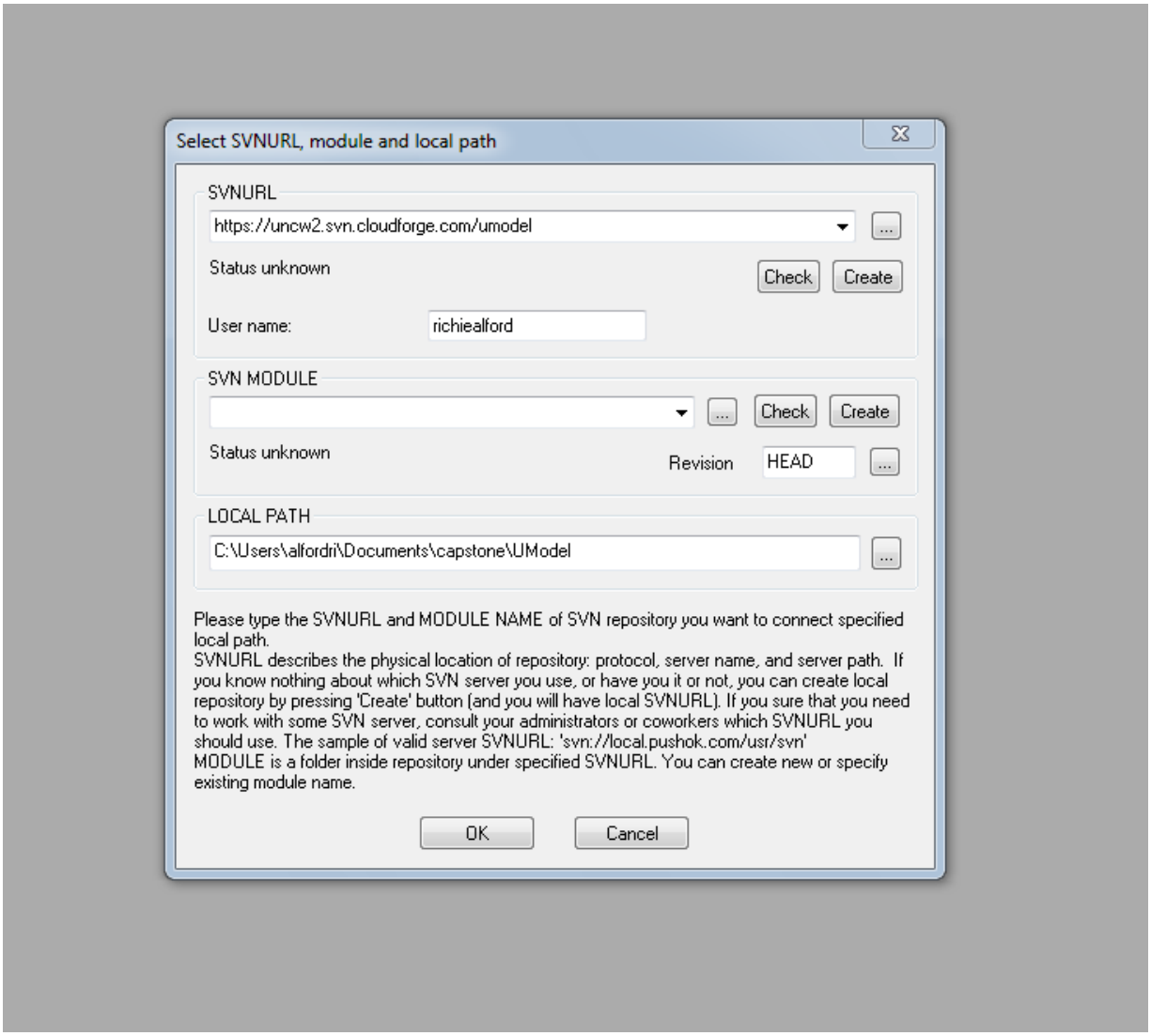
25. Sync the Java profile sequence PSM from the modified source code: Although Enterprise Architect supports code generation from sequence models, the evaluator could not establish how to achieve this functionality; therefore this step was not performed by the evaluator. The evaluator located the menu option for code generation from sequence diagram; however, it was disabled in Enterprise Architect.

4.2 Evaluation of Altova UModel

The evaluation of Altova UModel is documented by the evaluator in the following Procedure Log.

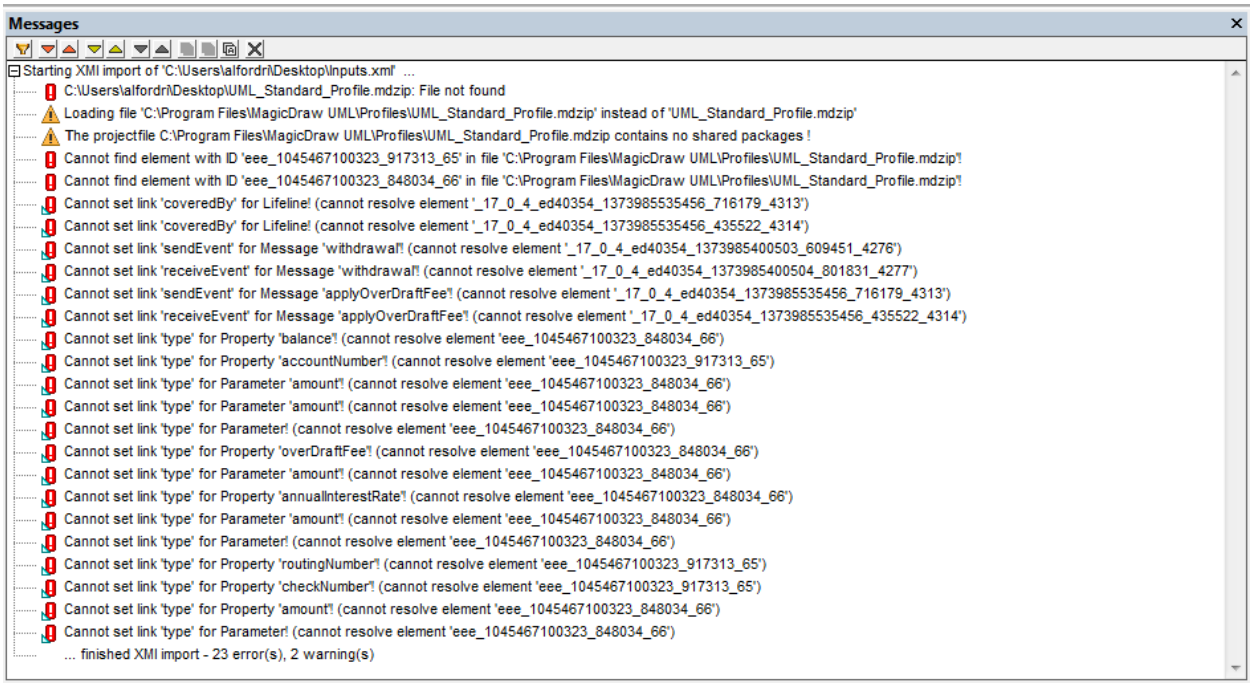
1. Setup version control settings in MDA Tool: UModel provides version control integration with Subversion only through select third party client plugins. PuskOk SVN SCC 1.5.1.3 subversion client was selected and installed by the evaluator to integrate Subversion with Altova UModel. Connecting to the Subversion repository was successful. Figure 24 illustrates the version control setup menu available from within UModel.

Figure 24: Altova UModel version control setup menu



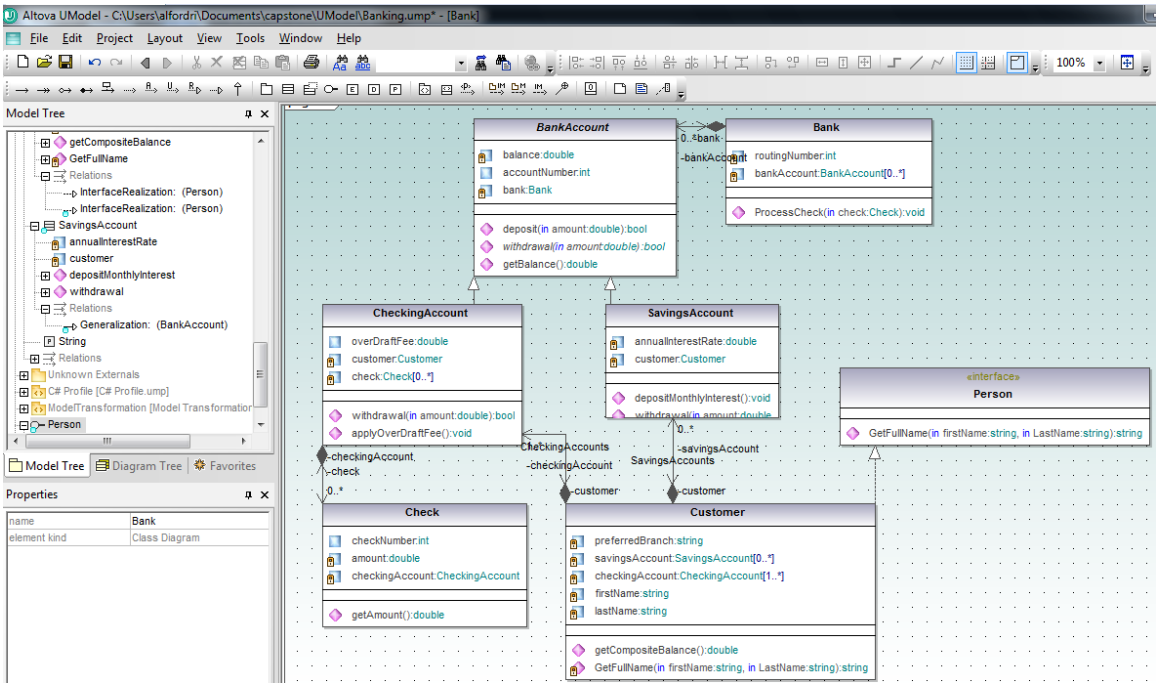
2. Import XMI of PIM of Banking from MagicDraw: Altova UModel provides a menu option for importing XMI into a new or existing project. After selecting the XMI file to import, UModel was able to determine that the XMI file was generated by MagicDraw and prompted the evaluator to include the default UML profile from MagicDraw. The status of the XMI import was documented by UModel in the Messages Window as illustrated in Figure 25. It reported 23 errors.

Figure 25: Altova UModel XMI Import messages



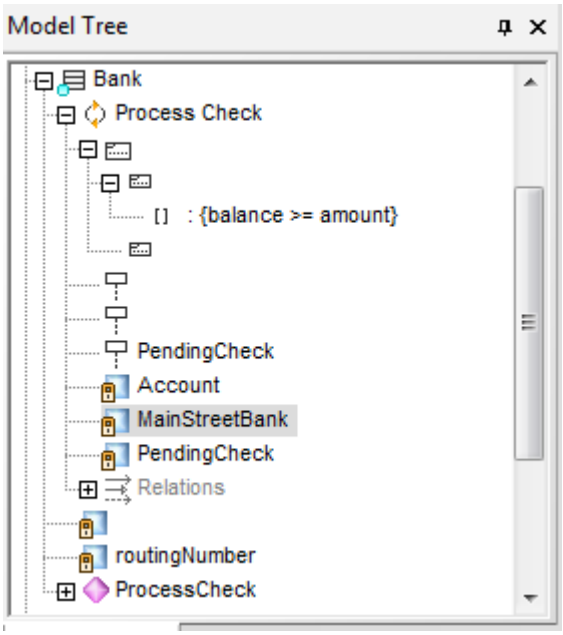
Upon reviewing the generated class diagram, the ‘type’ errors shown in Figure 25 are related to UModel failing to identify the property types for primitives. Figure 26 illustrates the class PIM generated by the XMI import.

Figure 26: Altova UModel class PIM from XMI



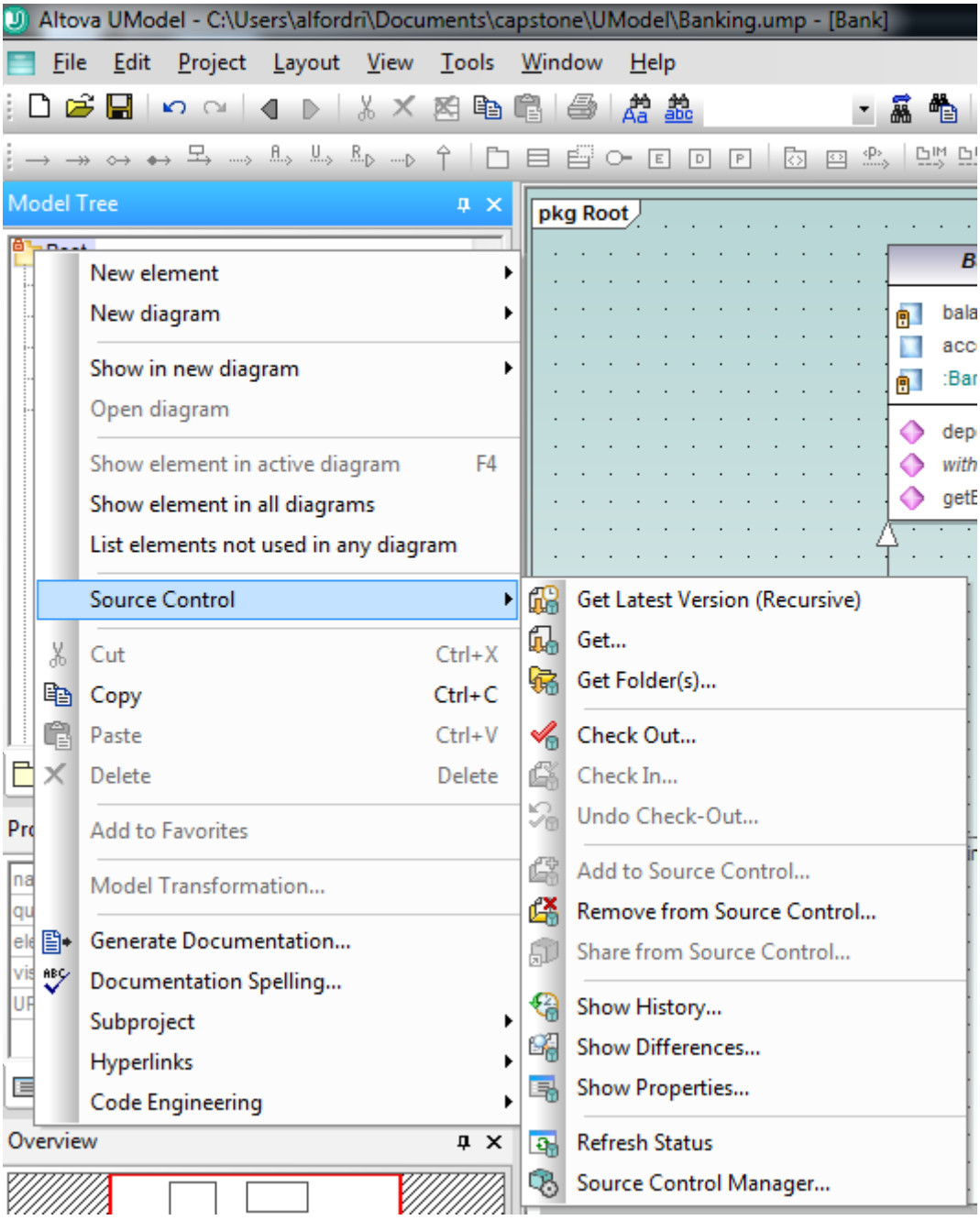
In order to gain access to primitive types in UModel, the user must apply a profile to the working namespace and then specify the type for the affected property. With the exception of the primitive type issue, the class diagram produced from the XMI import preserved all the object oriented aspects under evaluation. Notably, UModel also generated expected properties for the defined associations, creating arrays for one-to-many relationships. UModel did not produce a corresponding sequence diagram in the XMI import for the ProcessCheck() operation; however, there were stubs of the sequence diagram inputs located in the Model Tree pane as illustrated in Figure 27.

Figure 27: Altova UModel sequence diagram stubs in Model Tree



3. Commit to version control: Committing the generated models to version control within Altova UModel was successful. Version control commits with Altova UModel can be performed at the element level. Figure 28 illustrates the drop down menu options available for version control within Altova UModel.

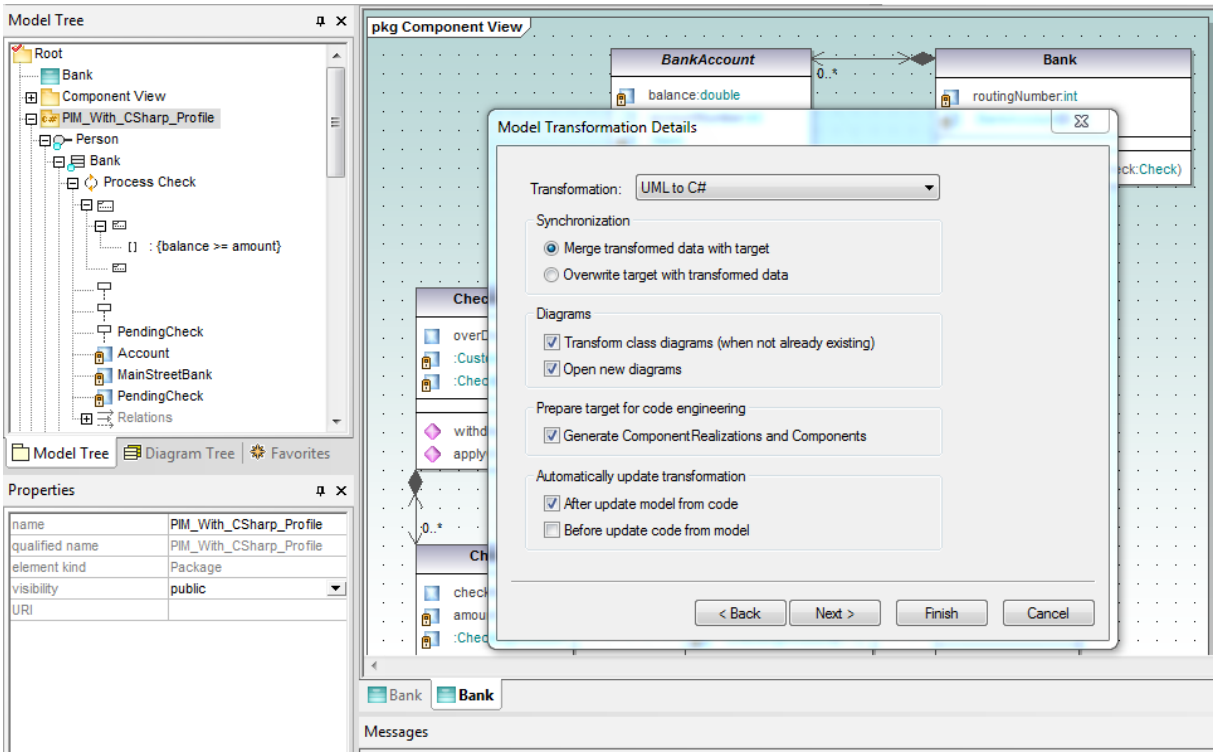
Figure 28: Altova UModel Version Control Check In



4. Retrieve PIMs from version control: Retrieving models from version control within Altova UModel was successful. Models are checked out within Altova UModel at the element level providing collaboration support for architect teams wishing to work with different model elements within the same project at the same time.

5. Generate C# PSM models from PIM of Banking: Due to the issue related to the primitive types discussed in step 2, the evaluator needed to make some manual modifications before generating the PSMs within Altova UModel. UModel requires that a package contains a targeted profile before model transformations can occur. The evaluator created a new package within the root namespace of the UModel project called PIM_With_CSharp_Profile, copied the PIM elements and models into this package, set C# as the package profile, and updated the primitive properties to match the corresponding C# profile primitive types. Next the evaluator selected the package PIM_With_CSharp_Profile and executed the Model Transformation command. The Model Transformation menu is illustrated in Figure 29.

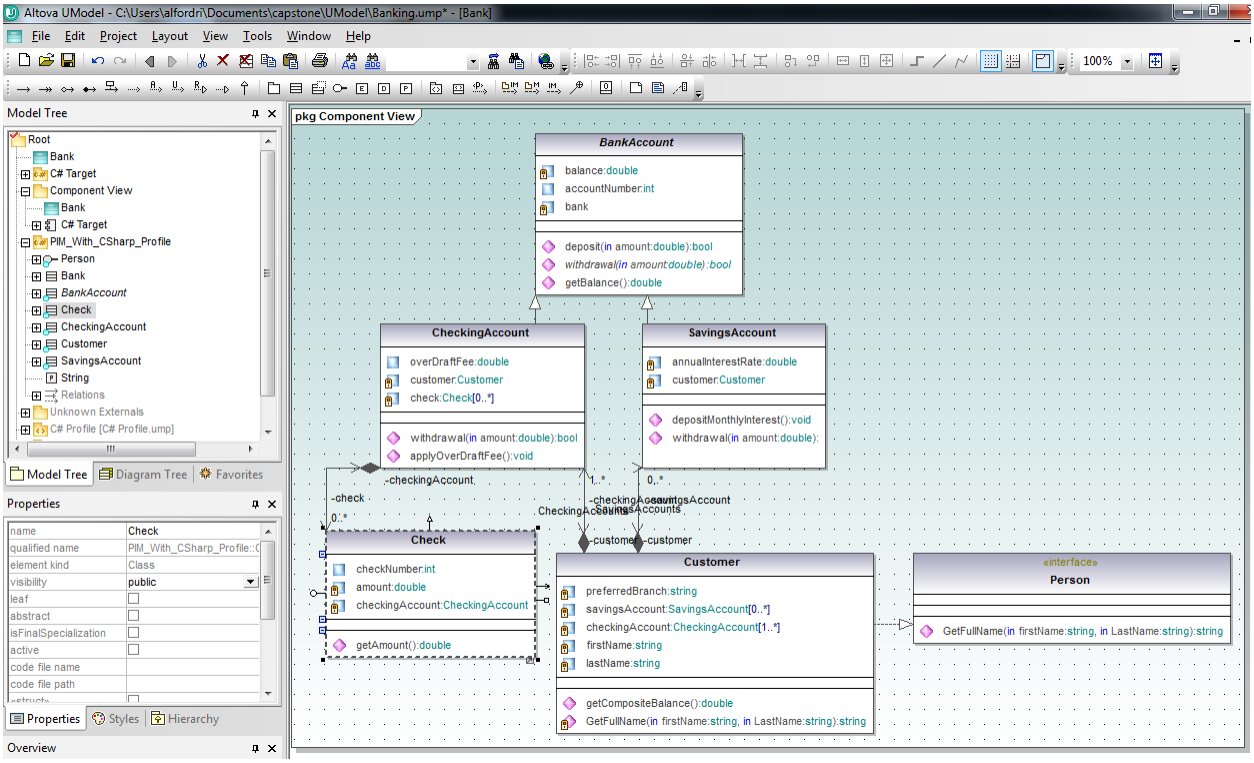
Figure 29: Altova UModel MDA Transform Screen 1



The results of the Model Transformation execution are illustrated in Figure 30. Based on review from the evaluator, the transformation from PIM to PSM targeting C# appears to be successful.

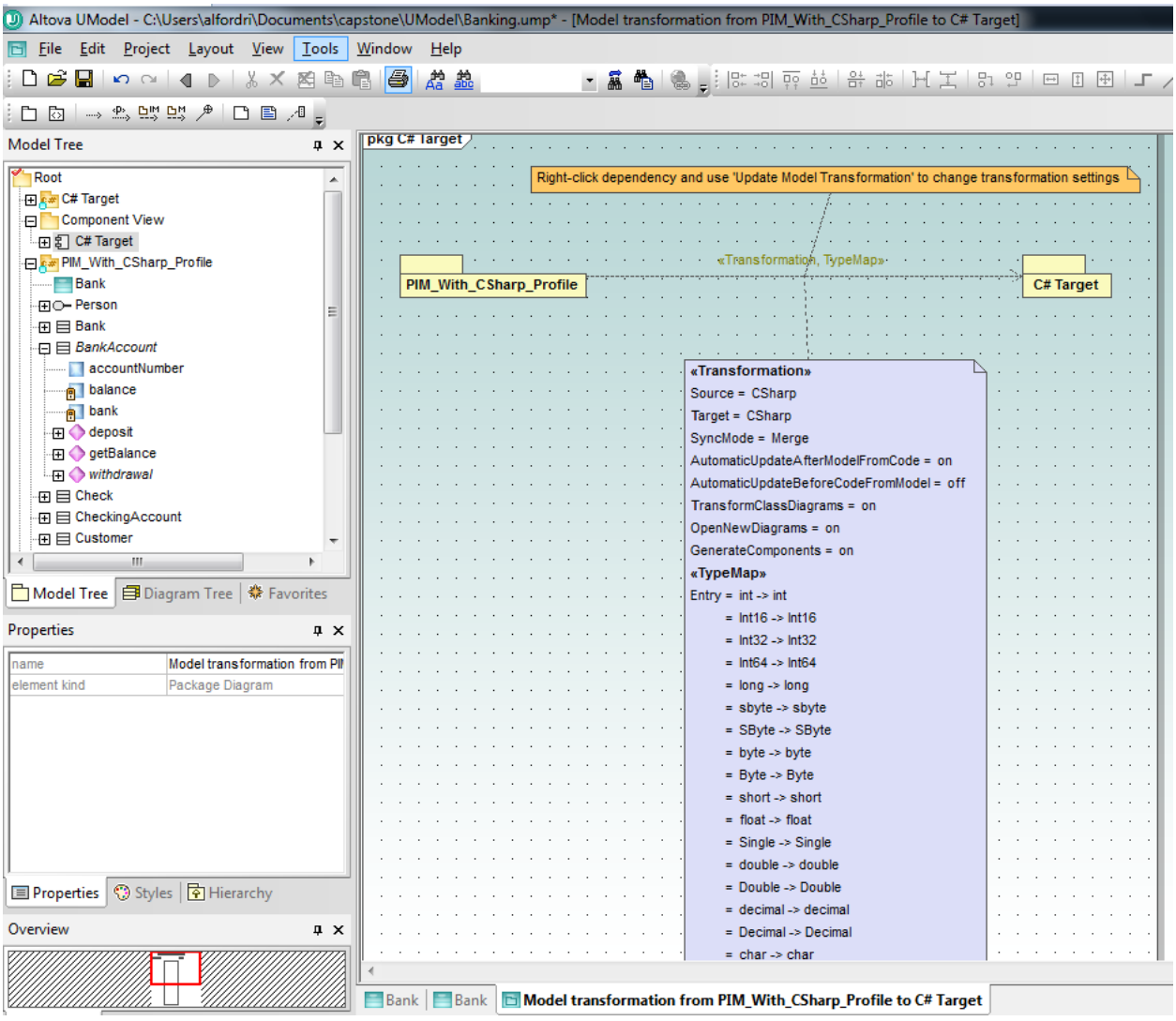
The sequence PIM describing the ProcessCheck() operation was not transformed during the Model Transformation process.

Figure 30: Altova UModel Class C# PSM from Model Transformation



UModel also created a component diagram illustrating the transformation settings and mappings applied as shown in Figure 31.

Figure 31: Altova UModel PSM to PIM Component Model Describing Transformation



6. Modify PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires adding a new class to the class PSM: Adding the Ledger class to the class PSM was straightforward and was accomplished by utilizing the class modeling tools within Altova UModel.

7. Sync the class PIM from the newly updated PSM: Altova UModel does not support syncing from PSMs to PIMs. It only provides forward transformations from PIMs to PSMs. It does

however state that it provides PIM syncing from PSM from Code. The evaluator could not determine how to test this feature.

8. Modify PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires modifying the sequence PSM named Process Check. Add Ledger as a participant to the Process Check diagram: Altova UModel did not produce the required sequence PSM; therefore, this procedure step is not required.

9. Sync the sequence PIM from the newly updated PSM: Altova UModel does not support syncing from PSMs to PIMs. It only provides forward transformations from PIMs to PSMs. It does however provide PIM syncing from PSM from Code. This will be tested in a later step.

10. Generate C# source code from PSMs: Altova UModel provides a menu option to produce source code from PSM packages illustrated in Figure 32 and Figure 33.

Figure 32: Altova UModel PSM to Code Targeting C#

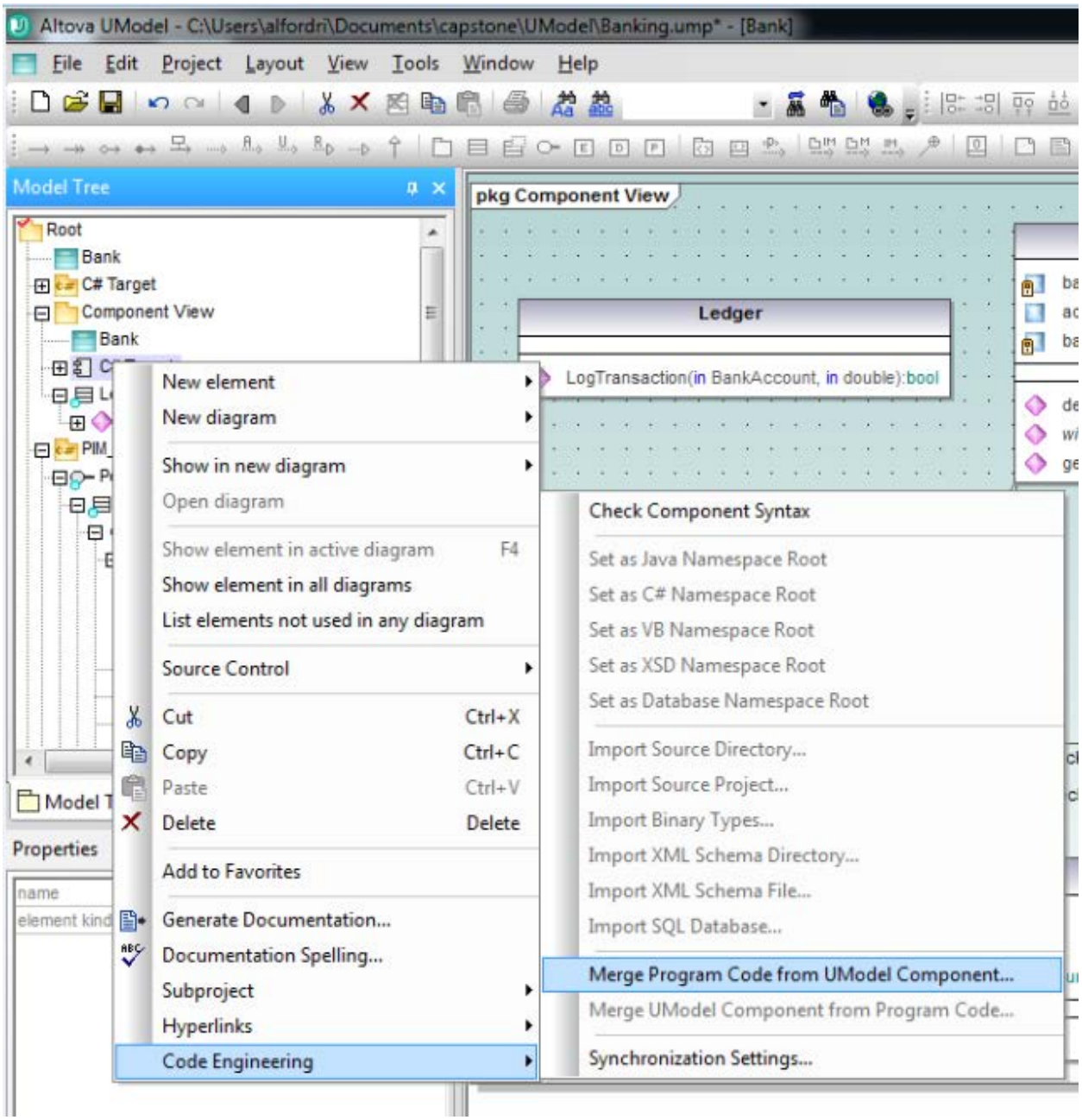
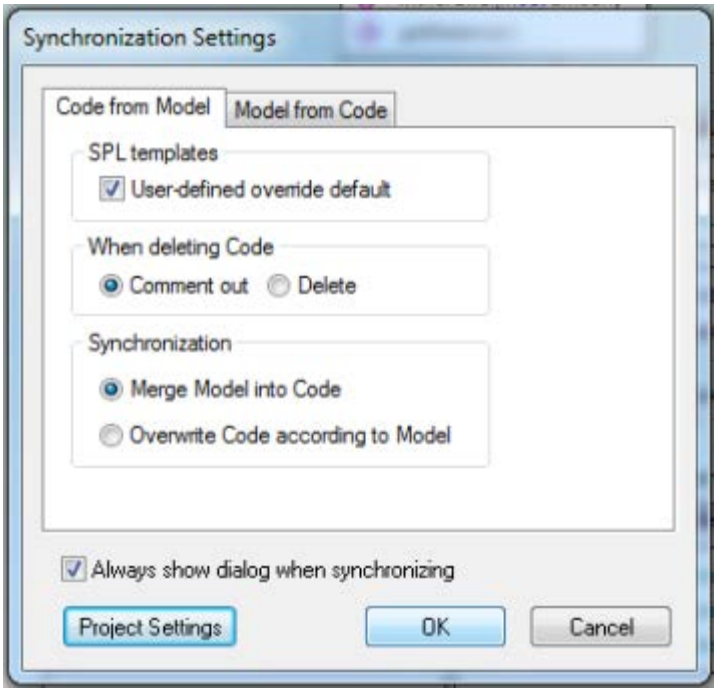


Figure 33: Altova UModel PSM to Code Targeting C# Synchronization Settings



11. Open C# source code project in Visual Studio or through integrated bridge provided by tool under evaluation. Create a new C# class BrokerageAccount inheriting from abstract class BankAccount. Add attributes and methods to the BrokerageAccount: Altova UModel provides an integration bridge to Visual Studio available as an additional download. The bridge is free and it is compatible with Visual Studio 2005, 2008, and 2010. Once installed, the generated source code from step 10 is merged into a new Visual Studio Project manually by the evaluator by creating a Visual Studio Solution and Project in the directories associated with UModel code generation project settings.

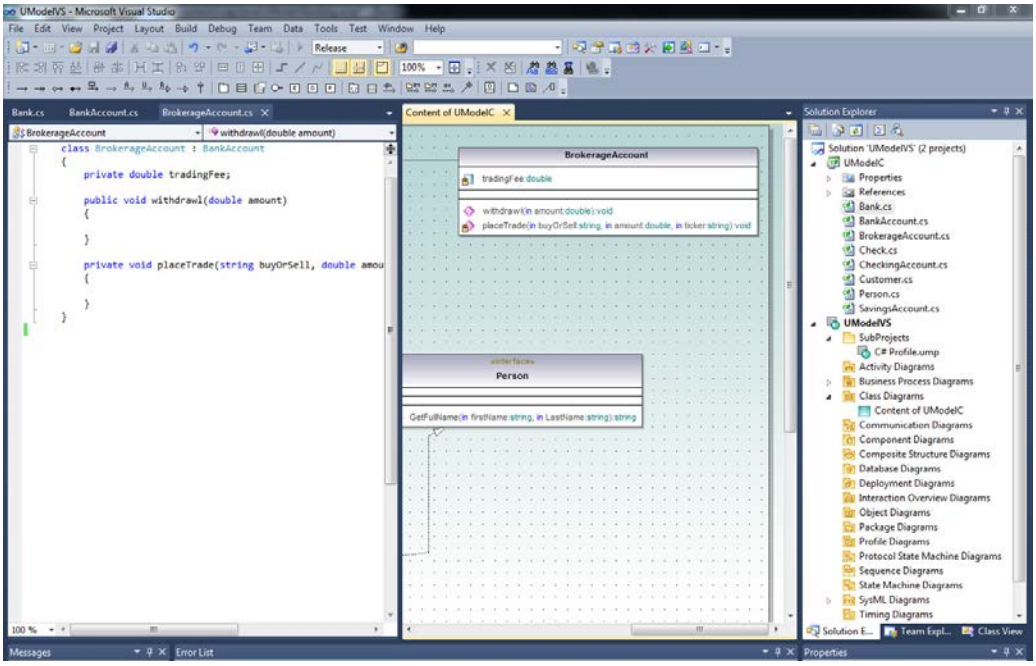
The evaluator inspected the generated code from within Visual Studio. All classes and interfaces were created during the code generation process resulting from step 10. UModel provides some clever comments within the generated source code. For example, operations defined in the PSM with return types are augmented with TODO comments as illustrated in Figure 34.

Figure 34: Altova UModel Code Generation Smart Comments

```
public class CheckingAccount : BankAccount
{
    public double overDraftFee;
    private Customer customer;
    private Check[] check;
    public bool withdrawal(double amount)
    {
        // TODO add implementation and return statement
    }
    public void applyOverDraftFee()
    {
        // TODO add implementation
    }
}
```

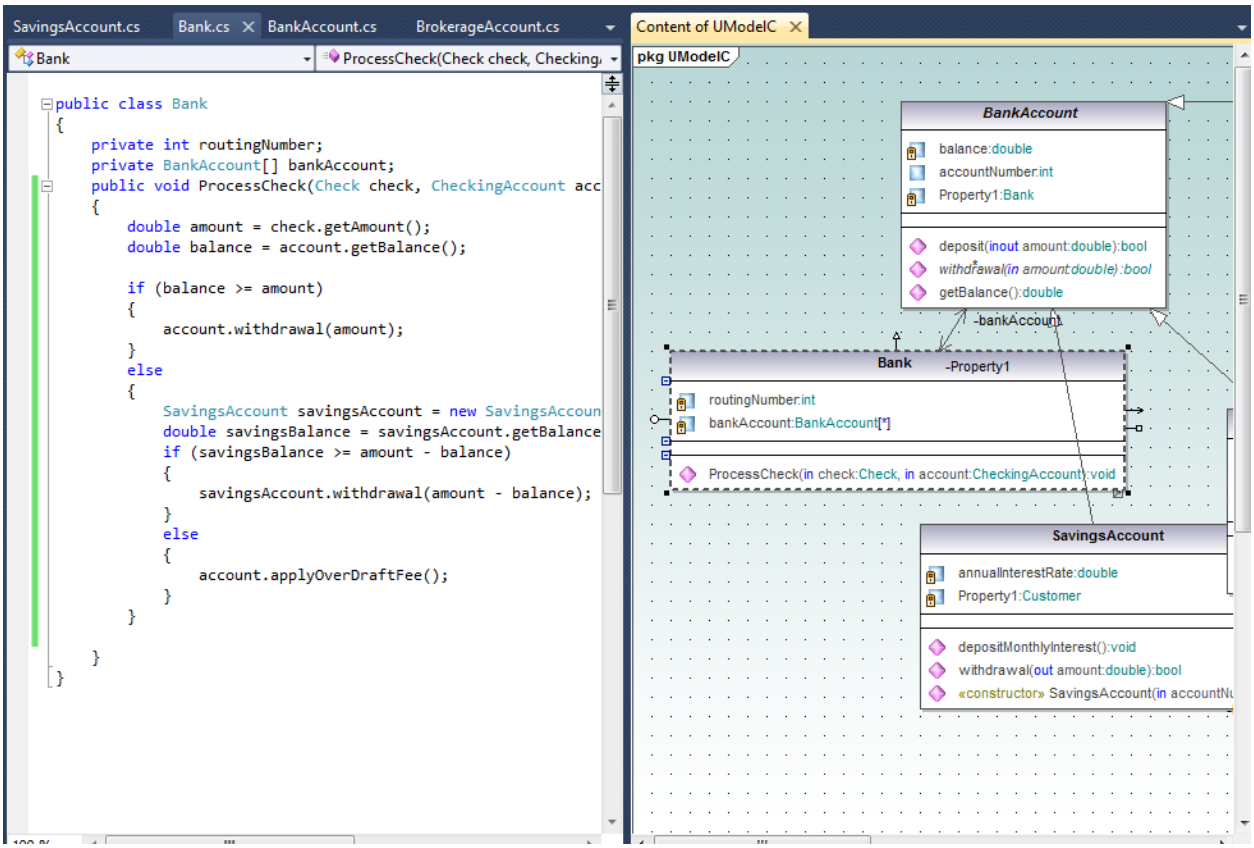
12. Sync the class PSM from the modified source code: The evaluator was able to fully implement the UModel bridge with Visual Studio. UModel provides real time syncing between class PSM and code from within the Visual Studio IDE, confirmed by creating the BrokerageAccount Class as source code. Figure 35 illustrates UModel's real-time syncing of model and code.

Figure 35: Altova UModel Visual Studio Bridge with Real-Time Sync Code to Model



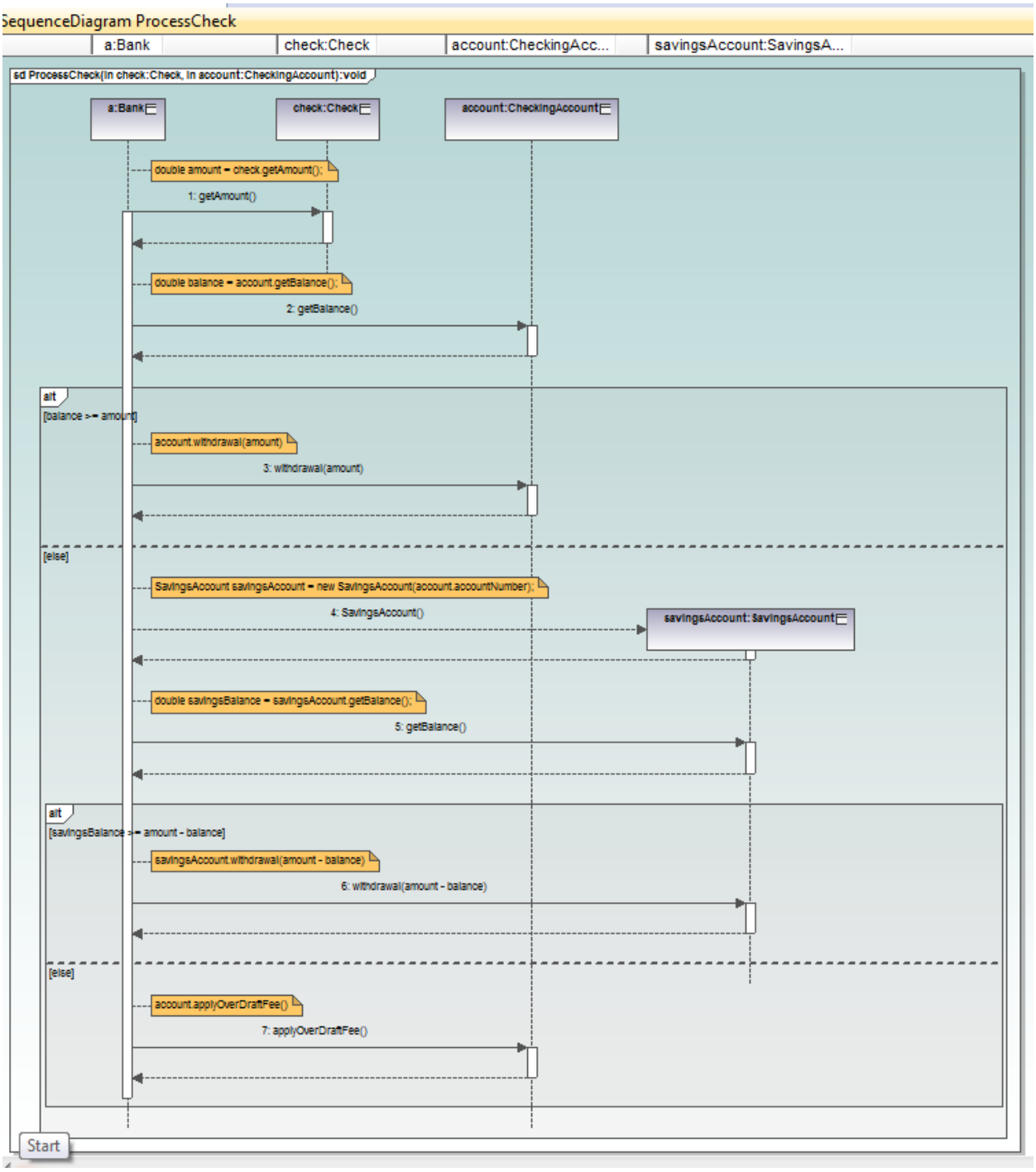
13. Open C# source code project in Visual Studio or through integrated bridge provided by tool under evaluation. Modify Bank method ProcessCheck(), if balance < amount, then attempt to recover the difference from account holder's savings account before applying overdraft fees: The evaluator made the modifications to the Bank class's ProcessCheck() method illustrated in Figure 36.

Figure 36: Altova UModel ProcessCheck Modification



14. Sync the sequence PSM from the modified source code: UModel provides operations from within the Visual Studio environment to create sequence diagrams from code and vice-versa. The evaluator selected the ProcessCheck() operation in the class PSM model and executed the menu option Generate Sequence Diagram from code. Figure 37 illustrates the sequence diagram produced by UModel.

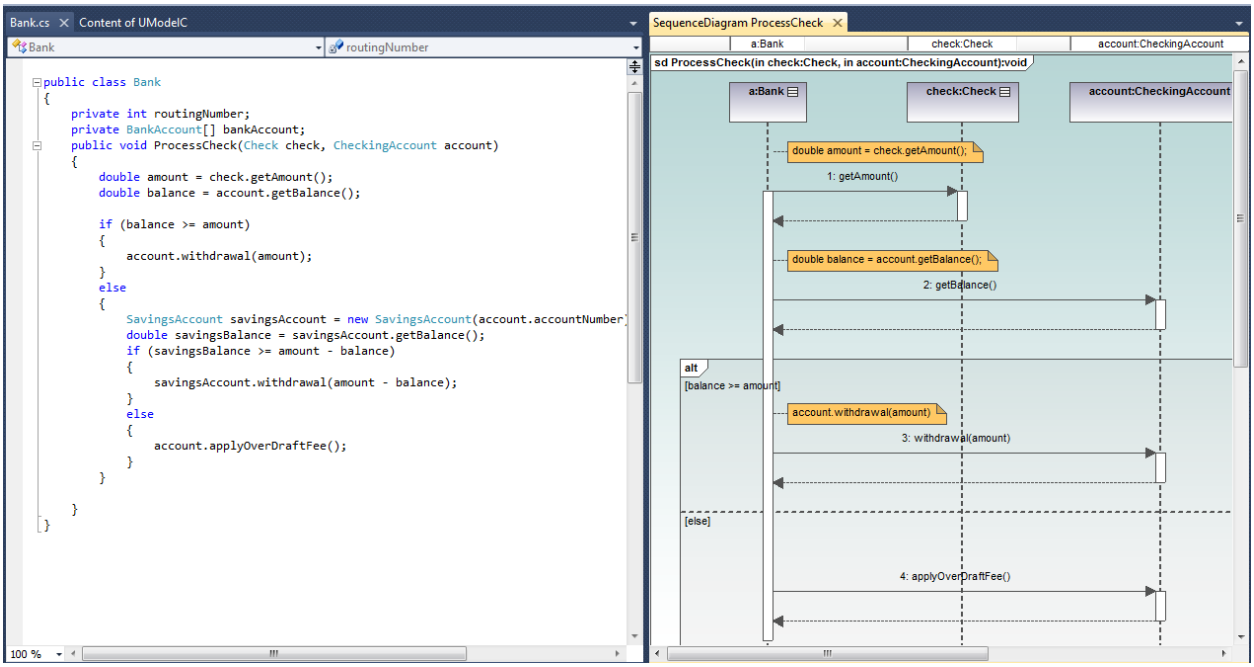
Figure 37: Altova UModel Sequence PSM from Source Code



With the success of generating the sequence diagram from code, next the evaluator modified the sequence diagram, removing the nested alt block to verify that the code synced with the model

modification. As illustrated in Figure 38, UModel successfully synced the C# code with the modified sequence diagram.

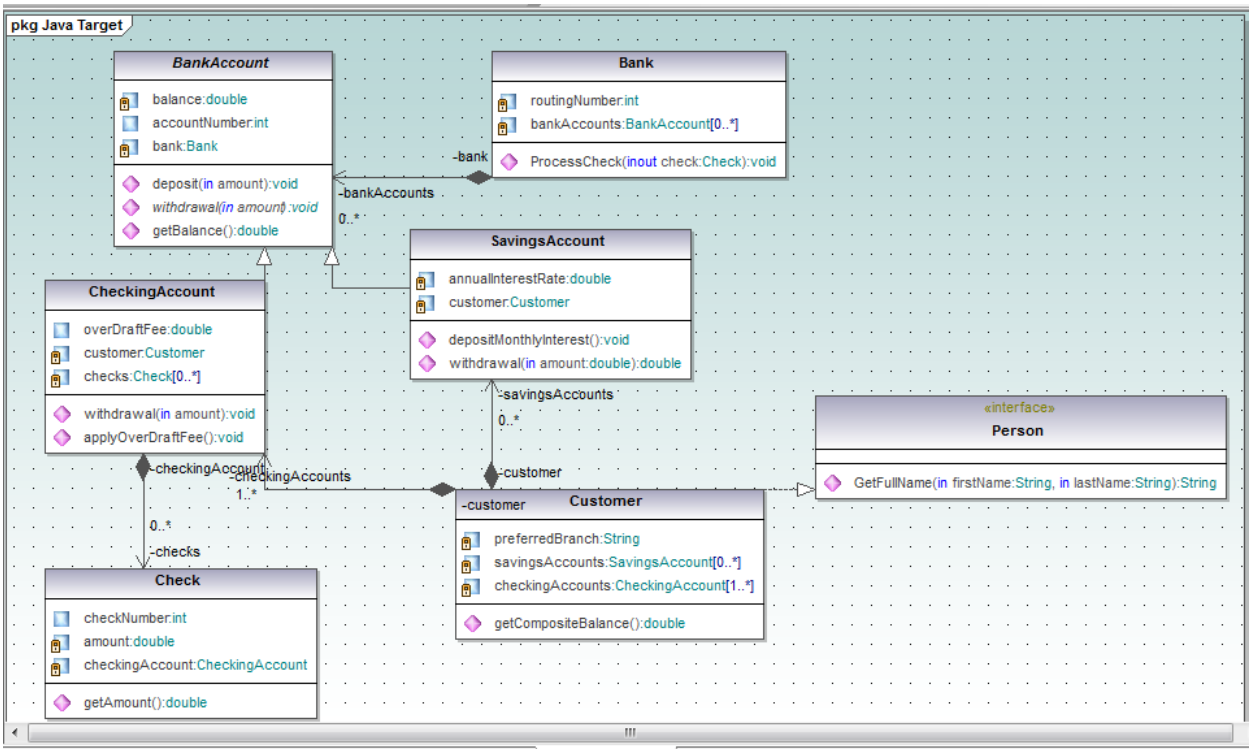
Figure 38: Altova UModel Sync Code from Sequence PSM modification



15. Export the models to XMI and attempt to import into MagicDraw: Altova UModel provides a menu option to export model elements as XMI. MagicDraw successfully imported the class diagram, sequence diagram, and related elements from the XMI export from Altova UModel.

16. Generate Java PSM models from PIM of Banking: The evaluator executed the same UModel screens presented in Figures 29 from step 5; this time targeting Java. The results of the Model Transformation execution are illustrated in Figure 39. Based on review from the evaluator, the transformation from PIM to PSM targeting Java appears to be successful. The sequence PIM describing the ProcessCheck() operation was not transformed during the Model Transformation process.

Figure 39: Altova UModel Class Java PSM from Model Transformation



17. Modify Java profile PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires adding a new class to the Java profile class PSM: Adding the Ledger class to the class PSM is straightforward and is accomplished by utilizing the class modeling tools.

18. Sync the Java Profile class PIM from the newly updated PSM: Altova UModel does not support syncing from PSMs to PIMs. It only provides forward transformations from PIMs to PSMs.

19. Modify Java profile PSM models to accommodate for a Ledger Class which logs withdrawals and deposits for a given BankAcount class. This step requires modifying the Java profile sequence PSM named Process Check. Add Ledger as a participant to the Process Check diagram: Altova UModel does not produce the required sequence PSM; therefore, this procedure step is not required.

20. Sync the Java profile sequence PIM from the newly updated Java profile PSM: Altova UModel does not support syncing from PSMs to PIMs. It only provides forward transformations from PIMs to PSMs.

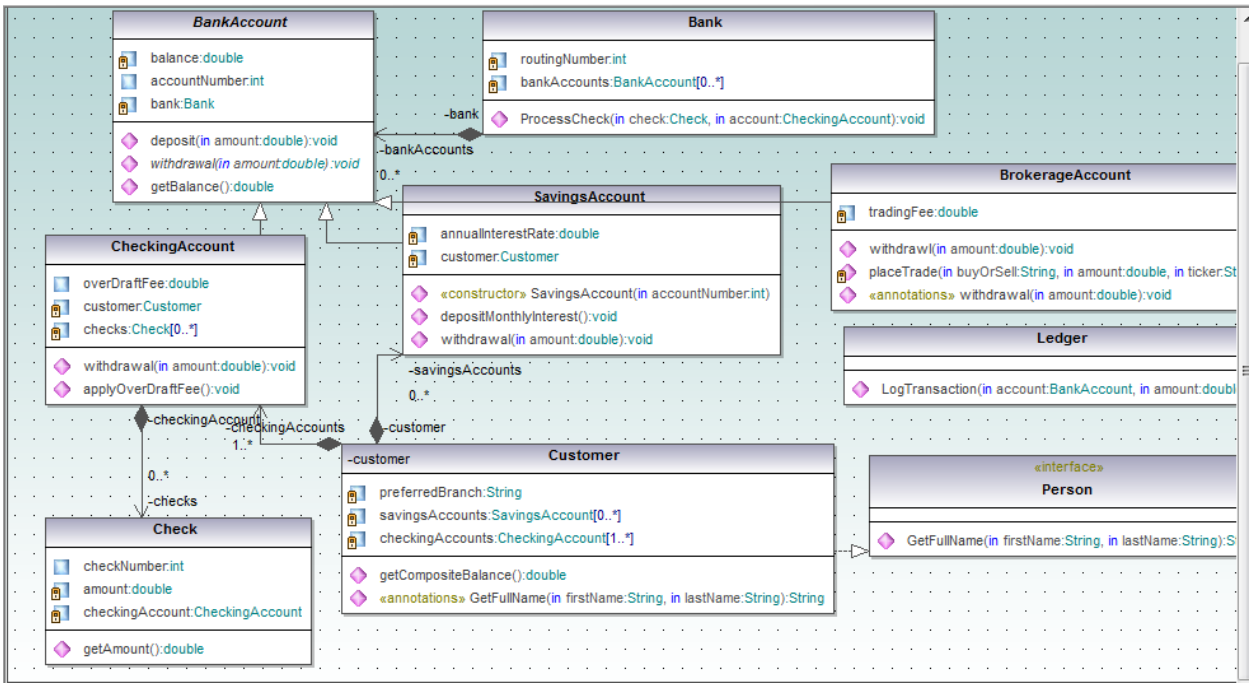
21. Generate Java source code from Java profile PSMs: Altova UModel provides a menu option to produce source code from PSM packages as described in step 10. The Java PSM is selected as the source model and Java is selected as the target source code. Code generation completed without error.

22. Open Java source code project Eclipse or through integrated bridge provided by tool under evaluation. Create a new Java class BrokerageAccount inheriting from abstract class BankAccount. Add attributes and methods to the BrokerageAccount: Altova UModel provides an integration bridge to Eclipse available as an additional download; it is free. Once installed, the generated source code from step 10 is merged into a new Eclipse Project manually by the evaluator by creating an Eclipse Workspace including a new Eclipse UModel Project. Unfortunately, the evaluator did not achieve the same success using the integration bridge for Eclipse as he did with Visual Studio; therefore, model to code integration evaluation for Java was conducted by toggling back and forth between UModel and Eclipse.

23. Sync the Java profile class PSM from the modified source code: Code to class PSM synchronization was successful in Altova UModel. As illustrated in Figure 40, the code to class PSM synchronization included the new Java class BrokerageAccount and other code based modifications.

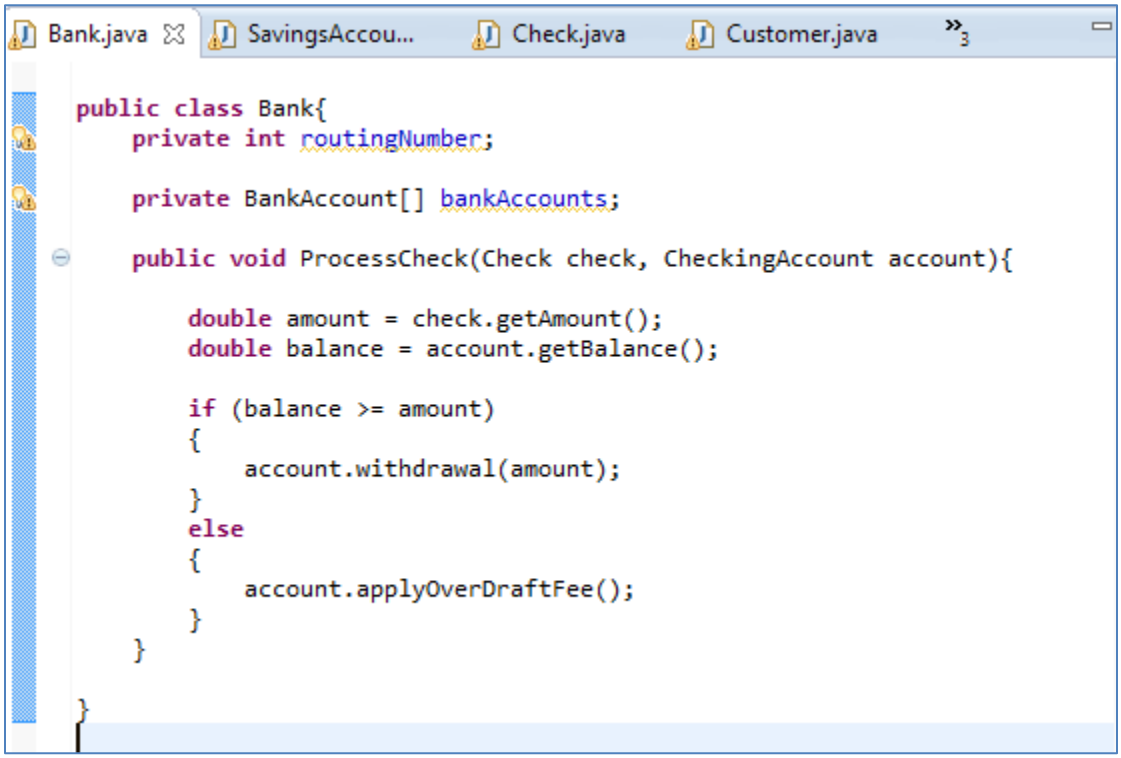
.

Figure 40: Altova UModel Synchronize Java Class PSM from Source Code



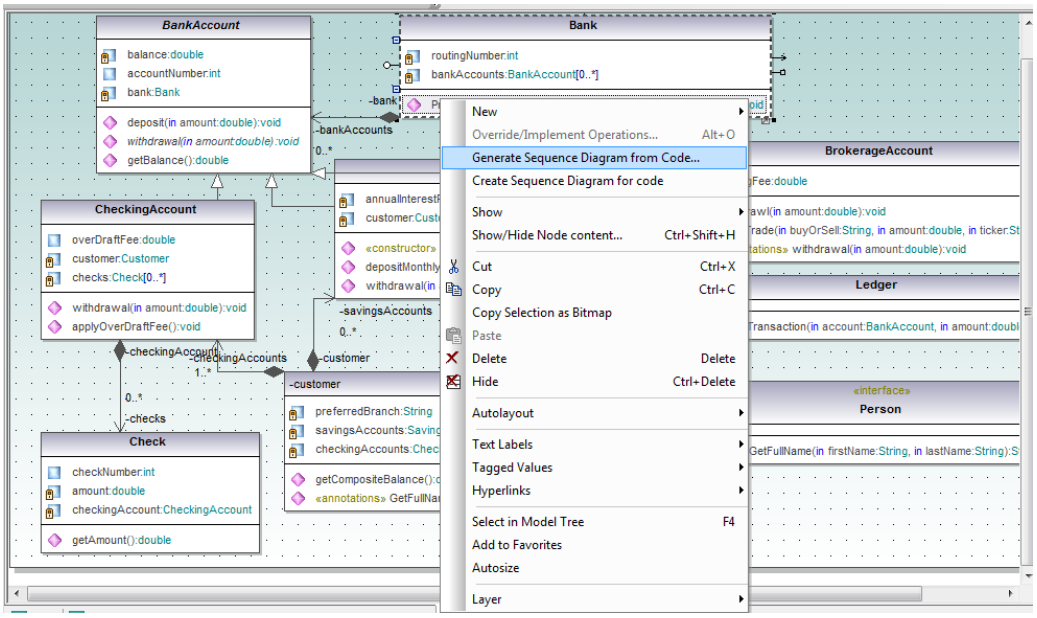
24. Open Java source code project in Eclipse or through integrated bridge provided by tool under evaluation. Modify Bank method ProcessCheck(), if balance < amount, then attempt to recover the difference from account holder's savings account before applying overdraft fees: Altova UModel provides an option to generate sequence diagrams for a given operation. Since the ProcessCheck sequence diagram was not produced by UModel in step 1, the evaluator added the appropriate code the Bank class operation ProcessCheck() as illustrated in Figure 41.

Figure 41: Altova UModel Java Bank.ProcessCheck() Code



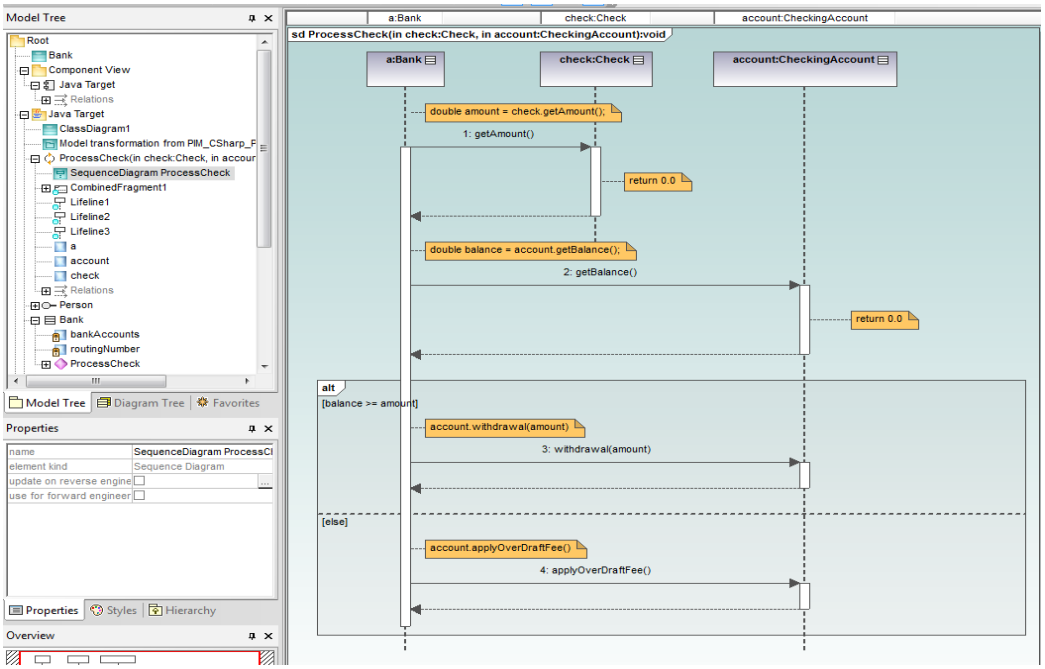
Next the evaluator selected the ProcessCheck() operation in the Java class PSM from within UModel and selected the menu option Generate Sequence Diagram from Code… illustrated in Figure 42.

Figure 42: Altova UModel Generate Sequence Diagram from Code



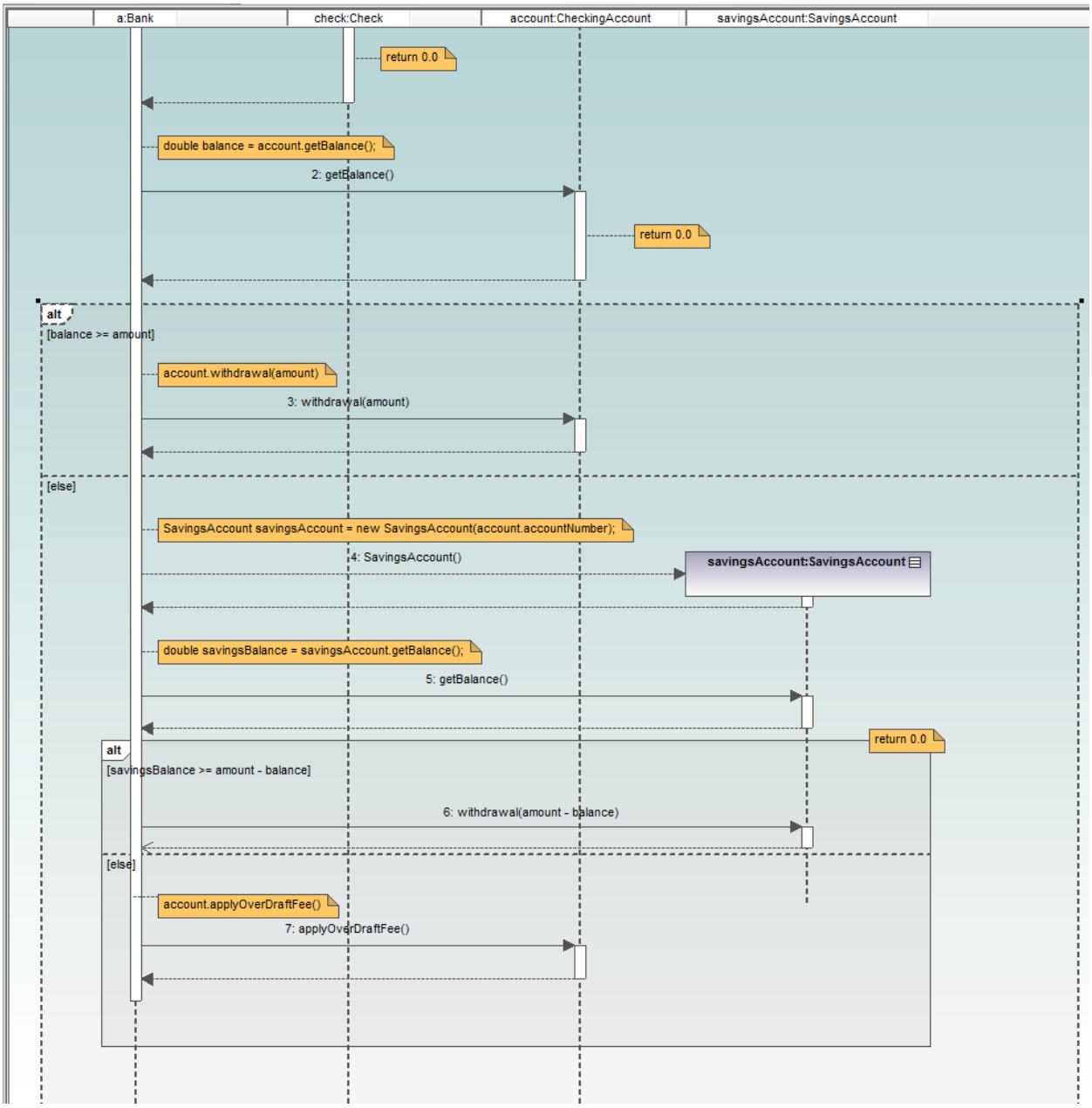
The sequence diagram corresponding to the ProcessCheck() operation was generated successfully. It is illustrated in Figure 43.

Figure 43: Altova UModel Sequence PSM from Code



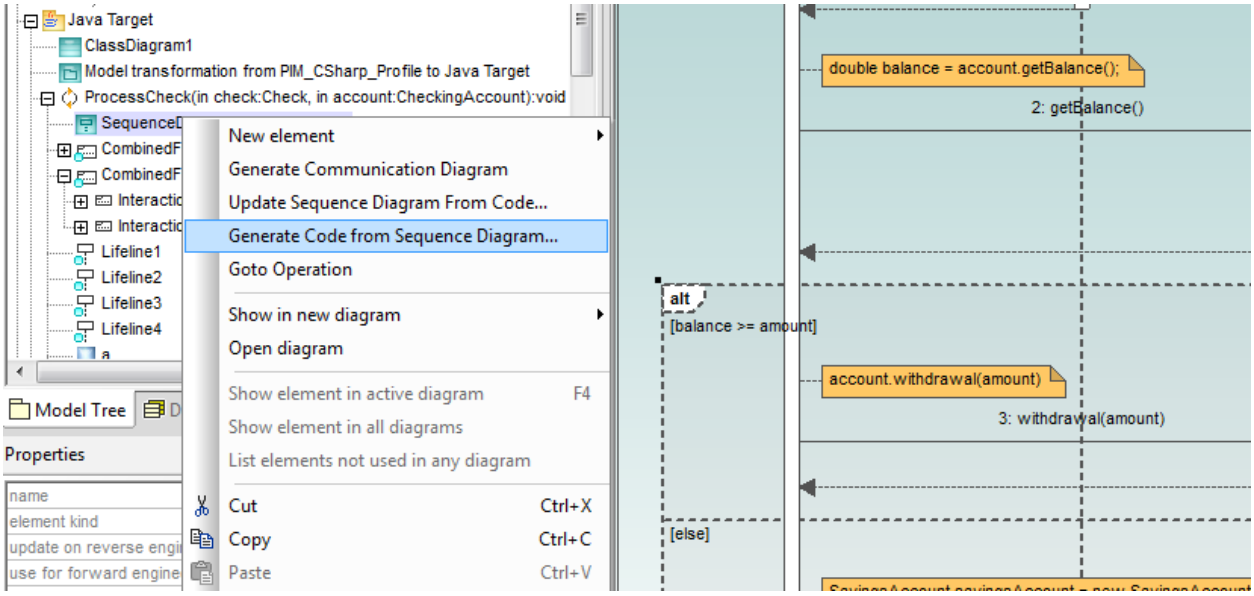
Next, the evaluator modified the generated sequence diagram by adding a nested alt fragment responsible for checking the customer's savings account for sufficient funds before applying the overdraft fee. This modification is illustrated in Figure 44.

Figure 44: Altova UModel PSM Sequence Diagram Modification



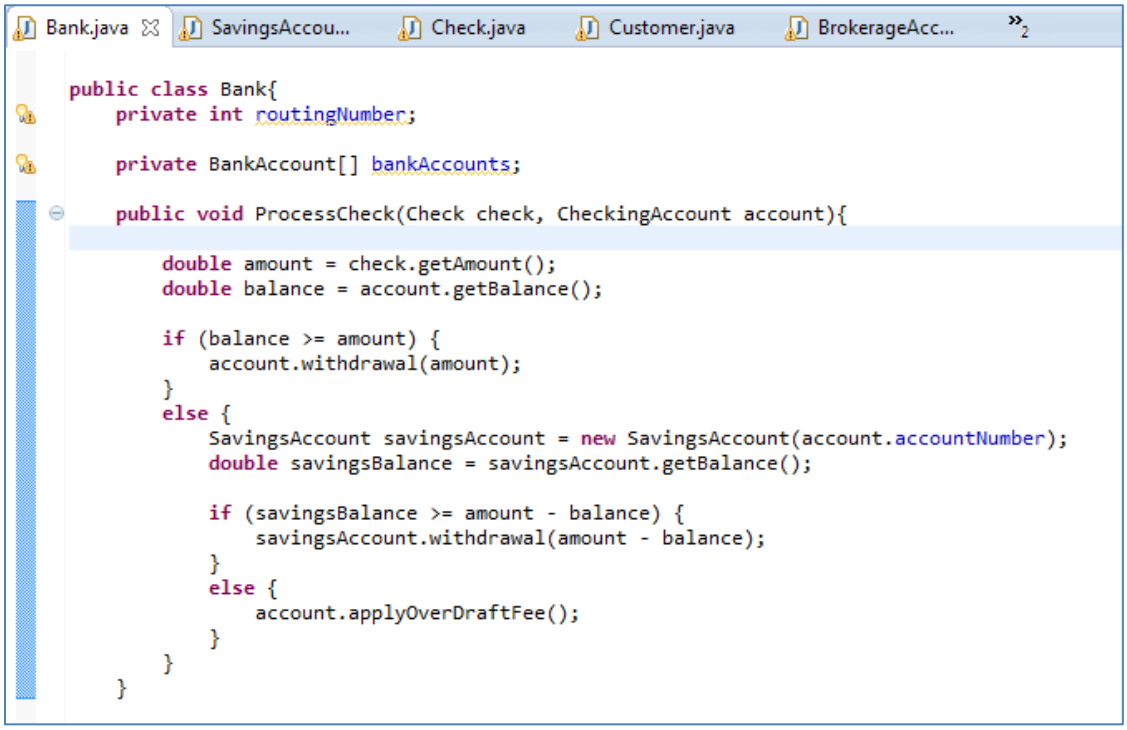
After saving the sequence diagram modification, the evaluator selected the Generate Code from Sequence Diagram… option illustrated in Figure 45.

Figure 45: Altova UModel Generate Code from Sequence Diagram



The evaluator reviewed the newly updated ProcessCheck() operation in the Bank class. As illustrated in Figure 46, UModel successfully updated the corresponding source code to match the sequence diagram modifications shown in Figure 44.

Figure 46: Altova UModel Sequence PSM Modification to Java Code

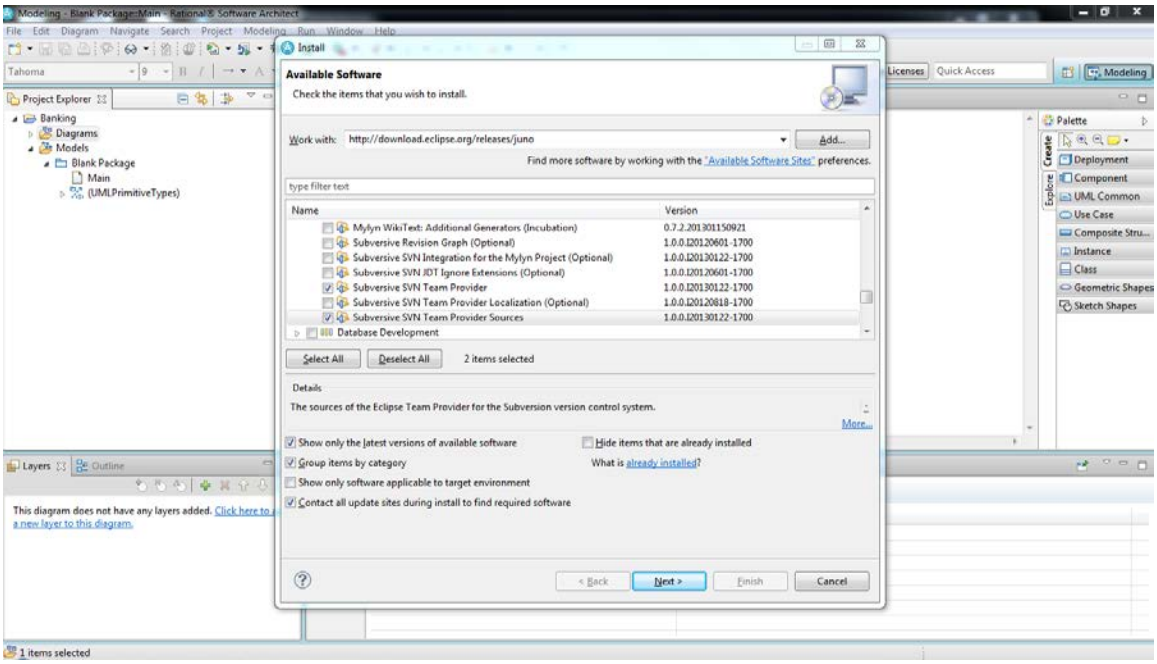


25. Sync the Java profile sequence PSM from the modified source code: As demonstrated in step 24, Altova UModel provides round-trip syncing between sequence PSMs and source code.

4.3 Evaluation of IBM Software Architect

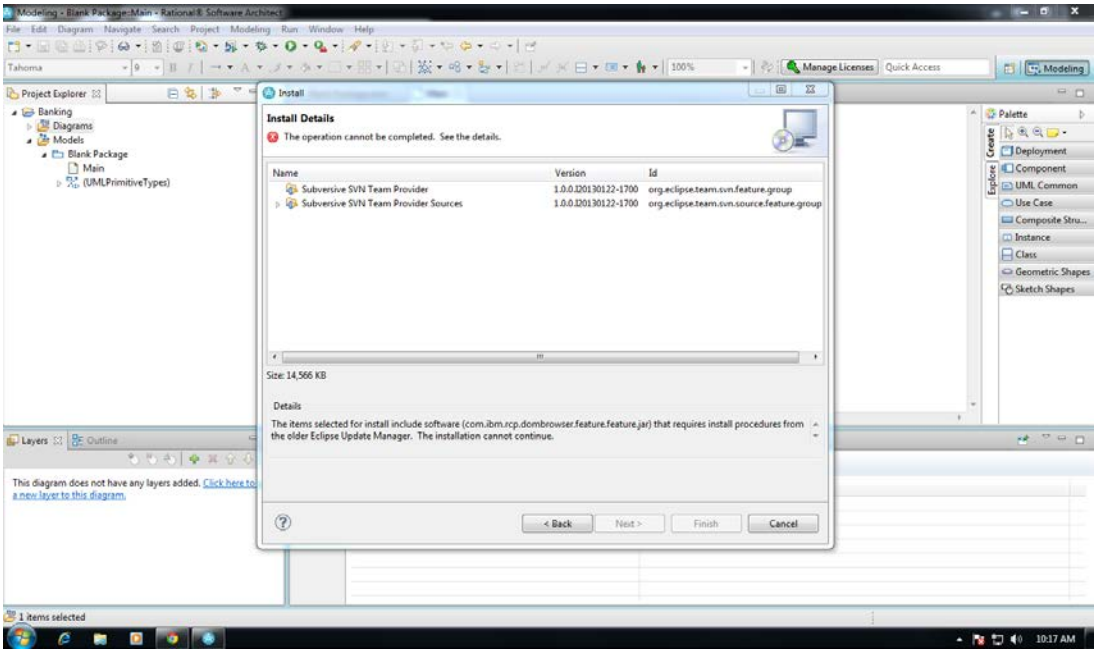
1. Setup version control settings in MDA Tool: IBM Software Architect integrates with Subversion through the Subversive software package available from the Eclipse project site. Figure 47 illustrates the menu selections for installing Subversive from within Software Architect.

Figure 47: IBM Software Architect Subversive Installation



The evaluator received the following error illustrated in Figure 48 upon installing Subversive.

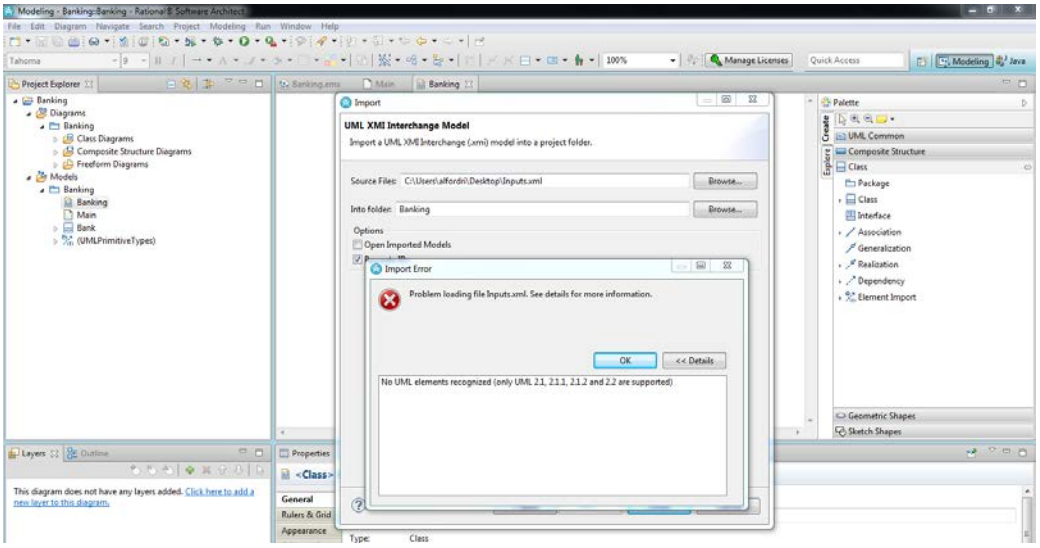
Figure 48: IBM Software Architect Subversive Install Error



The evaluator researched alternative options for installing Subversive; however, successful integration with Subversion through IBM Rational Architect was not achieved.

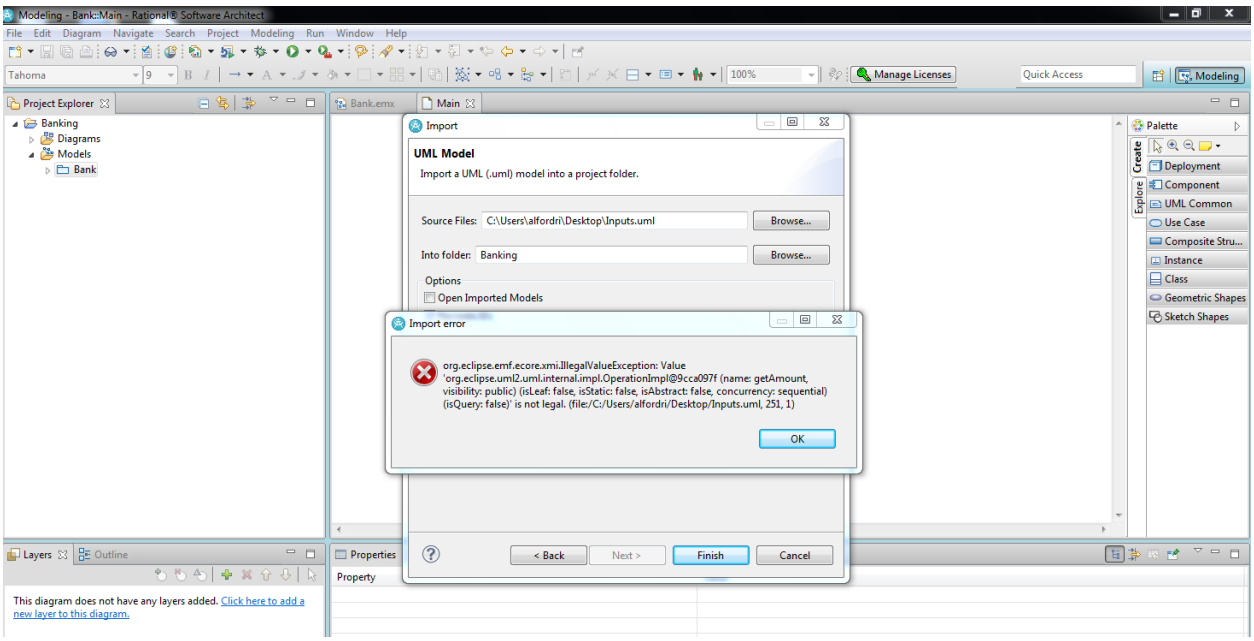
2. Import XMI of PIM of Banking from MagicDraw: IBM Software Architect provides XMI import capabilities. Software Architect does not support XMI 2.4 as illustrated in Figure 49.

Figure 49: IBM Software Architect XMI Import



To accommodate, the evaluator reverted back to MagicDraw and selected an Eclipse specific XMI export format named Eclipse UML2 XMI. Software Architect failed to import this XMI format as well. The error is illustrated in Figure 50.

Figure 50: IBM Software Architect XMI Import Attempt 2

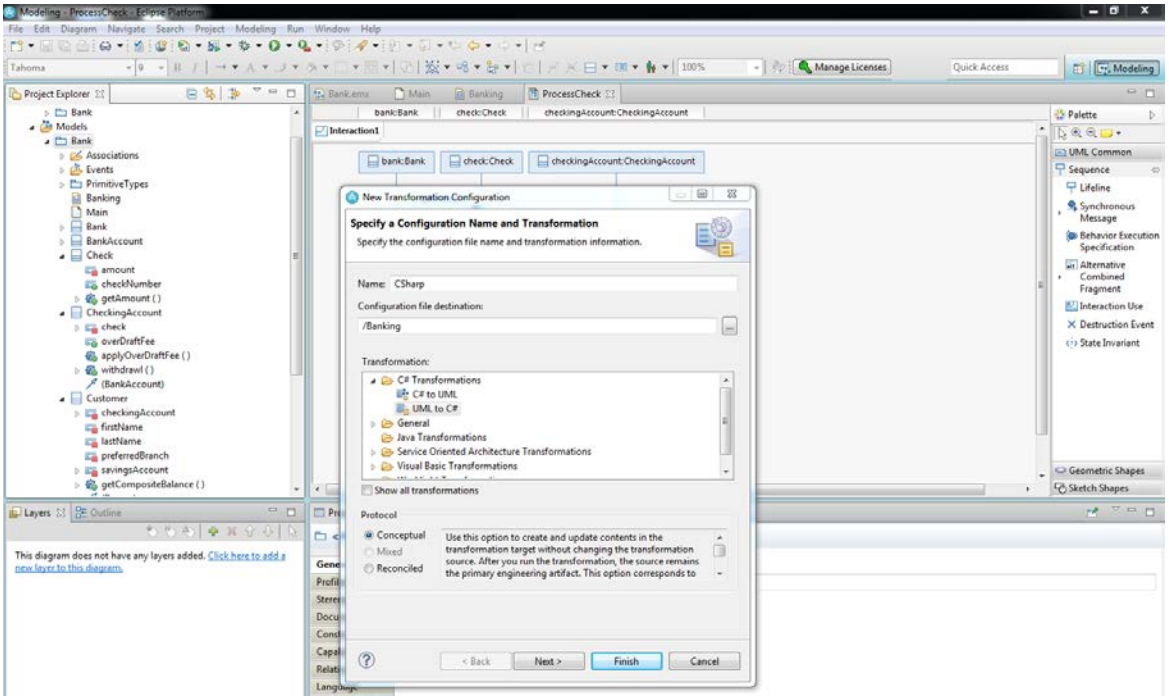


3. Commit to version control: Subversion integration was not achieved for this tool; therefore, this step is not required.

4. Retrieve PIMs from version control: Subversion integration was not achieved for this tool; therefore, this step is not required.

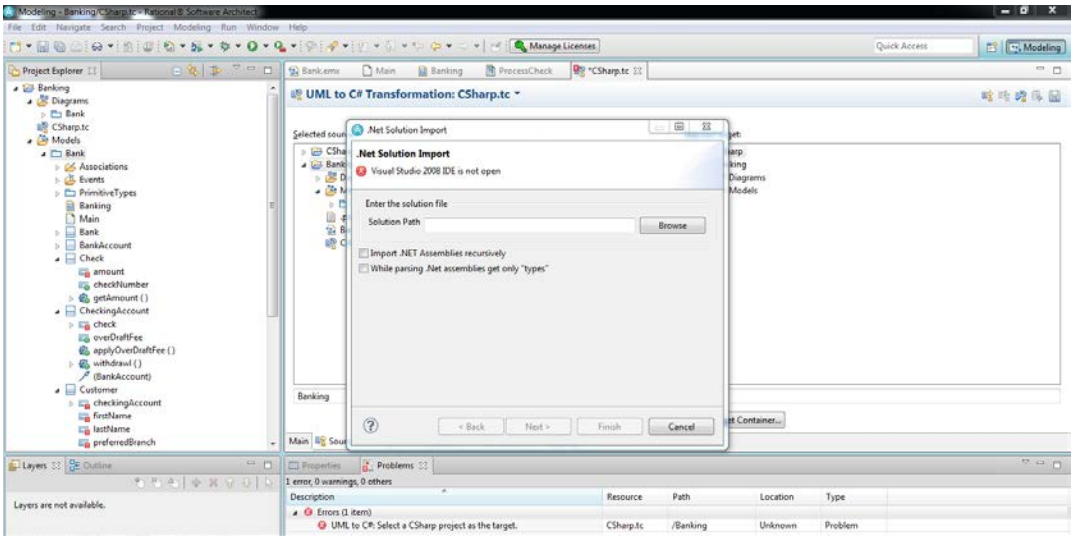
5. Generate C# PSM models from PIM of Banking: Since step 2 failed the evaluator composed the class and sequence diagrams within IBM Software Architect. The transform option was selected by right clicking on the Bank PIM located in the Project Explorer pane of Software Architect. Figure 51 illustrates the first of several transformation screens.

Figure 51: IBM Software Architect PIM to C# PSM Transformation



In order to generate the C# Target, Software Architect expects the user to have a Visual Studio 2008 solution file. The evaluation criteria specified in this study targets Visual Studio 2010. The evaluator attempted to map the transformation to a Visual Studio 2010 solution file; however, Software Architect halted with a project error shown in Figure 52.

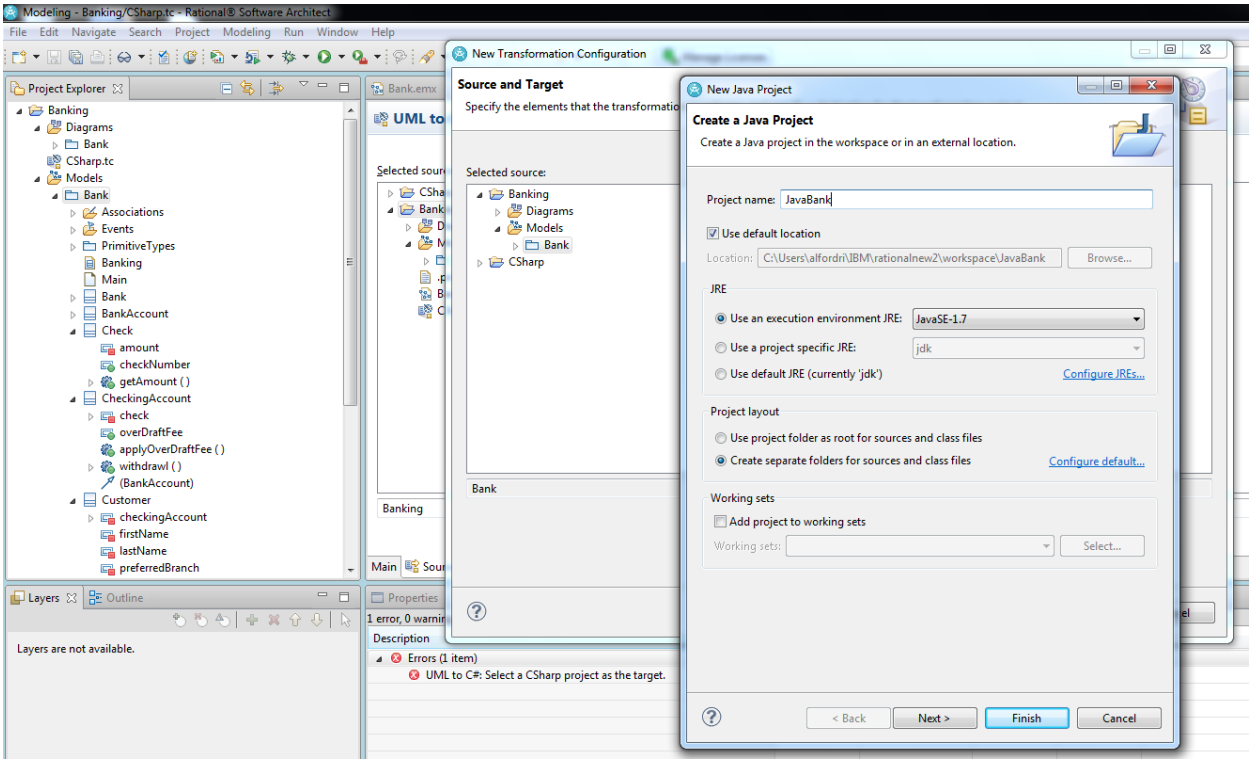
Figure 52: IBM Software Architect PIM to C# PSM Transform Error



Due to the inability to achieve a transformation targeting C# the evaluator moved on to step 16; the first step in the procedure log template targeting Java.

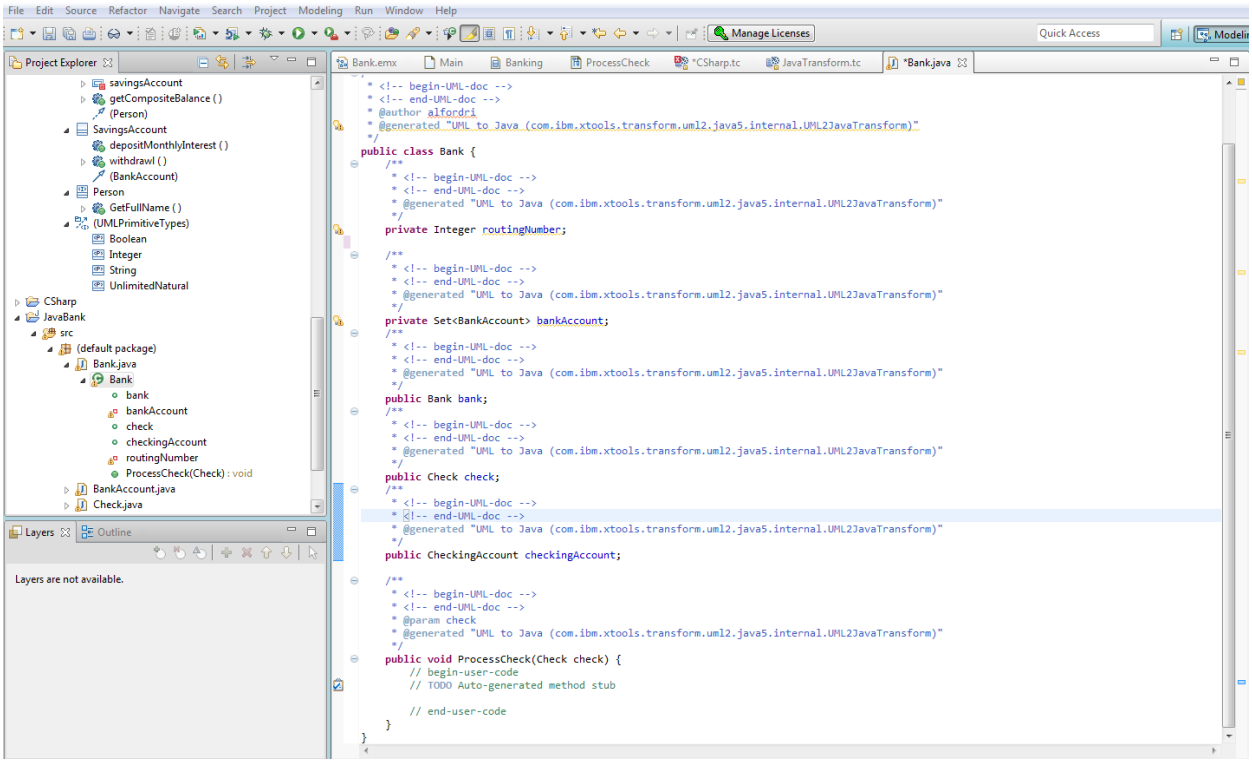
16. Generate Java PSM models from PIM of Banking: Setting up the PIM to Java PSM was achieved in the same manner for C# as illustrated in Figure 51. Next the evaluator is prompted to create a Java Project as the target location for the transformation as illustrated in Figure 53.

Figure 53: IBM Software Architect PIM to Java PSM



Software Architect produced source code artifacts based on the class PIM rather than transforming to PSM representation. The sequence diagram ProcessCheck was included in Software Architect's transformation; however, rather than populating the ProcessCheck() method with the expected behavioral code, the transformation merely added the Lifeline participants as class level variables in Bank.java. This is illustrated in Figure 54.

Figure 54: IBM Software Architect Generated Source Bank.java



The evaluator attempted to find a way to synchronize modified java source code back to the source models within Software Architect; however, this was not successful. It is possible that with further training with this tool could prove to be a viable MDA development option; however, the evaluator has spent more time working with this product than the other tool evaluations with minimal positive results.

4.4 Results from the Evaluations of Selected Tools

The results of the first metric, model transformation/code generation are compiled in Table 8 and 9 below.

Table 8: Results of Model/Transformation/Code Generation Capabilities

	Enterprise Architect	IBM Software Architect	Altova UModel
Structural PIM to/from PSM Transforms targeting C# Profile	Forward only transformations from PIM to PSM. Class diagram	Could not determine where/how to achieve this. The tool does provide PIM	Forward only transformations from PIM to PSM. Class diagram

	transformations were reasonably successful with the majority of elements preserved. This feature may prove valuable for one time migration of PIMs to PSMs. Associations were not reflected properly for 0-* relationships	to Source code transformations.	transformations were successful with the all elements preserved. This feature may prove valuable for one time migration of PIMs to PSMs. Associations were properly reflected for 0-* relationships.
Structural PIM to/from PSM Transforms targeting JAVA Profile	Forward only transformations from PIM to PSM. Class diagram transformations were reasonably successful with the majority of elements preserved. This feature may prove valuable for one time migration of PIMs to PSMs.	Could not determine where/how to achieve this. The tool does provide PIM to Source code transformations.	Forward only transformations from PIM to PSM. Class diagram transformations were successful with the all elements preserved. This feature may prove valuable for one time migration of PIMs to PSMs. Associations were properly reflected for 0-* relationships.
Structural C# Profile based PSM to/from Code Transforms targeting C# code	Structural PSM to Code and Code to Structural PSM synchronization was reasonably successful out of the box. With ample time and planning this tool could provide powerful model to code and code to model capabilities in the enterprise.	Not Applicable, Required an older version of Visual Studio	Structural PSM to Code and Code to PSM synchronization was extremely successful. Utilizing the Visual Studio bridge enabled real-time syncing between model and code for structural elements.
Structural Java Profile based PSM to/from Code Transforms targeting JAVA code	Structural PSM to Code and Code to Structural PSM synchronization was reasonably successful out of the box. With ample time and planning this tool could provide powerful model to code and code to model capabilities in the enterprise.	The evaluator could only achieve model to code transformations.	Structural PSM to Code and Code to Structural PSM synchronization was reasonably successful out of the box. With ample time and planning this tool could provide powerful model to code and code to model capabilities in the enterprise.
Behavioral PIM to/from PSM Transforms targeting C# Profile	Sequence diagrams were not included in the MDA transformation tool provided.	The evaluator could only achieve model to code transformations.	Sequence diagrams were not included in the MDA transformation tool provided.
Behavioral PIM to/from PSM Transforms targeting	Sequence diagrams were not included in the MDA	Could not determine where/how to achieve this.	Sequence diagrams were not included in the MDA

JAVA Profile	transformation tool provided.	The tool does provide PIM to Source code transformations.	transformation tool provided.
Behavioral C# Profile based PSM to/from Code Transforms targeting C# code	Enterprise Architect supports sequence, activity and interaction diagram transformations to /from code. The evaluator could not determine how to activate this feature for sequence diagrams.	Not Applicable, Required an older version of Visual Studio	Transformation between behavioral PSMs to code and code to behavior PSMs was extremely successful. Utilizing the Visual Studio bridge enabled real-time syncing between model and code for behavior elements.
Behavioral Java Profile based PSM to/from Code Transforms targeting JAVA code	Enterprise Architect supports sequence, activity and interaction diagram transformations to /from code. The evaluator could not determine how to activate this feature for sequence diagrams.	Model to Code Transformations in the context of sequence diagrams were minimum, only adding the Lifeline participants as class level variables leaving the actual method content empty.	Transformation between behavioral PSMs to code and code to behavior PSMs was extremely successful. Although the evaluator could not get the Eclipse Bridge to function as expected, syncing between behavior PSMs and code worked very well by toggling between UModel and Eclipse.

Table 9: Model Transformation/Code Generation Capabilities of MDA Tools – Range 0-2 (0 Lowest to 2 Highest)

	Enterprise Architect	IBM Software Architect	Altova UModel
Structural PIM to/from PSM Transforms targeting C# Profile	1	0	1
Structural PIM to/from PSM Transforms targeting JAVA Profile	1	0	1
Structural C# Profile based PSM to/from Code Transforms targeting C# code	2	0	2
Structural Java Profile based PSM to/from Code Transforms targeting JAVA code	2	1	2
Behavioral PIM to/from PSM Transforms targeting C# Profile	0	0	0
Behavioral PIM to/from PSM Transforms targeting	0	0	0

JAVA Profile			
Behavioral C# Profile based PSM to/from Code Transforms targeting C# code	1	0	2
Behavioral Java Profile based PSM to/from Code Transforms targeting JAVA code	1	0	2

Integration is the next metric result set. The results of the tool evaluations for this metric are detailed in Tables 10 and 11.

Table 10: Results of Integration Capabilities of MDA Tools

	Enterprise Architect	IBM Software Architect	Altova UModel
Eclipse Integration	This tool provides a plugin at an additional charge to integrate with the Eclipse IDE. The plugin functioned well allowing for rapid sync and toggling between code and model. It is well worth the extra $89-$145.	Deep integration with Eclipse as the tool itself is built into the Eclipse Environment.	This tool provides a plugin free of charge which integrates with the Eclipse IDE. The evaluator could not get this integration bridge to function properly, with more time it is possible that the evaluator could get this this bridge to function properly providing the same benefits as the bridge for Visual Studio.
Visual Studio Integration	This tool provides a plugin at an additional charge to integrate with Visual Studio. The plugin functioned well allowing for rapid sync and toggling between code and model. It is well worth the extra $89-$145.	This tool integrates with an older version of Visual Studio	This tool provides a plugin free of charge which integrates with Visual Studio. The plugin functioned exceptionally well allowing for real-time syncing between model and code.
Import of XMI Representation of UML Models	The XMI import of the class diagrams was successful with a few exceptions. This could be due to limitations from the input program Magic Draw. This feature would	The XMI import failed with multiple import format options.	The XMI import of the class diagrams was successful with a few exceptions. This could be due to limitations from the input program Magic Draw. This feature would

	be most useful for one-off transfers of models. The imported diagrams need to be verified by a user. Sequence diagram lost the majority of its defined elements during the import.		be most useful for one-off transfers of models. The imported diagrams need to be verified by a user. Sequence diagram was not imported by UModel via XMI.
Export of XMI Representation of UML Models	The export was successful with no exceptions.	The export was successful with no exceptions.	The export was successful with no exceptions.
Version Control Integration	Version Control with subversion was easy to use and install in the tool. The only limitation is that model packages must be checked in and out as a single unit. This could hinder collaborative development.	This product integrates with version control systems utilizing standard Eclipse integration packages. The evaluator could not get this working for subversion.	Version Control with subversion was easy to use and install in the tool. Additionally, the tool enables for check-in and at the element level allowing for collaboration between multiple architects.

Table 11: Integration Capabilities of MDA Tools – Range 0-2 (0 Lowest to 2 Highest)

	Enterprise Architect	IBM Software Architect	Altova UModel
Eclipse Integration	2	2	1
Visual Studio Integration	2	1	2
Import of XMI Representation of UML Models	1	0	1
Export of XMI Representation of UML Models	2	2	2
Version Control Integration	1	1	2

Usability is the next metric result set. The results of the tool evaluations for this metric are detailed in Tables 12 and 13.

Table 12: Results of Usability of MDA Tools

	Enterprise Architect	IBM Software Architect	Altova UModel
Tutorials	Enterprise Architect provides many easy to	Rational provides a large set of tutorials focused on	Altova provides a limited set of tutorials focused

	follow tutorials; however, there is very little content available specifically for MDA and MDD functionality.	creating custom model-model transformations.	within the MDA space.
General Documentation	Help menus were easy to use.	Help menus were easy to use.	Help menus were easy to use.
End User Experience	Enterprise Architect is a large software product with many features. They do a good job defining views that the user can select in order to limit the tool bars and functions to specific tasks. The tool did not freeze or crash during the evaluation. It appears to be built on top of Eclipse.	Software Architect froze multiple times during the evaluation. In general, the evaluator found the product difficult to use specifically as it relates to designing models and locating features.	UModel was extremely stable during the evaluation and navigating through the application was very easy.
Ease of installation	Easy. Installed in under 20 minutes	Moderate. Install took 8 hours.	Easy. Installed in 5 minutes.
Support Options	Email is the preferred support communication means and is available to all users. Registered users get preferred support.	Support portal open for read access to the public. Support tickets can only be submitted by users with an active support package.	Preferred support available for registered users with an active support package. User forums and FAQs available.

Table 13: Usability of MDA Tools – Range 0-2 (0 Lowest to 2 Highest)

	Enterprise Architect	IBM Rational	Altova UModel
Tutorials	1	2	2
General Documentation	2	2	1
End User Experience	2	0	2
Ease of installation	2	1	2
Support Options	2	1	1

Requirements are the next metric result set. The results of the tool evaluations for this metric are detailed in Tables 14 and 15.

Table 14: Results of Requirements of MDA Tools

	Enterprise Architect	IBM Software Architect	Altova UModel
Licensing Options	Per seat or floating	Per seat	Per seat or floating

Pricing	$699 seat or $849 floating Ultimate Edition includes IDE integration.	$4,320	$379
Supported Operating Systems	Windows	Windows, Linux, Mac	Windows

Table 15: Requirements of MDA Tools – Range 1-3 (1 Lowest to 3 Highest)

	Enterprise Architect	IBM Software Architect	Altova UModel
Licensing Options	3	2	3
Pricing	2	1	3
Supported Operating Systems	1	3	1

Chapter 5: Discussion and Lessons Learned

5.1 Project Rationale

This research project aims to evaluate the effectiveness of three software modeling and architecture tools within the context of their built-in MDA capabilities. Reflecting on my software development experiences over the past seven years it has become clear that software modeling and design, despite known advantages, is too often neglected within the enterprise. MDA attempts to bridge the gap between analysis and design with software development and implementation by promoting models to primary software artifacts. The results of this research project serves as a guide for me, and hopefully others, towards implementing MDA into existing software development methodologies. Further, it provides a critical evaluation of enterprise-grade MDA tools specific to a pre-defined set of functionality.

It is my near-term professional goal to integrate one of the MDA tools evaluated in this research into the existing software development methodology at my current place of employment. Over the long-term, I plan to expand my understanding of MDA, continue to refine its application, and spread its benefits within the enterprise.

5.2 Lessons Learned

There are still hurdles to overcome to fully realize MDA within the enterprise. As suggested by previous research, expertise with formal modeling and tool support for round-trip engineering remain primary challenges facing MDA. This research indicates that tool support for MDA is becoming viable within the enterprise specifically as it relates to the modeling and transformation of structural models, particularly class diagrams. Further, this research illustrates

that tool support for platform specific model driven development is robust and can be applied to both structural and behavior modeling.

Every tool is different and should be selected based on environment, resource availability, project scope, and budget. Levels of integration with popular IDEs and version control systems vary among the tools. Product complexity and extendibility add additional layers of features to be considered when selecting a tool to support MDA. For example, although IBM Software Architect performed poorly in regards to the procedure log template defined in this research, it does provide a deep level of extendibility enabling the architect to define their own set of model transformation mapping rules. Given the context of a larger software project, expert resources, and certain target platforms, it is likely that Software Architect would outperform the other tools evaluated in this research. MDA tool selection should be dependent on context of how and where it is to be implemented.

5.3 Addressing the Research Questions

This research evaluates the suitability of several current enterprise grade MDD tools within the context of MDA. As presented in the introduction on page 10, this project aims to address four research questions. These questions are addressed in the following sections.

5.3.1 How effective are current tools in regards to MDD within the framework of MDA?

One of the primary goals of this research is to evaluate the effectiveness of the selected tools in regards to their built-in support for MDA. All three of the tools lacked straightforward support for full round-trip engineering between PIMs, PSMs, and code. The tools proved to be most effective dealing with structural transformations between model and code, notably translating package and classes to from class diagrams. Tool support for behavior model

transformations varied. The research suggests that adopting a progressive approach to MDA within the enterprise starting with structural transformations between PSMs and source code. Additionally, behavioral transformations from source code to model can provide enterprises that lack existing behavioral diagrams immediate benefits specifically as it relates to explaining and visualizing code functionality. As tools mature and/or expert resources are brought into the enterprise the depth of MDA application can expand to encompass a more unified MDA process.

5.3.2 What new challenges are introduced by adopting MDD within the enterprise?

This research illustrates that adopting MDA presents several new challenges within the enterprise. The challenges identified include acquiring the needed talent to support MDA and tool support for round-trip engineering.

In order to effectively realize MDA it is imperative that the enterprise sources qualified personnel with expertise in software engineering specifically relating to UML modeling and software architecture. Some tools provide built-in model-to-model transformations while others require the architect to define mapping transformations. Depending on the level of control and complexity required for the project additional expertise may be required to develop custom mappings between models to effectively support required transformations. The acquisition of talented software engineers with the skill sets required to support MDA within the enterprise is challenging given the competiveness in today's high tech job market.

This research illustrates that tool support for synchronization between PSMs and source code is robust; however, round-trip engineering to and from PIMs to source code remains challenging particularly when leveraging built-in MDA functionality provided by the tools. It is

possible that this challenge could be addressed by developing custom transformations between PSMs and PIMs; yet, this would require additional level of expertise within the enterprise.

5.3.3 How easy/difficult is it to integrate the tools into an existing software development environment or toolset?

In order to seamlessly incorporate a MDD tool into the enterprise it should provide integration points with selected development environments and source control systems. From an MDA perspective the tools should also provide import and export functionality using a standard exchange format such as XMI. Each tool evaluated in this research provided integration bridges to two of the more popular IDEs; Eclipse and Visual Studio. The level of effort required to achieve integration between the tools and target IDEs varied. Some integration combinations successfully setup with minutes while others required extensive configuration. Overall the research shows that tool support for integration into existing software development environments is mature; however, due to limitations and preferences certain tools/IDE combinations are more robust than others. It is important to consider the depth of integration available for the enterprise's existing IDEs when selecting a MDD tool.

5.3.4 What are the relative strengths and weakness of the selected MDD tools?

Given the context of the procedure log template and metrics defined in this research, Altova UModel clearly lead the group followed by Sparx Systems Enterprise Architect and then IBM Rational Architect. All three of the tools lacked straightforward support for full round-trip engineering between PIMs, PSMs, and Code.

In regards to the first metric, Model/Transformation/Code Generation Capabilities, Altova UModel and Sparx Systems Enterprise Architect provided built-in MDA transformations

from UML based PIMs to target UML based PSMs. Both tools supported only structural transformations between PIMs and PSMs. IBM Software Architect did not provide built-in Model-to-Model transformations between PIMs and PSMs; however, it does support user defined model to model transformations. Although this feature was not evaluated, UModel and Enterprise Architect also provide mechanisms for the architect to modify existing mappings between PIMs and PSMs. All three tools provide robust model to code transformations and code to model transformations. UModel is limited to model transformations with class, sequence, and state diagrams while the other two products support a much wider collection of UML diagram transformations.

The three tools performed fairly well in regards to the second matrix, Integration Capabilities. UModel and Enterprise Architect outperformed Software Architect when importing XMI. All three products successfully exported valid XMI that was viewable in a third party modeling tool. In regards to IDE integration, UModel was the clear winner with Visual Studio providing real-time visualization and synchronization between model and code. Enterprise Architect provides integration plugins for Visual Studio and Eclipse; however, the evaluator was unable to successfully configure the plugins. IBM Software Architect is built on top of Eclipse which provides deep integration within this development environment. All three tools integrate with a variety of version control systems. Enterprise Architect limits check-in and check-out of modeling artifacts at the package level, a limitation worth noting.

From a usability perspective, Enterprise Architect and UModel were clear winners. Simple tutorials provided the evaluator with introductory knowledge needed to quickly test built-in MDA functionality inside the tools. Both of the tools were responsive to the end user. Software Architect became non-responsive on multiple occasions and the tutorials provided were

extremely involved. It is clear that Software Architect targets a more sophisticated user base willing to commit expertise and capital resources in order to realize the benefits of an extensible and integrated MDA tool within the Eclipse/Websphere ecosystem.

Product cost is a major factor for organizations with limited resources allocated to IT spending. UModel and Enterprise Architect are reasonably priced options for organizations with restricted budget resources; both available for purchase well under $1000.

5.4 Challenges Encountered

The challenges encountered during this research project were broad and wide. The greatest challenge was time. Understanding that these tools are complex products that often require years of experience to master, it was a challenge to evaluate their effectiveness within an allocated time frame of approximately forty hours per tool. Other challenges included having the required expertise in UML and setting up IDE integration within some of the tools.

Chapter 6: Conclusion and Future Work

In concluding this research, I have discovered that MDA is possible within the enterprise and there are viable products on the market to support its adoption. Further, I have found that each tool has specific advantages and disadvantages and tool selection should be made based on a variety of factors. Based on the environment, human resources, and budget at my current place of employment I plan to leverage Altova UModel as a tool for integrating MDA into our software development methodology.

There are many paths future research may take to build upon this project. One path could focus on implementation of a fully functional software program leveraging MDA with one of the tools evaluated in this project. Another option may focus on a specific metric such as Model

Transformations rather than evaluating the tools across multiple metrics. These paths would enable the researcher to spend more time on, and dig deeper into, specific MDA functionality within the tools. The selected tools also provide mechanisms for visually testing and debugging models, this is an exciting feature as well and it could be explored in more detail in future research.

References

1. *Model-driven Development of Complex Software: A Research Roadmap.* **France, Robert and Rumpe, Berhard.** 2007, 2007 Future of Software Engineering (FOSE '07)., pp. 37-54.

2. *From Code Centric to Model Centric Software Engineering: Practical case study of MDD infusion in a Systems Integration Company.* **Magarida Afonso, Regis Vogel, Jose Teixeria.** s.l. : Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, 2006.

3. **Object Management Group.** About OMG. [Online] http://www.omg.org/gettingstarted/gettingstartedindex.htm

4. **Mellor, Stephen, Scott, Kendall, Uhl Axel, and Weise, Dirk**. *MDA Distilled: Principles of Model Driven Architecture.* Addison Wesley, 2004.

5. **Object Management Group.** OMG's MetaObject Facility. [Online] http://www.omg.org/mof/

6. *Model-Driven Development: Where Does the Code Come From?* **Jicheng Fu, Wei Hao, Farokh B. Bastani, I-Leng Yen.** s.l. : 2011 IEEE Fifth International Conference on Semantic Computing, 2011, International Conference on Semantic Computing, pp. 255-262.

7. *Challenges in Deployment of Model Driven Development.* **Susanna Teppola, Päivi Parviainen, Juha Takalo.** 2009, 2009 Fourth International Conference on Software Engineering Advances, pp. 15-20.

8. *Model-Driven Development in the Enterprise.* **Uhl, Axel.** 1, s.l. : IEEE Software, Jan-Feb 2008, Vol. 25, pp. 46-49.

9. **Miles, Russ, Hamilton, Kim.** *Learning UML 2.0.* 2006. O'Rielly.

10. **Fowler, Martin. UML Distilled:** *A Brief Guide to the Standard Object Modeling Language. 3rd Edition. 200.* Addison Wesley

11. **Sparx Systems.** About US. [Online] http://www.sparxsystems.com/about.html

12. **Wikipedia**. Enterprise Architect (program). [Online] http://en.wikipedia.org/wiki/Enterprise_Architect_(Visual_Modeling_Platform)

13. **Wikipedia.** IBM. [Online] http://en.wikipedia.org/wiki/Ibm

14. **Wikipedia.** IBM Rational Rose Architect. [Online] http://en.wikipedia.org/wiki/IBM_Rational_Software_Architect

15. **IBM.** Rational Rose Architect. [Online] http://www-01.ibm.com/software/rational/products/swarchitect/features/?S_CMP=rnav

16. **Wikipedia.** Altova. [Online] http://www-01.ibm.com/software/rational/products/swarchitect/features/?S_CMP=rnav

17. **Altova.** Umodel. [Online] http://www.altova.com/umodel.html

18. **Wikipedia.** Umodel. [Online] http://en.wikipedia.org/wiki/UModel