High Level Support for Performance Engineering, and Model Integration and Model Transformation

Rainbow Yuhong Cai

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science, The University of Auckland, February, 2009.

Abstract

This thesis provides high level support for Performance Engineering in software architecture design via two research projects - Argo/MTE and MaramaMTE+. The Argo/MTE project extends the well established ArgoUML tool to support software architecture modelling and performance evaluation. The Argo/MTE research shows how commonly used Components Off The Shelf (COTS) tools can improve the usability and maintainability of an in-house technology, and how the use of standard model representations can improve tool integration. The MaramaMTE+ research integrates the traditional software architecture modelling with the web user behaviour modelling using the Marama meta-tool. The MaramaMTE+ research shows how model integration can extend the applicable domain of a software model; how a meta-tool can support efficient tool extension; and how to support automatic generation of web load testing plans.

This thesis provides high level support for Model Integration and Transformation via the research project MaramaCRelation. The MaramaCRelation research provides a structured approach to interconnect domain-specific models. It maintains the rational of an interconnection of domain-specific models; records semantics maintained and lost across the interconnected models; tracks the evolvement of modeling elements through the traceability across the interconnected models; and maintains behavior synchronization across the interconnected models.

The thesis has made contributions in software architecture design, software architecture performance evaluation, web load testing, and model integration and transformation. More specifically, the research of the thesis is aimed for improving the automatic support, analysis and design support, and systematic and structured support for Performance Engineering and Model Driven Engineering.

Acknowledgements

I guess my supervisors, Professors John Grundy and John Hosking, will never know how much I am grateful for their encouragement and support. But I still want to say a big thank you to them loudly, publicly. I have been distracted numerous times during my PhD research, and there was a time I almost gave up. They paid the maximum patience to me, and still held confidence on me even when I myself almost lost it. They generously shared ideas with me to help me to achieve results quickly; they went through research details with me when I could not clarify my ideas; they gave me financial support when I was broke; and they wrote me the best references when I was looking for a job. I owe my achievements to them. I will do my best to make them proud.

My warm thanks go to the department staff Robyn Young, Anita Lai, Lei Zhang, and Keith Johnston. Your great help saved me so many hassles during my working and studying in the department. I will not forget my fellow PhD students Karen Li, Richard Li, Fuad Tabba, Norhayati Mohd Ali, and Christian Hirsch. I want to thank you all for valuable discussions, productive collaborations, and memorable laugher.

To my husband, Christopher Xu, I am sure he knows how much I appreciate his life-long trust and support. This thesis is for you, Christopher!

Contents

Lists of Figures

List of Tables

Chapter 1 - Introduction

"Software Performance Engineering (SPE) is a systematic, quantitative approach to construct cost-effective software systems to meet their performance requirements" (Feldman et al, 2007). SPE aims for helping people to make decisions on architecture, design, and implementation; and it involves activities of analysis, modeling, estimation, evaluation, and comparison.

Performance engineering in software architecture design has become crucial in the development of large scale software systems. People want to evaluate, improve the performance of software architecture at the early stage of Software Development Life Cycle. SoftArch/MTE (Grundy and Cai, 2004; Grundy and Cai et al, 2001), the research in the master thesis of the author, provides an effective, efficient approach to evaluate the performance of software architecture. SoftArch/MTE allows architects to use a visual Architecture Description Language to sketch high-level system descriptions, including: client, server, database and host elements, and expected client requests and server and database services. From these descriptions, SoftArch/MTE automatically generates a reusable, deployable performance test-bed. The test bed is a fully functional distributed java application containing performance evaluation information. It can be deployed and run as a real distributed software system. The performance evaluation results can then be collected, stored, analyzed, and visualized (Grundy and Cai, 2004; Grundy and Cai et al, 2001).

The initial success of SoftArch/MTE prompts many problems for further research. Among them, the most interesting ones are:

1. The proprietary SoftArch/MTE tool can not deal with large industrial cases of architecture modeling and performance evaluation.
2. The performance evaluation technology of SoftArch/MTE does not support the process of model refinement and transformation, but generates test bed directly.

3. The performance evaluation technology needs to extend its applicable scope by integrating with other software modeling technologies.
4. In Model Driven Engineering, when transforming and integrating models, there is no structured high-level support.

The four problems motivated three individual yet intrinsically related research projects in this thesis, and they are: Argo/MTE, MaramaMTE+, and MaramaCRelation. Argo/MTE, a continuation of the SoftArch/MTE project, is aimed for industrial usage by extending an open-source UML CASE tool to provide high level support for software architecture analysis, design, and performance evaluation. The MaramaMTE+ project ports many of the Argo/MTE features to using the Marama meta-tool (Marama meta-tool, 2007). The MaramaMTE+ project also leverages the strength of the Stochastic Form Chart model for realistic client behavior modeling; and it uses a web crawler to synthesize Form Chart models. The challenges of complex model and tool integration involved in both Argo/MTE and MaramaMTE+ motivated the development of the MaramaCRelation project – a high-level support for Model Integration and Transformation (MI&T). The MaramaCRelation project also offers a central place to review, organize the main research issues involved in MI&T. Figure 1.1 briefly summarizes the problem domain of each project, and a detailed explanation of this figure is given in the following sections of this chapter.

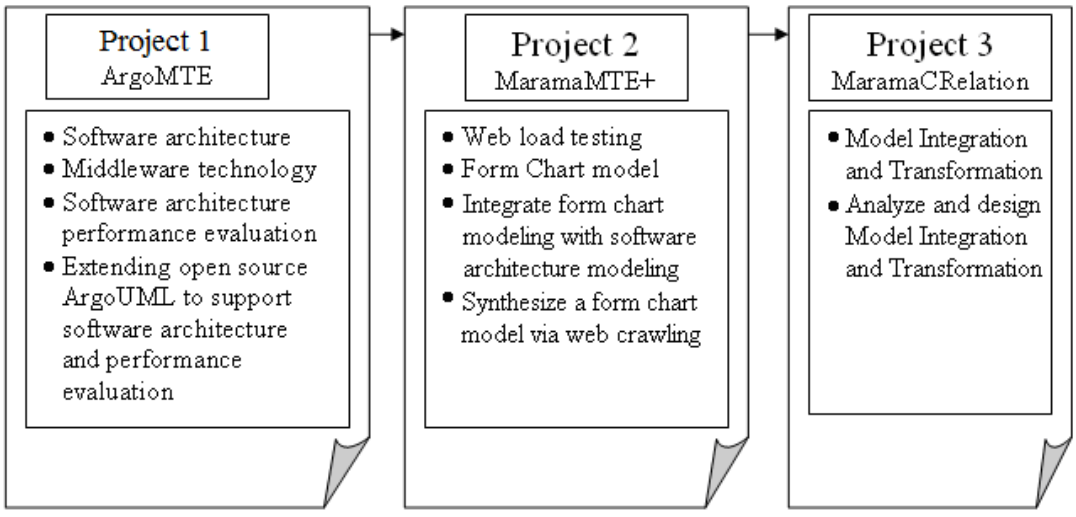


Figure 1.1. The three individual yet closely related research projects

1.1 The problem domain of the Argo/MTE project

The problem domain of Argo/MTE covers: middleware technologies, software architecture modeling, software architecture performance evaluation, and tool integration.

1.1.1 Using middleware in software architecture

Middleware, a structured software component, becomes an important part of web application software architecture (Oracle, 2006). Middleware technologies mask some kinds of heterogeneity of large distributed systems, including: heterogeneity of networks and hardware; heterogeneity of operating systems or programming languages, or both; and even heterogeneity among vendor implementations of the same middleware standard (Bakken, 2003). The popular middleware technologies include CORBA (OMG, 1995), RMI (Java RMI, 1999), J2EE (Java EE, 2007), .NET (MS .NET, 2007), and COM/DCOM (MS COM/DCOM, 2007).

With their increasing popularity and functionality, middleware technologies become more and more important in software architecture design (Oracle, 2006; Jackson et al, 2005; Feast, 2002). Today, software architects would always select one or more middleware products to develop large distributed computing systems rather than start from scratch. Faced with a number of middleware products, it is always hard for end-users to select the right one for the project being constructed (CSIRO, 2000). The Argo/MTE project supports software architects in their modeling of complex, middleware-based software architectures. Argo/MTE embodies the main components of a middleware technology in software architecture models, and supports the software architect in quickly evaluating and comparing the performance of various middleware technologies through comparing the performance of these software architecture models.

1.1.2 Software architecture modeling

The Argo/MTE tool models software architecture of middleware-based web applications. An Argo/MTE-styled software architecture model is designed to: 1) abstract away middleware-level component information from web applications and yet provide enough information for test bed generation, and hence decision making; 2) define the behavior of each component and allows components to interact with each other; 3) specify how the components interact with each other, and

omit the information that does not pertain to component interaction; and 4) comprise multiple views to analyze the possibly very complex software architecture design.

1.1.3 Software architecture performance evaluation

Decisions of software architecture design need to be made at the early stage of Software Development Life Cycle (SDLC) (White et al, 1997; MacKenzie, 2002). Performance is one of the main non-functional requirements to compare candidate architecture designs (Grundy et al, 2001; Liu et al, 2005). It is important to evaluate an intended software architecture design realistically, accurately, and efficiently. Argo/MTE was intended to provide tool support for evaluating the performance of middleware-based software architecture of web applications in particular. An Argo/MTE-styled software architecture model generates a fully functional test bed (consisting of java program, web pages, and database files) that contains performance evaluation information. The generated test bed can be executed to produce accurate evaluation results. The performance evaluation process is highly automated, and is seamlessly integrated with that of software architecture modeling (Cai et al, 2004).

1.1.4 Extending ArgoUML

Argo/MTE extended an open source UML CASE tool – ArgoUML (ArgoUML, 2003), to support software architecture modeling and performance evaluation. Argo/MTE leverages the strength of ArgoUML as it is a well-accepted, open source modeling environment. Argo/MTE added architecture-specific data to the existing UML meta-model, which allows software architecture modeling and performance evaluation to follow the same style as that of the existing UML modeling (e.g. UML class diagram modeling). Compared with SoftArch/MTE (Grundy and Cai et al, 2005) - the predecessor of Argo/MTE, Argo/MTE provides a much better modeling environment, and has greatly improved the flexibility, maintainability, and usability of the technology of test bed generation and performance evaluation.

1.2 The problem domain of the MaramaMTE+ project

The problem domain of MaramaMTE+ covers: the Marama meta-tool, web load testing and Form Chart modeling, and automatic web reverse engineering.

1.2.1 Marama meta-tool and MaramaMTE

The Marama meta-tool developed by Grundy et al (Marama meta-tool, 2007) is a set of tools for building diagramming applications in Eclipse. It provides facilities to specify complex diagram-based meta-models, shapes and connectors, and views. It also supports: complex behavior specification via OCL constraints, visual event handlers, and a comprehensive API (Marama meta-tool, 2007). The Marama meta-tool allows software development organizations to develop Domain-Specific Software Tools effectively and efficiently. The Marama meta-tool aims for rapid prototyping of multi-view, multi-user diagramming applications with live update and easy end-user accessibility.

The feasibility of the Marama meta-tool is demonstrated through the development of selected proof-of-concept domain-specific tools, and MaramaMTE (MaramaMTE, 2007) is one of the successful and complex examples. MaramaMTE is a domain-specific tool built on top of the Marama meta-tool. It re-implements the key features of software architecture modeling and performance evaluation of Argo/MTE. The Marama meta-tool makes it possible for MaramaMTE to efficiently interact with other domain-specific modeling technologies to leverage their strength for software architecture modeling and performance evaluation.

1.2.2 Web load testing and the Form Chart model

Web Application Load Testing (WALT) is an important part of web performance engineering. WALT measures response time, throughput, and availability of a target website from a client's perspective (usually a web browser) (Apache JMeter, 1999; WebLOAD, 2003; Proxy-Sniffer, 2008). It needs to be undertaken rigorously before a robust cost-effective website can be constructed.

MaramaMTE supports WALT via its integration with the formal Stochastic Form Chart modeling (Draheim et al, 2006). A *Form Chart* model is a technology-independent bipartite state diagram, and is used to simulate web user behavior (Draheim et al, 2006). It describes at a high level what the user sees as system output, and what he or she provides as input to the system. A Stochastic Form Chart additionally provides probabilities around user interactions with web forms and their responses to these interactions. Essentially this provides a probabilistic model of user-website interactions.

Like most of the existing technologies of software architecture modeling, MaramaMTE is originally focused on the structure of the main components of a software system (e.g. server components); and is lack of realistic analysis and design of client behavior. Combining Stochastic Form Chart modeling with MaramaMTE's software architecture modeling provides a powerful model-based performance estimation approach. The user behavior model can be used to evaluate a web application's performance under loading at design time before significant implementation expenses are incurred.

1.2.3 Automatic web reverse engineering

MaramaMTE+ was designed to extend MaramaMTE in two key aspects: 1) to automate the process of Form Chart model structuring, because manually modeling a Form Chart model is tedious and error-prone; and 2) to generate complex web load testing plans and scripts. MaramaMTE+ uses a web crawler to extract structural information from a running target website and aggregates the collected data to generate Form Chart models, which efficiently provides users with skeletons of the Form Chart models. Users then manually augment the generated skeletons to complete their Stochastic Form Chart models. A complete Form Chart model is then used by MaramaMTE+ to generate a *client* side program for a MaramaMTE-styled *architecture* test bed, as well as testing plans for a specific target third party testing tool such as Apache JMeter.

1.3 The problem domain of MaramaCRelation

The problem domain of MaramaCRelation covers: model driven engineering, model integration, and model transformation.

1.3.1 Model Driven Engineering in Software Engineering

In software development, Model-Driven Engineering (MDE) refers to a range of development approaches that use models as a primary form of expression (Schmidt, 2006). The models involved in MDE range from platform-independent models at high abstraction level (e.g. business requirement model, business process model) to platform-dependent models at low abstraction level (e.g. UML class model, UML sequence model). The software engineering models, much more than the intuitive "box-and-line" diagrams, can: 1) contain a certain level of detail and then code is written by hand in a separate step; 2) contain executable actions; and 3) generate code ranging from system skeletons to complete, deployable products. For example Argo/MTE and MaramaMTE+ both use a variety of abstract models

of architecture, process, and website usage and OO design; and these models are used to synthesize detailed code and scripts for performance test beds.

1.3.2 Model integration

Model integration is an important operation in MDE. Software modeling technologies cover every aspect of software development lifecycle at various abstraction levels (BPMN, 2004; Li et al, 2007; Krutchen, 1995; Garlan et al, 1997; Medvidovic et al, 1996). Model integration combines models with different target domains to serve a more comprehensive target domain for operational, tactical, and strategic purposes. Model integration has been researched through model comparison (Soto, 2008; Briand, 1998; IBM, 2005), model merging (Sabetzadeh et al, 2006), and multi-model coordination and synchronization (Kirwan et al, 2008). For example, MaramaMTE+ integrates a domain-specific architecture modeling language (based on that from Argo/MTE) with a Stochastic Form Chart model, using each modeling language for its intended domain of discourse (architecture structuring and website predictive interaction respectively). These models are integrated to provide a multi-view approach to web architecture modeling and usage prediction.

The MaramaCRelation approach uses the CRelation model to capture the rationale that motivates model integration, and uses the captured rationale as a central place to review and reorganize the main issues involved in the model integration. For example, MaramaCRelation allows users to model interconnection relationships between architecture and form chart models in MaramaMTE+, and maintain traceability and behavior synchronization between the models.

1.3.3 Model transformation

Model transformation, a process of converting one model to another, is a core technology in MDE. It involves extensive research in: transformation languages (ATLAS Transformation, 2006; XSLT Transformation, 2001; Csertan et al, 2002), traceability (Falleri et al, 2006; Amar et al, 2008), behavior and view synchronization (Garcia, 2008; Xiong, 2007**;** Sendall, 2004), and consistency management (Kuster, 2004; Sanchez et al, 2008).

Model transformation is meaningful because the involved models share common semantics conceptually. A transformation preserves certain semantics of the source model in the transformed target model. The

preserved semantics can be presented significantly differently in the transformed model; the traceability can be easily lost; and the behavior and view synchronization among models is hard to achieve. The MaramaCRelation approach captures the rationale that motivates the model transformation, and uses the captured rationale to support a flexible traceability mechanism, as well as a behavior and view synchronization mechanism for the model transformation. For example, in MaramaMTE+ the combined architecture and form chart models are high-level abstract models that need to be transformed into lower-level code and scripts to support performance evaluation of the architecture model under the specified predictive usage models. MaramaCRelation provides a model and associated infrastructure to support analysis, design, and beyond of the transformation process.

1.3.4 Analyze and design MI&T

The MaramaCRelation approach supports analysis and design of MI&T, and reviews some of the main issues involved in MI&T at a central place from a high abstraction level. The MaramaCRelation uses the CRelation model to: visually represent the rationale behind MI&T; decompose the traditionally monolithic transformation scripts; associate the traditionally isolated rules and templates in the context of a broader model; and differentiate what from how to integrate and transform models. The MaramaCRelation approach supports model integration and transformation in the same way that Object Oriented Analysis (OOA) and Design (OOD) technologies support Object Oriented Development. It views MI&T at a high level, which raises the level of automated, flexible support for the traceability as well as behavior and view synchronization during MI&T. In MaramaMTE+, for example, MaramaCRelation can provide visual mappings between architecture, form chart, OO design, and EJB models. It allows transformations between these to be modeled precisely and declaratively. It organizes the inter-model transformation and consistency rules using inter-model entities and relationships. Finally, it provides infrastructure support for realizing model transformations, traceability, and behavior synchronization for the MaramaMTE+ tool.

1.4 Outline of thesis

Chapter 2 reviews the related work. It introduces the important concepts used in the research, and identifies the problems that motivated the research.

Chapter 3 introduces the main features of the SoftArch/MTE project, and specifies the motivations for the research in this thesis.

Chapter 4 examines the development of Argo/MTE. The improvements of Argo/MTE from SoftArch/MTE are detailed.

Chapter 5 presents a case study of using Argo/MTE to support software architecture modeling and performance evaluation. Argo/MTE-styled software architecture is showcased; the performance evaluation test bed is generated and executed; and the performance results are collected and discussed.

Chapter 6 introduces the Marama meta-tool and MaramaMTE. The basic ideas of the Marama meta-tool are discussed. The main improvements from Argo/MTE to MaramaMTE are explained.

Chapter 7 examines the development of MaramaMTE+. The process of generating Form Chart model skeleton via web crawling is detailed. The process of automatically generating web load testing plan is well explained.

Chapter 8 specifies the problem domain of the MaramaCRelation project. It introduces the background and motivation of the research, and highlights of the research. The chapter prepares readers for the MaramaCRelation project.

Chapter 9 presents the CRelation modeling. The chapter presents the used terms and concepts of the CRelation model. It introduces the abstract syntax and semantics of the CRelation model through a running example.

Chapter 10 discusses the MaramaCRelation prototyping tool. It presents the mechanisms and algorithms of how the MaramaCRelation prototyping tool supports the CRelation modeling.

Chapter 11 presents two running case studies of using MaramaCRelation to support MI&T. One case study is small and has been used to explain the MaramaCRelation approach throughout chapters 8, 9, and 10. The other one is more comprehensive and can demonstrate that the MaramaCRelation research has achieved the proposed requirements.

Chapter 12 evaluates the MaramaCRelation approach by using a comprehensive questionnaire.

Chapter 13 discusses the interesting future research questions raised from the MaramaCRelation research.

Chapter 14 finishes the thesis with a comprehensive discussion of the achieved results throughout the thesis.

Chapter 2 - Related Work

As is introduced in chapter one, the research in the thesis is focused on Performance Engineering and Model Driven Engineering (MDE). More specifically, Performance Engineering is researched through software architecture modelling, software architecture performance evaluation, web reverse engineering, and web load testing; and MDE is researched through Model Integration and Transformation (MI&T), software engineering environments, and model semantics representation and checking. This chapter reviews the leading work of these related areas by introducing their concepts, goals, and problems; introduces the essential background for understanding the research in the thesis; and discusses the problems and goals that motivate the research in the thesis.

2.1 Software architecture modelling

The software architecture of a software system is the structure or structures of the system (Len Bass et al, 2003). Software architecture describes intended software by using structural elements, architectural components, subsystems, sub-assemblies, parts or "chunks" (Bachmann et al, 2000). It must support the functional requirements of the software, and take into account of the dynamic behavior of the software. It must also concern about the attributes of the intended system, including performance, security, scalability, and flexibility or extensibility (Software Architecture, 2008).

2.1.1 Architecture Description Language

Software architects often use an Architecture Description Languages (ADLs) to model architectures. Nenad Medvidovic and Richard N. Taylor reviewed and compared a substantial group of ADLs (Medvidovic et al, 2000), and some of them are listed in Table 2.1. Other ADLs include UML (UML, 1996), Booch Notation (Booch, 1994), xADL (Khare et al, 2001), and SoftArch/MTE (Grundy and Cai et al, 2001).

ADL	Definition
ACME (Garlan et al, 1997)	supporting architectural interchange, predominantly at the structural level
Aesop (Garlan, 1995)	supporting the use of architectural styles
C2 (Medvidovic et al, 1996**)**	supporting the description of user interface systems using an event-based style
Darwin (Magee et al, 1995)	supporting the analysis of distributed message-passing systems
Rapide (Luckham et al, 1995)	allowing architectural designs to be simulated, and has tools for analyzing the results of those simulations
SADL (Moriconi et al, 1997)	providing a formal basis for architectural refinement
UniCon (Shaw et al, 1995)	generating Glue code for interconnecting existing components using common interaction protocols
Wright (Allen, 1997)	supporting the formal specification and analysis of interactions between architectural components

Table 2.1. Some of the ADLs reviewed by Nenad Medvidovic and Richard N. Taylor (Medvidovic et al, 2000)

2.1.2 Conceptual concepts of software architecture modeling

Software architects model software architecture based on a set of conceptual concepts, including: components, connectors, configurations, views, and styles. All available ADLs support those conceptual concepts to certain extent.

2.1.2.1 *Components*

"In software architectures, components represent the primary computational elements and data stores of a system" (Medvidovic et al, 2000). Typical software components include: clients, servers, filters, objects, blackboards, and databases. A component describes its features mainly using: *interfaces, semantics, types, and constraints*.

Component interfaces define points of interaction between a component and its environment. The interface specifies the services (messages, operations, and variables) a component provides and needs. An interface point in SADL (Moriconi et al, 1997) or Wright is a port (Allen, 1997), and in UniCon a player (Shaw et al, 1995). In SoftArch/MTE, the interface of a component is consisted of a set of properties that define how the component interacts with the other model elements to construct a fully functional distributed system (Grundy and Cai et al, 2001).

Component types abstract and encapsulate functionality into reusable blocks. A component type needs to be instantiated and can be instantiated multiple times in a single architecture. It may also be reused across architectures. All of the ADLs distinguish component types from instances. For example, Rapide defines types in a separate type language (Luckham et al, 1995). SoftArch/MTE defines types in a separate domain-specific meta-model (Grundy and Cai et al, 2001).

Component semantics is a set of associated information that performs analysis, enforces architectural constraints, and ensures consistent mappings of architectures from one level of abstraction to another. Component types can be viewed as a part of component semantics. Component semantics needs to be defined at a high level model. All ADLs support specification of component semantics, although to varying degrees. For example, UniCon express semantic information in component property lists (Shaw et al, 1995). SoftArch/MTE implicitly specifies semantic information in the code generators of the supporting tool together with documents written in natural language (Grundy and Cai et al, 2001).

"A constraint is an assertion about a system or one of its parts, the violation of which will render the system unacceptable (or less desirable) to one or more stakeholders" (Clements, 1997). Component constraints can also be viewed as a part of component semantics. An example stylistic invariant is C2's requirement that a component has exactly two communication ports, one each on its top and bottom sides (Medvidovic et al, 1996). SoftArch/MTE provides constraints in a meta-model (where types and semantics are also specified). A sample SoftArch/MTE constraint requires that, in multi-tier client server architecture, a client program must be associated with at least one remote application server program(Grundy and Cai et al, 2001).

2.1.2.2 *Connectors*

"Connectors are used to model interactions among components and rules that govern those interactions" (Medvidovic et al, 2000). Some ADLs treat connectors as first-class entities (e.g. Aesop (Garlan, 1995), UniCon (Shaw et al, 1995)), some of them treat connectors as second-class entities (e.g. UML, SoftArch/MTE). A connector describes its features mainly using: *interfaces*, *types, semantics*, and *constraints*.

A connector's interface is a set of interaction points between the connector and the components. Connector interfaces enable proper connectivity of components, as well as their interaction, which helps to reason about architectural configurations. In general, when treated as first-class entities, connectors support explicit specification of connector interfaces. For example, connector interface points in ACME, Aesop, UniCon, and Wright are *roles*, which are named and typed (Medvidovic et al, 2000). When treated as second-class entities, connectors have fairly simple functionality and do not support explicit interfaces. For example, in UML and SoftArch/MTE connectors do not have explicit interfaces.

Connector types abstract and encapsulate component communication, coordination, and mediation decisions. Connector types categorize complicated interaction protocols, and make them reusable both within and across architectures. For example, ACME, Aesop, C2, SADL, and Wright base connector types on interaction protocols (Medvidovic et al, 2000). SoftArch/MTE connectors are also typed, but the types do not support generic interaction protocols, and only constrain what types of components that can be connected (Grundy and Cai et al, 2001).

Connector semantics is a set of associated information that enable component interaction analysis, consistent refinement of architectures across levels of abstraction, and enforcement of interconnection and communication constraints (Medvidovic et al, 2000). Connector semantics must be defined at high level. Connector types can be viewed as part of component semantics. All ADLs support specification of connector semantics, although to varying degrees. For example, SADL provides a constraint language for specifying style-specific connector semantics (Moriconi et al, 1997). SoftArch/MTE implicitly specifies semantic information in the supporting tool together with documents written in natural language (Grundy and Cai et al, 2001).

Connector constraints ensure adherence to intended interaction protocols, establish intra-connector dependencies, and enforce usage boundaries. Connector constraints can also be viewed as part of connector semantics. A simple constraint of a connector is a restriction on the number of components that interact through the connector. For example, C2 imposes a restriction that each connector port may only be attached to a single other port **(**Medvidovic et al, 1996**)**. SoftArch/MTE imposes a restriction that two typed components can only be linked by appropriate types of connectors (Grundy and Cai et al, 2001).

2.1.2.3 *Configuration*

Architectural *configurations*, or topologies, are connected graphs of components and connectors that describe architectural structure (Medvidovic et al, 2000). Architectural configurations must support features including: understandable specifications, compositionality, refinement and traceability, heterogeneity, scalability, evolability, dynamism, and constraint (Medvidovic et al, 2000). The features that are most related to the research in the thesis include:

Refinement and Traceability — A main goal of software architecture modeling is to bridge the gap between informal, "boxes and lines" diagrams and low-level application design activities (e.g. design and implementation). It is important to enable correct and consistent refinement of architectures into executable systems, and maintain the traceability of changes across levels of architectural refinement. Compared with other ADLs, SADL and Rapide support refinement and traceability more extensively. SADL uses its maps (see Figure 2.1) to prove the correctness of architectural refinements. The mapping refines components and connectors from Level 1 architecture (arch_L1) to Level 2 architecture (arch_L2). Overall, ADLs provide limited support for refinement and traceability. Many of them generate code directly from architectures without showing the refining process and maintaining traceability (e.g. SoftArch/MTE, C2).

```
arch_map MAPPING FROM arch_L1 TO arch_L2
BEGIN
  comp --> (new_comp)
  conn --> (new_comp!subcomp)
  port --> ( )
  . . .
```

Figure 2.1. A refinement mapping declared in SADL (Medvidovic et al, 2000)

Heterogeneity — An ADL needs to deal with the ever-increasing complexity and variety of software architecture concerns. It is important that ADLs provide facilities for architectural specification and development with heterogeneous components and connectors. For example, C2 currently supports development in C++, Ada, and Java **(**Medvidovic et al, 1996**)**, while SoftArch/MTE supports development in a list of middleware technologies including CORBA, RMI, J2EE, and .NET (Grundy and Cai et al, 2001).

Evolvability — As software systems continuously evolve, so do their architectures. ADLs must provide evolution support to maintain the validity of software architectures when components and connectors are incrementally added, removed, replaced, and reconnected. Most existing ADLs and their supporting toolsets are very rigorous; and they provide limited support for architecture evolution, such as component addition, and incomplete architecture designs. For example, in Darwin (Magee et al, 1995), MetaH (Binns et al, 1996), Rapide (Luckham et al, 1995), and UniCon (Shaw et al, 1995), compilers, constraint checkers, and runtime systems have been constructed to raise exceptions if architecture is incomplete. xADL is a highly extensible ADL (Khare et al, 2001). It uses a set of XML schemas to define an initial set of architectural concepts. The schemas are modular and extensible. Each architectural concept is defined in a separate schema and each individual schema can serve as the basis for further extension. SoftArch/MTE uses a domain-specific meta-model to support the evolution of the target software architectures. The meta-model can be extended and modified to support the evolution of software architecture.

2.1.2.4 *Architectural styles*

Patterns and styles help people to reuse well-established knowledge. Very often, people choose a certain software architectural style to describe software architecture. An Architectural Style defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style defines a vocabulary of components and connector types, and a set of constraints on how they can be combined (Shaw et al, 1996). Table 2.2 lists several well-known architectural styles and their main characters.

Architectural Style	Definition
client-server	representing software architecture where a component interacts with other components by requesting their services, and the communication between the components is typically a bidirectional pairing of "uses" relationships (Taylor et al, 2008)
pipe and filter	showing data flow architectures based on graphs of pipes and filters (Taylor et al, 2008)
blackboard	representing a family of software systems that are based on shared data space and a set of knowledge sources (Taylor et al, 2008)
multi-tier architecture	representing a client-server architecture in which, the presentation, the application processing, and the data management are logically separate processes (Taylor et al, 2008)

Table 2.2. Sample architectural styles

Most ADLs support architectural styles. For example, Wright supports pipe-and-filter architectural style (Allen, 1997). SoftArch/MTE supports multi-tier architectural style (Grundy and Cai et al, 2001). Figure 2.2 illustrates a 3-tier SoftArch/MTE architecture design. The components enclosed in the green rectangle represent the presentation tier. Components enclosed in the red rectangle represent the application processing tier. Components enclosed in the blue rectangle represent the data tier. In this architecture design, web users send requests through the presentation tier. The presentation tier sends user requests to application servers, and receives feedback from the application processing tier. The application processing tier contacts with data tier to retrieve and store data.

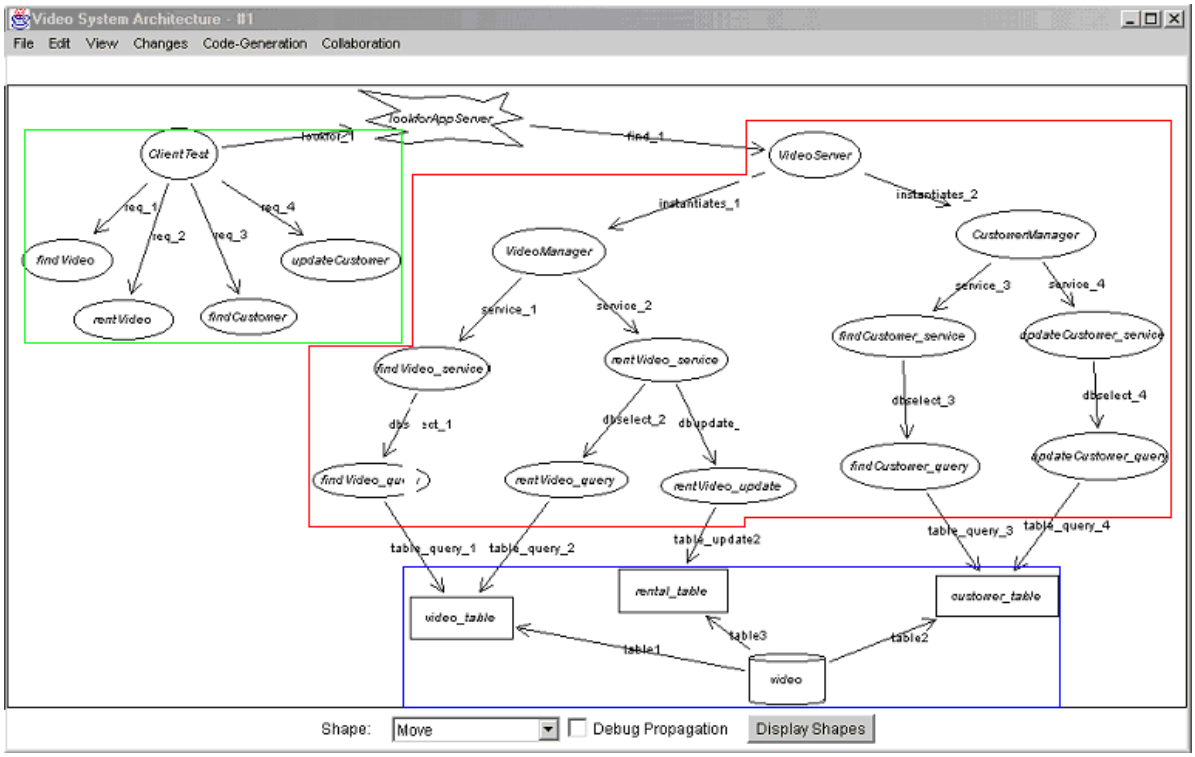


Figure 2.2. A sample SoftArch/MTE architecture design (Grundy and Cai et al, 2005).

2.1.2.5 *Views*

Different stakeholders of a software system have different concerns and view points of software architectures. Software architecture needs to be organized in views to allow stakeholders to focus on their specific concerns. Table 2.3 lists a set of well-accepted architecture views and their interested stakeholders.

View	Abilities	stakeholders
functional/logic	describing what the system should provide in terms of services to its users.	developers and project managers
development/structural	representing software module organization (hierarchy of layers, software management, reuse, constraints of tools)	developers and project managers
physical/deployment	representing how to map software architecture to underlying hardware (Topology, Communication)	technicians, system engineers
user action/feedback	defining how users interact with the system, such as how to enter data, send request, put feedback.	end-users, business analysts
data	explaining how the data flow happens in the software	data analysts, developers and project managers

Table 2.3. Views and their interested stakeholders

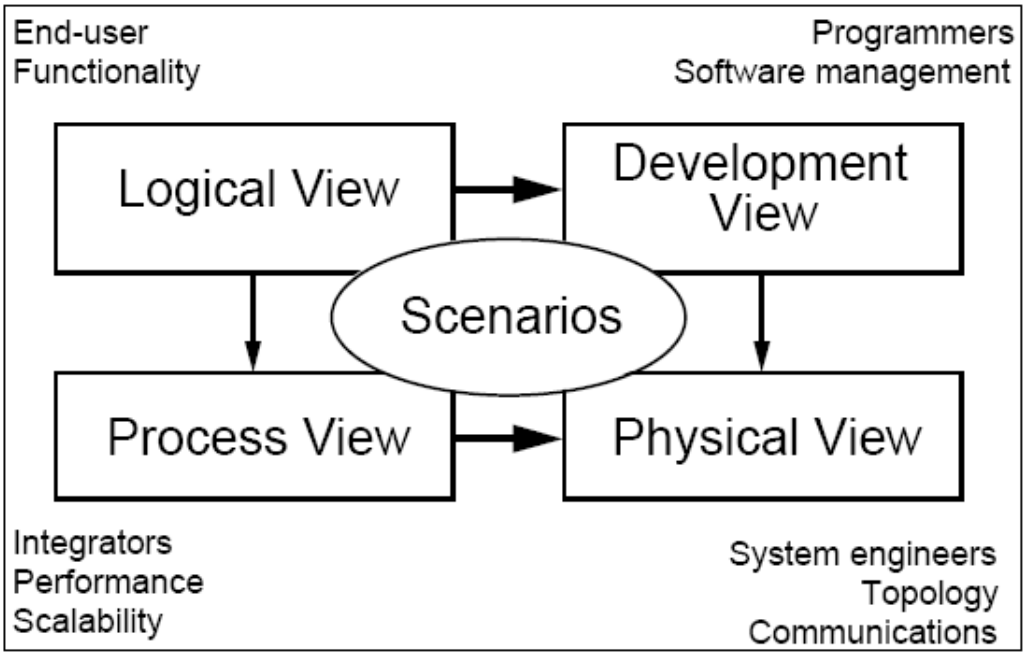


Figure 2.3. "4+1" views of software architecture (Kruchten, 1995)

Most ADLs support only functional/logic view of an architecture (in textual or graphical or both format), although their functional/logic views may vary (Medvidovic et al, 2000). The existing ADLs rarely support more views. The multi-view feature of software architecture is best demonstrated by the "4+1" view model (Kruchten, 1995). As is illustrated in Figure 2.3, the "4+1" view model describes the architecture of software-intensive systems by using multiple, concurrent views. The four views of the model are logical, development, process and physical view. In addition, selected use cases or scenarios are utilized to illustrate the architecture. The views of software architecture must be well-coordinated in their presentation, behavior, and data.

2.2 Software architecture performance evaluation

Software architecture design and performance evaluation have become crucial in the development of large scale systems (ECPerf, 2002; Gorton et al, 2000; Grundy and Cai et al, 2001). Validation of non-functional requirements is particularly critical. System performance is one of the most challenging requirements to validate (Gorton et al, 2000; Nimmagadda et al, 1999; Petriu et al, 2000). In general, there are three main types of performance evaluation approaches for software systems; and they are benchmarking, rapid prototyping, and simulation.

Benchmarking (Balsamo et al, 2002; Balzer, 1985; SPEC benchmarks, 2002) uses a fair and useful set of metrics to differentiate candidate systems. For example, SPECjAppServer2002 is a benchmark for measuring the performance of Java Enterprise Application Servers by using a subset of J2EE APIs in a web application (ECPerf&JDBC Benchmark, 2002). It gives Java users an objective and representative benchmark for measuring Enterprise JavaBeans (EJB) container in a J2EE 1.3 compatible server. Other J2EE-related benchmarks include SPECjbb, SPECjvm, SPECMail (SPEC benchmarks, 2002). Benchmarking technologies provide objective goals for the applications that are closely related to the benchmark applications. For example, the SPEC J2EE benchmark applications are only useful for J2EE-related technologies. Benchmark applications are not always available for new technologies of software development. Benchmarking technologies can not provide instant evaluation results or guidelines for designing new software products.

Rapid Prototyping (RP) (Hu et al, 1997) develops partial software applications to implement performance-critical parts of the code e.g. network-centric and database-intensive. The CSIRO

middleware technology evaluation project (CSIRO, 2000) uses RP to evaluate middleware technologies. RP is a simplified process of software development, and much effort is expected for even simple prototypes. Most rapid prototypes are manually built and are not flexible. Whenever the software design is changed, the prototypes also need to be changed (mostly manually).

Both benchmarking and RP are empirical approaches. They evaluate the whole or part of real software systems. Simulation Approaches obtain formal models (Markovian, 1986) of the intended distributed applications, and use the models to estimate the performance of the real applications (Markovian, 1986). The Simulation Approaches simulate performance instead of testing real software systems. Their accuracy varies widely, and it is difficult to obtain effective formal models for applications such as databases.

2.3 Web application reverse engineering

Reverse engineering is a process of analyzing a subject system to identify the system's components and their interrelationships, for the purpose of creating representations of the system in another form or at a higher level of abstraction (Chikofsky et al, 1990). The main objectives of web application reverse engineering include: pattern abstraction, re-documentation, and architecture recovery (Patel et al, 2007). Pattern abstraction is focused on improving the quality of web application source code. It involves activities such as analyzing web application source code; identifying instances of commonly used patterns and styles, and resulting in abstract representations of fragments of source code; and discovering human inspired concepts and then linking them to implemental concepts. Re-documentation is the process of generating accurate documentation from existing, undocumented software. Architecture Recovery aims for obtaining an understanding of the structural aspects of a system's architecture. WARE (Di Lucca et al, 2001) and Revangie (Draheim et al, 2005) demonstrate the main activities involved in web application reverse engineering.

WARE (Di Lucca et al, 2001) is an approach that extracts information of a web application, and abstracts documentation that describes the physical and conceptual structure of the application. Figure 2.4 illustrates the WARE's reverse engineering process. During extraction process, WARE analyzes the source code of web application components (client and server pages, script modules) statically as well as

dynamically, extracts the information that is needed for building up analysis models, and stores the extracted information in a relational database (Tramontana et al, 2002).

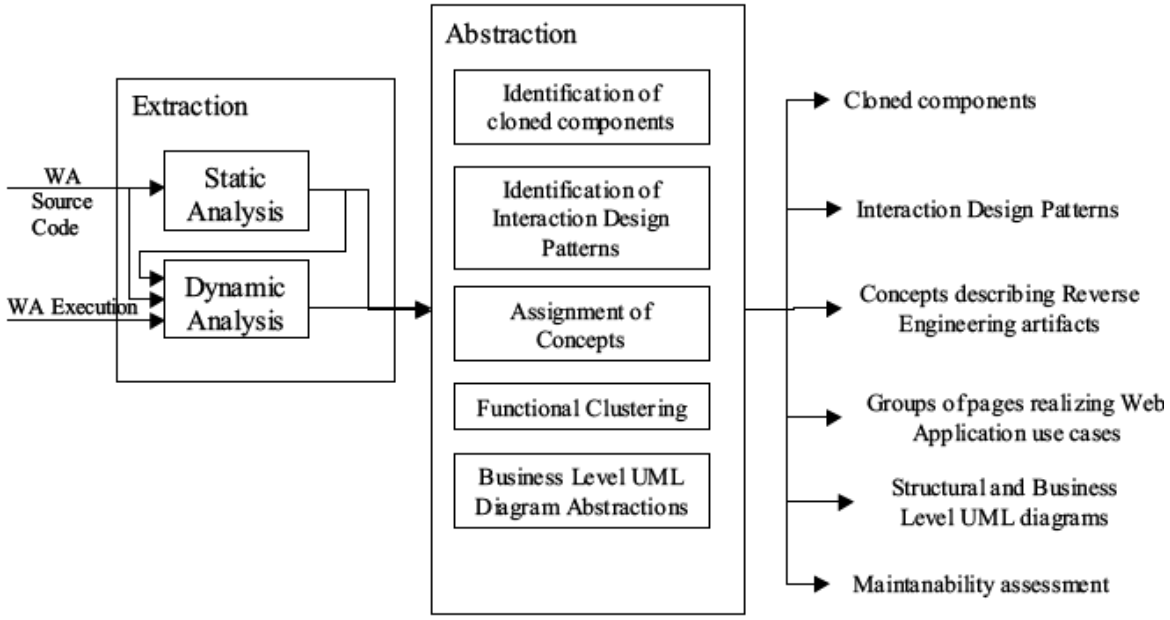


Figure 2.4. Web Application Reverse Engineering process (Tramontana et al, 2002)

The retrieved information is processed during the abstraction process. Clone analysis examines if there are cloned pages in the web application, and may suggest reengineering to eliminate page duplications. Web Application user interfaces need to be analyzed to identify Interaction Design Patterns. The existing concepts used in the web application need to be retrieved, analyzed, and exploited to support the assignment of concepts to reverse engineering artifacts. The pages of the web application need to be clustered in subsets; the subset cohesion must be maximized; and the coupling between the subsets must be minimized. In order to recover business level UML Diagrams, WARE build up attributes, methods and relationships of UML diagrams by analyzing the data a user inputs by a form, the data exchanged between Web Application pages, the data flow between the application and the databases. WARE can recover the information such as Cloned components, Interaction Design Patterns, and Concepts describing Reverse Engineering artifacts. The extracted and abstracted information is then used to evaluate some quality characteristics of web applications such as maintainability.

Revangie (Draheim et al, 2005) supports source code independent reverse engineering of dynamic websites. Revangie has three modes of operation to extract form-oriented information of a web application, including the crawl mode, snoop mode, and guide mode. The crawl mode operates automatically on the client side like an automated web browser. It uses an HTTP client to request pages, submit values, and analyze the trace of submitted values and visited pages. The snoop mode can either operate as a proxy server or a façade to the web server. It monitors the HTML communication of one or more users to collect data of actual sessions of a web application. The snoop mode is user-driven. The guide mode tries to combine the advantages of crawl and single-user snoop mode: automation and the possibility to enter form data manually. The goal of all three modes is to obtain information to build a form-oriented analysis model to describe the user interface of web applications.

The form-oriented user interface model is a typed, bipartite state machine. In the form-oriented model, one set of states represents client-side web pages, and the other set represents server actions that generate the pages. The recovered form-oriented models can be, for example, exploited for the purpose of requirements engineering and load test development (Draheim et al, 2005). Figure 2.5 shows a form-oriented model of a web application. The model starts at web page “login” that should contain a form for submitting user’s login information to server action “checkpw”. If the submitted information is invalid, the user is shown another instance of the “login” page; otherwise the user is forwarded to page “home”. The “home” page can be navigated to either the “buyCars” page or the “buyBikes” page. The “buyCars” and “buyBikes” pages represent all the possible web screens showing lists of cars and bikes respectively; and they both use the same action” buy” to process the submitted data (Draheim et al, 2005). The form-oriented model can be used to analyze realistic user behavior in web load testing.

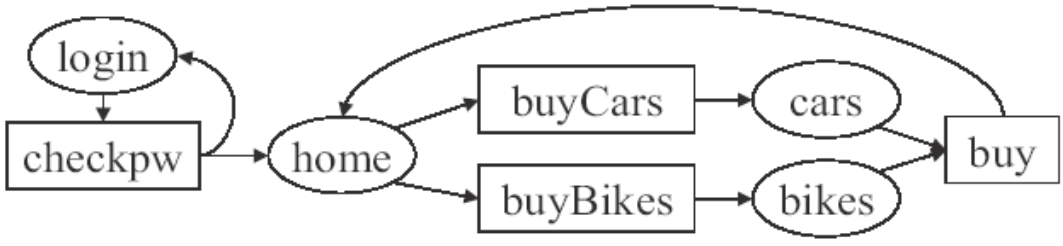


Figure 2.5. Form Chart model example (Draheim et al, 2005)

2.4 Web Application Load Testing

Web Application Load Testing (WALT) is an important part of web performance engineering. WALT measures response time, throughput, and availability of a target website from a client's perspective (usually a web browser). Load testing needs to be undertaken rigorously before a robust cost-effective website can be achieved (Mensace et al, 2002). A wide range of load testing tools (Apache JMeter, 1999; Cai et al, 2004; Draheim et al, 2006; Microsoft, 2002; Mensace et al, 2002; Subraya et al, 2000) and generic performance and reliability engineering tools (Denaro et al, 2004; Smith et al, 2005; Sprenkle et al, 2005) have been developed. The core functionality of these tools includes modelling client behaviour as well as constructing load testing plans. Almost all such tools support only a fairly basic model of client behaviour, which provides a sequence of requests on a website arranged into repeating groups, allows multiple threads to mimic large numbers of client browsers, and supports limited control logic depending on the website response. Some tools support parameterisation of loading tests to allow configuration of different test cases and test data, but the configuration is limited. To date, limited formal client loading models for web applications have been developed, but they rarely are used to generate web load testing plans (Draheim et al, 2006; Draheim et al, 2003; Subraya et al, 2000).

A load testing plan describes a series of steps a load testing will execute. For example, a complete JMeter test plan consists of: one or more Thread Groups, logic controllers, sample generating controllers, listeners, timers, assertions, and configuration elements (Apache JMeter, 1999). JMeter load testing plans can be complicated and tedious. Although the JMeter tool provides a user friendly GUI, it is still time consuming and error-prone to construct testing plans. To achieve accurate testing results, a load testing plan should model *realistic* behaviour of web users. Most existing load testing tools, focused on operational goals of a running loading test, do not support users to analyze if their plans can realistically capture the web user behaviour. For example, JMeter can support complicated testing plans, but it does not assist people to analyze if the plans realistically capture the user behaviour.

2.5 Model Driven Engineering

Model Driven Engineering (MDE) refers to a range of development approaches that use software modeling as a primary form of expression. The main principals of MDE include: 1) well-defined models, instead of third-generation languages (e.g. java, C++), are the main forms to express systems for enterprise-scale solutions; 2) the building of systems is developing a set of models at different layers.

The models are organized into an architectural framework, and can be transformed between each other (normally from high platform-independent models to lower platform-dependent models); and 3) the integration and transformation between models must be formally underpinned at meta-model level, and need to be done automatically through tools (OMG MDA, 2001).

The principles of MDE are best demonstrated by MDE's well-known incarnation – Model Driven Architecture (MDA) (OMG MDA, 2001). MDA is an architectural framework of the model-driven software development. It separates business and application logic from underlying platform technology by using OMG's (Object Management Group) established standards, including Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Enterprise Distributed Object Computing (EDOC), Software Process Engineering Metamodel (SPEM), and Common Warehouse Metamodel (CWM)). Through MDA, high level platform-independent models (must be UML models or other associated OMG modelling standards) can be realized on virtually any platforms, open or proprietary, including Web Services, .NET, CORBA, J2EE, and others. MDA allows business and technical aspects of an application or integrated system can each evolve at its own pace (OMG MDA, 2001).

MDE is a broader concept than MDA. On top of the focuses of MDA including: technical variability by making a difference between platform independent and platform dependent models, and defining transformations between these models, MDE is also focused on application-domain variability by adding modeling dimensions for subject areas (e.g. order entry, customer portal, back-end administration) and architectural aspects (e.g. data, presentation, security, business rules, workflows) (Hann, 2008).

2.5.1 Domain-specific modelling languages

Domain-Specific Modelling (DSM) is a core technology of MDE. It refers to constructing models by using concepts that represent things in the application domain, not concepts of a given programming language. Domain-Specific Modelling Languages (DSMLs) capture the domain abstractions and semantics, which allow developers to work directly with domain concepts. DSMLs use type systems to formalize the application structure, behaviour, and requirements within particular domains, such as software-defined radios (PrismTech, 2008), avionics mission computing (Balasubramanian et al, 2006), online financial services (Tolvanen, 2008), warehouse management (Deng et al, 2003), and the domain of middleware platforms (Grundy and Cai et al, 2001).

DSMLs use meta-models to capture the concepts of an application domain, define the relationships among concepts in the domain, and precisely specify the key semantics and constraints associated with these domain concepts. For example, SoftArch/MTE models the domain of middleware-based multi-tier online business (Grundy and Cai et al, 2001; Grundy and Cai et al, 2005). It uses a domain-specific meta-model to capture the domain-specific concepts, define how the concepts are connected, and specify semantics to check the validity of the captured domain-specific concepts.

2.5.2 Model transformation technologies

Model transformation is a core technology in the MDE paradigm. Model transformation is the process of converting one model to another. The OMG QVT standard (OMG QVT, 2001) for model transformation has been supported by a wide range of model transformation solutions, including ATL (ATLAS Transformation, 2006), XSLT (XSLT Transformation, 2001), VIATRA (Csertan et al, 2002). ATL is a model transformation language specified both as a meta-model and a textual concrete syntax. It is a hybrid of declarative and imperative. An ATL transformation program is composed of rules that define how the source model elements are matched and navigated to create and initialize the elements of the target models. XSLT, an XML-based language, is used for the transformation of XML documents into other XML or "human-readable" documents. An XSLT transformation style sheet contains XSLT program text (or ‘source code’ in other languages, such as HTML and Java). It describes a collection of template rules and other directives that guide the XSLT processor in the production of the output document. VIATRA is a model transformation-based framework supporting the systems designed using UML. The transformation language of VIATRA is consisted of graph patterns, graph transformation rules, and control structure. Graph Transformation (GT) is the primary means for elementary model transformation steps. GT provides a rule and pattern-based manipulation of graph based models. The application of a GT rule on a given model replaces an image of its precondition (left-hand side, LHS) pattern with an image of its post condition (right-hand side, RHS) pattern, and additional actions can be executed (Varr’o et al, 2003).

Both the ATL and XSLT transformation programs/scripts are textual, and deal with models through the models’ textual format (e.g. XML format). A VIATRA transformation program contains graphic (e.g. graph patterns) as well as textual information (e.g. textual rules, pattern-based manipulation language),

and deals with models through the models' graphic format. ATL and XSLT are focused on transforming models, while VIATRA explores broad range of issues related to model transformation, including: checking consistency, completeness, and dependability requirements.

2.5.3 Model integration technologies

Model integration combines the strength of various domain-specific software models to serve a more comprehensive target domain for operational, tactical, and strategic purposes. It means different things in different context, and roughly falls into two categories: "deep" integration and "functional" integration (Geoffrion, 1996).

Deep Integration (DI) produces a single new model that combines two or more given models. DI, also expressed in the literature as model merging, is to form a new model that must be represented in the same definitional formalism as the given models; the new model must semantically match the modeller's intentions as much as possible. The main concerned issues involved in DI include model comparison, difference highlighting and model merging. IBM's Rational Software Architect (IBM Rational, 2008) is a representative support tool for domain-specific model merging.

Functional Integration (FI), in contrast, does not yield a new model but leaves the given models as they were. FI superimposes a computational agenda for coordinating calculations over the involved models, typically directing certain models' outputs to other models' inputs while specifying the order of computations (Geoffrion, 1996). The main focuses of FI are: consolidating previously self-evolved domain modelling knowledge, supporting multi-model coordination and synchronization, and finding new uses for the existing domain modelling knowledge. FI has not been researched as intensively as deep integration. Most of FI research is done case by case without systematic and structured support.

2.5.4 Semantics representation and checking

Semantics of a model is consisted of semantic domain and semantic mapping. A semantic domain provides a set of expressions that have well defined meaning and behaviors. Semantic mapping refers to a process that maps abstract syntax to the semantic domain (Chen et al, 2004). Figure 2.6 illustrates the relationships among abstract syntax, concrete syntax, and semantic domain.

Explicit, formalized semantics of models can help users to correctly represent their meaning in the models, and allow models to be read and understood by machines. Some modelling languages (e.g. Web Ontology Language (W3C, 2004) and Web Service Modelling Languages (W3C, 2005)) have explicit, formal semantics, as well as well-formed abstract syntax. But for many other modelling languages, their semantics is often implicitly defined by its model interpreter, and needs to be explained by documents written in a natural language (Grundy and Cai et al, 2001; UML, 1996). For example, the UML provides informal descriptions and insights into the semantics through its abstract syntax. Although the implicit UML semantics is far from being satisfactory and needs to be improved, the UML document (UML, 1996) is practical, and can provide sufficient information for experienced users to gain knowledge about the meaning of the constructs of UML. SoftArch/MTE also does not have explicit, formal semantics. SoftArch/MTE has a fairly small semantic domain, as well as a small set of abstract syntax. It is practical to introduce its semantics in natural language, and explains semantics together with abstract syntax, which reduces the learning load brought by separate, formal semantics descriptions.

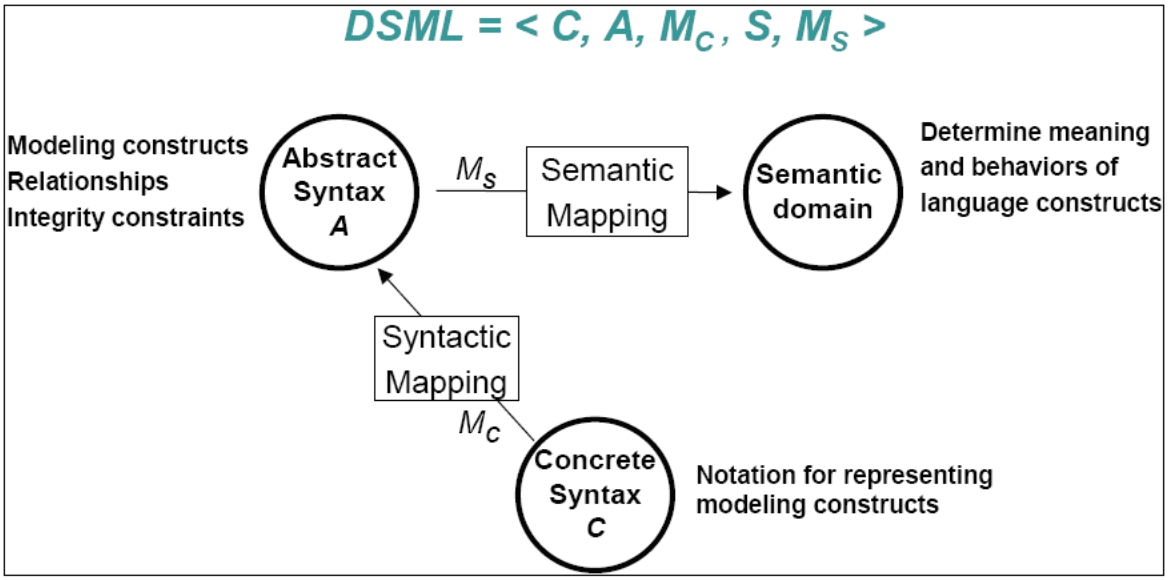


Figure 2.6. Relationships between syntax and semantics (Chen et al, 2004)

Semantic Consistency needs to be maintained during Model Integration and Transformation (MI&T). Most existing MI&T technologies are focused on achieving operational goals and provide little support to maintain semantic consistency during MI&T. However, consistency maintenance has been argued as the essential basis for Model Driven Architecture (MDA) or more generally for Model Driven

Engineering (MDE) (Pieter Van et al, 2005; Graaf et al, 2007). Consistency maintenance allows modellers to specify: what to transform and integrate, what the conditions are for an intended transformation and integration, and what the expected results are after the transformation and integration.

Semantic Consistency provides requirements for software modellers to analyze and design an intended MI&T. Without semantic consistency requirements, software modellers can only produce rules and templates driven by ad-hoc operational goals and based on personal experiences. Without appropriate analysis and design, MI&T may cause unnecessary semantic inconsistencies.

Currently, consistency maintenance is mainly done through consistent transformation or consistency comparison (Egyed, 2001). Consistent transformation ensures consistency via well-defined transformation steps where source models are transformed into target models in a manner that guarantees consistency (Engels et al, 2002). Consistency comparison is done after transformation. Sample consistency comparison approaches include VisualSpecs (Cheng et al, 1994), JViews (Grundy et al, 1998), and ViewIntegra (Egyed, 2001_2). Both VisualSpecs and JViews convert graphical models into either a formal language (VisualSpecs) or a data repository (JViews) in which they perform consistency analyses. ViewIntegra rewrites (through reverse modelling or model generation) the source (target) model into the target (source) domain, and then compares both versions of target models (source models).

2.5.5 Multi-view Support Software Engineering Environments

Multi-view Support Software Engineering Environments (MSSEEs) allow people to develop software products with different concerns, at various abstraction levels, and based on different domain-specific knowledge. Popular MSSEEs include Rational Rose (IBM, 2001) and Eclipse (Eclipse 2001). Although different multi-view systems have different foci (Rational Rose is OO analysis and design oriented, Eclipse is tool development oriented), they share the common issues: 1) data consistency management; 2) view synchronization; and 3) event propagation management. Those issues have been widely researched; and the representative work includes MViews (Grundy et al, 1996), VIATRA2 (Kocsis et al, 2006), FUJABA (Fujaba, 2007), and Spearmint (Ulrike Becker-Kornstaedt et al, 1999). MViews uses a central repository to hold a base model; and the base model can then be represented in different views by

different visual notations. VIATRA2 also keeps a base model, although the base model evolves in different views. FUJABA uses the same model to maintain the integrity of data that is represented differently in different views. Most of the multi-view approaches use low-level (source code level or database level) common data repositories, which normally need to be updated at programming level whenever new domain-specific knowledge (new target domains) need to be supported. In those approaches, synchronization is normally implemented by low level code-bound event-handling systems.

2.6 Summary

Software architecture modelling is aimed at raising the abstraction level of software development. A software architecture design provides far more analysis and design support for software development than intuitive "box-and-line" diagrams. Each of the reviewed ADLs addresses certain concerns at architectural level, and allows software architects to model well-defined software architecture, which, in turn, can provide well-structured guidance for the other stages of Software Development Life Cycle. Software architecture performance evaluation is focused on improving the performance of architecture design. Architecture designs with rigorous performance evaluation are highly desirable and influential on the quality of the final software systems. The Argo/MTE project of the thesis supports software architecture modelling and performance evaluation in a different way from benchmarking, rapid prototyping, and simulation. It models software architecture of an intended software system, and generates fully functional architecture level test bed of the system. Argo/MTE improves the integration between software architecture modelling and the other UML modelling.

Web reverse engineering is aimed for improving the existing web applications to achieve better source code structure, better scalability and performance, and better maintainability. The reviewed web reverse engineering technologies explain the main concerns and show the solutions in this area. The MaramaMTE+ project of the thesis focuses on web load testing modelling and testing plan generation. The project captures realistic client behaviour through form chart modelling, automatically synthesizes structure of form chart model, and generates testing plans for a third party tool (e.g. JMeter).

Model Driven Engineering (MDE) is a big paradigm where models, instead of third-generation languages, are the main form of expressions in software development. MDE is a sensible context to host active research areas including domain-specific modelling languages, model transformation, model

integration, and model-transformation-based software tools. The problems found in those research areas motivate the research of the MaramaCRelation project. The MaramaCRelation project of the thesis provides analysis and design support for MI&T, construction of multi-view support software systems, and consistency maintenance. The MaramaCRelation approach is intended to provide better solutions for some of the main issues involved in MI&T.

Chapter 3 - Thesis Motivation

The SoftArch/MTE project, done as part of the author's Masters research, provides an approach for software architecture modeling and performance evaluation. It allows users to model software architecture of a middleware-based web application; generates a fully functional performance test bed from the software architecture model with client and server code, database configuration and deployment scripts; and automates the performance evaluation process of test bed generation, compilation, and deployment and performance metric result collection. This chapter presents an overview of the SoftArch/MTE performance evaluation technology, and identifies a number of challenges that motivated the research in this thesis. Specifically, the challenges include: the need to use a well-established CASE tool platform (e.g. ArgoUML, Eclipse); the need to better capture architecture patterns and models for reuse; the need to better capture user interaction with web applications; the need to develop a higher level of model integration and transformation support; and the need to better support traceability and consistency management in multi-view tools.

3.1 SoftArch/MTE target domain

SoftArch/MTE is aimed for modeling and evaluating the architecture of middleware-based web applications (Grundy and Cai et al, 2001; Grundy and Cai et al, 2005). A SoftArch/MTE software architecture model abstracts main-stream middleware technologies to architectural level concepts, and provides essential information to generate a fully functional test bed (middleware-based web applications). A test bed carries performance metric information to measure the performance of the interested part(s) of the software architecture. SoftArch/MTE allows software architects to compare performance of available middleware technologies, and generate the best-of-practice functional prototype for the intended web applications.

Consider the development of an on-line video shop (Grundy and Hosking, 2000), where the main tasks include: supporting an on-line video store library; supporting customer on-line video search/reservation; and supporting staff in-store video rental management tasks. Some example interfaces for such a system are illustrated in Figure 3.1(a). One candidate architecture design for the system is shown in Figure 3.1(b). In this example, video store staffs use desktop applications connecting to the database(s). Customers interact with user interfaces that connect to application servers, which, in turn, are connecting to possibly other servers and one or more databases e.g. holding staff, customer, video, and video rental details. Data processing may be centralized or spread across clients or servers. Middleware may be Java RMI, DCOM, CORBA, or J2EE. Server objects may be COM, CORBA or Enterprise Java Beans. Data management may use relational, object or XML databases, or files.

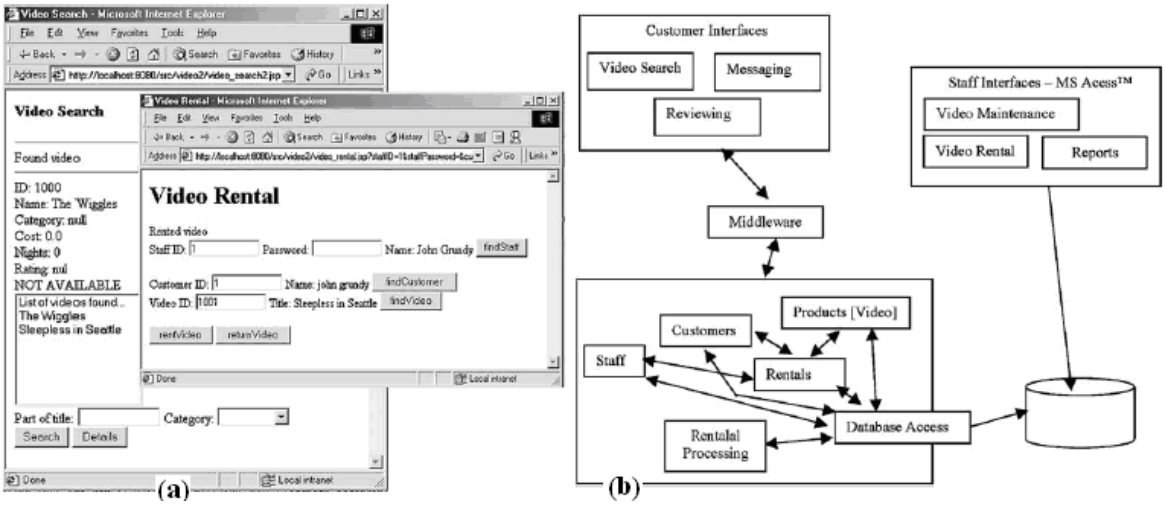


Figure 3.1. Parts of a simple on-line video system (Grundy and Hosking, 2000)

A software architect typically has some performance constraints that any chosen architecture design must meet. Such constraints might include maximum number of users, and response time for different user requests and data processing services. Further constraints may include hardware and software constraints e.g. must run on Windows/LINUX machines; must run on low-end desktop machine; must run over 56kbps modem connection; and must use either CORBA or DCOM middleware protocol, must use SQL Server™ 7 database.

To determine a suitable architecture for a system, including appropriate middleware and database choices, an architect typically relies on the past experience. SoftArch/MTE generates performance test-beds (essentially rapid prototypes) directly from software architecture descriptions/models; and it enables architects to quickly and iteratively understand the performance impacts of their architecture-level design decisions.

3.2 SoftArch/MTE overview

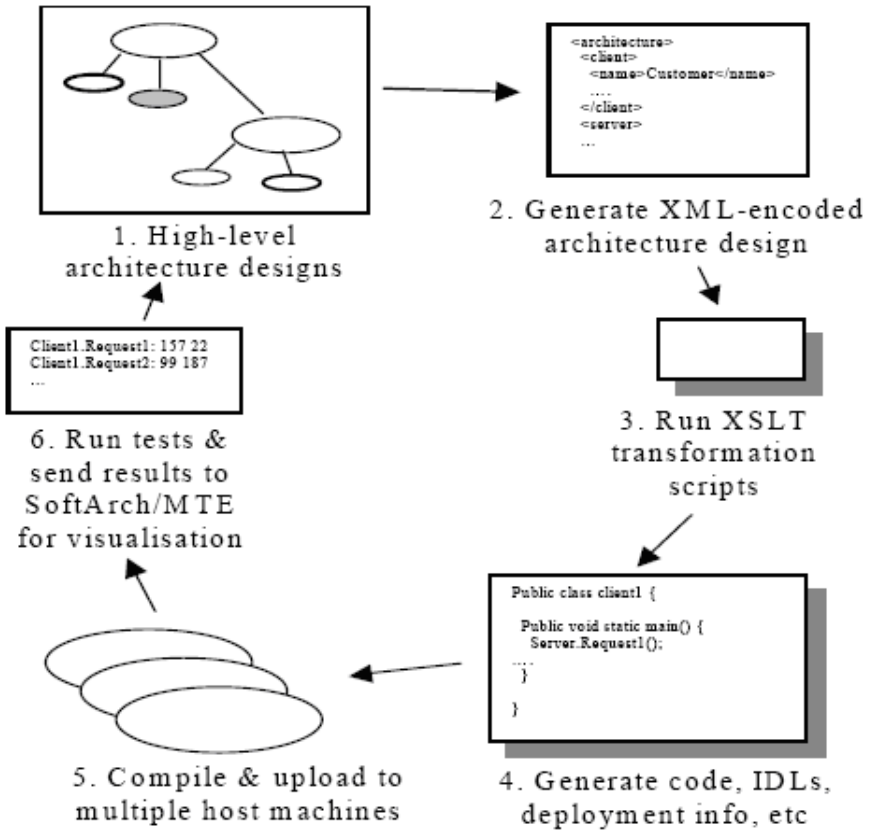


Figure 3.2. Outline of the SoftArch/MTE architecture performance analysis process (Grundy and Cai et al, 2001)

Figure 3.2 outlines the process whereby software architects obtain performance results from test beds (real code) generated by SoftArch/MTE. Steps 2-6 are fully automated. The architect first constructs a high-level architecture design that specifies architectural components and connectors such as: clients, servers, remote server objects and database tables, client-server, server-server, client/server-database

requests and server services, and various kinds of connectors between these architectural abstractions (e.g. belongs-to, runs-on, network connection) (1). The components and connectors specify various properties: client, server and database host machine; number and frequency of requests (e.g. 1000 times; continuous; every 0.25 seconds; etc); database table and request complexity (e.g. one row select; 100 row select/update; one row insert/delete etc); and middleware protocol (e.g. CORBA using Visibroker 4.0; TCP/IP socket using textual XML document; etc). The abstraction types of those components and connectors are specified in extensible SoftArch/MTE meta-models. The architect instructs SoftArch/MTE to generate an XML encoding of the architecture model (2). The XML-encoded architecture model is then passed through a number of XSLT (XML style sheet transformations) scripts (3), which generate Java, C++, Delphi etc code, along with CORBA and COM IDL files, EJB deployment descriptors, database table creation and population scripts, and compilation and start-up scripts (4). This generated code is a fully working performance test bed.

The compiled (deployable) client and server program code is then uploaded to the specified client, server, and database host machines; all the host machines run a SoftArch/MTE deployment agent and can receive the test bed code. The deployed client and server programs and appropriate database servers are started on all hosts. The clients wait for a SoftArch/MTE signal (via their deployment agent), or a scheduled start time, to begin execution i.e. sending requests to servers. Once the tests complete, the deployment agents collect results (usually from client and server program output files) and send them to SoftArch/MTE (6). SoftArch/MTE annotates architecture diagrams in various ways to highlight the performance measures captured from running the generated test beds. SoftArch/MTE can also generate performance summary analysis reports and invoke a 3rd party data visualization tool to show performance details and summary charts (we use MS Excel™ to do this). Multiple test run results using different middleware, databases and client/server request can be visualized together. Architecture designs and their performance results can also be versioned within SoftArch/MTE; users can compare the performance results of the different architecture design options (Grundy and Cai et al, 2005).

3.3 SoftArch/MTE meta-model

SoftArch/MTE uses domain-specific meta-models to define abstractions of domain-specific knowledge. Figure 3.3 shows a sample SoftArch/MTE meta-model for e-commerce applications. In the meta-model, a software architect specializing in e-commerce system domain defines abstractions (basic parts,

components, or modeling types) of e-commerce systems including: *Client*, *Request*, *AppServer*, *RemoteObj*, *RemoteService*, *DatabaseServer*, and *Database*. Relationships between abstractions also need to be defined. Each abstraction's characters and behaviors are defined by a set of properties as well as the abstraction's relationships with other abstractions.

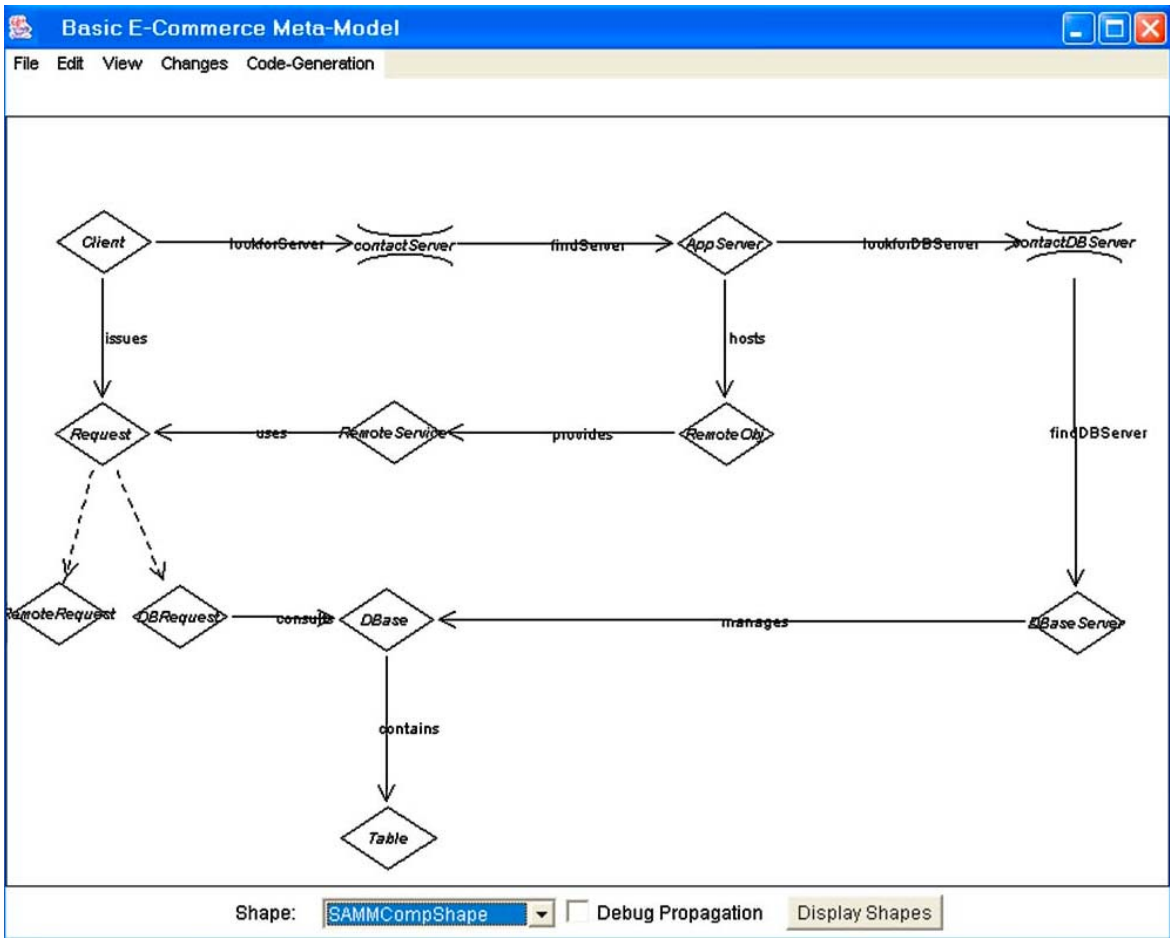


Figure 3.3. A sample SoftArch/MTE meta-model for e-commerce applications (Grundy and Cai et al, 2001)

A number of properties for each of these architectural abstractions need to be designed. Figure 3.4 illustrates the properties of the abstractions in Figure 3.3. The properties may be structural properties e.g. names, middleware types used, and roles. The structural properties are denoted with "AP" (Architectural Parameter) in Figure 3.4. The other properties are related to performance evaluation e.g. number of times to call a server request, number of concurrent threads to create for a client or server, pause duration (if any) between making requests, number of rows and columns a database table has, number of

rows expected to be returned or updated by a database query, and number of arguments a server request expects. These properties are denoted with "CG" (Code Generation) in Figure 3.4. A property may have single value, multiple values, or expressions. Both structural and performance evaluation properties are used by the SoftArch/MTE code generator for the generation of test-bed code and scripts.

Element Type	Main Attributes	Property Description
Client	ClientType (AP, CG) Threads(CG)	Type of a client. For example, a client can be a browser, a CORBA client, or a RMI client. Number of clients that will be running during performance testing.
RemoteRequest	RemoteServer (AP, CG) RemoteObject(AP, CG) RemoteMethod(AP, CG) RecordTime(CG) TimesToCall(CG)	The name of remote server, to which the remote request is launched The name of remote object, which provides services required by the remote request The name of remote service, which completes the remote request A switch/boolean that defines if the performance testing results are recorded or not. Repetition time of certain operations during performance testing
DBRequest	QueryType (AP, CG) Dbase(AP, CG) Table(AP, CG) TimesToCall(CG) RowsReturned(CG) RecordTime(CG) Caching	Type of query for test bed and performance testing, such as 'select', 'update', 'insert', etc. The name of queried database The name of queried table Repetition times of certain operations during performance testing Number of returned results of the database request A switch/boolean that defines if the performance testing results are recorded or not. A switch/boolean that defines if query results are cached or not
AppServer	RemoteObjs (AP, CG) Type (AP, CG)	Names of all objects this application server hosts Type of the application server, such as CORBA, RMI, and J2EE……
RemoteObj	Type (AP, CG)	Type of the remote object. Type could be CORBA, RMI, and EntityBean……
RemoteService	Arguments (CG) Threading (CG) ConcurrencyControl (CG) RecordingTime (CG)	Arguments used by the service/operation A Boolean that records if this service symbols multi-threads character or single-thread character A Boolean parameter that records if the service has concurrency control or not A switch/boolean that defines if the performance testing results are recorded or not.
DBServer	Dbases (AP) ServerType (AP, CG)	Name of all databases this database server hosts Type of database server, such as MSSQL, Cloudscape, and ORACLE……
DBase	Name (AP, CG) Type (AP, CG)	Database name Type of database server, such as MSSQL, Cloudscape, and ORACLE……
Table	Name (AP, CG) PrimaryKey (CG) Rows (CG) Columns (CG)	Table name The primary key of the table Number of rows of data expected in table Number of columns expected in table

Figure 3.4. Sample SoftArch/MTE meta-model abstractions and properties (Grundy and Cai et al, 2001)

3.4 SoftArch/MTE architecture model

A SoftArch/MTE architecture model is an instance model of a SoftArch/MTE meta-model. An architect is able to model architectures using the abstractions of a domain-specific meta-model. The SoftArch/MTE code generator must understand the used meta-model. An architect can parameterize software architecture designs with available component and connector properties. For example, the

architect may specify a method call between client request and server service is implemented by CORBA and 100 calls are to be made; and having code generated to implement this in their test bed. The architect may then change this to RMI and 250 calls for generation of a new test bed and subsequent test run. The architect can easily define a more specialized kind of architectural component or provide further characteristics about a component in a domain-specific meta-model, but the test bed code generation scripts need to be modified if this is done.

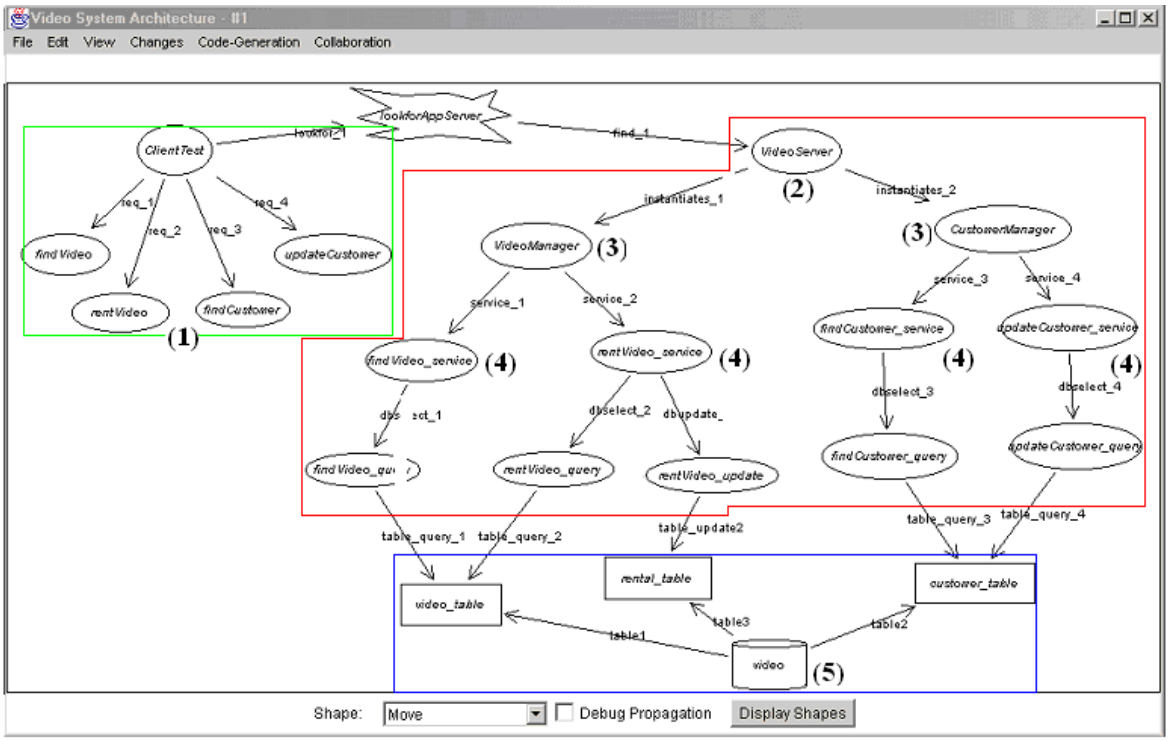


Figure 3.5. Sample high-level distributed on-line video software architecture (Grundy and Cai et al, 2005)

Figure 3.5 shows an example 3-tier architecture design for part of the on-line video system. The design uses the SoftArch/MTE meta-model displayed in Figure 3.3. In this example, staff and customer client programs have a number of requests that they can make on remote services e.g. find video/customer/rental, add/update rental item, and update customer details (1). The requests can be simple (one remote call) as well as complicated (involving several remote requests). For each client, the architect can specify its property values, such as assigning a value to the property "Threads" to simulate the number of the video shop users. Similarly, for each client request, the architect can specify a number

of times to call the remote service(s) and time to pause (if any) between invocations. This information configures the server loading tests, which will be run by SoftArch/MTE's generated performance test-beds. The server side components include a video application server (2) and its remote objects (3). The remote objects provide services (4) that retrieve appropriate data from the video database (5) for client requests. The video application server can be CORBA, RMI, or J2EE. The database can be MS Access, Oracle, or Microsoft SQL server. The architecture model in Figure 3.5 can generate CORBA, RMI, or J2EE applications that use MS Access, Oracle, or Microsoft SQL databases.

3.5 SoftArch/MTE performance evaluation

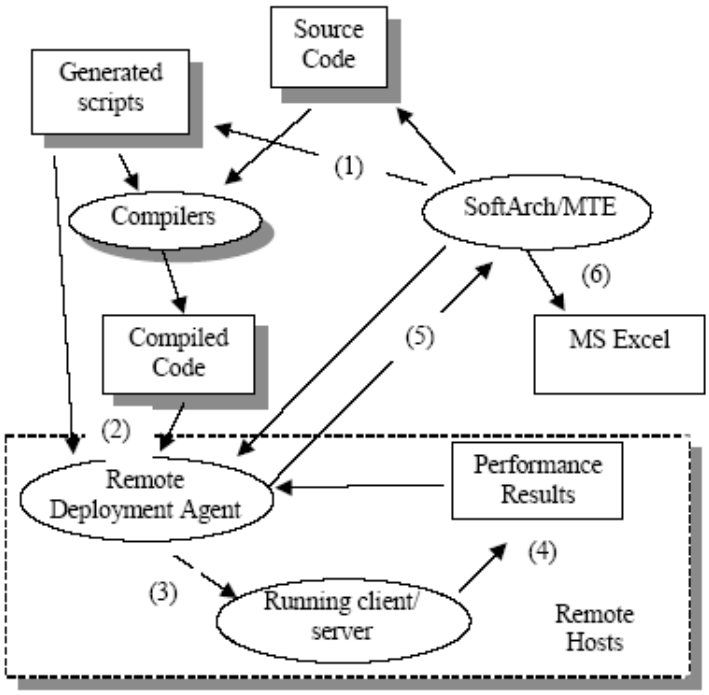


Figure 3.6. System deployment and test run process (Grundy and Cai et al, 2001)

SoftArch/MTE architecture models also generate DOS batch files to coordinate the tedious and error-prone process of test bed compilation, deployment, execution, and result collecting. Figure 3.6 outlines the SoftArch/MTE performance testing process. A generated test bed is compiled by SoftArch/MTE, using generated compilation scripts (1). The compiled code/IDLs/deployment descriptors, together with the scripts to configure and deploy these on a host, are up-loaded to remote client and server hosts using the remote SoftArch/MTE deployment agents (2). On each host machine, the deployment agent

organizes the uploaded code and associated scripts into suitable directory structures. It runs the scripts to properly configure the host machine, its database and registries, and the deployed test-bed code. The client and server programs are then run: CORBA, RMI and other server programs are started; EJB, JSP and ASP components are deployed into J2EE and IIS servers; database servers started and database table initialization scripts run; and finally clients are started (3). The clients look up their servers and then await SoftArch/MTE sending a signal (via their deployment agent) to run, or may start execution at a specified time. Clients send servers requests, logging performance timing for different requests to a file (4). Servers like-wise log the time taken to execute their remote methods and database operations. Performance results are currently collected in comma-separated value text files. These results are sent back to SoftArch/MTE after tests have completed (5). SoftArch/MTE collects the test results, and aggregates them using simple data processing algorithms to form a unified result set. The results are associated with SoftArch/MTE architecture model instances using data annotation facilities built into the SoftArch modelling tool's repository. SoftArch/MTE then uses data visualization techniques (e.g. 3rd party tools like MSExcel™) to display the results of the performance tests to architects (6).

3.6 Problems with SoftArch/MTE

A number of problems were identified during the development of SoftArch/MTE. These include: using proprietary tool platform; limited support for reuse of meta-models and code generation scripts; lack of analysis of user interaction with systems; limited multi-view integration; limited model integration and model transformation; and low-level implementation of cross-view traceability, constraint checking and consistency management. The main problems that motivated the three projects of the thesis are introduced as follows:

1) Using proprietary tool platform

The SoftArch/MTE tool is proprietary. It uses JVIews (Grundy et al, 1998), a multi-view support programming framework, to implement back-end model, model representation, model persistency, XML processing, user interface, and so on. The user interface and modeling style of the SoftArch/MTE tool may cause huge learning load for tool users. The initial feedback of SoftArch/MTE users showed that it was highly desirable to transplant the automatic performance evaluation technology to well-established tool platforms (e.g. ArgoUML, Rational Rose, and Eclipse). Leveraging well-established platforms can improve the quality of the SoftArch/MTE performance evaluation technology from both users' and tool developers' perspective. This desire motivated the Argo/MTE and MaramaMTE+ projects, where the

performance evaluation technology was ported to ArgoUML platform (in the Argo/MTE project) and Eclipse platform (in the MaramaMTE and MaramaMTE+ projects) respectively. Argo/MTE, MaramaMTE, and MaramaMTE+ have hugely improved the quality of the performance evaluation technology in terms of user interface, model exchange, code generation flexibility and reusability, domain-specific knowledge integration and transformation, and tool maintainability.

2) Lack of the support for user behavior modeling

SoftArch/MTE is focused on modeling and evaluating the server-side parts of a web application. It has very limited capability to model how clients (e.g. web users) interact with software systems (e.g. web applications). This motivated the MaramaMTE (a project done by John et al. but closely based on SoftArch/MTE) project that integrates architecture modeling with Form Chart modeling. Form Chart models are used to specify web user behavior and generate realistic load testing plans. The load testing plans can then run against the test bed of other parts (e.g. application servers and database servers) of the web application software architecture design, or run against a running legacy web application. The MaramaMTE project extends the applicable scope of the SoftArch/MTE performance evaluation technology; and motivated the MaramaMTE+ project to synthesize Form Chart models through web crawling.

3) Lack of support for model refinement

SoftArch/MTE generates java code directly from architecture models without showing the refining process, which makes the code generation very obscure, hard to understand, and hard to modify. A well-structured refining process is highly desirable to: improve the flexibility and maintainability of code generation; improve the chance of leveraging the strength of other modeling technology (e.g. Object Oriented Design); and support transformation between architecture models and their test beds. The problem of lack of a refining process was further identified in the development of Argo/MTE, MaramaMTE, and MaramaMTE+, which motivated the MaramaCRelation project. The MaramaCRelation project supports model refinement through model integration and transformation. The MaramaCRelation project provides a high-level support for model integration and transformation, which, in turn, helps to construct flexible, extensible model refinement.

A sound refinement mechanism also requires the support for traceability, behavior synchronization, and semantic consistency between the models involved in a refining process. SoftArch/MTE's one-off code

generation process does not support traceability, behavior synchronization, and semantics consistency management between architecture models and their test beds. SoftArch/MTE's poor support for traceability, behavior synchronization, and consistency management during code generation was further identified in the development of Argo/MTE, MaramaMTE, and MaramaMTE+, which motivated MaramaCRelation to specify conceptual model relationships, and support traceability and consistency management between related models.

3.7 Summary

The SoftArch/MTE research provides a high-level, extensible architectural modeling language. It encodes architecture designs in XML; it uses a set of extensible XSLT transformations scripts to transform the XML-encoded architecture design into test bed client and server programs as well as compilation/deployment scripts. A deployment agent running on available client and server hosts is used to automatically upload compiled systems and to configure and deploy them. Test runs are performed, and the performance results are automatically captured and aggregated by the SoftArch/MTE tool. These results are visualized by either the annotations in the architecture design diagrams within the SoftArch/MTE tool, or by a 3rd party application like MS Excel™. The SoftArch/MTE tool has been used to model a number of distributed systems, generate performance test-beds for these models; capture results and compare the performance measures obtained from hand-implemented, completed distributed systems. The initial results have demonstrated that SoftArch/MTE technology provides a useful, accurate automated architecture performance analysis approach for these distributed systems. The problems found in the development of the SoftArch/MTE technology are generic in the areas of software architecture modeling and software architecture performance evaluation; and they need systematic solutions. Those problems motivated the research of projects Argo/MTE, MaramaMTE+, and MaramaCRelation. Argo/MTE redeveloped the technology of software architecture modeling and performance evaluation in the well-established ArgoUML environment. It has improved the original SoftArch/MTE technology in many ways. MaramaMTE redeveloped SoftArch/MTE technology in Eclipse environment by using the Marama meta-tool (Marama meta-tool, 2007), which hugely improved the chance of model and tool integration. The MaramaMTE project is extended by the MaramaMTE+ project that provides automatic support to synthesize form-chart models, and generation of load testing plans. The MaramaCRelation project is aimed for providing high level support for model integration and transformation, which, in turn, supports structured, flexible model refinement.

Chapter 4 - Argo/MTE Performance Engineering Tool

The research of Argo/MTE extends the SoftArch/MTE research. Argo/MTE is aimed for industrial usage of the SoftArch/MTE performance evaluation technology by redeveloping and improving the technology in the well-established ArgoUML CASE tool (ArgoUML, 2003). This chapter identifies a range of problems that arose from scaling the SoftArch/MTE research on performance test bed generation; presents approaches used to solve these problems in Argo/MTE; reports on deployment and evaluation of Argo/MTE architecture designs; and discusses effectiveness of the used solutions.

4.1 SoftArch/MTE motivating Argo/MTE

After the initial success, SoftArch/MTE has been used on several industrial projects where software architecture becomes complicated. NetPay (Dai et al, 2007; Cai et al, 2004) is one of the tested large size software systems.

4.1.1 Sample target project – NetPay

NetPay is an on-line micro-payment software system (Dai et al, 2007). By using the NetPay system, a customer buys a collection of "e-coins" (virtual money) from a broker; and the coins are cached in an "e-wallet" (virtual wallet) on the customer's machine. The customer, when buying many small-cost items from a vendor, pays for these transparently by passing the e-coin information to the vendor. Periodically the vendor redeems the e-coins with the broker for "real" money. E-coins can be transparently exchanged between vendors when the customer moves to another site (Dai et al, 2002).

Figure 4.1 shows a component-based architecture of the NetPay system. When developing such software, architects must be able to model architecture, including clients, servers, machines, networks, protocols, caching, databases, messages, and user interfaces. The architecture needs to be specified in various levels of detail, from overview, refining into successively more detailed designs. Architects would be

interested in getting support to gauge likely design performance, even from early, high-level designs (Grundy and Cai et al, 2001). The NetPay architecture in Figure 4.1 will be explained in more details in chapter 5.

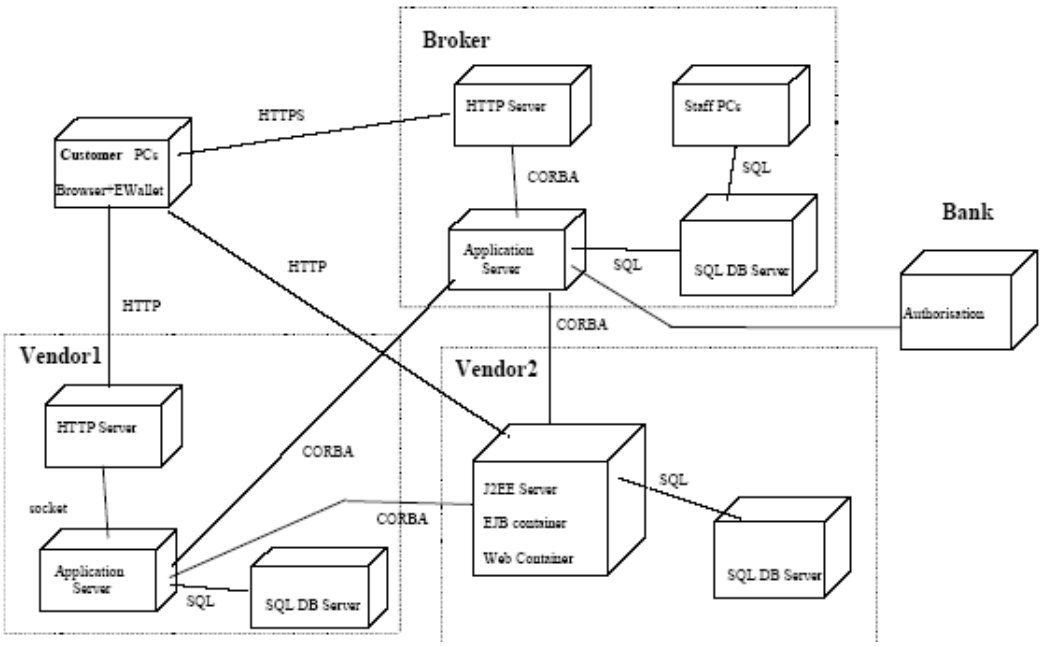


Figure 4.1. NetPay micro-payment system architecture (Cai et al, 2004)

4.1.2 SoftArch/MTE deficiencies

In the SoftArch/MTE research, the technology of test bed generation and performance evaluation is tied up with the SoftArch/MTE tool. The experiences and attempts to use the SoftArch/MTE performance evaluation technology on industrial complicated projects such as NetPay revealed major deficiencies, including:

- *Non-standard design tool and modeling notation*

  The SoftArch/MTE tool is an experimental proof-of-concept tool. While it is proved suitable for experimenting with the concepts of test bed generation and performance evaluation, the SoftArch/MTE tool has poor usability, which causes heavy learning load for tool users. It also has limited integration with other CASE tools, which makes it hard to leverage the strength of other well-established modeling technologies.

- *Proprietary XML architecture model format*

The SoftArch/MTE tool saves model designs in an ad-hoc XML format developed only for in-house experimental work. This makes it difficult to exchange architecture designs with other tools. The SoftArch/MTE architecture model ad-hoc XML format and test bed generation scripts (XSLT scripts) are also excessively tangled together, which makes code generation very inflexible.

- *Poor support for evolvement of code generation*

SoftArch/MTE hard-codes monolithic control logic for a domain-specific meta-model to co-ordinate the processes of code generation, compilation, deployment, and result capture and visualization (steps 3-6 in Figure 3.2). When the domain-specific meta-model evolves to support new middleware technologies (e.g. JSP and ASP web server components, web services WSDL descriptions and deployment scripts), the SoftArch/MTE tool does not provide structured support for tool developers to evolve the meta-model's code generation scripts and control logic.

- *Proprietary test bed deployment tool*

SoftArch/MTE has its own Java deployment tool to package test bed components, deploy components, unpack deployed components (if it is necessary), run test bed, and collect evaluation results. SoftArch/MTE generates DOS batch files from architecture models; and the deployment tool uses the DOS batch files to control the evaluation process. The proprietary deployment tool proved to be too difficult to adapt to different deployment environments, and it lacked robust fundamental code deployment and test control facilities.

4.2 An Overview of Argo/MTE usage

Argo/MTE is designed to solve above problems. Argo/MTE is aimed to become an "industrial strength" performance test bed generation tool by extending a well-established CASE tool – ArgoUML. ArgoUML is open source; well-structured and extendable at both diagramming and modeling levels; it uses common data representation standards such as XMI; and its cognitive support could be used to provide architecture design process support (Robbings et al, 1999; Robbings et al, 1998).

Figure 4.2 provides an overview of the Argo/MTE architecture and its usage. Multiple Argo/MTE domain-specific meta-models can be defined using a new Argo/MTE meta-tool (an ArgoUML tool

extension based on the SoftArch/MTE meta-tool), each providing a different set of architecture modeling abstractions and code generators (1). These meta-model abstractions are stored using an extended form of ArgoUML's Meta-data Interchange (XMI). Argo/MTE allows tool users to draw, modify, refine, and revise software architecture designs, again using a new architecture modeling tool (an ArgoUML tool extension based on the SoftArch/MTE architecture design tool) (2). Architecture models are developed using one or more meta-models and multiple design views. Each Argo/MTE design encodes essential data to generate a test bed (a distributed software system) for a given level of abstraction. The test bed not only implements fundamental functional requirements of the intended system, but also carries performance evaluation information. Meta-models and architecture models are saved in an extended XMI format (3).

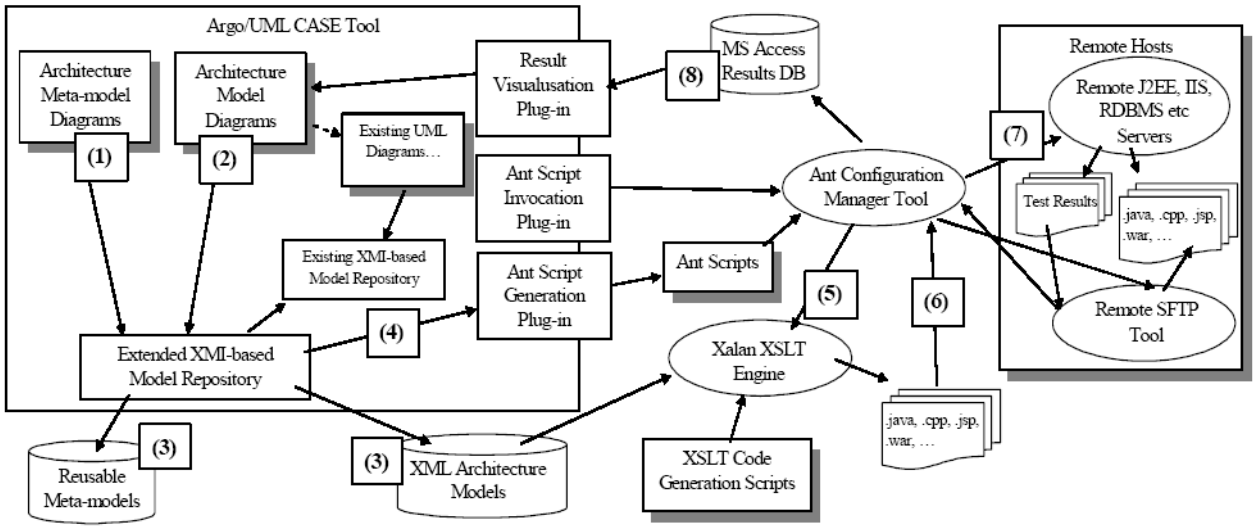


Figure 4.2. Overview of Argo/MTE architecture (Cai et al, 2004)

In addition to the generation of a test bed, an architecture model also generates a set of Ant (Apache Ant, 2004) configuration management tool scripts to perform the test bed's compilation, deployment, test initiation and results capture (4). The XMI-encoded software architecture model is transformed into a range of Ant-related files and scripts (5). A set of XSLT scripts and the Xalan (Apache Xalan, 2004) XSLT engine are needed to perform this work. The functional test bed code is compiled and deployed to multiple host machines (6). Performance tests are then run producing text files capturing the performance profiling results (7). The results are downloaded and captured in an MS Access database,

producing an archive of architecture model/performance results over time. The result database is queried, and performance results for a single or multiple performance test runs are visualized using various graphs and architecture model annotations (8).

4.3 Argo/MTE extending ArgoUML

Figure 4.3 illustrates the three important extensions made to ArgoUML. Argo/MTE extends the existing UML meta-model with a set of predefined elements to support Argo/MTE-styled architecture modeling. Argo/MTE extends the ArgoUML tool by adding a domain-specific meta-model specification tool to support domain-specific meta-modeling. Argo/MTE extends the ArgoUML tool by adding an architecture design tool to support architecture modeling.

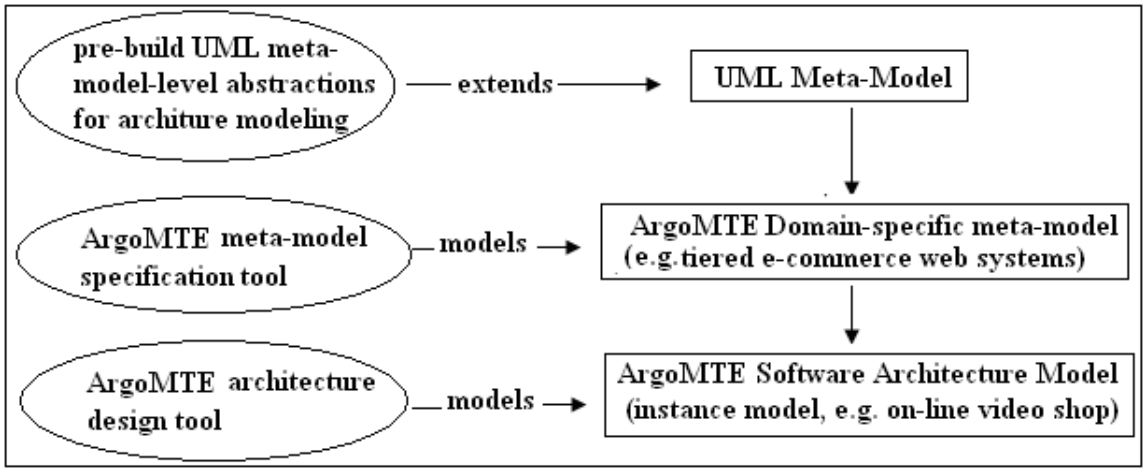


Figure 4.3. Extending ArgoUML

4.3.1 Extending UML meta-model to support architecture-specific modeling

Figure 4.4 illustrates the pre-defined UML meta-model level abstractions for architecture modeling, including: objects (*ArchOperHost*), nodes (*ArchHost*), operations (*ArchOperation*), attributes (*ArchAttribute*) and two types of properties (i.e. *ArchitecturalParameters*, *TestingParameter*).

ArchOperHost: an architectural abstraction that hosts operations. For example, a remote object of a CORBA application server in the NetPay system (refer to Figure 4.1) contains a set of business operations, and can be abstracted to an ArchOperHost.

ArchOperation: an architectural abstraction of operations/logic of an ArchOperHost. For example, a business operation of a CORBA remote object in the NetPay system (refer to Figure 4.1) can be abstracted to an ArchOperation.

ArchAttrHost: an architectural abstraction that hosts attributes (e.g. tables in a database). For example, a database in the NetPay system (refer to Figure 4.1) contains a set of tables, and can be abstracted to an ArchAttrHost.

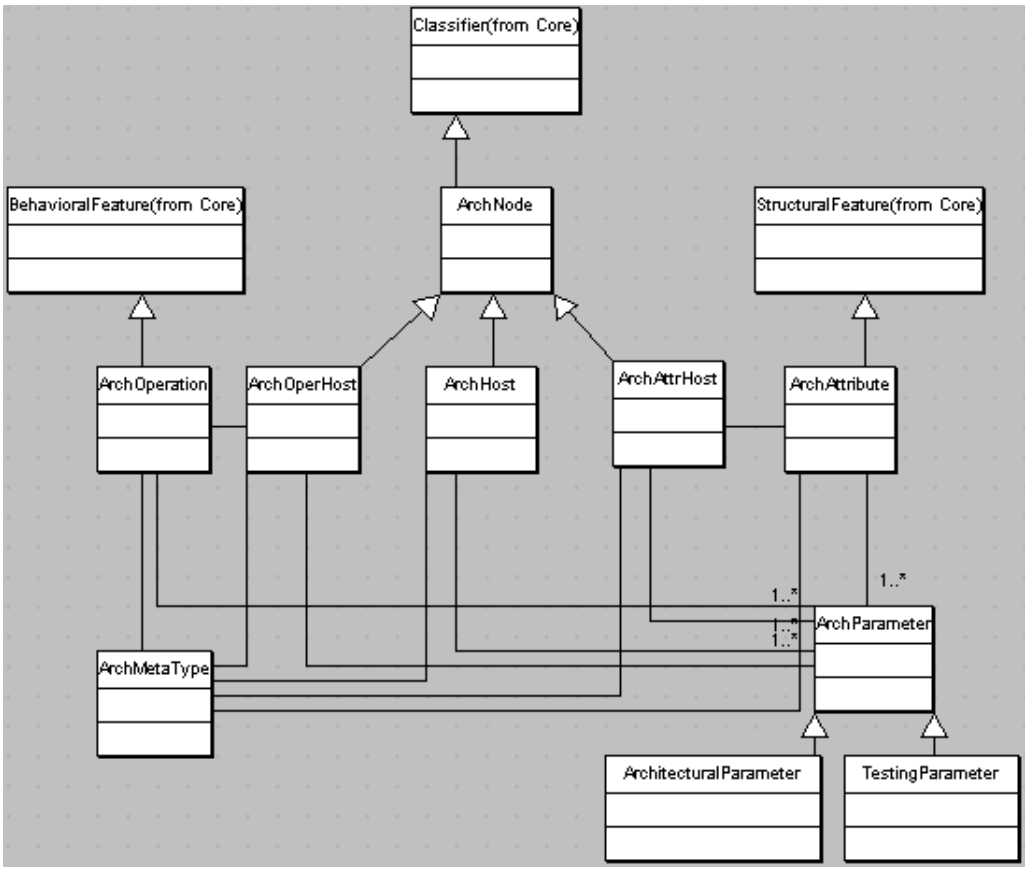


Figure 4.4. Extending UML meta-model with Argo/MTE architecture modelling abstractions

ArchAtrribute: an architectural abstraction of attributes for an ArchAttrHost. For example, a table of a database in the NetPay system (refer to Figure 4.1) can be abstracted to an ArchAttribute.

ArchHost: an architectural abstraction that hosts ArchOperHosts and ArchAttrHosts. For example, the CORBA NetPay Broker application server (refer to Figure 4.1) hosts a set of ArchOperHosts (such as CORBA remote objects), and can be abstracted to an ArchHost.

ArchParameter: an architectural abstraction of properties for ArchOperHosts, ArchAttrHosts, ArchOperations, and ArchAttributes. There are two types of ArchParameter; and they are ArchArchitectureParameter and ArchTestingParameter.

ArchitecturalParameter *(similar to AP in SoftArch/MTE, refer to section 3.3.2 of chapter 3):* *representing* properties that represent architectural/structural information. For example, a client request needs architectural parameters to define which remote server it calls, and which remote service it requests.

TestingParameter *(similar to CG in SoftArch/MTE, refer to section 3.3.2 of chapter 3):* representing properties that are related to performance measurements. For example, a client request needs testing parameters to define how many repetitive calls need to be made, and how long it takes for the request to get the response of the server.

ArchMetaType: an abstraction similar to the UML stereotype. An ArchMetaType represents a domain-specific abstraction. Sample ArchMetaTypes for the e-commerce domain can be Client, AppServer, and RemoteObject.

4.3.2 Adding a domain-specific meta-model specification tool

Same to SoftArch/MTE, Argo/MTE uses domain-specific meta-models to abstract domain-specific knowledge by specifying abstraction types and their relationships. Each such modeling type defines a set of architectural and testing parameters. Architectural parameters define structural data of an architecture design, while testing parameters provide data related to performance measures. A well-defined domain-specific meta-model is essential to ensure an architecture design has adequate, low redundancy information for test bed generation. Argo/MTE extends ArgoUML to build the Argo/MTE meta-modeling tool. Figure 4.1 shows a sample Argo/MTE meta-model for the domain of tiered e-commerce web systems (e.g. NetPay). This meta-model includes abstractions for clients, databases, application

servers, remote objects, and other architecture modeling types. The domain abstractions and their properties are listed in Table 4.1(similar to Figure 3.4). Element attributes annotated with "AP" are "ArchitecturalParameters" (refer to Figure 4.4) representing structural information. Ones marked "TP" are "TestingParameters" (refer to Figure 4.4) representing performance measures. Figure 4.5 shows that Argo/MTE meta-tool has similar look and feel of UML modeling, which reduces the learning load of users' domain-specific meta-modeling.

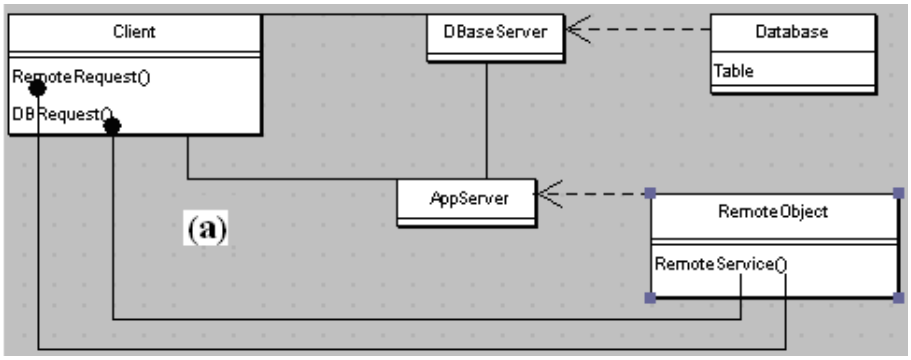


Figure 4.5. An Argo/MTE e-commerce-specific meta-model

Abstraction	Properties	Property Description
Client (typed by ArchOperHost)	Name (AP)	name of the client
	Type (AP)	type of the client, e.g. browser, CORBA, RMI
	Threads (TP)	simulate the number of clients
RemoteRequest (typed by ArchOperation)	Name (AP)	name of the request
	RemoteServer (AP)	name of the remote server the request tries to contact
	RemoteObject(AP)	name of the remote object the request tries to contact
	RemoteService(AP)	name of the remote service the request tries to contact
	WarmUp (TP)	how long the request needs to wait before it contacts the remote server
	RecordTime (TP)	if the response time needs to be saved or not
AppServer (typed by ArchHost)	Name (AP)	name of the server
	Type (AP)	type of the server, e.g. CORBA, RMI, J2EE, http-enabled
RemoteObject (typed by ArchOperHost)	Name (AP)	name of the remote object

	Type (AP)	type of the remote object, e.g. CORBA, RMI, J2EE
RemoteService (typed by ArchOperation)	Name (AP)	name of the service
	RemoteServer (AP)	name of the server the service tries to contact
	RemoteObject(AP)	name of the remote object the service tries to contact
	RemoteService(AP)	name of the remote service the service tries to contact
	WarmUp (TP)	how long the service needs to wait before it responds a request
	RecordTime (TP)	if the operation time needs to be saved or not
DBaseServer (typed by ArchHost)	Name (AP)	name of the database server
	Type (AP)	type of the database server, e.g. MS Access, MS SQL
Database (typed by ArchAttrHost)	Name (AP)	name of the database
Table (typed by ArchAttribute)	Name (AP)	name of the database table
DBRequest (typed by ArchOperation)	Name (AP)	name of the request
	RemoteDBServer (AP)	name of the database server this request tries to contact
	RemoteDB(AP)	name of the database this request tries to contact
	RecordTime (TP)	if the response time needs to be saved or not

Table 4.1. A sample Argo/MTE meta-model abstractions and their properties

4.3.3 Adding an architecture design tool

The Argo/MTE architecture design tool was developed by specializing the class diagramming and collaboration diagramming tools from ArgoUML. This approach provides architects with design tools similar to the look and feel of the ArgoUML toolset. Part of the complex, distributed NetPay system architecture is shown in Figure 4.6 to illustrate this tool. The architecture modeling notation extends the UML class and collaboration diagram appearance and layout. A class icon-like representation of architectural components is used. The Argo/MTE architecture model (Figure 4.6) comprises components (rectangles), requests and services of components (labels), associations (solid black lines) and hosting associations (dashed lines). Each component and association is typed by a domain-specific abstraction looking like the UML stereotype. Each modeling element (e.g. components, operations, and attributes) has a set of properties derived from its type. The architecture in Figure 4.6 comprises a customer PC-hosted browser and payment client (1), a broker (2), and a vendor site (3). The client browser accesses a

vendor's web pages (4), which, in turn, access application server components (5) communicating with a database (6). Components are associated via relationships. More details about this architecture model will be explained in chapter 5.

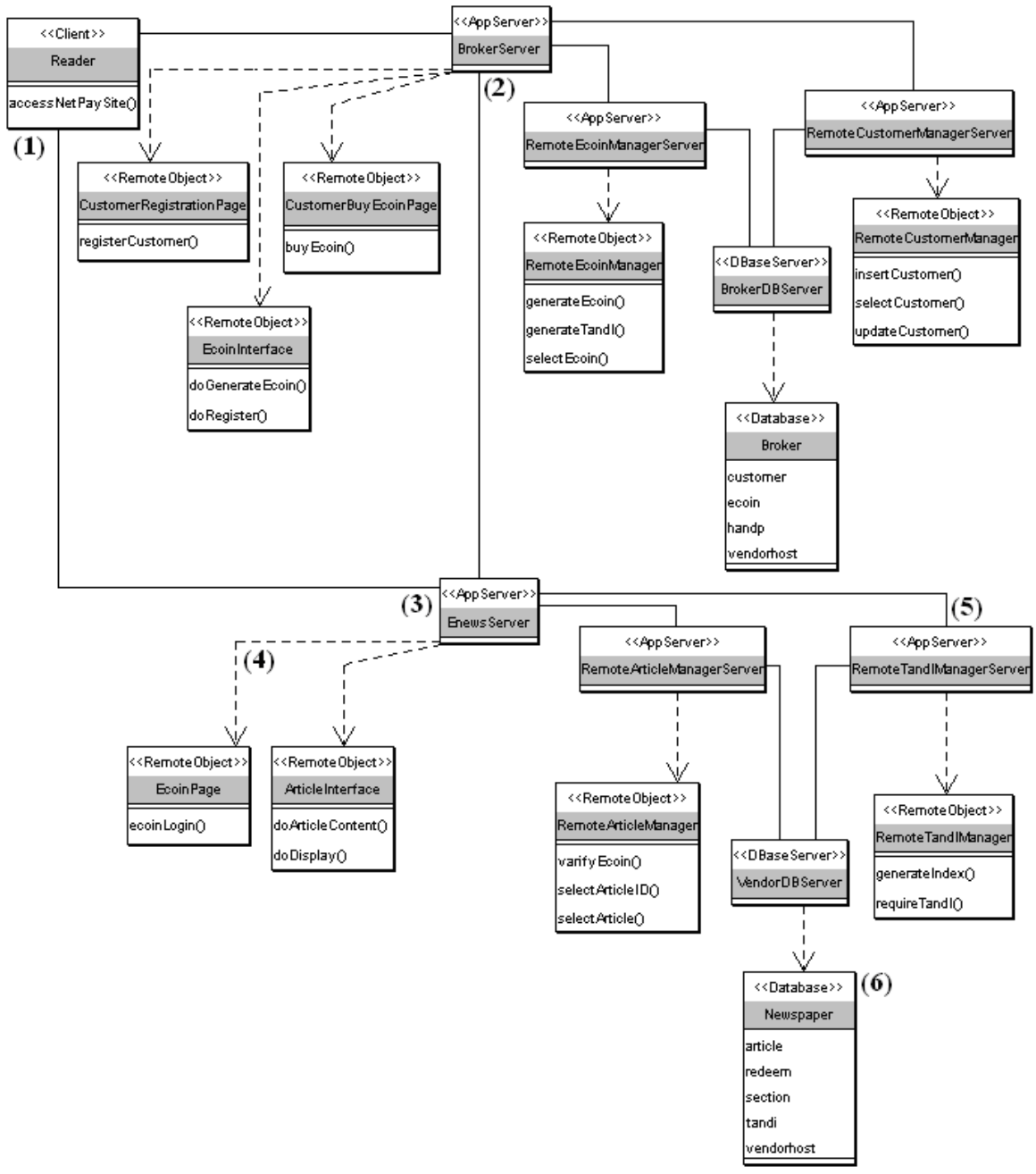


Figure 4.6. Part of the Argo/MTE architecture model for the complex micro-payment system

Argo/MTE supports multiple views for complex architecture specifications. Figure 4.7 shows two sample collaboration views of the architecture model, which both visualize/specify the same collaboration process between client requests and server services. In Figure 4.7(1), the collaboration/messaging view is overlaid with the structural architecture design. In Figure 4.7(2) the collaboration/messaging view is displayed separately. Collaboration views provide complementary support for users to specify how the structural modeling information collaborates to complete concrete tasks.

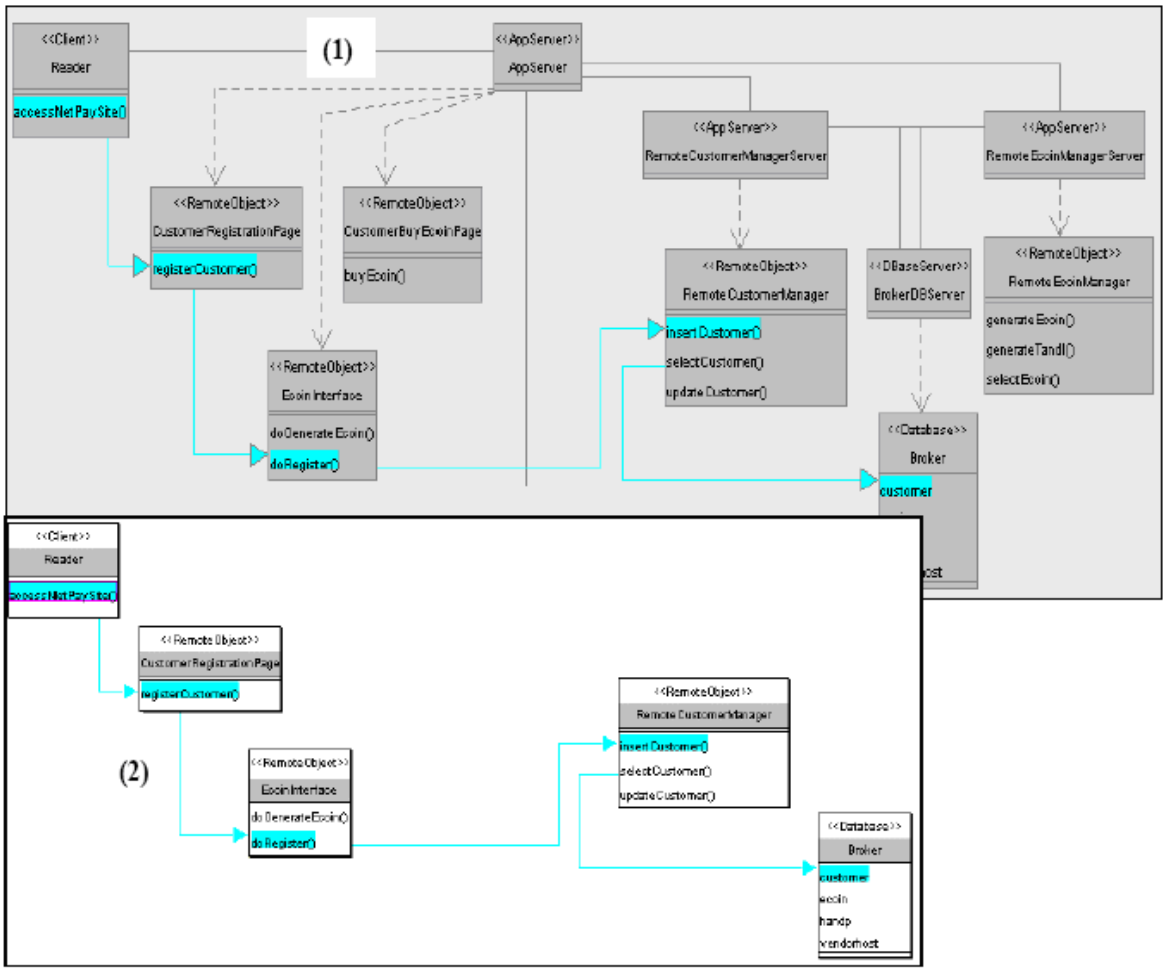


Figure 4.7. Sample collaboration views of an Argo/MTE architecture design

4.4 Data format of the Argo/MTE architecture model

SoftArch/MTE architecture models are saved as a proprietary XML format (a sample XML file is shown in Figure 4.8). The schema of the architecture XML files is based on the used SoftArch/MTE domain-specific meta-model. It uses abstraction types (e.g. *Client* in line 2, *RemoteRequest* in line 6) and their properties (e.g. *Name* in line 3, *Threads* in line 5) as the main tags. Whenever the domain-specific meta-model is changed (e.g. change of meta-type name), the schema of architecture XML is changed, and the related SoftArch/MTE XML reader and writer need to be changed as well (manually). The problem is getting worse when modeling larger systems, where types are complicated. The architecture XML files also have the fundamental problem that only the SoftArch/MTE tool could ever generate and read them. Argo/MTE is aimed at representing the architecture models in a more standardized format, and eventually making Argo/MTE model data exchangeable with other XMI-supporting CASE tools.

```
1  .....
2  <Client>
3   <Name>Reader</Name>
4   <Type> browser</Type>
5   <Threads>20</Threads>
6   <RemoteRequest>
7     <Name>registerCustomer</Name>
8     <RemoteServer>Broker</RemoteServer>
9     <RemoteObject>CustomerRegistrationPage</RemoteObject>
10    <RemoteService>registerCustomer</RemoteService>
11    <WarmUp >10</WarmUp >
12    <RecordTime >true</RecordTime >
13  </RemoteRequest>
14 </Client>
15 .....
```

Figure 4.8. Sample SoftArch/MTE architecture design XML file

For now, Argo/MTE extends ArgoUML's XMI representational format, and uses the extended XMI-format to record Argo/MTE architecture designs. The architecture-specific elements of the extended UML meta-model (shown in Figure 4.5) consist of the main tags of architecture design XMI files (illustrated in Figure 4.9). Instead of being the main tags, the domain-specific abstraction types are recorded as the values of tag *ArchMetaType* (see lines 31 and 48 in Figure 4.9). This extended XMI format is more stable than the proprietary SoftArch/MTE architecture model data format, because the evolvement of an Argo/MTE domain-specific meta-model will not influence the reader and writer of the Argo/MTE tool but the values of *ArchMetaType*. For example, when users modify a domain-specific meta-model by changing the name of an abstraction type or adding more testing or architectural

parameters to a type, the Argo/MTE tool can save the architecture design without changing the model reader and writer of the tool. Using *UML meta-model level* architecture elements to record Argo/MTE architecture design models separates domain-specific abstractions from architecture model data format, which leaves the Argo/MTE domain-specific meta-models to be focused on: domain-specific abstractions, their code generation scripts, and the logic that coordinates the code generation scripts when generating fully functional test bed.

```
2   .....
3   <ArchOperHost name="Reader" xmi.id="xmi.7"
4   xmi.uuid="-126--40-38--114-47cadf:f8dc488e98:-7ff5">
5     <ArchOperations>
6       <ArchOperation>
7         <ArchOperation.TestingParameters>
8           <ArchOperation.TestingParameter>
9             Boolean RecordTime=true
10          </ArchOperation.TestingParameter>
11          <ArchOperHost.TestingParameter>
12            Integer WarmUp =10
13          </ArchOperHost.TestingParameter>
14        </ArchOperation.TestingParameters>
15
16        <ArchOperation.ArchitecturalParameters>
17          <ArchOperation.ArchitecturalParameter>
18            String Name=registerCustomer
19          </ArchOperation.ArchitecturalParameter>
20          <ArchOperation.ArchitecturalParameter>
21            String RemoteServer=Broker
22          </ArchOperation.ArchitecturalParameter>
23          <ArchOperation.ArchitecturalParameter>
24            String RemoteObject=CustomerRegistrationPage
25          </ArchOperation.ArchitecturalParameter>
26          <ArchOperation.ArchitecturalParameter>
27            String RemoteService=registerCustomer
28          </ArchOperation.ArchitecturalParameter>
29        </ArchOperation.ArchitecturalParameters>
30
31        <ArchMetaType>RemoteRequest</ArchMetaType>
32      </ArchOperation>
33    </ArchOperations>
34
35    <ArchOperation.TestingParameters>
36      <ArchOperHost.TestingParameter>
37          Integer Threads =20
38      </ArchOperHost.TestingParameter>
39    </ArchOperation.TestingParameters>
40    <ArchOperHost.ArchitecturalParameters>
41      <ArchOperHost.ArchitecturalParameter>
42        String Name=Reader
43      </ArchOperHost.ArchitecturalParameter>
44      <ArchOperHost.ArchitecturalParameter>
45        String Type =browser
46      </ArchOperHost.ArchitecturalParameter>
47    </ArchOperHost.ArchitecturalParameters>
48    <ArchMetaType>Client</ArchMetaType>
49  </ArchOperHost>
50  .....
```

Figure 4.9. A sample Argo/MTE architecture design XML file

4.5 Test bed generation and domain-specific meta-model evolvement

Figure 4.10 illustrates the framework of Argo/MTE code generation. The extended UML meta-model provides architecture model data format (1) and works as the meta-model for domain-specific meta-modeling (2). A domain-specific meta-model records abstractions of an interested domain (in this case, tiered middleware-based web systems); it provides appropriate code generation script(s) (e.g. Client.xslt, RemoteRequest.xslt) for each of its abstraction; and it provides control logic to coordinate the code generation scripts to generate functional code. The role of domain-specific meta-models in test bed generation will be explained in detail through a case study in Chapter 5.

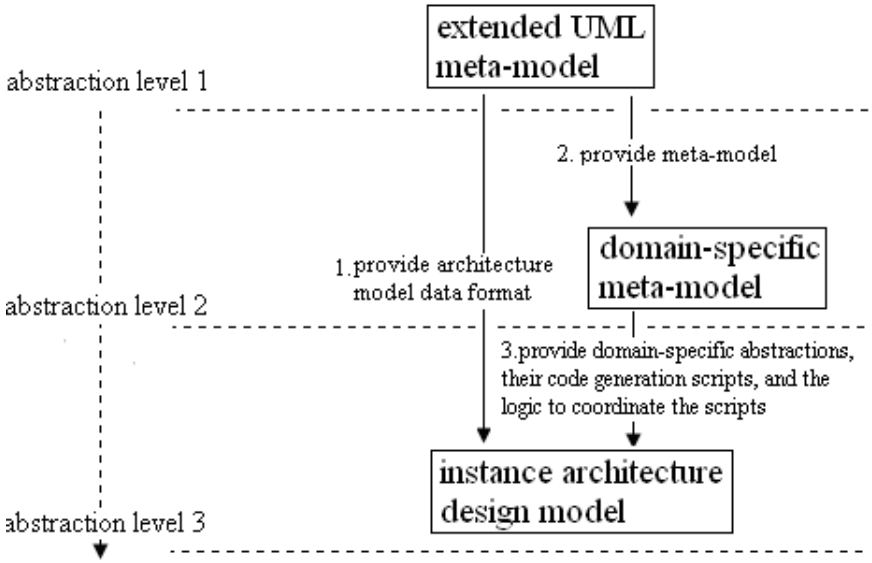


Figure 4.10. Framework for Argo/MTE code generation

An Argo/MTE domain-specific meta-model may evolve with new architectural concerns coming up. When a domain-specific meta-model evolves, its code generation scripts and logic need to evolve. At this stage, Argo/MTE provides a conceptual framework to support the evolvement. Figure 4.11(b) illustrates the conceptual framework that can support the evolvement of the meta-model in Figure 4.11(a).

In Figure 4.11(b), steps (a) and (b) are preparation steps. Before extending a domain-specific meta-model to support new domain-specific concerns, tool users/developers need to analyze the new concerns to decide their performance-critical atomic functional code part (step (a)). Tool users/developers derive

the intended new domain-specific abstractions and construct programs (mainly XSLT scripts) to bridge the intended abstractions with their functional code part (e.g. java source code) (step (b)).

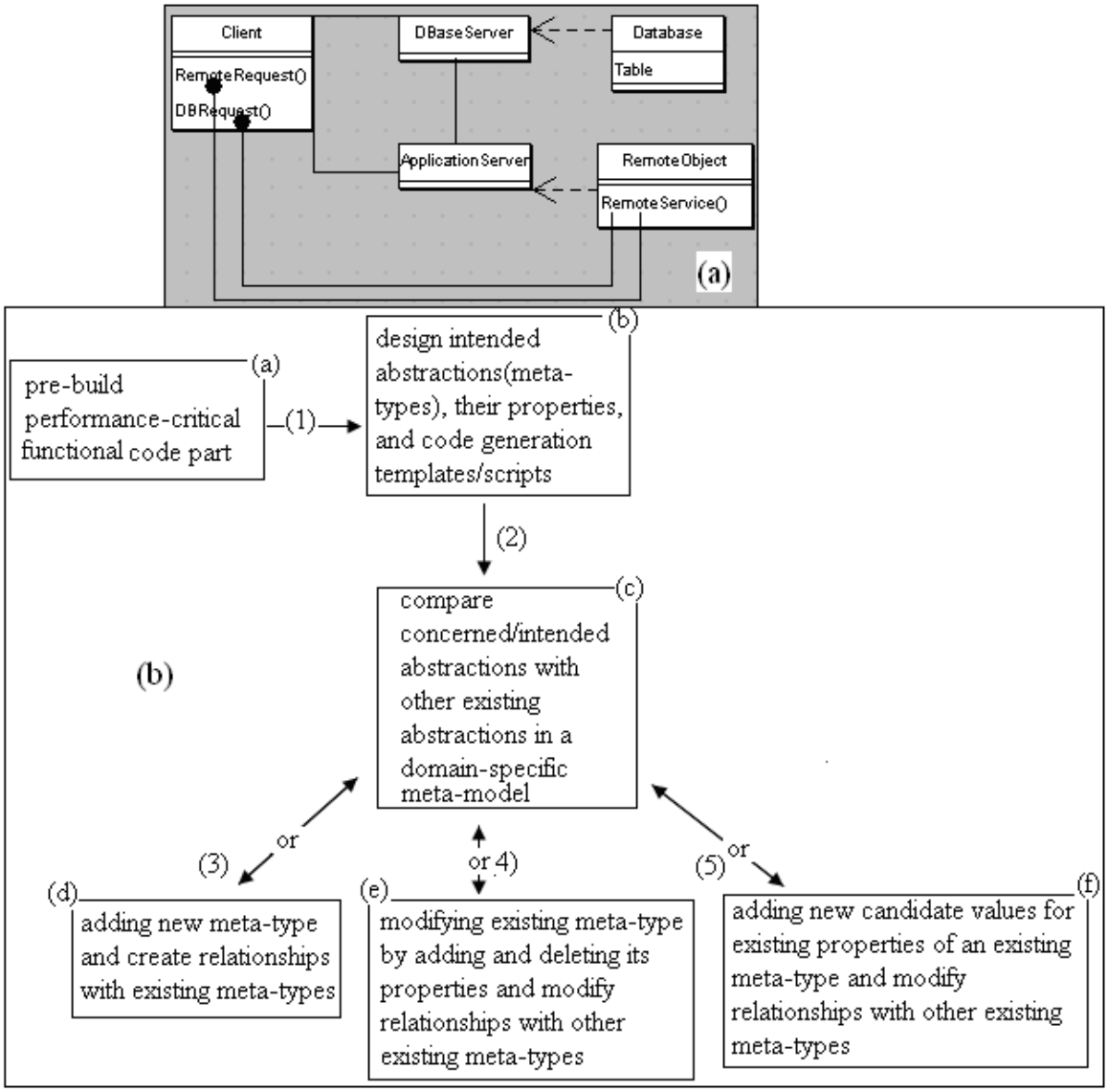


Figure 4.11. (a): a sample Argo/MTE meta-model for tiered web systems; (b) a conceptual framework to evolve Argo/MTE domain-specific meta-models

After the preparation, tool users/developers start to evolve the existing meta-model to support the new intended abstractions, and they have three options to do so: 1) none of the existing abstractions is even close to the intended abstraction, and a new abstraction need to be added to the existing meta-model

(step(d)); 2) one existing abstraction is similar to the intended abstraction, and can represent it via slight modifications of the existing properties (such as adding and/or deleting properties)(step (e)); 3) an existing abstraction is very close to the intended abstraction, and can represent the intended abstraction by adding and/or deleting a few candidate values for some of its properties (step(f)). The three options allow users to extend the domain-specific meta-model in different scopes to make sure that new meta-modeling information goes to the right places without causing much redundant modeling information.

A domain-specific meta-model must always be validated while it is evolving. For example, for every newly-added modeling abstraction, users need to check how it influences the existing meta-types/abstractions, such as, if the newly added abstraction is able to work with the existing abstractions as they are, or if the existing abstractions need to be modified before they can work with the newly added abstraction. This evolvement-validation cycle is represented by the bi-directional arrows (3), (4), (5) in Figure 4.11(b).

This conceptual structured framework helps users to develop, modify, and reuse domain-specific meta-models in the Argo/MTE's meta-model specification tool. The conceptual framework will be supported by the Argo/MTE tool in the future work.

4.6 Automating performance evaluation process

Besides test bed generation, the performance evaluation process involves test bed compilation, deployment, execution, and result collecting. Argo/MTE uses third party tools (mainly Ant tool) to automate the tedious and error-prone evaluation process. The efficiency and maintainability of Argo/MTE evaluation process have been hugely improved from those of SoftArch/MTE.

4.6.1 Generating Ant build files

The test bed performance evaluation process is complicated. For example, to successfully compile a web system (e.g. NetPay), the order of compiling each part of the system is important, as some parts rely on the compiled results of other parts (such as a CORBA object can not be compiled until its IDL file is compiled). SoftArch/MTE hard-codes a set of monolithic guidelines (logic) to generate DOS batch files from an architecture model, and uses the DOS batch files to automate the process of test bed

performance evaluation. As the complexity of the test bed increases, the hard-coded monolithic guidelines are hard to evolve, and DOS batch files are extremely hard to manage.

Argo/MTE uses the Ant tool (Apache Ant, 2004) to improve the flexibility and manageability of the performance evaluation process. Ant build files are well-structured; and their target actions are functionally independent. They can manage complex dependencies among the parts of a large test bed program at each stage of compilation, deployment, and execution.

Figure 4.12(a) illustrates a small piece of generated Ant build file for compiling the *BrokerServer* and *EnewsServer* of the generated NetPay test bed. Details of the build file will be explained through the case study in Chapter 5. Figure 4.12(b) describes a conceptual framework for Argo/MTE to manage the evolvement of build file generation. Steps (a) and (b) are preparation steps. Each meta-model abstraction must bring pre-built scripts to generate its individual Ant build files for various operational tasks, including compilation.xml, deployment.xml, and resultCollection.xml (step a). The meta-model itself must provide control logic to coordinate the individual build files of the abstractions (step (b)). When the meta-model evolves (refer to Figure 4.11), users need to update the pre-built generation scripts of each abstraction (by replacing old ones or adding new ones); and they must also validate the control logic to coordinate all the involved scripts to synthesize appropriate Ant build files for the process of test bed performance evaluation (step (c)),

```
<!-- ================ Compile BrokerServer server======================== -->
<target name="BrokerServer" depends="RemoteEcoinManagerServer,RemoteCustomerManagerServer">
    <javac srcdir="src/Broker" destdir="${build.home}/WEB-INF/classes"
        debug="${compile.debug}" deprecation="${compile.deprecation}"
            optimize="${compile.optimize}">
        <classpath refid="compile.classpath"/>
    </javac>
</target>
<!-- ================== Compile EnewsServer server================ -->
<target name="EnewsServer" depends="RemoteTandiManagerServer,RemoteArticleManagerServer">
    <javac srcdir="src/enews " destdir="${build.home}/WEB-INF/classes"
        debug="${compile.debug}" deprecation="${compile.deprecation}"
            optimize="${compile.optimize}">
    <classpath refid="compile.classpath"/>
    </javac>                                                    (1)
</target>
```

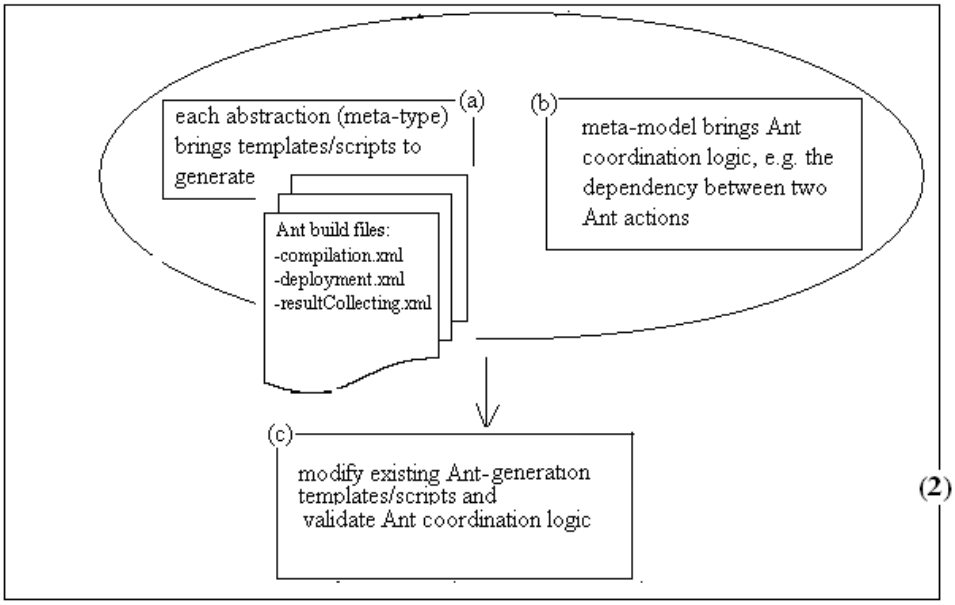


Figure 4.12. (1) sample Ant build file snippet; (2) a conceptual framework to manage the evolvement of Ant script generation

4.6.2 Managing automated performance evaluation Process

Figure 4.13 illustrates the Argo/MTE process of test bed deployment, execution, and results management. Argo/MTE instructs Ant to upload and initialize generated test client and server programs, scripts, IDLs and database scripts (1).The generated Ant build script is run with "deployment" parameter, resulting in multiple file uploads to remote hosts using a local SFTP client and remote SFTP servers(2). Each remote host has another generated Ant script uploaded as part of this deployment process. This is used by a remotely-deployed Ant build engine. The Ant build engine, running on each host to initialize deployed programs and configuration scripts, synchronizes the start of multiple client programs (3). Results from performance evaluation are captured as text files (4). The Argo/MTE Ant engine downloads these results via SFTP (5). It updates a test database by inserting performance results for each model item grouping them by test run (6).

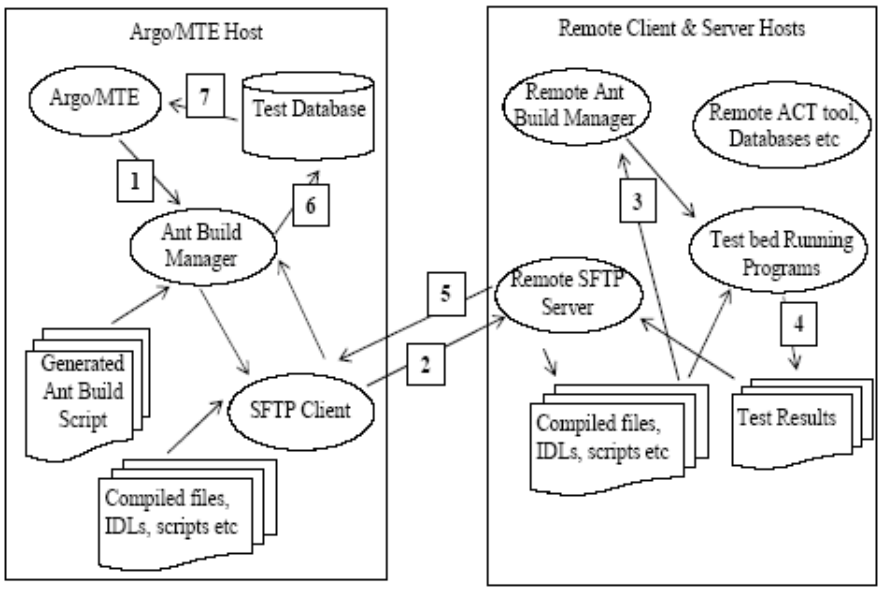


Figure 4.13. Argo/MTE test execution & results capture.

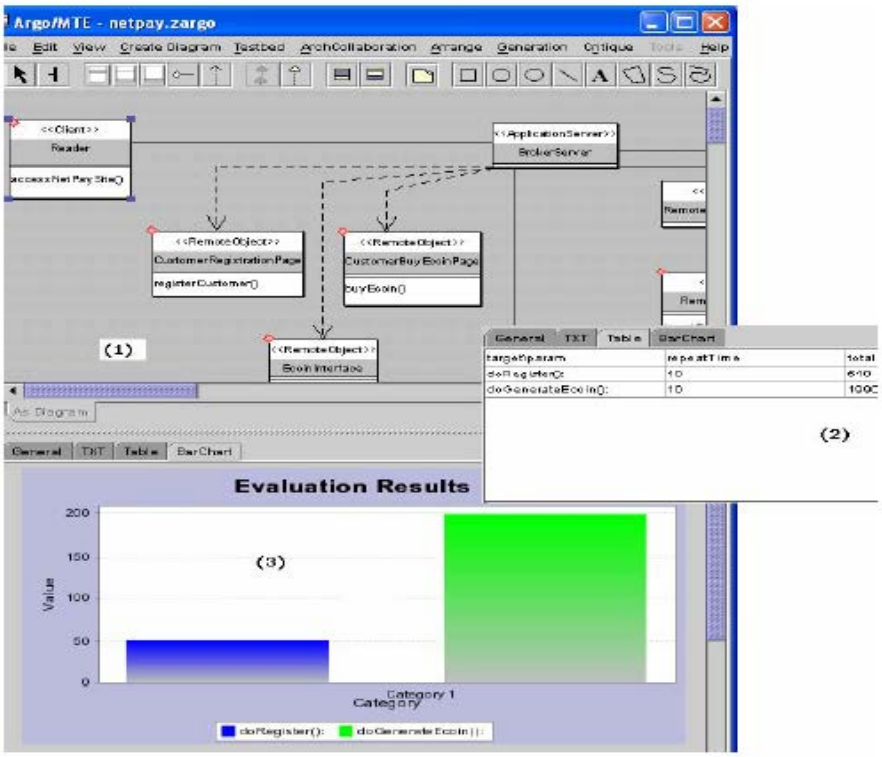


Figure 4.14. Example of result visualization

The result visualization component queries the test result database and displays test run results as graphs or annotations to architecture model diagrams. This is shown in Figure 4.14 where architecture components with performance results available are annotated with a small box at the top left corner (Figure 4.14(1)). The collected results can then be examined in tabular (Figure 4.14(2)) or graph (Figure 4.14(3)) form within Argo/MTE. The visualization component closely integrates the result visualization support into Argo/MTE architecture model diagrams (7).

4.7 Discussion and Conclusions

4.7.1 General Discussion of Argo/MTE

Argo/MTE has been used to model and test several software architectures and has compared generated performance results against that of actual implemented applications for accuracy. Applications modeled include several variants of thick and thin-client versions of an on-line video application (Grundy and Cai et al, 2001), a Java Pet Shop application (MSDN, 2002), substantial parts of NetPay (Dai et al, 2007; Cai et al, 2004), and several architectural approaches to Enterprise Application Integration (EAI) support system (Grundy and Bai et al, 2003). Argo/MTE successfully modeled these diverse architectures. The meta-modeling tool permitted users to define allowable modeling abstractions and tailor meta-models for ever-growing interested target technologies (e.g. from RMI, CORBA, to J2ee, .NET). The structural architecture modeling facilities were predominantly used to define clients and their requests, multi-tier servers, server objects, web components and relationships, and databases and tables. More complex architectures like the EAI and NetPay systems used multiple views with collaboration to manage the modeling complexity. Argo/MTE also presents conceptual framework to support the evolvement of test bed generation and Ant files generation as the domain-specific meta-model evolves.

J2EE and .NET test bed code is generated for each modeled and tested system. Generated test beds are run by using one or more SQL Server 2000database servers. Some of the tested applications had pre-existing implementations in both J2EE and .NET (video system and Pet Shop); others had implementations in Java, J2EE, Java Messaging Service and CORBA (NetPay and the EAI application). Generated performance tests were run against the original, hand-implemented applications. In general, performance results obtained from the generated test bed code are accurate, with detailed Argo/MTE models producing performance results within 20-40% of the hand implemented applications. Larger variances occurred with systems with complex business logic (conditional execution of substantial

remote object and database services) and complex transaction processing logic as these violate Argo/MTE's assumption of low overhead of such code. For some implementation technologies, including Java Messaging Service and .NET web services, Argo/MTE had only rudimentary code generators, resulting in inaccurate generated code. Overall, the performance results obtained from Argo/MTE's generated test beds were reasonably close to those obtained when running the exact same clients against the real implemented system servers. As the code generation scripts/templates encode "the best-of-practice", test bed can also help to discover implementation errors in the real system developed by less experienced developers when Argo/MTE test bed results are wildly different from those obtained from the real system. Correction of these programming errors resulted in much closer performance result correlation. Argo/MTE's performance test database proved useful to capture all test results in one place and allow complex analysis and result visualization.

The conceptual frameworks (Figure 4.11 and Figure 4.12) provide structured guidelines to extend code generation facilities with the ever-evolving domain-specific meta-model. The frameworks need to be elaborated to get full tool support in the future. Argo/MTE can annotate architecture models with the evaluation results, and present evaluation results through the seamless integration with third party tools (e.g. MS Access, MS excel). The performance visualization support is still basic and needs improving.

The XMI extensions, based on the UML meta-model extensions, are arbitrary, although they are a significant improvement on the proprietary architecture model format of SoftArch/MTE. The format used may require revision as standardization occurs in the representation of architecture information in UML and XMI.

Extensions to the UML meta-model, test bed evolvement management and Ant-file evolvement management are all much more structured and flexible than with SoftArch/MTE, which allows Argo/MTE to deal with more complex architecture design models than the SoftArch/MTE. Most of the systems tested by Argo/MTE had several servers and databases, with numerous remote operations per server, even with greatly simplified architectural models. Compared with SoftArch/MTE, Argo/MTE is more user-friendly, better structured, easier to manage, and more efficient.

4.8 Summary

Applying SoftArch/MTE automated performance test bed generation tool to industrial case studies proved problematic. It was found that while this automated software engineering technique was applicable to the case study domains, the proof-of-concept tool had many problems when trying to scale it. Argo/MTE was developed to integrate the test bed generation approach into an open source, UML-based CASE tool. Extensions of UML modeling notations and data representations of models are used to describe architectures. A number of third-party tools, including XSLT, Ant, SFTP and MS Access, are used to realize the performance test bed generator support in a much more scalable and flexible way. Using Argo/MTE on several large industrial case studies indicates these approaches have generally been successful in scaling SoftArch/MTE test bed generation approach.

Chapter 5 - Using Argo/MTE - NetPay Case Study

This chapter uses NetPay as a comprehensive case study to examine the features of Argo/MTE. The case study demonstrates how Argo/MTE specifies software architecture of complicated web systems; generates a high abstraction level test bed; deploys and executes the distributed test bed; and collects and stores performance evaluation results. The case study also compares the performance evaluation results of a generated NetPay test bed with those of a legacy NetPay software system.

5.1 NetPay review

As is introduced in Chapter 4, NetPay is a micro payment system to charging for web content (typically) for situations with a small cost-per-use/ high use-frequency. An example use of NetPay is on-line newspaper purchase, where users may intend to spend large numbers of small amounts of money at web sites in exchange for various content or services (Dai et al, 2007; Cai et al, 2004). A NetPay micro-payment system includes customers (e.g. newspaper readers), vendors (e.g. on-line e-newspapers) and a broker. The broker is responsible for the registration of readers, and for crediting the e-newspaper's account and debiting the reader's account. Other main concepts in a NetPay system include: e-coin (electronic money used in NetPay system), and e-wallet (cached e-coin information), and touchstone (e-coin transaction history) (Dai et al, 2007).

Figure 5.1 illustrates the architecture of the NetPay system (Cai et al, 2004). The Broker hosts a database that holds the information of: customer and vendor account, generated coins and payments, and redeemed coins and micro-payments made (buying coins and redeeming money to vendors). Through a set of CORBA interfaces, the Broker application communicates with vendor application servers when they request touchstones and redeem e-coins. The Broker server also communicates with one or more bank servers to authorize micro-payments (customer buying coins or broker paying vendors when

redeeming spent coins). The customer can access the Broker to buy e-coins, and check their e-wallet balances and transaction history (Dai et al, 2007; Cai et al, 2004).

The customer accesses the broker and vendor servers through a web browser. The customer's e-wallet (cached e-coin information) resides on the vendor server, and is transferred from the broker to each vendor he/she is buying content from. When the customer buys e-coins the Broker's application server updates the customer's e-wallet. When the customer purchases information, the vendor's web server accesses e-coin information using his/her e-wallet.

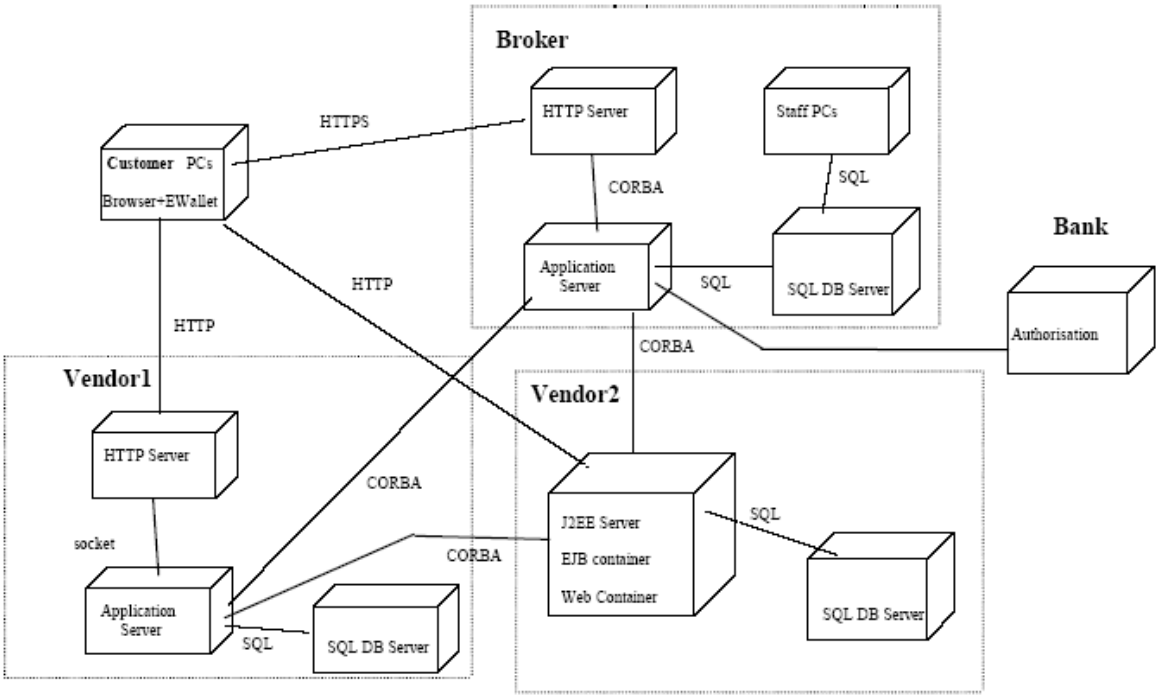


Figure 5.1. Basic NeyPay software architecture (Cai et al, 2004)

The vendor sites provide an http-enabled application server. The Vendors provide web pages with contents that need to be paid for. In order to verify the e-coins being spent and to redeem the spent e-coins, the Vendor application server needs to access the Broker application server via CORBA to obtain information. Vendor application servers communicate with each other to pass on e-coin info via a CORBA interface. Vendors may use quite different architectures. In the example above, Vendor #1 uses a web server, custom application server and relational database. Vendor #2 uses a J2EE-based

architecture with J2EE server providing Java Server Pages (web services) and Enterprise Java Beans (application server services) (Cai et al, 2004).

After developing the NetPay system architecture and a prototype, we wanted to determine its scalability and performance (mainly through average response time) under heavy loading. Argo/MTE was used to retrieve the software architecture model of the NetPay system and generate performance test beds to empirically evaluate NetPay performance. As part of this work we wanted to assess both the performance of the legacy NetPay system as well as the performance of the Argo/MTE-generated test bed under Argo/MTE-generated client loadings. To this end we:

- developed an architecture model of the NetPay system in Argo/MTE
- generated client load test application code and ran these against the legacy NetPay system
- generated server-side code from the NetPay architectural model and ran the same NetPay-generated client load test applications against it
- compared the results obtained by these two performance evaluations
- made modifications to the legacy NetPay code base to correct performance faults discovered during this process
- re-ran the client loading tests against the existing, modified NetPay application and compared the results with those from our generated server-side code

5.2 Modelling NetPay system in Argo/MTE

Figure 5.2 shows part of the Argo/MTE architecture design of the CORBA-based NetPay software architecture in Figure 5.1. This architecture model uses the e-commerce domain-specific meta-model introduced in section 4.3.2, Chapter 4. Like Figure 5.1, the model contains three main parts, including the “Reader” (typed as *Client*), the “BrokerServer” (typed as *AppServer*), and the “EnewsServer” (typed as *AppServer*). Unlike Figure 5.1, this model only contains one vendor. In Figure 5.2, the “Reader” accesses the “BrokerServer” to buy e-coins, and accesses the “EnewsServer” to purchase web contents by using the e-coins.

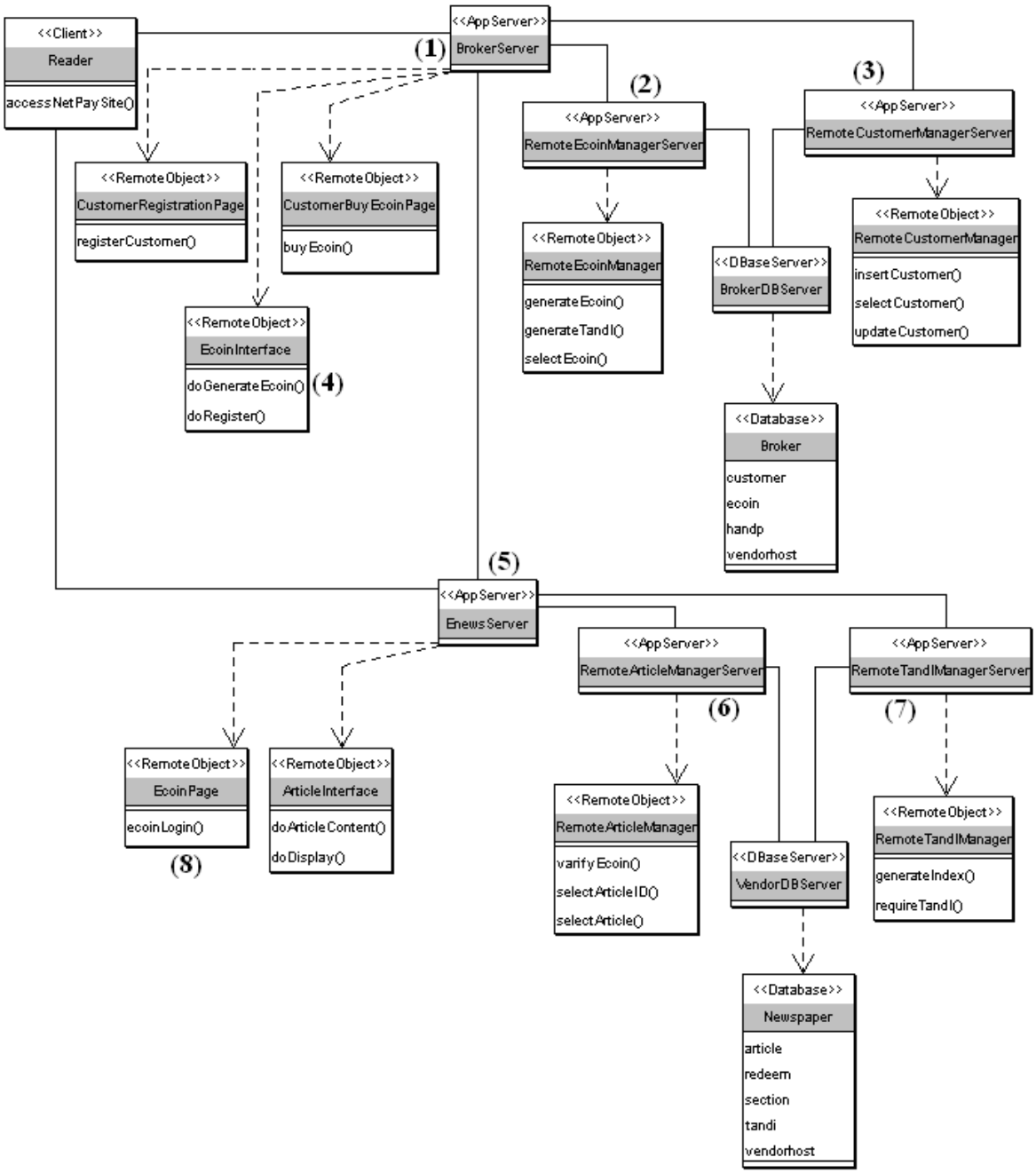


Figure 5.2. Partial architecture of NetPay in Argo/MTE

The two remote objects "CustomerRegistrationPage" and "CustomerBuyEcoinPage" of the "BrokerServer" (1) provide services for users to register themselves with the NetPay service and purchase e-coins respectively. The "BrokerServer" provides clients with further services from the

“RemoteEcoinManagerServer” (2) and the “RemoteCustomerManagerServer” (3) via the “EcoinInterface” (4). The “RemoteEcoinManagerServer” (2) hosts the “RemoteEcoinManager” that provides e-coin-specific operations, such as “generateEcoin”, “generateTandI”, and “selectEcoin”. The “RemoteCustomerManagerServer” (3) hosts the “RemoteCustomerManager” that provides customer-specific operations, such as “insertCustomer”, “selectCustomer”, and “updateCustomer”. Customers and their e-coin status are recorded in the “Broker” database on the database server “BrokerDBServer”.

The “EnewsServer” (5) accepts e-coins via the “EcoinPage” (8); the “EnewsServer” requests web contents from the “RemoteArticleManagerServer” (6) via the “ArticleInterface”; and it processes customers’ e-coin status with the help of the “RemoteTandIManagerServer” (7). The “RemoteArticleManagerServer” (6) hosts the “RemoteArticleManager” that first verifies customers’ e-coin status (via operation “verifyEcoin”) then presents web contents (via operations “selectArticleID” and “selectArticle”). The “RemoteTandIManagerServer” (6) hosts “RemoteTandIManager that checks the security stamp and credit status (via operation “requireTandI”) of an e-coin and updating a customer’s e-coin status (via operation “generateIndex”). The “Newspaper” database records transaction-related information (e.g. web contents of “article” and “article section”, e-con status of “redeem”).

Although only just a part of the NetPay system, Figure 5.2 shows considerable complexity. There are six application servers involved, two of them (“BrokerServer” and “EnewsServer”) must be http-enabled (e.g. J2EE, .Net server), and the others can be simple application servers (e.g. CORBA, RMI, etc) as well as complicated http-enabled ones. Inter-component and inter-service communications are also complicated, including communications between the reader and the broker, the broker and the vendor, the broker and other more specific servers (“RemoteEcoinManagerServer”, “RemoteCustomerManagerServer”), and the vendor and other more specific servers (“RemoteArticleManagerServer” and “RemoteTandIManagerServer”).

5.3 NetPay test bed

5.3.1 Test bed generation rules, scripts, and logic

An Argo/MTE domain-specific meta-model must provide rules, scripts, critics, and test bed generation logic. As is illustrated in Figure 5.3, each domain-specific abstraction must provide a textual rule to

describe what targeted concept to generate code for (e.g. the concept of application server of CORBA, the server-side remote object of CORBA, the client of CORBA) (1). The abstraction must provide XSLT scripts to define how to generate functional code for the concept (2). The meta-model itself provides critics (through hard-coded logic) to validate software architecture design and coordinate test bed generation (3). The meta-model itself also needs to provide test bed generation logic to determine the structure of the generated test bed (4).

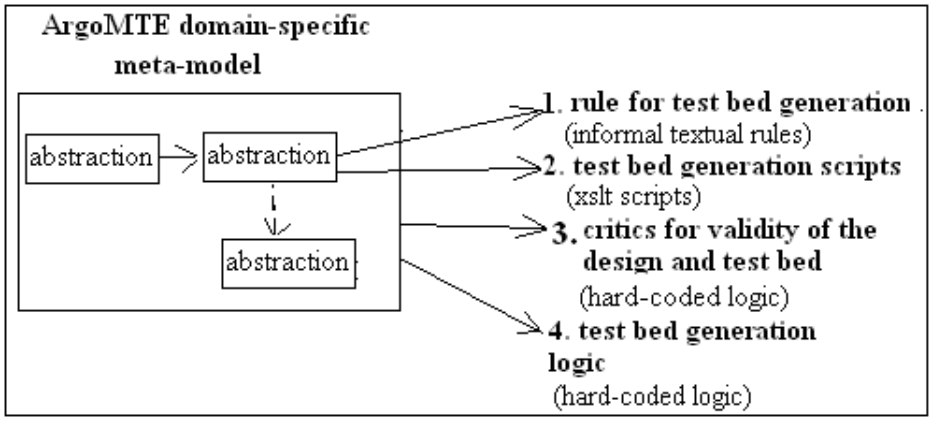
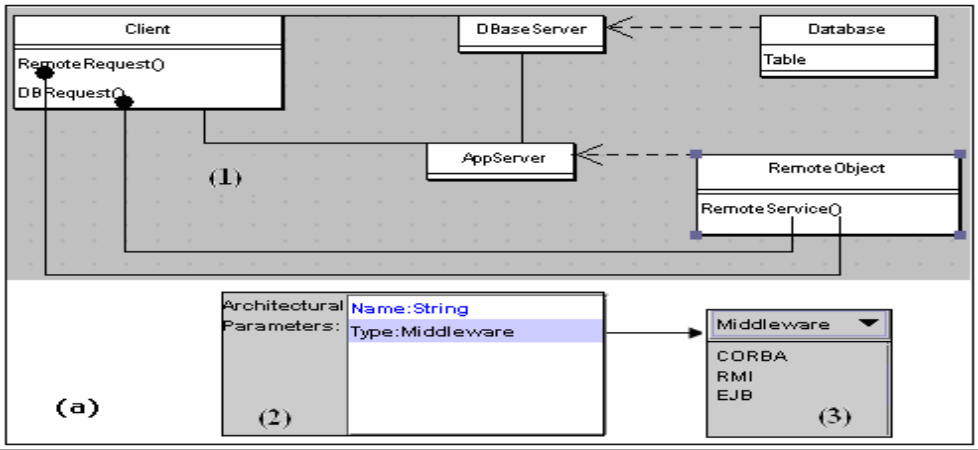


Figure 5.3. Argo/MTE meta-model supporting test bed generation

An abstraction's test bed generation rules and scripts are based on the used extended-UML meta-type (refer to section 4.3.1 of chapter 4) and the used middleware technologies. For example, in Figure 5.4(a)(1), abstraction RemoteObject is an ArchOperHost and has two properties "Name" and "Type" (Figure 5.4(a)(2)). The property "Type" represents the middleware technologies (also as targeted technologies of the evaluation) supported by Argo/MTE, including CORBA, RMI, and EJB (Figure 5.4(a)(3)). So, the rule for designing test bed generation scripts for abstraction RemoteObject is "*that the combination of ArchOperHost and CORBA middleware targets at generating code for the concept of CORBA server-side remote object; the combination of ArchOperHost and RMI middleware targets at generating code for RMI server object; and the combination of ArchOperHost and EJB targets at generating code for session bean and entity bean*". The rule then leads to the design of the abstraction's test bed generation scripts such as, corba_object.xsl, rmi_object.xsl, entityBean.xsl, jsp.xsl, sessionBean.xsl. The various middleware-specific scripts of an abstraction can be refactored to make it easier to organize XSLT scripts.




```
- <xsl:template match="ArchOperHost">
  - <xsl:choose>
    + <xsl:when test="./ArchitecturalParameters/Type='JSP'">
    - <xsl:otherwise>
        package
        <xsl:value-of select= .//ArchitecturalParameters/Host" />
        ; import java.util.*; import java.io.*; import java.net.*; import java.sql.*; import
        supportFiles.*; import org.omg.CosNaming.*; import
        org.omg.CosNaming.NamingContextPackage.*;
      + <xsl:for-each
          select="//ArchOperations/ArchOperation/SubOper/ArchitecturalParameters/UsingRemServer
          [generate-id()=generate-id(key('distinct-remServer',.))]">
      + <xsl:choose>
        <xsl:apply-templates mode="CORBA" select="." />
        <xsl:apply-templates select="//ArchOperation" />
        }
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
- <xsl:template match="ArchOperation">
    public void
    <xsl:value-of select="./ArchitecturalParameters/Name" />
    (
    <xsl:apply-templates mode="normal" select="./ActualParameters" />
    ) {
    <xsl:variable name="recordingTime" select="//RecordingTime" />
    <xsl:variable name="repetition" select="//Repetition" />
  - <xsl:if test="$recordingTime = 'true'">
      boolean recordingTime =
      <xsl:value-of select="$recordingTime" />
      ; long startTime = System.currentTimeMillis();
    </xsl:if>
  - <xsl:if test="$repetition !='null'">
      int repetition =
      <xsl:value-of select="$repetition" />
      ;
    </xsl:if>
    <xsl:apply-templates select="SubOper" />
    long endTime =System.currentTimeMillis(); System.out.println( "
    <xsl:value-of select="./ArchitecturalParameters/Name" />
    (): " + (endTime - startTime)); }
  </xsl:template>
- <xsl:template match="SubOper">
    <xsl:variable name="dbQuery" select="./ArchitecturalParameters/QueryType" />
    <xsl:variable name="remMethod" select="./ArchitecturalParameters/UsingRemMethod" />
  - <xsl:choose>
    - <xsl:when test="$dbQuery">
        <xsl:apply-templates mode="dbRequest" select="." />
      </xsl:when>
    + <xsl:when test="$remMethod">
    </xsl:choose>
  </xsl:template>
- <xsl:template mode="dbRequest" match="SubOper">
    try {
    <xsl:apply-templates mode="composeQuery" select="." />
    } catch (Exception ex) { ex.printStackTrace();}
  </xsl:template>
```

(b)

Figure 5.4. Argo/MTE meta-model and code generation scripts

Figure 5.4(b) illustrates the refactored test bed generation script (an XSLT script) for abstraction RemoteObject. The script merges the common parts of all the ArchOperHost-related scripts (as highlighted in grey in Figure 5.4(b)). It points/directs to the middleware-specific XSLT files when necessary (e.g. highlighted in cyan in Figure 5.4(b)). It provides java code for the ArchOperHost and its operations (ArchOperations) in the appropriate templates (e.g. the area highlighted in yellow).

The e-commerce meta-model provides *critics* to validate the architecture model and coordinate the test bed generation. For example, in Figure 5.2, the "registerCustomer" operation of the "CustomerRegistrationPage" wants to use the service "doRegister" of the "EcoinInterface", which actually assumes that the operation "registerCustomer" requires the existence of the remote service "doRegister". If the remote service "doRegister" does not exist, the model will not be able to generate a valid test bed. At this stage, Argo/MTE meta-model developers need to hard code a set of critics/logic for generating valid and functional test bed.

The e-commerce meta-model also needs to provide *test bed generation logic* to determine the structure of a test bed. More specifically, the logic needs to define if an abstraction should generate a folder of files or a single file, and what files a folder should contain. At this stage, the logic needs to be hard-coded in the Argo/MTE tool.

5.3.2 Generated test bed

Figure 5.5 illustrates the structure of the test bed generated from the Argo/MTE NetPay architecture design (refer to Figure 5.2). In this case, the *test bed generation logic* defines that each instance of domain-specific meta-types "DBaseServer" and "AppServer" generates a folder of files. So the test bed contains two DBaseServer folders, including BrokerDBServer and VendorDBServer, and six AppServer folders, including BrokerServer, EnewsServer, RemoteArticleManagerServer, RemoteCustomerManagerServer, RemoteEcoinManagerServer, and RemoteTandIManagerServer (left of Figure 5.5). The supportFiles folder contains a set of pre-built files to work with the generated files (test bed).

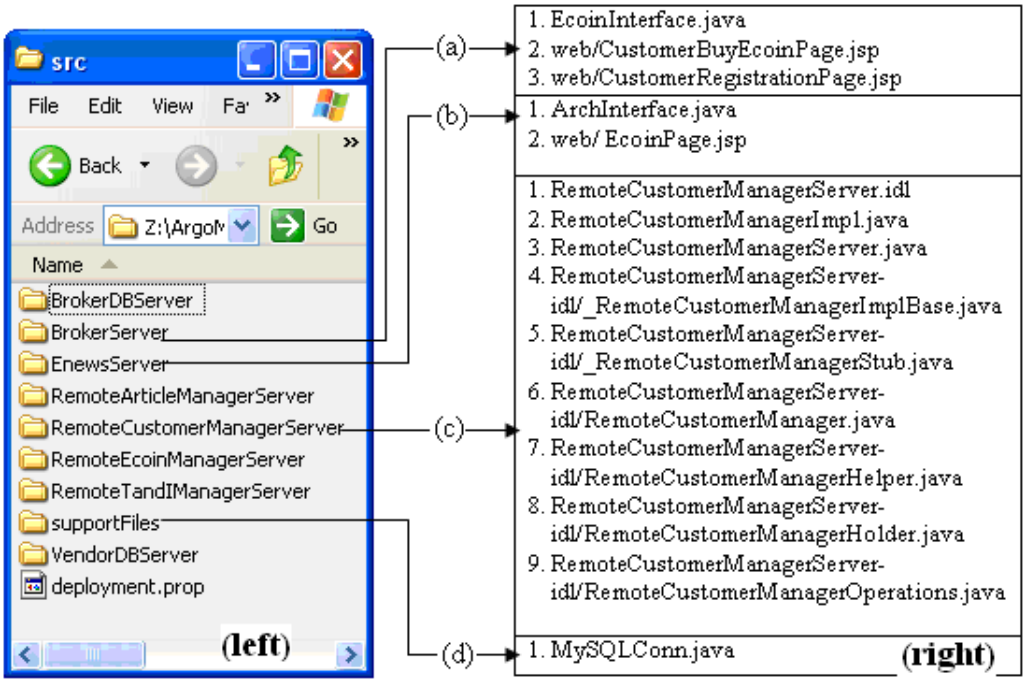


Figure 5.5. Structure of the generated NetPay test bed

Folder BrokerServer contains (a) the EcoinInterface.java that encodes the "EcoinInterface" of Figure 5.2. The CustomerBuyEcoinPage.jsp and CustomerRegistrationPage.jsp encode the "CustomerBuyEcoinPage" and the "CustomerRegistrationPage" of Figure 5.2 respectively. Folder EnewsServer contains (b) the ArticleInterface.java file that encodes the "ArticleInterface" of Figure 5.2; and the EcoinPage.jsp encodes the "EcoinPage" of Figure 5.2.

In the NetPay architecture design of Figure 5.2, the "RemoteCustomerManagerServer" is a CORBA server. Figure 5.5 shows (c) that folder RemoteCustomerManagerServer contains the RemoteCustomerManagerServer.idl (CORBA IDL file), the RemoteCustomerManagerServer.java (CORBA server main program), and the RemoteCustomerManagerImpl.java (CORBA remote object program). More CORBA-related files are generated in the folder of RemoteCustomerManagerServer-idl. Folders RemoteArticleManagerServer, RemoteEcoinManagerServer, and RemoteTandIManagerServer contain the similar files as folder RemoteArticleManagerServer.

Figure 5.5 also illustrates that Argo/MTE provides a set of pre-built support files for large distributed applications to make test bed fully functional (d). Support files, such as MySQLConn.java, are independent on the software system under modelling.

The CustomerRegistrationPage.jsp in the folder BrokerServer of Figure 5.5 is illustrated in Figure 5.6. The highlighted cyan areas show the functional information of the JSP file. The highlighted grey area in line 11 shows that this JSP file uses java bean class BrokerServer.EcoinInterface that encodes the remote object "EcoinInterface" of the "BrokerServer" of Figure 5.2. The highlighted grey area in line 26 shows that this JSP uses service "doRegister" of remote object "EcoinInterface" of Figure 5.2.

```
2  <HTML>
3      <HEAD>
4          <TITLE><xsl:value-of select="./ArchitecturalParameters/Name"/></TITLE>
5      </HEAD>
6      <H3><xsl:value-of select="./ArchitecturalParameters/Name"/></H3>
7      <BODY bgcolor="white">
8      <%@ page import="java.io.PrintStream,java.io.FileOutputStream"%>
9      <%@ page import="BrokerServer.*"%>
10     <jsp:useBean id="EcoinInterface" scope="session"
11                                          class="BrokerServer.EcoinInterface"/>
12     <%
13         PrintStream out_stream = null;
14         try{
15             out_stream = new PrintStream( new FileOutputStream(
16                                     "CustomerRegistrationPage result.txt"));
17             System.setOut(out_stream);
18         }
19         catch(Exception exp){System.out.println("Error in EcoinInterface.java");}
20         boolean recordingTime =true;
21         long startTime = System.currentTimeMillis();
22         int repetition =20;
23         try{
24             for(int i=0; i<repetition; i++)
25             {
26                 EcoinInterface.doRegister();
27             }
28         }
29         catch(Exception exp){System.out.println("Error in EcoinInterface.java");}
30         long endTime =System.currentTimeMillis();
31         System.out.println( " Do Registration: " + (endTime - startTime));
32     %>
33         registerCustomer takes time <%=(endTime-startTime)>;
34     </body>
35 </HTML>
```

Figure 5.6. CustomerRegistrationPage.jsp

```
13 public class RemoteCustomerManagerImpl extends
14         RemoteCustomerManagerServer_idl._RemoteCustomerManagerImplBase
15 {
16   public void insertCustomer(int  param2,String paraml)
17   {
18     boolean recordingTime =true;
19     long startTime = System.currentTimeMillis();
20     int repetition =10;
21     try {
22       for(int i=0; i<repetition; i++)
23       {
24         MySQLConn.getInstance().executeQuery("insert into customer
25                     values('1000', 'JohnDoe','student', '1234567')");
26       }
27     } catch (Exception ex) { ex.printStackTrace();}
28     long endTime =System.currentTimeMillis();
29     System.out.println( " insertCustomer(): " + (endTime - startTime));
30   }
31
32   public void selectCustomer(int  param2,String paraml)
33   {
34     boolean recordingTime =true;
35     long startTime = System.currentTimeMillis();
36     int repetition =10;
37     try {
38       for(int i=0; i<repetition; i++)
39       {
40         MySQLConn.getInstance().executeQuery("select * FROM customer WHERE ID = 1");
41       }
42     } catch (Exception ex) { ex.printStackTrace();}
43     long endTime =System.currentTimeMillis();
44     System.out.println( " selectCustomer(): " + (endTime - startTime));
45   }
46
47   public void updateCustomer(int  param2,String paraml)
48   {
49     boolean recordingTime =true;
50     long startTime = System.currentTimeMillis();
51     int repetition =10;
52     try {
53       for(int i=0; i<repetition; i++)
54       {
55         MySQLConn.getInstance().executeQuery("update customer set column2
56                                             =\""+"password"+"\""+" WHERE ID = 1");
57       }
58     } catch (Exception ex) { ex.printStackTrace();}
59     long endTime =System.currentTimeMillis();
60     System.out.println( " updateCustomer(): " + (endTime - startTime));
61   }
62 }
```

Figure 5.7. RemoteCustomerManager.java

Figure 5.7 illustrates part of the generated RemoteCustomerManagerImpl.java of the "RemoteCustomerManager" of Figure 5.2. The highlighted grey code in lines 16, 32, and 47 encode the three operations of this remote object. The highlighted yellow code in lines 24 and 25 implements a

simple database query. A test bed does not focus on the complexity of business logic but focus on the performance-intensive parts (e.g. establishment of remote connections, inter-server requests and services) of a distributed web system. The highlighted cyan code from line 18 to 20 and from line 28 to 29 is performance evaluation code recording the time consumed for this service to complete.

Figure 5.8 illustrates a sample support file, MySQLConn.java, which is used by test beds to handle the establishment of database connection and disconnection, and execute database queries.

```
5 public class MySQLConn
6 {
7
8     protected boolean connected = false;
9     private Connection con;
10    private  String driverName = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
11    private String dbName = "jdbc:odbc:broker";
12    private static MySQLConn instance = new MySQLConn();
13
14    public MySQLConn()
15    {}
16
17    public static MySQLConn getInstance()
18    {
19        return instance;
20    }
21
22    public void connect(String database) throws Exception
23    // connect to MySQL database server
24    {
25        if(connected)
26            return;
27        else {
28         Class.forName(driverName);
29         con = DriverManager.getConnection("jdbc:microsoft:sqlserver://ycai003:
30                   1433;DatabaseName="+database+";user=honghong;password=honghong");
31         con.setAutoCommit(true);
32         connected = true;
33       }
34   }
35
36   public void disconnect() throws Exception
37   // disconnect from MySQL database server
38   {
39       con.close();
40       connected = false;
41   }
42
43   public boolean isConnected()
44   {
45       return connected;
46   }
47
48   public synchronized void execute(String sql) throws Exception
49   {
50       Statement stmt = con.createStatement();
51       stmt.execute(sql);
```

Figure 5.8. MySQLConn.java

5.4 Test bed compilation, deployment, execution, and result collecting

Argo/MTE uses the Ant tool to automate the tedious process of test bed compilation, deployment, execution, and result collecting (Cai et al, 2004; Cai and Grundy et al, 2004). Besides the test bed source code, an Argo/MTE architecture model also generates Ant build files for various tasks involved in the evaluation process. An Argo/MTE meta-model must provide the rules, scripts, and logic for Ant build file generation. As is illustrated in Figure 5.9, each meta-model abstraction provides a textual rule to describe which targeted concept to generate Ant files for (e.g. the concept of application server of CORBA, server-side remote object of CORBA, client of CORBA). The abstraction provides XSLT scripts to define how to generate functional Ant build files for the targeted concept. The meta-model itself provides critics to coordinate individual Ant build files for the various tasks involved in the performance evaluation process.

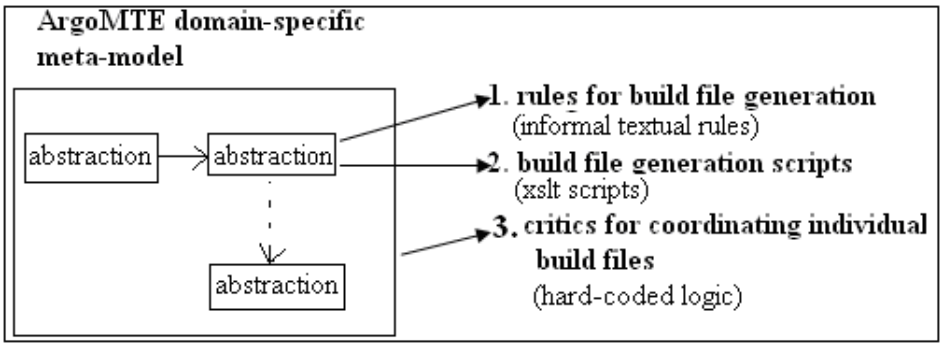


Figure 5.9. Argo/MTE meta-model supports build file generation scripts and logic

An abstraction's build file generation rules and scripts are based on the used extended-UML meta-type (refer to section 4.3.1 of chapter 4) and the used middleware technologies. For example, in the e-commerce meta-model in Figure 5.10, abstraction AppServer (with UML meta-type ArchHost) (1) has five properties (2). The property "Type" contains the information of the supported middleware technologies (the targeted technologies to be evaluated) including CORBA, RMI, and EJB (3). The rule for designing the build file generation scripts for the abstraction is "*the combination of ArchHost and CORBA middleware must target at generating Ant file for CORBA server; the combination of ArchHost and RMI middleware must target at generating Ant file for RMI server; and the combination of ArchHost and EJB middleware must target at generating Ant file for EJB server*". The rule then leads to the design of the build file generation scripts including, corba_server_compile.xsl,

corba_server_deploy.xsl, rmi_server_compile.xsl, and rmi_server_deploy.xsl. For each task involved in the evaluation process (e.g. compilation, deploying, running, and result collecting), the individual scripts of an abstraction need to be coordinated to synthesize the build files for the whole test bed application. Currently, a domain-specific meta-model provides hard-coded logic to coordinate individual Ant build files and synthesize build files for the test bed.

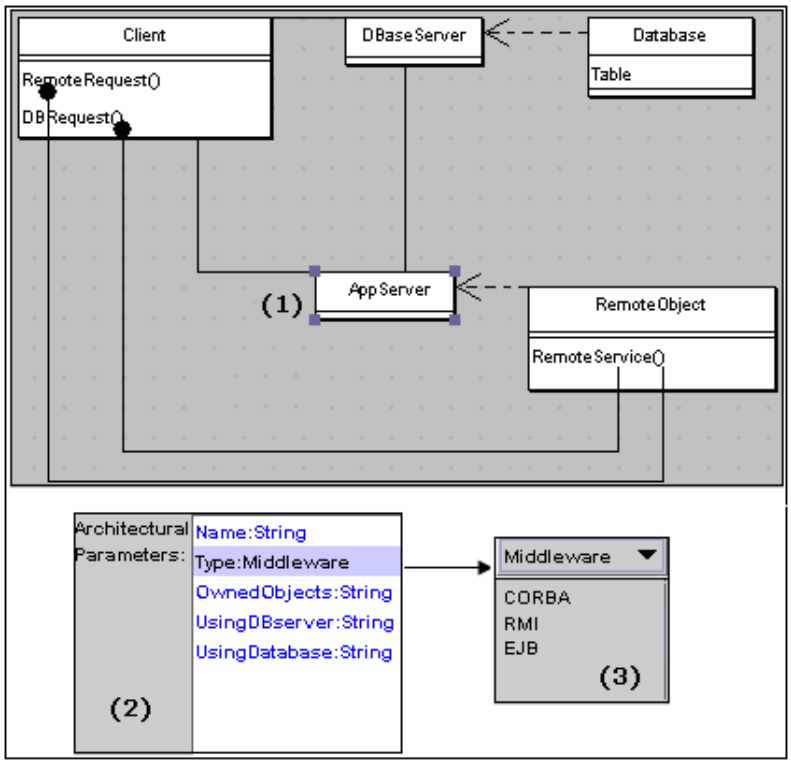


Figure 5.10. Sample meta-type and the middleware technologies

Figure 5.11 illustrates part of a synthesized Ant build file for the test bed compilation. The file is consisted of the individual compiling information of each application server (highlighted in grey, e.g. BrokerServer, RemoteEcoinManagerServer, RemoteArticleManagerServer), as well as the dependency among them (highlighted in cyan) such as the compilation of BrokerServer depends on the compilation of RemoteCustomerManagerServer and RemoteEcoinManagerServer.

Figure 5.11. Sample NetPay Ant build file compile.xml

Figure 5.12 illustrates how Argo/MTE automates deployment, execution, and result collecting. To get ready for deployment, Argo/MTE uses the generated Ant build files to organize the test bed and its support information into folders (Figure 5.12(a)) (e.g. deploy_BrokerServer, deploy_EnewsServer), and package them into deployable parts (e.g. BrokerServer.jar, EnewsServer.jar). Argo/MTE then uses the Ant tool to deploy the packaged parts on the active hosts (computers that can run the deployed test bed parts) through the directed arrows (1) in Figure 5.12(b). After all parts are deployed, a fully functional web application is established. The clients can access the running test bed and launch requests. The performance evaluation results of the interested server-side operations are collected and sent back to the performance evaluation center – the place that runs Argo/MTE through the directed arrows (2).

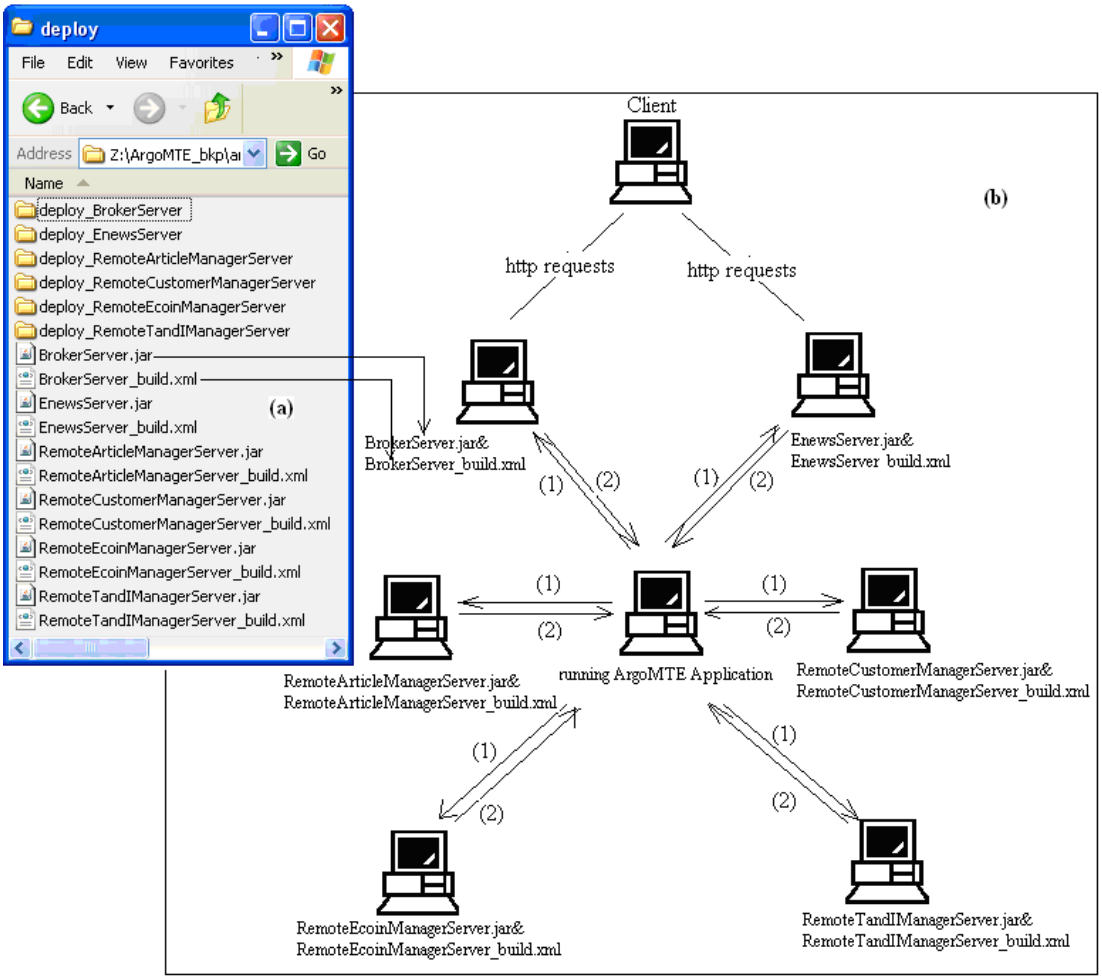


Figure 5.12. Argo/MTE deploys and runs a test bed, and collects testing results

5.5 Sample Performance Evaluation Results

Argo/MTE can be used to compare the performance of a legacy system with that of an architecture model. Table 5.1 and Table 5.2 present sample evaluation results towards both a legacy NetPay system as well as a NetPay test bed. The legacy system is done by a novice CORBA java programmer with little consideration of performance optimization. Referring to the architecture design in Figure 5.2, Table 5.1 presents the evaluation results of service "doRegister" of remote object "EcoinInterface". Table 5.2 presents the evaluation results of service "doDisplay" of remote object "ArticleInterface". When evaluating how a remote service responds a client request, we launch multiple client threads (to simulate multiple users) towards the targeted remote service. The targeted remote service can be repeated multiple times (to simulate complicated business logic such as multiple database queries, refer to line 20 Figure 5.7). Then we will check that averagely how long it takes for each client to get the response of the targeted remote service.

Table 5.1 records the results of 15 test runs. For each test, we used 20 threads to represent 20 client requests for the remote service "doRegister" of remote object "EcoinInterface". Through the 15 tests, the test bed performed well and responded less than 1093 ms (occurred in test 13). The tests on the legacy system brought interesting results. After two tests on the legacy system, the third and fourth tests saw the sharp drop of the performance. It took unbearable 22870ms (occurred in test 3) and 22683 ms (occurred in test 4) for a client to get response. In this situation, all the servers needed to be restarted to carry on the following tests. After restarting all the involved servers, test 5 to test10 saw the reasonable response from the server until the server performance dropped sharply again in tests 11 and 12 and all servers needed to be restarted again. Figure 5.13 visualizes the Table 5.1 information in a MS Excel chart. It shows that overall, the test bed shows stable performance through the 15 tests; and the legacy system takes more time to respond. While it is reasonable for a real system to perform slightly poorer than a test bed, because of more complicated business logic and large amount of real data passed around, the reasons for the abnormal hike of the response time in tests 3, 4, 11, and 12 need to be found out. The legacy system was then reviewed. It was found out that instead of using the same established client server remote connection, the legacy system tried to establish a client server remote connection in multiple places. The legacy system tried to establish much more remote connections than it really needed, which made the legacy system very remote-connection-intensive and easily brought down the servers. After the performance evaluation, the legacy NetPay system was then improved.

Test	Value of test bed(ms)	Value of real system (ms)
1	1015	4060
2	1016	3556
3	984	26870(suddenly, the performance drops steeply)
4	984	22683 (at this stage, all servers need to be restarted including tomcat, name server, and all CORBA servers)
5	1062	4760
6	999	3970
7	859	4021
8	859	5061
9	1078	4841
10	968	3250
11	953	22573 (suddenly, the performance drops steeply)
12	984	22948 (at this stage, all servers need to be restarted including tomcat, name server, and all CORBA servers)
13	1093	3760
14	1062	4970
15	1125	4021

Table 5.1. Evaluation results of remote service "doRegister"

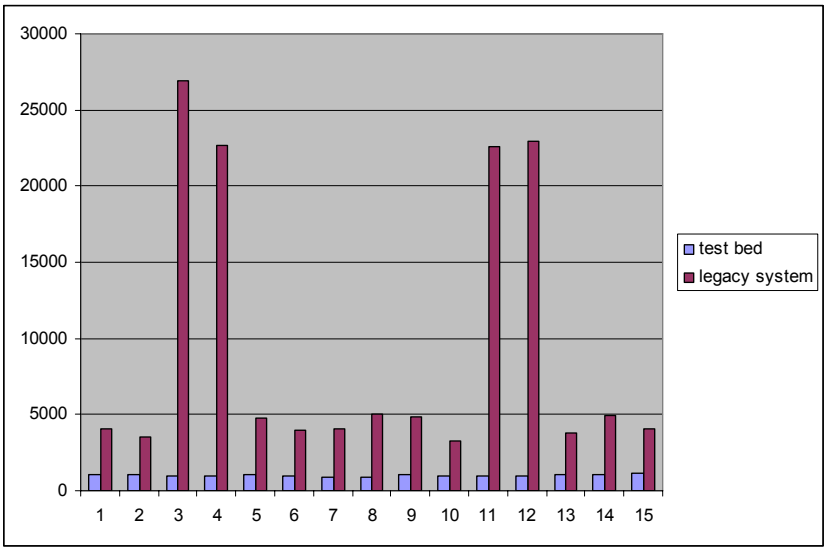


Figure 5.13. Evaluation results of remote service "doRegister"

Test	Value of test bed (ms)	Value of real system (ms)
1	4972	14530
2	5967	38590 (suddenly, the performance drops steeply)
3	5185	34152
4	30890	34543
5	30906	34168
6	30890	33927(at this stage, all servers need to be restarted including tomcat, tnameserver, and all CORBA servers)
7	4904	13250
8	4873	38600
9	4686	34471
10	14780	33829
11	30719	34547
12	30891	34173 (at this stage, all servers need to be restarted including tomcat, tnameserver, and all CORBA servers)
13	5013	16445
14	6051	38990
15	5870	36540

Table 5.2. Evaluation results of process "doDisplay"

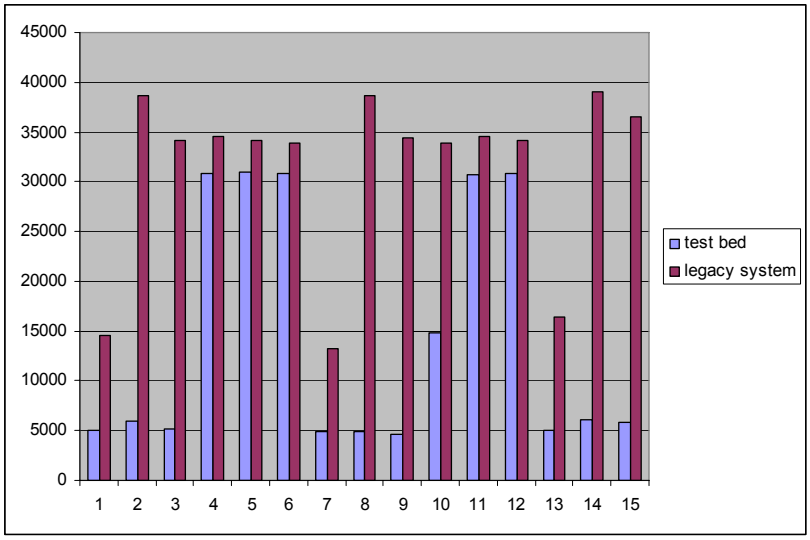


Figure 5.14. Evaluation results of "doDisplay"

Table 5.2 records the results of 15 tests against the test bed and the legacy system (before it was improved). For each test, we used 20 threads to multiple 20 client requests for remote service "doDisplay" of remote object "ArticleInterface". Through the 15 tests, both the test bed and the legacy system suffered abnormal performance and the servers needed to be restarted. After three tests on the test bed, the fourth, fifth, and sixth tests saw the sharp drop of the performance. It took unbearable

30890ms (in test 4) and 30906 ms (in test 5) for a client to get response. In this situation, the servers did not work appropriately at all and needed to be restarted to carry on the following tests. After restarting all the involved servers, test 7 to test 9 saw the reasonable response until the server performance dropped sharply again in tests 11 and 12, where all servers needed to be restarted again. The legacy system performed even worse than the test bed. The servers could reluctantly work for one test (e.g. tests 1, 7, and 13) after each restarting of the servers. Figure 5.14 visualizes the Table 5.2 information in a MS Excel chart. The unstable performance of the test bed means: 1) the architecture design is not stable itself (for example, too many inter-server communications); and 2) more robust code needs to be generated even at architectural level. The evaluation results upon the test bed require the improvement of both the architecture design, as well as the code generation scripts of the used meta-model. An improved test bed can be used as benchmark to guide the improvement of the legacy system. For the legacy system, on top of the possibly poor architecture design, the unnecessary intensive requests for remote-connection and too many inter-server communications cause the even worse performance than that of the test bed.

5.6 Summary

The case study introduces details of how Argo/MTE specifies and evaluates software architecture. In the case study, the complicated NetPay system contains 6 remote application servers and two database servers. The 6 remote application servers are heterogeneous, e.g. the “BrokerServer” and “EnewsServer” are JSP-enabled web servers, and all others are CORBA application servers. Argo/MTE generates the complicated and fully functional NetPay test bed including JSP pages, CORBA-related java files, and Ant build files. The test bed is intended to encode the best-of-practice implementation of the architectural-influenced parts of a software system, which is useful in both forward engineering and reverse engineering. In forward engineering, architects can send designers a test bed as functional prototypes to follow. In reverse engineering, a test bed can be a benchmark for a legacy system. In fact, in the case study, NetPay test bed did act as a benchmark for the legacy NetPay system and helped finding out coding problems that caused the poor performance of the legacy NetPay. The case study demonstrates the strength of Argo/MTE to work with complicated software systems.

Chapter 6 - Review of Marama Meta-Tool and MaramaMTE

MaramaMTE+ is the second main project in the thesis. Projects Marama meta-tool and MaramaMTE are two prerequisites for understanding the MaramaMTE+ project. The Marama meta-tool, developed by John, G. et al (Marama meta-tool, 2007), is an Eclipse-based tool set that supports efficient development of domain-specific tools. The MaramaMTE project, developed by John, G. et al (MaramaMTE, 2007) redevelops the technology of test bed generation and performance evaluation by using the Marama meta-tool, and supports Form Chart modeling. This chapter reviews the basic concepts of the Marama meta-tool and the MaramaMTE project to get ready for understanding the MaramaMTE+ research.

6.1 Marama meta-tool

The Marama meta-tool is an Eclipse-based tool set to support efficient development of domain-specific tools (Marama meta-tool, 2007). It provides a framework to catch the common issues in tool development, including: visual notations, underlying modeling elements, events triggered by modeling elements, users, and visual notations. The Marama meta-tool is intended to support experienced tool developers to quickly construct the basic visual modelers of a domain-specific modeling tool within one day, and then they can move to develop specific/advanced facilities of the tool such as specification of backend code generators, complex editing, or behavioral constraints.

Figure 6.1 shows how the Marama meta-tool realizes domain-specific visual modeling tools. A tool developer specifies an intended domain-specific tool (such as Argo/MTE-like tool) in an XML file (containing modeling elements, associations, model driven events, user defined events, etc) using a text editor or third party modeling tool (1). The Marama meta-tool Eclipse plug-ins read in the tool specification to configure the intended visual modelers (2). The meta-tool creates a data model and one or more graphical editors conforming to the tool specification (3). The constructed domain-specific modeling tools use Eclipse GEF to realize the graphical editors, and use Eclipse EMF to represent model

and diagram state. By using the OMG XMI common exchange format (via EMF's built-in capabilities), the meta-tool saves or loads model and diagram state to XML files or an XML database (4) (Grundy and Hosking et al, 2006).

The Marama meta-tool has been used to implement a wide range of domain specific tools, such as Marama Torua (MaramaTorua, 2007) and MaramaMTE (MaramaMTE, 2007). In each case users have been able to rapidly implement basic modelers for the complex Eclipse-based tool, with typically a several hundred-fold increase in productivity over coding the tool with the standard GEF and EMF frameworks.

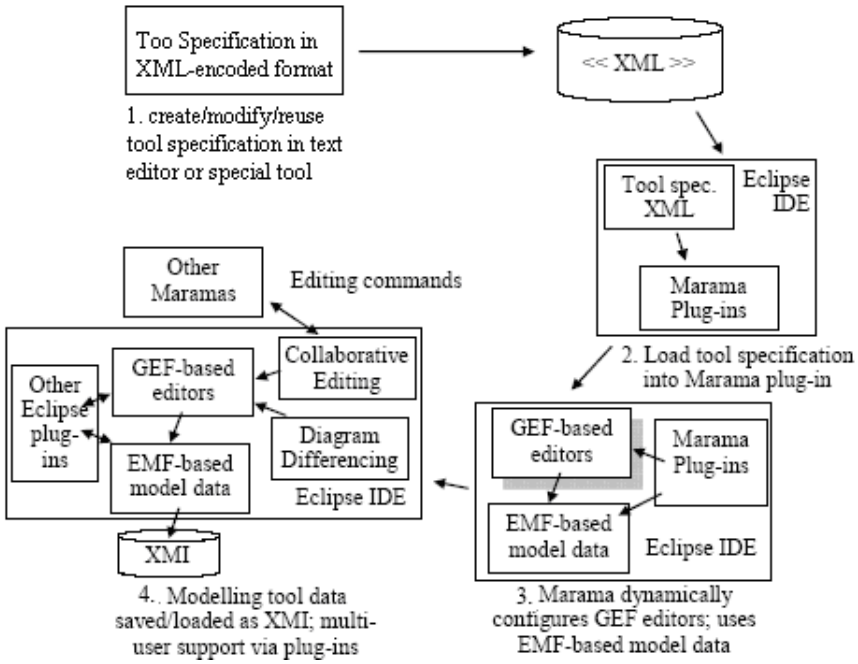


Figure 6.1. The Marama approach to realizing Eclipse-based visual language tools (Grundy and Hosking et al, 2006)

6.2 MaramaMTE

The MaramaMTE tool, developed by John G. et al (MaramaMTE, 2007; Draheim et al, 2006), is a domain-specific tool implemented using the Marama meta-tool. It redevelops the main features of Argo/MTE and supports test bed generation and performance evaluation of middleware-based software systems. The improvement of MaramaMTE from Argo/MTE lies in two aspects:

1) *The Marama meta-tool generated a large part of the MaramaMTE tool, while Argo/MTE required tedious programming to extend ArgoUML*

Figure 6.2, Figure 6.3, and Figure 6.4 show the main activities involved in developing the MaramaMTE tool. Firstly, as illustrated in Figure 6.2, a MaramaMTE domain-specific meta-model (previously Argo/MTE domain-specific meta-model, refer to section 4.3.2, Chapter 4) is specified in the *tool definer* of the Marama meta-tool. This contains abstraction entities such as *ApplicationClent* and *RemoteObject* (green rectangles), and abstraction associations such as *ClientServer* and *ServerObject* (pink rectangles).

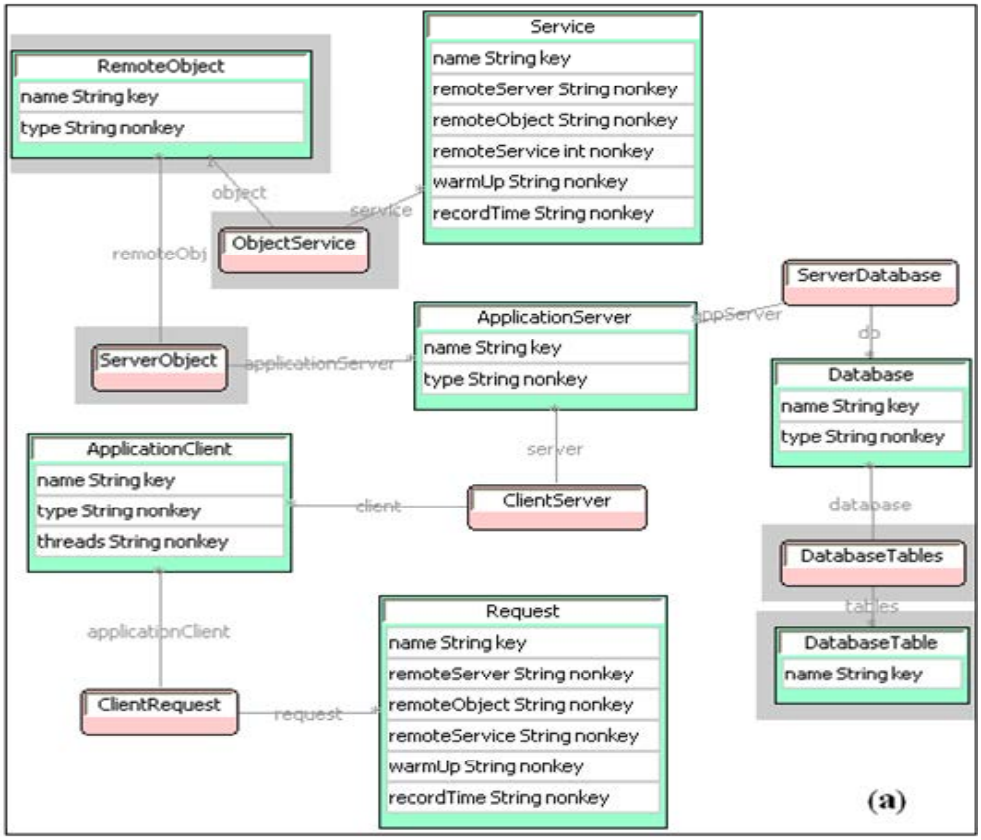


Figure 6.2. MaramaMTE meta-model

Secondly, as illustrated in Figure 6.3, visual notations for each abstraction entity and abstraction association are specified using the *shape definer* of the Marama meta-tool. For example, the cyan rectangle in Figure 6.3 is the visual notation of abstraction *ApplicationClient* in Figure 6.2. The grey rectangle containing properties in Figure 6.3 is the visual notation of abstraction *Request* in Figure 6.2.

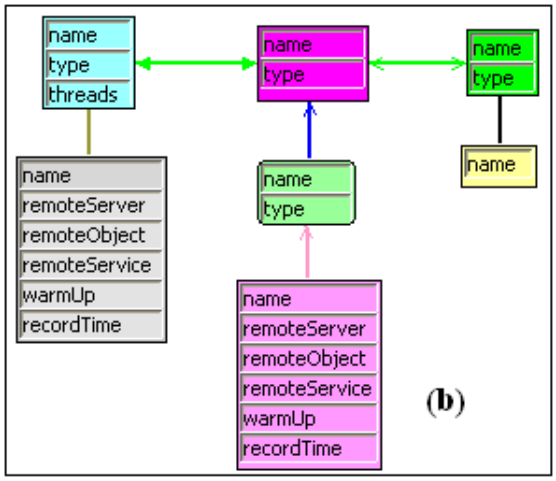


Figure 6.3. visual notations for the MaramaMTE meta-model

Thirdly, as illustrated in Figure 6.4, users construct view/diagram types by mapping shapes, connectors, and event handlers to appropriate model entities and associations using *view type definer* of the Marama meta-tool. Figure 6.4 only illustrates part of the mapping between meta-model abstractions and their visual notations. The basic facilities of the MaramaMTE tool, including drawing, event triggering, model saving and loading, are then complete. Tool developers can then experiment with and tailor the basic tool to support its advanced features such as detailed diagram editing (such as hide or show properties of an abstraction), code generation, event handling, and model checking.

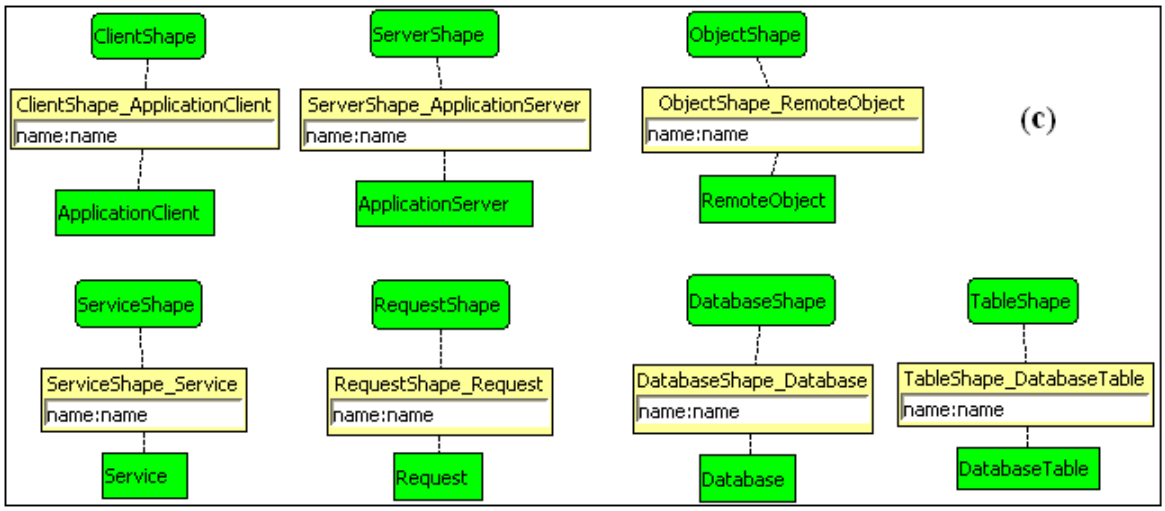


Figure 6.4. MaramaMTE view type

2) *The Marama meta-tool provides a good platform to integrate domain-specific models and tools at both framework level as well as functional level, which helps to integrate MaramaMTE with other domain-specific modeling technologies.*

The MaramaMTE technology needs to leverage the strength of other modeling technologies to improve its quality, usability, and maintainability. For example, like SoftArch/MTE and Argo/MTE, MaramaMTE architecture models consist of components with properties, and the property values need to be constrained in order to generate a valid functional test bed. A sample structural constraint in an e-commerce architecture model could be that "*when a client launches a RemoteRequest to a RemoteService, the value of property 'RemoteService' of the RemoteRequest must be equal to the value of property 'Name' of the interested RemoteService*". Those constraints are tedious to develop, and relatively informal. Both SoftArch/MTE and Argo/MTE hardcode those constraints, and tool users have a heavy workload of programming with the tool-API.

MaramaTatau (Liu et al, 2007) is an add-on mechanism to the Marama meta-tool set, which allows users to specify OCL-styled structural constraints upon models (e.g. MaramaMTE domain-specific meta-models). MaramaTatau uses the EMF OCL (MDT, 2008) framework to implement a dynamic compiler and interpreter for OCL specifications. As all domain-specific tools built in the Marama meta-tool-based (e.g. MaramaMTE) are EMF based, this means the EMF-based MaramaTatau can be easily integrated with those domain-specific tools (e.g. MaramaMTE) to provide them with a formalized approach to define and manage structural constraints.

MaramaMTE also extends its applicable scope via integration with other Marama-based domain-specific tools. For example, the integration of MaramaMTE with service composition tools such as BPMN and ViTABal-WS (Grundy, Hosking, and Li et al, 2006) allows users to evaluate performance of service-oriented architectural models.

6.3 MaramaMTE supporting Form Chart modeling

MaramaMTE also extends its application scope to support Web Application Load Testing (WALT) via integration with Form Chart modeling (Draheim et al, 2006). A *form chart* model is a technology-independent bipartite state diagram used to simulate user behavior. It describes, at high level, what the user sees as system output, and what he or she provides as input to the system (Draheim et al, 2006).

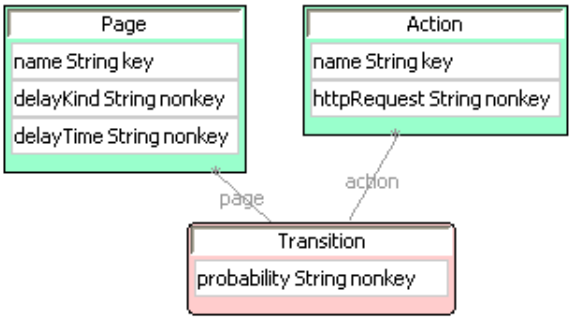


Figure 6.5. The simplified FormChart meta-model

Figure 6.5 shows the simplified Form Chart meta-model. The meta-model consists of abstractions Page, Action, and Transition. Abstraction Page represents possible states of a website; abstraction Action represents website server side information, including: components, their behavior, and their response to requests; and abstraction Transition represents association between pages and actions. A Form Chart model, an instance model of the Form Chart meta-model, captures the structure of the target website from users' perspective and can be augmented with probabilities to capture client behavioral interaction with web (Draheim et al, 2006). The original Form Chart models have limited intentions. MaramaMTE supports a semantically-improved version of Form Chart models, *stochastic Form Chart models*, to formally model realistic user behavior. A stochastic form chart model extends a basic model with stochastic functions to describe navigation, time delays, and user input (Draheim et al, 2006).

Considering about the java Pet Store application (PetStore, 2002), an online store where users can buy pets on the internet. Figure 6.6(a) is the simplified MaramaMTE architecture model of the Pet Store application. This architecture design, at a high level of abstraction, shows the main components of the Pet Store system, including client-side components and the main server-side components (including the application server "Pet StoreAppServer" and its objects "SignOn", "MainServlet", and "RequestProcessor"). The architecture model is focused on the server side multi-tier structure, and can be richly enhanced by providing operations for each functional component and setting up a wide variety of component properties. However, the client side model (the "user" component in Figure 6.6(a)) contains little more than a sequence of remote requests (not shown). MaramaMTE leverages the strength of the Form Chart modeling to provide effective, realistic user behavior model to enrich the original MaramaMTE test beds.

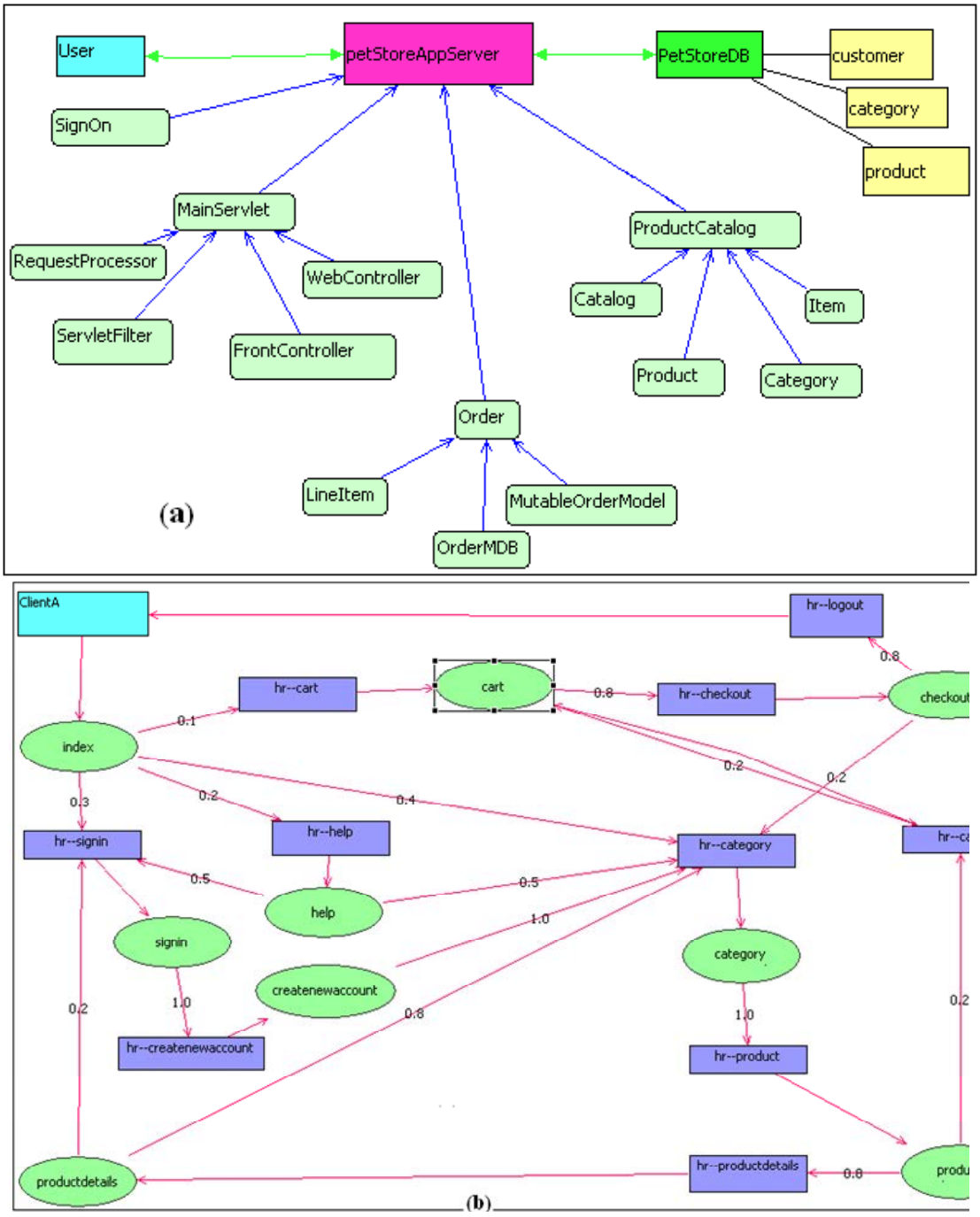


Figure 6.6. (a) High-level view of Pet Store software architecture; (b) sample Stochastic Form Chart loading

Figure 6.6(b) shows a sample MaramaMTE form chart model diagram representing user behavioral interaction with the Pet Store. The model contains: a starting state (top left rectangle: "ClientA"),

various Form Chart pages (ovals, including “index”, “cart”, signin”, “productdetails”), and actions (rectangles, including “hr--signin”, “hr--cart”) allowing movement between the Form Chart pages. The engineer captures probabilities of moving from a given web form via actions e.g. from “index” to “signin” 0.3; from “index” to “help” 0.2. For each Form Chart page, various properties of the page are captured, such as the URL address, request delay kind, and request delay time (Draheim et al, 2006).

6.4 Summary

The Marama meta-tool supports efficient construction of basic features of domain-specific tools such as visual notations, underlying modeling elements, and events triggered by modeling elements, users, and visual notations. It captures the common tasks of developing domain-specific visual modeling tools; provides a tool definer, a shape definer, and a view type definer to support users to specify an intended tool; and generates the basic visual modelers of the intended tool. The Marama meta-tool makes it easy to redevelop the technology of test bed generation and performance evaluation in MaramaMTE. It also improves the performance evaluation technology through the integration with the Form Chart modeling technology and other software modeling technologies.

Chapter 7 - MaramaMTE+, Synthesizing Client Load Models for Performance Engineering via Web Crawling

MaramaMTE+ extends MaramaMTE to support effective web load testing through web reverse engineering. This chapter (1) presents the motivation for MaramaMTE+; (2) introduces how MaramaMTE+ reverse engineers web applications and supports web load testing; and (3) describes the design and implementation of the MaramaMTE+ toolset, and compares the efficiency of MaramaMTE+ with that of traditional web load testing technologies (e.g. Apache JMeter and Microsoft Web Application Stress tool).

7.1 Introduction of MaramaMTE+

MaramaMTE+, a continuation of MaramaMTE, supports web load testing through reverse engineering a running website. MaramaMTE+ uses a web crawler to extract structural information from a target website to generate the structure of a Form Chart model; users manually augment the Form Chart model with appropriate property values; the synthesized Form Chart model is then used to generate testing plans for a third party load testing tool such as Apache. JMeter.

7.2 Motivation and related work

MaramaMTE integrates traditional architecture modeling with Form Chart modeling (MaramaMTE, 2007; Draheim et al, 2006). It prompted two interesting problems: 1) MaramaMTE users must construct Form Chart models manually and modify them incrementally to reflect changing website structure and user behavior; 2) In MaramaMTE, Form Chart models only generate Java client load test bed to enrich the server-focused performance evaluation technology. As a formal analysis technology, Form Chart modeling has the potential to generate test plans for other web load testing technologies; and the potential needs to be explored.

The manual construction of a Form Chart model in MaramaMTE is both error-prone and time consuming, especially when large websites are being re-engineered. In addition, it is desirable to use Form Chart modeling to support more web load testing technologies. To reduce the bottleneck around user behavior model specification and improve load test technology integration, the following requirements for an improved tool (e.g. MaramaMTE+) are needed:

- to capture a realistic model of web application usage, an analysis model is needed to relate actual website structure (from a user perspective) to possible form-level interactions (via e.g. a Form Chart)
- the structure of the analysis model needs to be automatically generated from the target website instead of manually built
- tool support must allow users to easily change client load model testing parameter values then generate multiple testing plans and scripts automatically
- 3rd party load testing tools (e.g. JMeter) should be used leveraging their capabilities for large web application stress testing, and their client loading plans and scripts (e.g. JMeter testing plans) need to be generated from analytical formal models (e.g. Form Chart load models)
- the tool should be well-integrated within a generic performance engineering environment (e.g. MaramaMTE), so a realistic client behavior model can influence the design of other parts of a software system (e.g. the server-focused software architecture)

A number of practical load testing tools have been developed for stress-testing web and other software applications. One example is Apache JMeter (Apache JMeter , 1999). JMeter offers both a textual and a GUI environment for users to construct testing plans and scripts. JMeter testing plans can be reasonably complex. They can simulate and analyze a variety of load scenarios to obtain a quantitative insight into a website's loading characteristics. However, JMeter does not support formal behavior analysis modeling, so users have to construct their testing plans based on ad-hoc testing goals. Microsoft's WebApplication Stress Tool (WAS) (Microsoft, 2002) is a simple load and stress testing tool that can be closely integrated with Visual Studio and other development environments. It does not support as complex testing plans as JMeter. It provides simple load modeling capability via a test setting wizard resulting in a testing plan with limited flexibility.

Several experimental web application testing tools have been developed that try to encapsulate load models in various ways (Barford et al, 1998; Denaro et al, 2004; Elbaum et al, 2003; Smith et al, 2005). These tools address the target system from web developer perspective; and they require web developers to construct performance models of the system under test (usually UML or similar architectural models or black-box services). Those models are normally part of the forward engineering, and are not directly related to the existing web applications. Some web load testing analysis tools, such as Surge (Barford et al, 1998), generate representative web requests that are based on analytical models of web use. Those analysis tools (e.g. Surge) rarely provide visual notations to relate the target website to web load analysis. Some web analysis tools try to obtain realistic testing plans by recording user behavior then replaying or analyzing it (Sprenkle et al, 2005). The recorded user behavior may be close to reality, but the raw behavior data needs to be abstracted to a suitable client loading model for the purpose of user behavior analysis and data exchange. No existing web application load testing tools provide the functionality demanded by the requirements described above.

7.3 MaramaMTE+ approach

Figure 7.1 illustrates how MaramaMTE+ uses a web crawler to automatically generate the structure of a Form Chart model; how the basic Form Chart model is manually augmented; and how to generate load testing plans and scripts from the Form Chart model. Initially the target website is crawled (1) by a 3rd party web crawler adapted for the purpose. The web crawler generates http request data from the target website (2), and this website structural data is stored in a database of possible requests to the target web site (3). The extracted http requests are used to synthesize an initial Form Chart model which is then imported into MaramaMTE+ (4) and a default layout applied to generate one or more Form Chart diagrams. This initial Form Chart model is then manually augmented by the performance engineer using the MaramaMTE+ Form Chart diagramming tools to specify probabilities and other stochastic parameters. The complete model may be versioned to allow variations of the parameters and user behavior to be modeled for comparison (5). Load testing plans are then generated for third party web application stress-testing tools (e.g. Apache JMeter or Microsoft Web Stress Tool) (6). In addition, MaramaMTE+ can generate server-side test beds of the target website. Stress-tests are run against the target web application, and results are collected by the stress-testing tool (either the MaramaMTE+ tool or other third party tools) (7). These test results are shown either inside MaramaMTE+ or via a third party visualization tool (8) and may be stored for future reference and comparison (9).

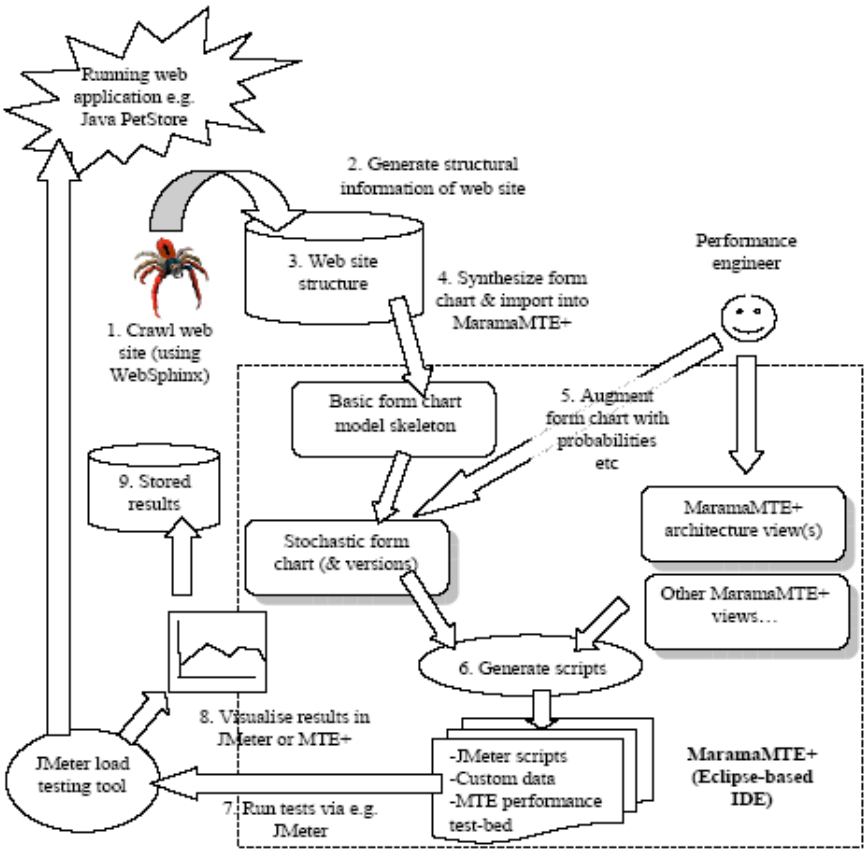


Figure 7.1. Crawling websites to extract Form Charts and generating stress-tests with MaramaMTE+ (Cai et al, 2007)

Web Crawlers have long been used to explore structure of websites from a user's perspective. MaramaMTE+ uses WebSphinx (Miller et al, 1998) to extract structural information from websites, including: the main screens, screen content, hyper links, and http requests plus parameters and values from a web application. The extracted information is collected into a crawler result database, which makes the website structural data available for further use. The website structural data can be retrieved from the database to generate the basic structure of a Form Chart model. The generated Form Chart needs to be manually augmented by adding additional data, such as action flow transition probabilities and MaramaMTE-specific code generation data (parameters and values). The enriched Form Chart model is a stochastic Form Chart containing sufficient information to generate load testing plans of the

targeted website, including: JMeter testing plans, Microsoft Web Stress Tests, and MaramaMTE+ thick client testing plans (Java applications).

7.4 Example usage

The Java Pet Store reference application (PetStore, 2002), a de facto benchmark application for performance evaluation technologies, is used as an example to illustrate MaramaMTE+ approach. From an end user (shopper) perspective, basic interactions with the Pet Store website include: users sign on; they browse the catalog; they buy pet(s) by putting them into a shopping cart; they check out; and receive purchase confirmation. It is obvious that the distribution of the types of user-website interactions is not linear. For example, "browse catalog" will be the most frequent interaction, just as in a real shop, there will always be more browsers than buyers. The "buy pet" interaction is likely to be more frequent than "check out", because a buyer may buy more than one pet before checking out. The stochastic Form Chart model(s) for the Pet Store application must capture these nuances. In addition, multiple models may be needed for different kinds of users e.g. business vs. personal shoppers or situations, e.g. Christmas vs. February, providing even more fine-grained client load modeling for the web application.

7.4.1 HTTP request extractions

MaramaMTE+ uses WebSphinx to extract Pet Store structural information into an http requests database. The user runs WebSphinx as a java application invoked from MaramaMTE+ (Figure 7.2(a)). The user supplies the target website address to the crawler (Figure 7.2(b)). The crawler explores Pet Store website to get information such as: main screens, hyper links among screens, and http requests and their parameters and values. MaramaMTE+ collects data into a purpose-built crawler request database. The crawler database contains tables to offer easy data access when the database is needed to generate Form Chart model. Figure 7.2(c) shows the "http request" table that holds http requests and their requested web pages, and Figure 7.2(d) shows the "page" table that mainly holds URL addresses of the crawled web pages.

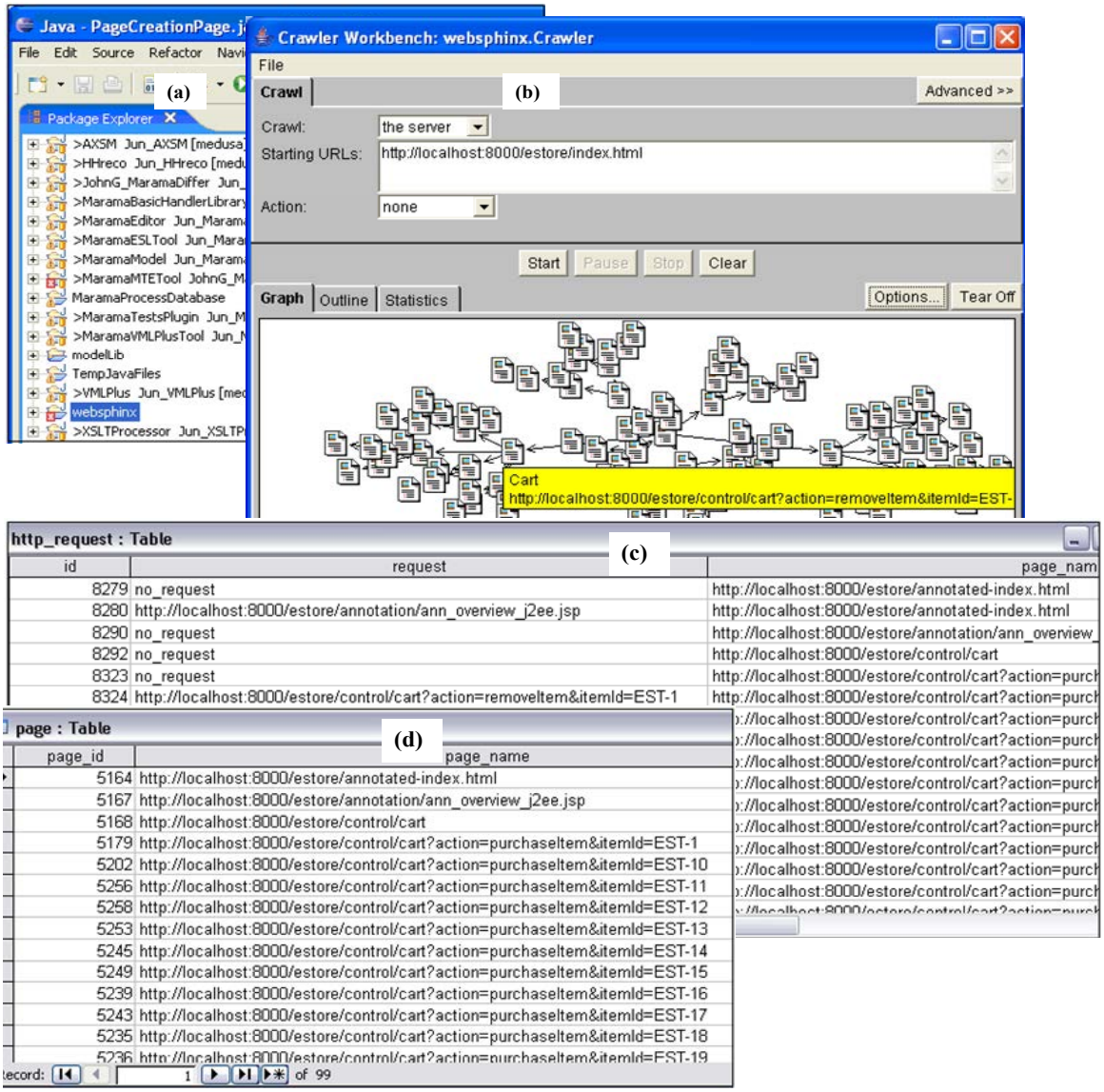


Figure 7.2. MaramaMTE+ using WebSphinx to extract structural information from the Pet Store web application

7.4.2 Form Chart extraction

MaramaMTE+ retrieves data from the database, and uses the data to generate structure of a Form Chart model. Figure 7.3, Figure 7.4, Figure 7.5, and Figure 7.6 show the main steps of Form Chart model generation. The user starts the generation process by opening a wizard “Import Pages” (in Figure 7.3). The user points to the database of the interested target website, and imports available web pages. The web pages are transformed into correspondent Form Chart *Pages*, and added to the Form Chart

diagram/model (in Figure 7.3). Note that the pages in the Form Chart model do not represent individual web pages as seen by the user, but classes of web pages that share the similar structure and similar set of web form parameters with possibly different values.

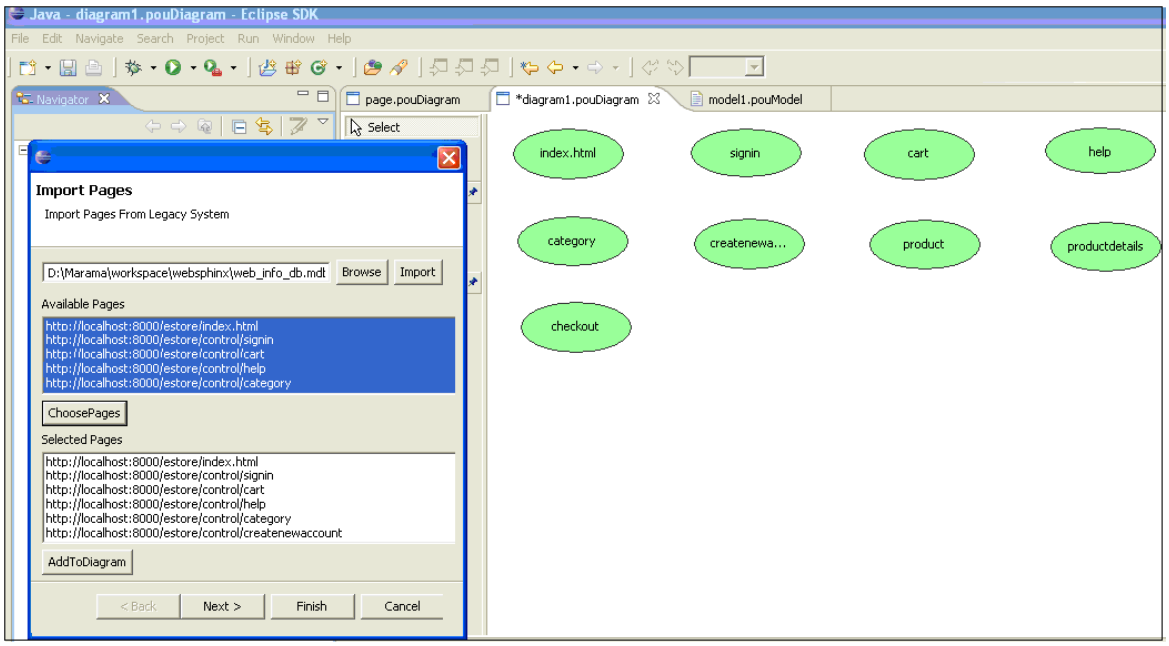


Figure 7.3. Generating Form Chart pages by importing website pages

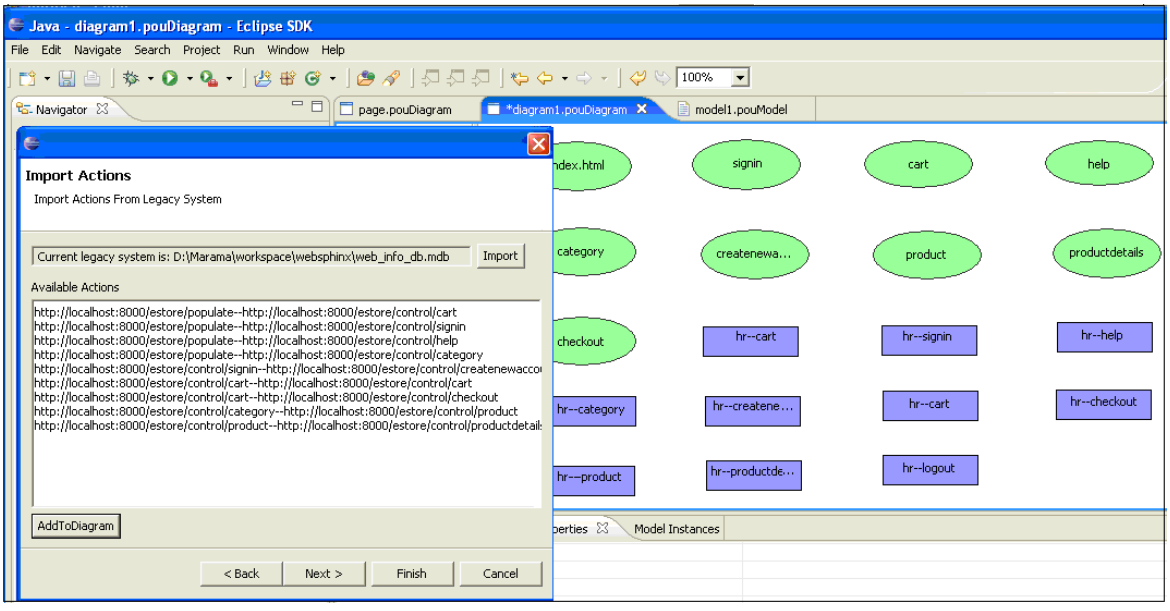


Figure 7.4. Generating Form Chart actions by importing http requests

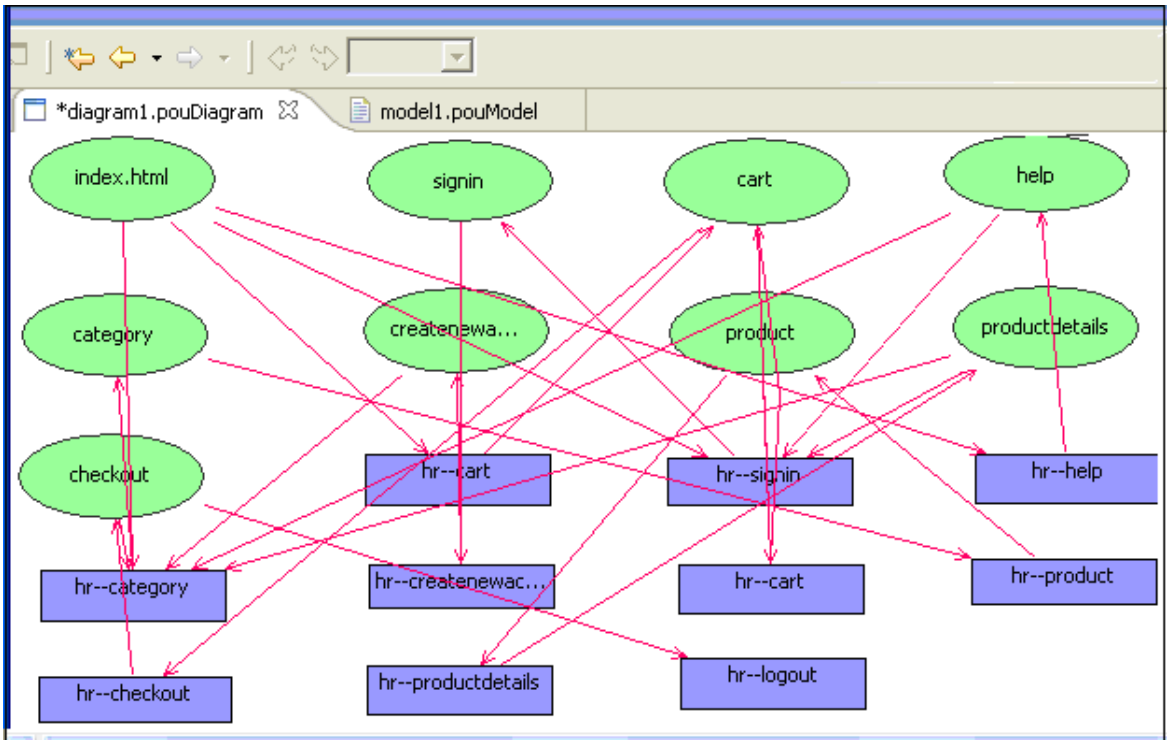


Figure 7.5. Generating Form Chart transitions by importing http requests

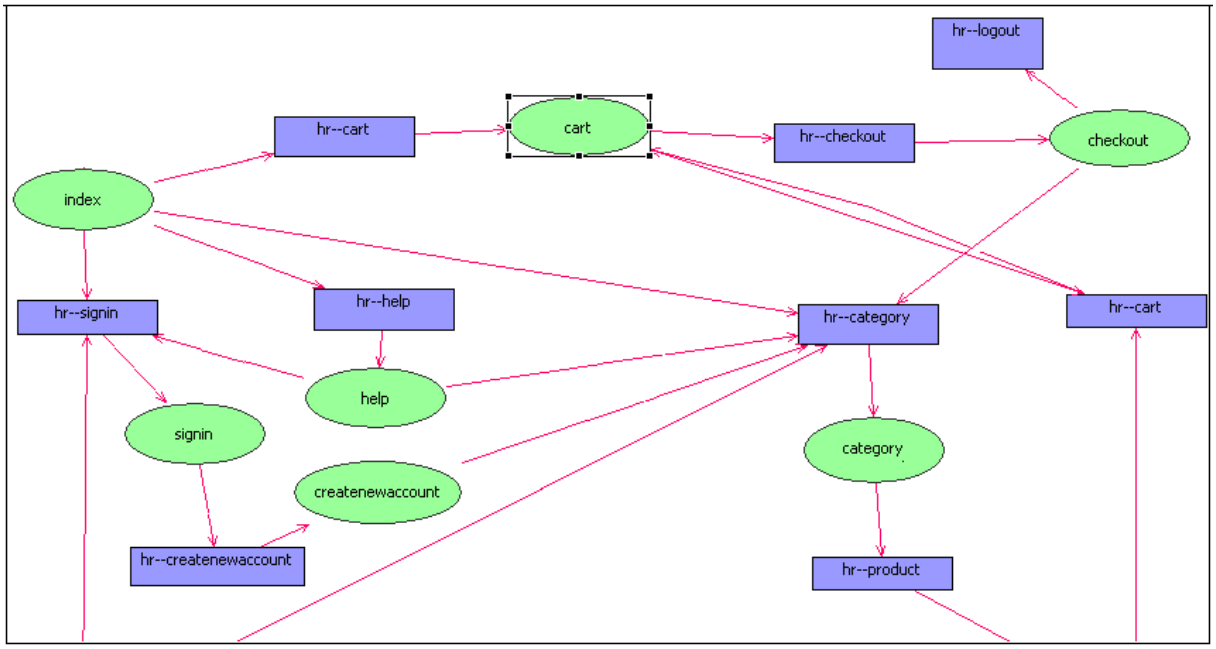


Figure 7.6. Manually adjusted generated Form Chart model

The wizard then allows tool users to add *Actions* to the intended Form Chart model (Figure 7.4). The user is given a list of available actions (based on the http requests the chosen pages can launch), and can add Action components to the Form Chart model. The wizard then allows tool users to add *Transitions* between Page components and their Action components (Figure 7.5). The generated basic Form Chart model (Figure 7.5)), a perfectly correct Form Chart model, is far from ideal aesthetically, and requires manual rearranging to improve its readability (Figure 7.6).

7.4.3 Form Chart augmentation

MaramaMTE+ is intended to use a Form Chart model in two ways: (1) using it as an independent model to generate testing plans and scripts for third party load testing tools (e.g. Apache JMeter); and (2) using it together with other domain-specific models (e.g. the architecture design models, business process models and service composition models) to generate comprehensive performance evaluation test beds. For either purpose, a generated Form Chart model needs to be augmented by appropriate properties for code generation. For example, MaramaMTE+ stochastic Form Chart Transitions use a property "Probability" to model the chance of users requesting a particular web page (refer to Figure 6.3). This property may be computed in various ways e.g. randomly within a specified range value, normal curve distribution, or from monitored web site usage (Draheim et al, 2006). Form Chart pages also require properties to specify such stochastic information including "delayKind", and "delayTime" (refer to Figure 6.3) (Draheim et al, 2006). Tool users thus need to flesh out the generated basic Form Chart models (e.g. Figure 7.6) with suitable property values such as probabilities (currently empirical data) on all Transition links. The basic models also need to be augmented by adding some MaramaMTE+-specific modeling components. Figure 7.7 shows the augmented stochastic Form Chart. The *ClientA* component does not belong to a generic Form Chart model. It represents a client-side start-up component for loading test code generation, as all testing plans need an entry point and various configuration properties. The *quit* actions are also manually added to the generated model to describe the real client behavior.

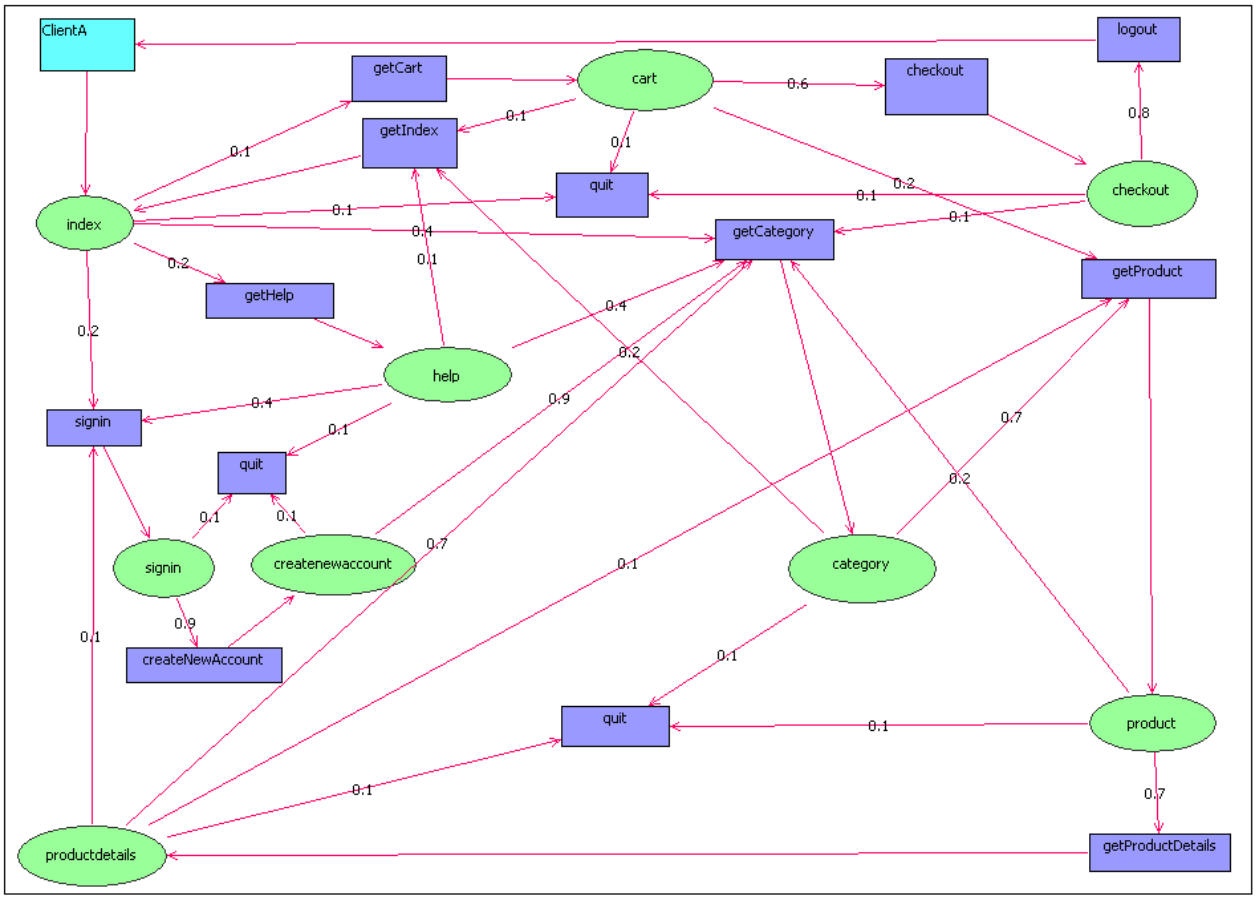


Figure 7.7. A synthesized Pet Store Form Chart model

7.4.4 Form Chart History-Sensitive Supplementary Model

MaramaMTE+ also extends MaramaMTE to capture user behavior history. User behavior history (Draheim et al, 2006) is how users make decisions to take paths to visit various web pages. In a shopping session, users' previous visiting/behavior may influence their next move. Figure 7.7 is a history-free model, where each page has only one state and is not influenced by users' behavior history. In the history-free model, page "product" (oval shape labeled with "product") may launch three http requests (outgoing Transitions to three square shape actions). The empirical probabilities for each of those http requests are: 0.2 (to action *getCategory*), 0.1 (to action *quit*), and 0.7 (to action *getProductDetails*) respectively. However, in reality, the probability distribution of page *product* to its three connected actions varies with users' behavior history, which is illustrated in a complementary history-sensitive model in Figure 7.8. In the history-sensitive model, the numeric suffix of each page

name represents the history-sensitive state of the page. The history-sensitive model shows three decision paths (users behavior history) related to page "product", including, ***Path 1:*** *page "cart_1" -> page "product_1" (leading to the state of product_1);* ***Path 2:*** *page "cart_1" -> page "product_1" -> page "productdetails_1" -> page "product_2" (leading to the state of product_2);* ***path 3:*** *page "cart_1" –> page "product_1" –> page "category_1" –> page "product_3" (leading to the state of product_3).* Three paths are three different contexts in which users make different decisions for the next move. For example, page *product* has three states *product_1*, *product_2*, and *product_3*. Each state is identified by a set of probability distribution of http actions leading to the form page. MaramaMTE+ supports users to use the history-free model as the base model, and then refine it by one or more correspondent history-sensitive model(s) to model more realistic user behavior and generate realistic load testing plans.

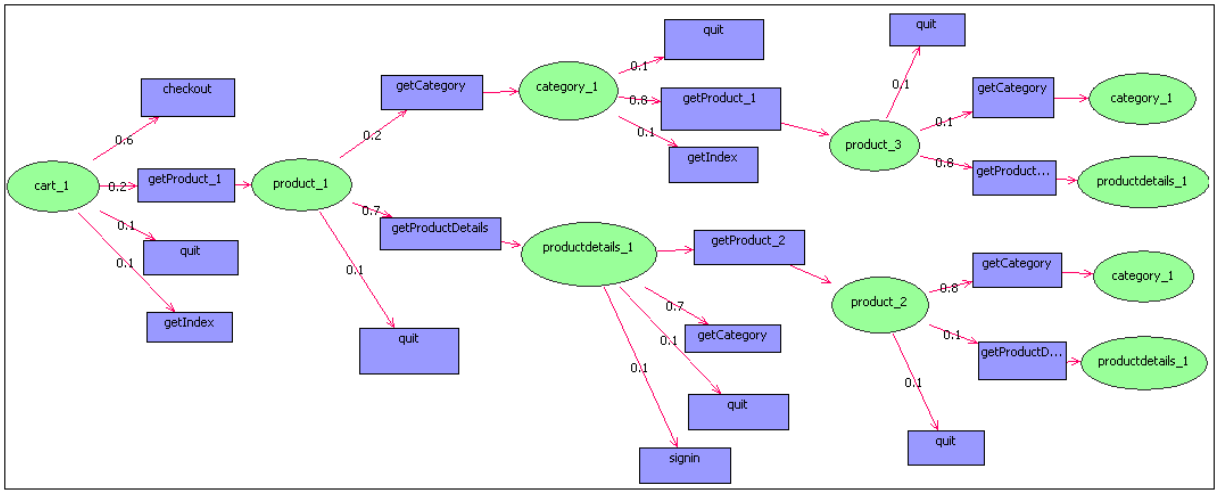


Figure 7.8. A supplementary decision model

7.4.5 Generating load testing plans

MaramaMTE+ generates load testing plans, and their associated scripts or programs for a range of third party tools. One example is the generation of Apache JMeter testing plans and associated scripts. A JMeter testing plan consists of: one or more thread groups, logic controllers, sample generating controllers, listeners, timers, assertions, and configuration elements (Apache JMeter, 1999). When generating a JMeter test plan, each Form Chart Page component represents the state of the website; each Action component represents an http request to obtain certain web page; and each Transition specifies the possibility of the Page launching the Action. Element properties such as "Probability" and "http

request" are used to generate the logic controllers of the JMeter testing plan. More details of JMeter testing scripts generation will be explained in section 7.5. Figure 7.9(a) illustrates a small piece of JMeter testing scripts generated from the Form Chart model in Figure 7.7 and Figure 7.8. This generated test plan can be loaded into the JMeter test tool's GUI environment, as shown in Figure 7.9(b). A JMeter plan does not need to be loaded into JMeter's GUI interface but can, instead, be executed directly through a command line in MaramaMTE+. MaramaMTE+ can also generate a test-bed-specific client side Java application using the Pet Store Form Chart model, as illustrated in Figure 7.9(c) and Figure 7.10.

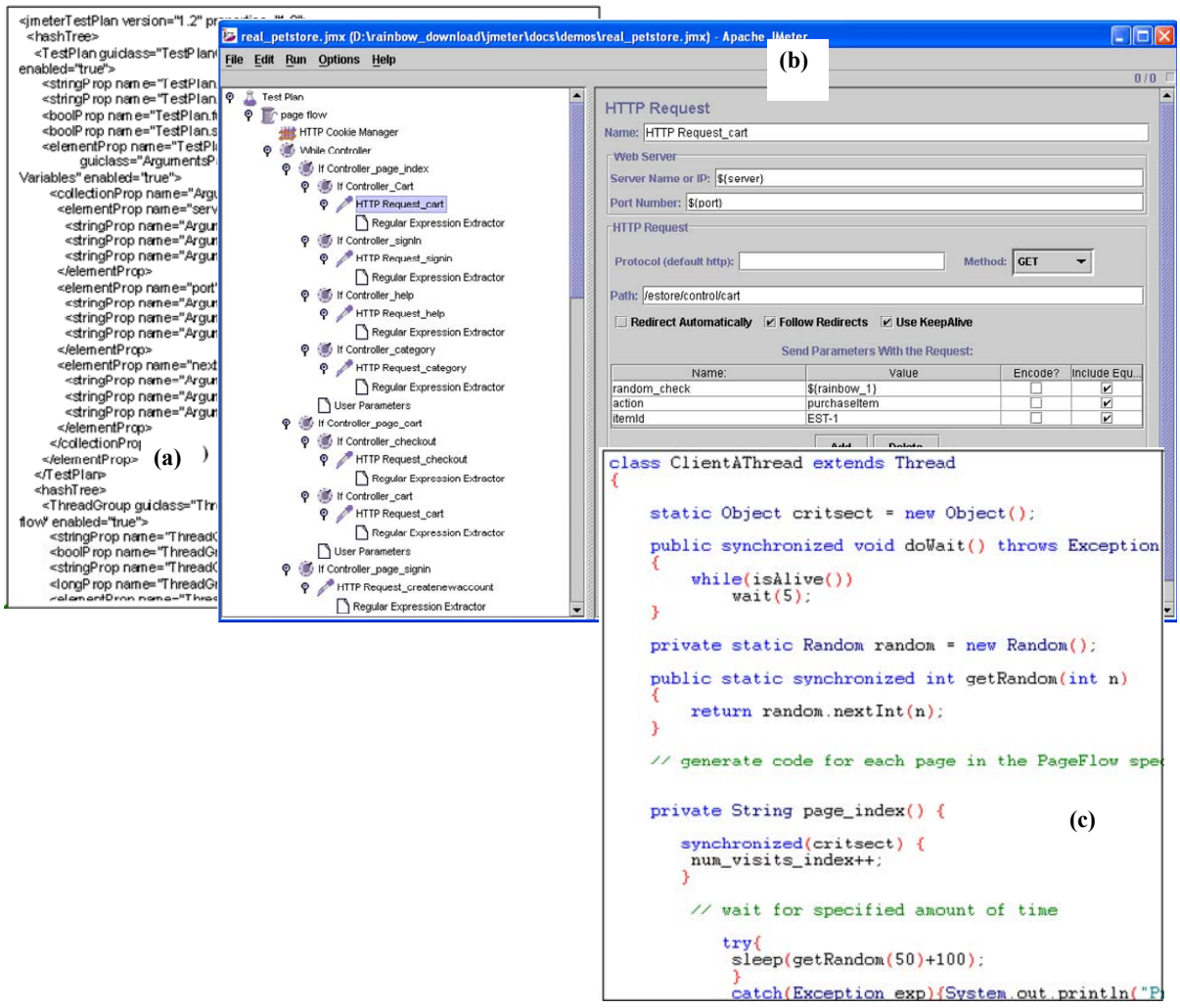


Figure 7.9. JMeter test plan, JMeter, and test bed client application

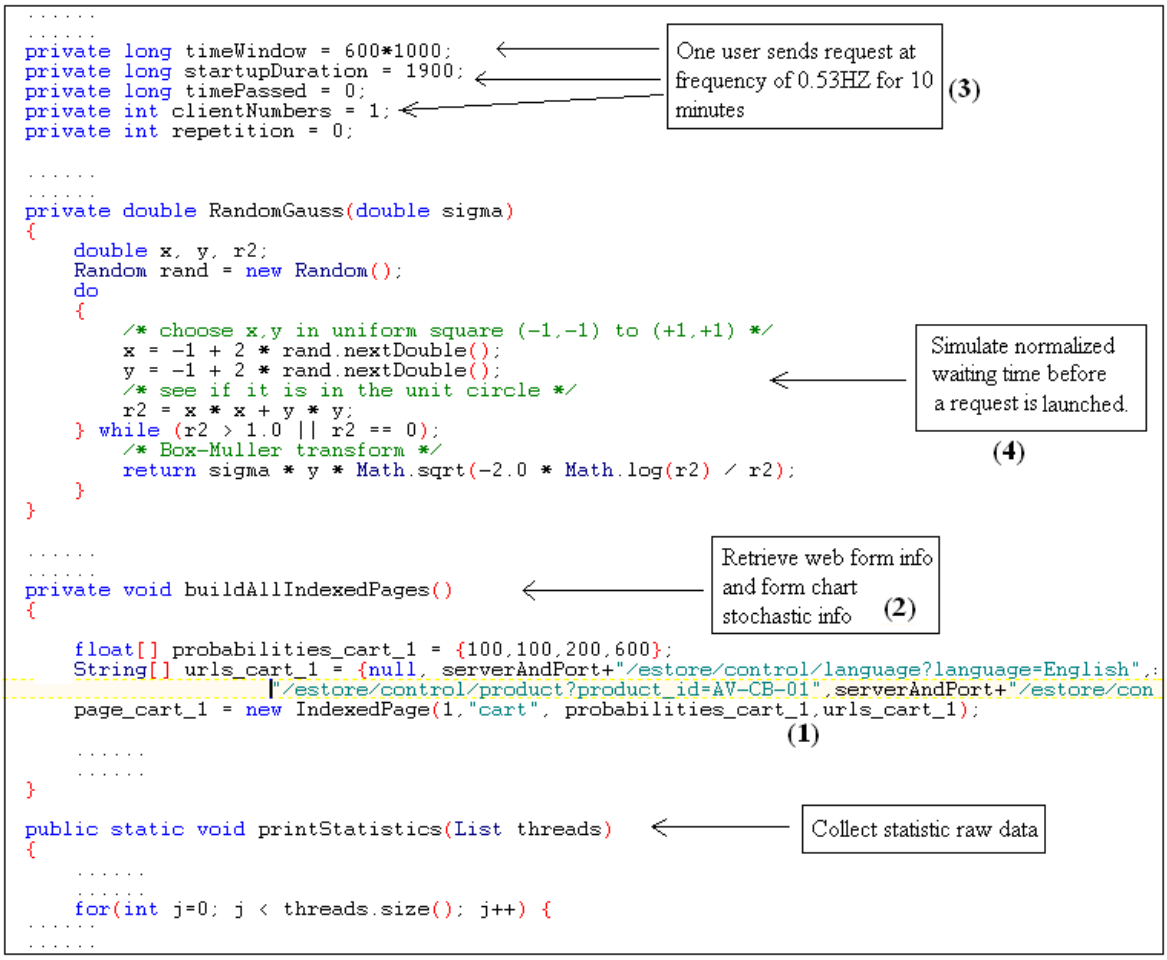


Figure 7.10. Generated load testing java program

Figure 7.10 specifies how Form Chart model properties are translated into the information of a java load testing program (part of test beds). The Form Chart Page components and their properties have been translated into http requests towards the target web application server (1); the actions and transitions have been translated into control logic implementing a state machine (2); the client component has been translated into the test plan configuration and properties (3); the waiting time for launching a http request is simulated by stochastic data (4). The generated program encodes a set of load testing parameters including "clientNumbers", "timeWindow", "startupDuration" (3). Parameter "clientNumbers" represents how many customers will be simulated in the load testing plan. Parameter "Time Window" represents how long one load testing will last. Parameter "startupDuration" represents

the time delay between two requests, and often interchanges with parameter "Request Launch Frequency" representing frequency that each customer will launch http request. These three testing parameters come form the "Client" component in the Form Chart model. Software engineers can set up various testing parameter values to get raw results to analyze web performance.

7.4.6 Running generated load tests

Generated test plans and scripts can be run against a running J2EE Pet Store application. Table 7.1 shows a testing plan's three testing parameters and their values. Client Number represents that a testing plan simulates 1 user to continuously launch requests to the Java Pet Store server. Time Window represents that the testing plan must launch requests continuously for 10 minutes. Request Launch Frequency (RLF) represents the frequency the requests should be launched. Each test plan must choose a RLF within the range from 0.5 to 1 HZ. The values of these testing parameters are chosen empirically. For each test run, users need to choose a value for Client Number, Time Window, and Request Launch Frequency. Figure 7.11 illustrates a test run's raw evaluation data using the generated java test program (in Figure 7.10) running against a legacy J2EE Pet Store application. It shows how many times a particular Form Chart page has been visited (visits), how long it takes for the page to respond to all those http requests (total time), and the average time taken to respond one request.

Client Number	1
Time Window (minute)	10 minutes
Request Launch Frequency (HZ)	0.5, 0.53, 0.56, 0.59, 0.62, 0.67, 0.71, 0.77, 0.83, 0.9, 1

Table 7.1. Load testing parameters

```
Test Running...
Statics summary:
  cart: visits = 93, total time = 17590, ave time = 187
  category: visits = 1420, total time = 304835, ave time = 214
  checkout: visits = 61, total time = 13643, ave time = 220
  createnewaccount: visits = 286, total time = 49537, ave time = 172
  help: visits = 164, total time = 29899, ave time = 181
  product: visits = 1083, total time = 197822, ave time = 182
  productdetails: visits = 807, total time = 142454, ave time = 176
  signin: visits = 307, total time = 58508, ave time = 189
  index: visits = 917, total time = 149188, ave time = 162
```

Figure 7.11. Sample load testing raw result data of java Pet Store

The Pet Store website responds differently when RLF is changed, and the results are shown in Figure 7.1 and Figure 7.13. Figure 7.12 shows how the response time of each page increases with the increase of RLF. For most of the visited pages, the response time is doubled when the request frequency goes from 0.5 HZ to 1HZ. For all the tested pages, 1HZ seems to be the highest frequency the tested Pet Store server can handle. When the frequency is increased to 1.2HZ the Pet Store server becomes extremely unstable and is easily brought down. Figure 7.13 shows the distribution of average response time of each page. This figure shows that all web pages have similar response time, and the performance of the Pet Store server is well balanced. Page "category" has slightly longer response time as it responds the request with the largest amount of data. How to analyze data and apply analysis results to improve website performance is beyond the scope of MaramaMTE+. However, MaramaMTE+ supports formal analysis modeling of web users' behavior and testing plan generation, which improves the efficiency and effectiveness of web load testing.

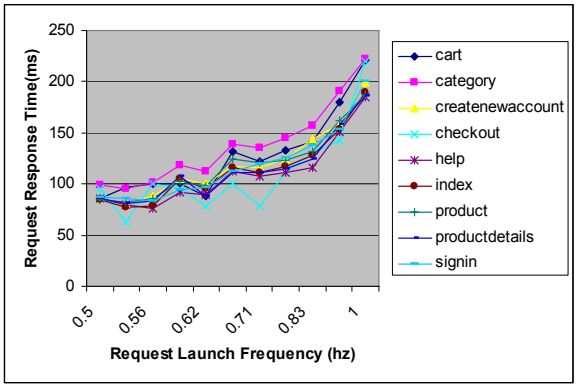


Figure 7.12. Request Response Time changes with Request Launch Frequency

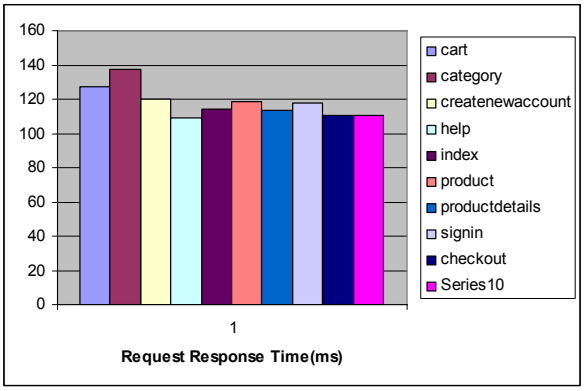


Figure 7.13. Distribution of average Request Response Time for Web Pages

7.5 MaramaMTE+ design and implementation

MaramaMTE+ is a set of Eclipse IDE plug-ins implemented using the Marama meta-tool development framework. The architecture of MaramaMTE+ is illustrated in Figure 7.14. The Marama meta-tool provides framework for developing diagram editors (1). Two key diagram types are used – the Form Chart model and the architecture model (2). Diagrammatic editors are instantiated to edit these models using the Eclipse Graphical Editing Framework. The "WebSphinx coordinator" component starts up

WebSphinx crawler tool (an independent Eclipse plug-in) (3); and also works with a "DatabaseProcessor" component to collect crawled target web application structural information and store it in the crawler database (8). The "DatabaseProcessor" component in the "MaramaModelGenerator" manages database connections, and accepts all crawled data from WebSphinx and saves it in the crawler database (4). The "DiagramGenerationManager" component retrieves website information from the web crawler database to generate Form Chart model entities, associations and their visual icons (5). The "Algorithm" component arranges the generated visual icons and connectors into a basic Form Chart diagram layout. A simple layout algorithm is currently used to arrange the pages and actions one after another as illustrated previously. Other layout algorithms such as force-directed layout algorithms provided by the CCVisu 3rd party package (Beyer, 2005) have also been tried.

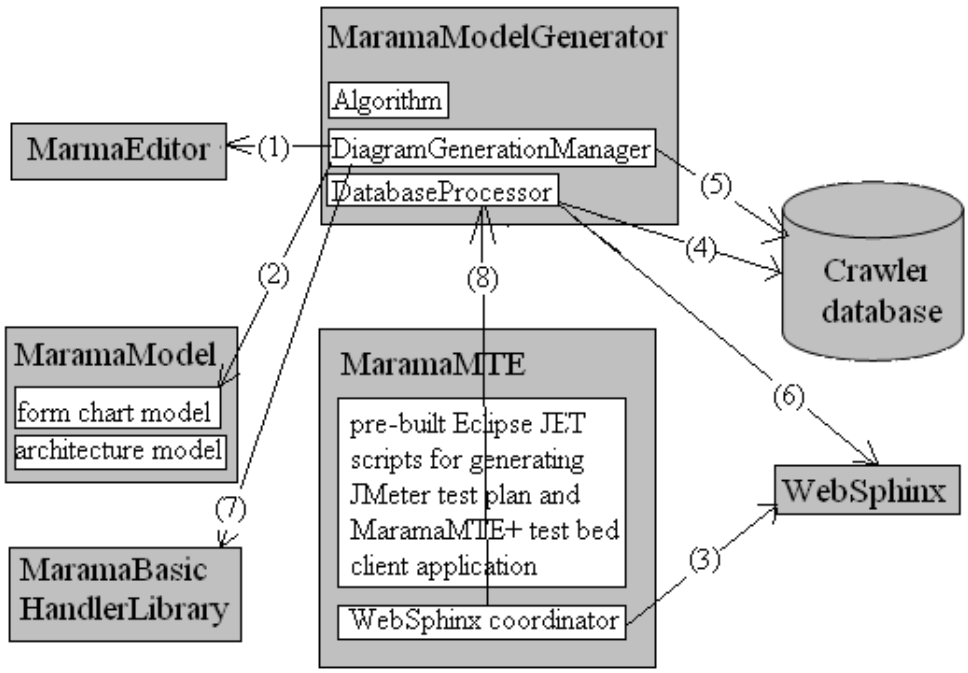


Figure 7.14. High-level architecture of MaramaMTE+

MaramaMTE+ uses Eclipse Java Emitter Templates (JET) scripts to generate code, including test plans and scripts from Form Chart models, as well as server-focused architectural test bed from architecture models. JET uses a subset of the Java ServerPages (JSP) syntax making it easy to write the required code generation templates. MaramaMTE+ traverses a Form Chart model and transforms each element

into a set of target load testing tool abstractions. Figure 7.15 shows a JET script generating the basic information of a JMeter test plan from the *Client* component and its properties. The basic information includes the name of the test plan, how many threads the test simulates, and the test run timing monitors (WhileController) (1). The first page (the homepage of a website) in the Form Chart model is then transformed into an initial http request on the target web application (IfController), which includes the target URL and the URL parameters and example values encoded in the Form Chart model (2). An action is translated into a HTTP sampler (3). The Transitions from Pages to Actions generate decision logic in the JMeter test script (through JMeter's RegexExtractor, UserParameters, and more), which implements a state machine model of user behavior. The probabilities of transitions are realized through JMeter's BeanShellTimer, Gaussian Random Timer, and more.

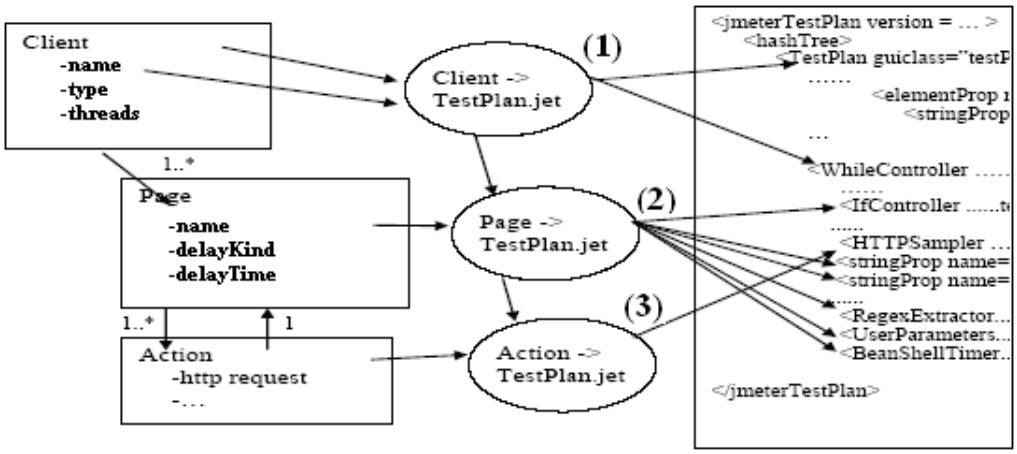


Figure 7.15. JMeter test plan generation from MaramaMTE+ Form Chart model.

7.6 Discussion

MaramaMTE+'s effectiveness for supporting realistic client load modeling and test plan generation has been evaluated. MaramaMTE+ has been used to synthesize a formal model of client loading for several web-based systems, including the Java Pet Store, NetPay, and the web site of the Department of Computer Science of The University of Auckland. Table 7.2 presents some initial empirical evaluation results. In Table 7.2, *Work Efficiency* compares the effort needed to manually construct JMeter test plans with the effort needed to augment an extracted Form Chart model then generate JMeter test plans. The work was undertaken by an experienced software engineer who knew each target system well. The manually created JMeter test plans were done using the JMeter GUI editor rather than replay/capture

tool. Efficiency gains between 5-6 times of using MaramaMTE+ were demonstrated. The Form Chart models developed for each targeted website also bring long-term benefit for users, because Form Chart models are easy to understand and maintain than JMeter test plans and scripts.

System	Work Efficiency	
	effort to manually build test plans	effort to augment an extracted Form Chart model then generate test plans
Java Pet Store	18 hours	3 hours
NetPay	25 hours	3 hours
Department of Computer Science website	15 hours	2.5 hours

Table 7.2. Empirical comparison results

Most current web performance engineering tools require considerable knowledge of the system under test to formulate and build appropriate loading test plans and scripts, which requires much effort especially for systems under change, large systems, or systems the performance engineer is unfamiliar with. In contrast, MaramaMTE+ provides a structured and automatic way to retrieve target system information and allows users to build up testing plans efficiently. As shown by preliminary results with MaramaMTE+ in Table 7.2, effort is much lower for generating a client load model with it than using JMeter's GUI test plan designer directly.

Key advantages to MaramaMTE+ approach include: its ability to extract model structure from a web application via web crawling; model-based generation of 3rd party stress testing tool test plans and scripts; and ability to run and compare web application performance under numerous different loading models accurately and efficiently. MaramaMTE+ extracts most information of an intended Form Chart model directly from a web application, which greatly reduces errors and time taken to develop client behavior models. Engineers can even build different versions of stochastic Form Charts for all or part of a web application to analyze its performance. Generating test plans and scripts for 3rd party stress testing tools allows MaramaMTE+ to leverage their advanced features for load testing. For example, JMeter provides sophisticated measurement, reporting, distributed test execution and test scheduling support features that MaramaMTE+ is able to reuse directly with little effort. However 3rd party tool

limitations also need to be dealt with. Most web application stress testing tools have less rich client behavioral models than MaramaMTE+ Form Charts. Thus Form Chart models may need to be simplified when generating test scripts for target load testing environment. Sometimes implementing Form Chart-specific behavior is quite complex in the 3rd party testing tool. For example, to implement a probabilistic state machine in JMeter proved to be quite challenging.

The extracted Form Chart structure can be very large for large websites. This issue is mitigated in MaramaMTE+ by allowing any number of partial Form Charts to be rendered in diagrams. Automatic layout of the extracted Form Chart diagrams is currently rudimentary and needs to be improved. In addition, support for semi-automatic grouping of large website structures into multiple Form Chart diagrams is needed to manage large website load testing.

For large websites, it can be a complex process to augment the extracted Form Charts with probabilistic information about user behavior. A key area of future work is to infer such stochastic Form Chart parameters from profiled target web application behavior. It is planned to monitor the actual usage of websites using tools to provide real user session histories with large numbers of http requests. These logs can then be analyzed to infer transition probabilities improving accuracy of the client behavior model. The example data in user session http requests also provide realistic sequences of parameter values to invoke web server pages. This will still allow performance engineers to change these probabilities or to specify alternative versions of the Form Chart for the same application.

The ultimate goals of MaramaMTE+ include: from developer perspective, reverse-engineering server-side architecture of legacy web applications into MaramaMTE+ architecture designs; from end user perspective, reverse engineering legacy web applications into Form Chart models; forward engineering optimized intended server-side architecture models for the legacy system; and comparing the legacy server-side software models with the optimized server-side architecture models. MaramaMTE+ would provide a structured and automated performance engineering environment to improve the performance of server-side architecture of legacy systems.

7.7 Summary

MaramaMTE+ is an innovative approach to automate the process of retrieving website structural data, generating Form Chart model(s), and generating load testing plans. This chapter discusses the effectiveness of the approach through a case study, where the running Pet Store website is crawled; structural data is extracted; a Form Chart model is automatically generated and manually augmented; JMeter testing plans are generated and executed; and load testing results are collected. MaramaMTE+ helps to reverse engineer legacy web applications; and efficiently run web loading tests. MaramaMTE+ shows potential to effectively and rigorously compare the performance of a legacy web application with that of an optimized software architecture model of the legacy system.

Chapter 8 - MaramaCRelation Introduction

Model Integration and Transformation (MI&T) can extend applicable scope of domain-specific knowledge, and combine different types of domain-specific knowledge to serve a broader problem domain. The researches of ArgoMTE, MaramaMTE, and MaramaMTE+ are all motivated by MI&T. ArgoMTE integrates the UML modeling technology with the technology of test bed generation and performance evaluation. MaramaMTE and MaramaMTE+ integrate the technology of test bed generation and performance evaluation with web user behavior analysis modeling (Form Chart modeling). All these researches support code generation - the simplest model transformation. MI&T in those researches are done at programming level and lack of structured high level support. This chapter introduces the MaramaCRelation approach – a high level support for MI&T. This chapter introduces the problem domain of the proposed MaramaCRelation approach; identifies the problems found in traditional technologies of MI&T; discusses the requirements for the MaramaCRelation approach; and highlights the main features of the proposed approach.

8.1 Problem statement

As is reviewed in chapter 2, Model Driven Engineering (MDE) covers a large range of research areas, including Model Integration, Model Transformation, and Multi-View Support. Model Integration generally falls into two categories of deep integration and functional integration. Model Integration in Argo/MTE, MaramaMTE, and MaramaMTE+ is functional integration, because those tools leverage the strength of different modeling technologies (without changing them) to extend their original applicable scopes. Model Transformation is the core concept involved in MDE. Model Transformation technologies transform one model to another at the same or different abstraction level. The researches of ArgoMTE, MaramaMTE, and MaramaMTE+ generate code by using popular transformation technologies such as XSLT, JET. The researches also involve many model transformation activities, such as transforming proprietary SoftArch/MTE architecture models to ArgoMTE architecture models,

and transforming MaramaMTE+ architecture models to Form Chart models. Although they have different focuses, Model Integration and Model Transformation share many similarities: 1) they both extend the applicable scope of a model; 2) need to maintain the rational of the integration or transformation; 3) record semantics maintained and lost during the integration or transformation; 4) track the evolvement of modeling elements; and 5) synchronize the involved models.

8.2 Motivation

Software engineering domain-specific models are widely used to model various facets of a software system, such as business process modeling (Li et al, 2007; BPMN, 2004), software architecture modeling (Grundy and Cai et al, 2001; Cai et al, 2005), requirement modeling (Castell et al, 1998), and object oriented design modeling (UML, 1996). These models differ in semantics and address issues of the intended software system in various ways. MI&T are effective approaches to interconnect domain-specific models to leverage their strength to build software systems, which is demonstrated by the following two cases:

Case 1: architecture modeling and performance evaluation (e.g. MaramaMTE+ architecture modeling) vs. web user behavior modeling (e.g. Form Chart modeling)

1. Model software architecture of a software system in a MaramaMTE+ architectural model
2. Model web user behavior of the same system in a Form Chart model
3. Interconnect the two models conceptually
4. Check the semantic consistency of the above two models during interconnecting
5. Generate client-side program from the Form Chart model, and server-side program from the MaramaMTE+ architectural model. Combine the generated program from both models to analyze web user behavior and evaluate the performance of the intended software architecture design

Case 2: business process modeling (BPMN) vs. architecture modeling and performance evaluation (MaramaMTE+) (*Grundy, Hosking, and Li et al, 2006****)***

1. Model business processes of a service-oriented software system in a BPMN model
2. Model software architecture of the same service-oriented software system in a MaramaMTE+ architecture model
3. Interconnect the two models conceptually

4. Check the semantic consistency of the above two models during interconnecting
5. Generate performance evaluation program from the MaramaMTE architectural model to evaluate the performance of the business process model

In each case, the domain-specific knowledge is self-evolved and left intact during model integration. Coupling them together extends each model's applicable scope; makes both models functionally connected in a broader target domain; and consolidates self-evolved domain-specific knowledge.

The two cases above help to identify the following problems found in MI&T:

- ***rational of MI&T is lost in the operational code***

  Models can be integrated and transformed because they share common conceptual semantics. The shared conceptual semantics is part of both the source and target models, and is the rationale and main resource for the intended MI&T. The shared semantics leads to building up interconnections between the source and target models, and guiding their coordination (including structural, behavioral, and functional) over time. Shared semantics needs to be captured and well-formed to effectively initiate and guide the intended MI&T.

  In most of Model Integration projects (Cai et al, 2004; Grundy, Hosking, and Li et al, 2006), the rationale of model integration is entangled with the operational code that implement the integration. The same thing happens to model transformation. The rationale of transformation is entangled with the functional code of the transformation (e.g. XSLT scripts, ATL scripts). As it is lost in operational code, the rationale of MI&T can not provide support for the interested issues involved in MI&T, including semantics maintenance, traceability, and behavior synchronization. Currently, those issues are treated as isolated operational tasks of MI&T.

- ***no high-level guidelines for tool extension***

  MI&T requires tool support. In most cases, it requires extending an existing tool to support new modeling technology. For example, Rational Rose and ArgoUML are extended continuously to support new modeling knowledge. Tool extension is mostly done at programming level. Tool developers set up functional requirements and then implement them. In ArgoMTE, tool developers use programming framework (e.g. GEF, Novosoft UML) to support architectural modeling; in

MaramaMTE and MaramaMTE+, tool developers use Marama meta-tool API to support Form Chart modeling. The developer-oriented programming-intensive approach of tool extension needs to be improved. Tool users need to take part in tool extension by providing high-level guidelines for tool extension, including semantics maintenance, traceability, and behavior synchronization.

- ***low level semantic consistency check***

During MI&T, semantics of model elements may evolve, and semantic consistency needs to be checked. The requirements for semantic consistency may vary in different circumstances, but semantic consistency check is a very important task, and is supported in tools like Rational Rose, ArgoUML, ArgoMTE, MaramaMTE, and MaramaMTE. In those tools, the support for consistency check is done at implementation level via programming framework and mechanisms. Tool users need to take part in the development of consistency check mechanism by supporting high-level, structured analysis of semantic maintenance during MI&T.

- ***traceability is maintained through low level centralized data repositories***

During MI&T, source and target model elements may look very different. They do not have to have the same names or the same other obvious property values. Maintaining traceability among models is a very important task. Most of comprehensive software engineering tools support traceability, including Rational Rose, ArgoUML, ArgoMTE, MaramaMTE, and MaramaMTE+. To support traceability, those tools use low-level centralized repositories (mainly at source code level or database level) to hold commonly shared data structure. Their traceability maintenance mechanisms are tool-API-dependent, and need to be updated manually when the involved models evolve or when new domain-specific technologies need to be supported. A flexible, high level mechanism is required to support traceability of MI&T.

- ***behavior synchronization across the models is maintained through coding***

When models are interconnected (via either model integration or transformation), their behaviors often need to be synchronized. Many comprehensive software engineering tools support behavior synchronization among models, including Rational Rose, ArgoUML, ArgoMTE, MaramaMTE, and MaramaMTE+. Again, those tools use low-level programming framework and mechanisms to support behavior synchronization. Tool users need to take part in the development of behavior

synchronization mechanisms by providing high level, well-structured information of the events triggered by MI&T.

- ***maintainability of transformation programs is very poor***

Model transformation programs and scripts (e.g. ATL, and XSLT) are textual files. Transformation templates and rules are tedious, error-prone, not categorized, not well designed, and hard to understand and maintain. They are a list of ad-hoc operations to complete model transformation. The maintainability of transformation programs and scripts needs to be improved.

- ***model transformation programs can serve more purposes***

The main goal of model transformation programs and scripts (e.g. ATL and XSLT) is to transform models from one format to another. Transformation programs contain well-structured information of: model element mapping, mapping conditions, model elements involved in each mapping, and relationships among mappings. The structured information can become good resources to study the main concerns of MI&T, including semantics consistency, traceability, and behavior synchronization. However, tool developers have not explored the potential of the structured information. They use the transformation scripts to execute one-off tasks, and then treat the concerns of MI&T as individual functional goals. It will be interesting to see how transformation programs can help to reorganize those main MI&T concerns.

- ***existing MI&T lacks analysis and design support***

Object Oriented (OO) software products can be produced without OO analysis and design methodologies. But OO analysis and design methodologies allow users to make use of the best practice OO knowledge in the intended system. Software products that are well analyzed and designed have much better qualities than ad-hoc development. Most MI&T technologies, similar to OO development technologies, realize MI&T at operation/implementation level without analysis and design support. Users usually construct ad-hoc rules and templates to get the job done. Integration and transformation rules and templates are generally hard to maintain, improve, and reuse. An analysis and design model for MI&T is needed for software modelers to make use of the best practice knowledge in designing good quality rules and templates for MI&T.

8.3 Requirements for structured, high level support for MI&T

The problems addressed above motivate the following requirements for a high-level support for MI&T:

- ***MI&T needs to be analyzed and designed at meta-model level***

  Most domain-specific models use meta-models (e.g. MOF-based meta-models (MOF, 2008), VPM-based meta-models (Varr'o et al, 2003)). A meta-model provides abstract syntax and semantics (including entities, associations, properties, constructs, semantic constraints, and rules). Domain-specific models are instance models of their meta-models, and exploit surface notations that embody the semantics defined in their meta-models. Any structured high-level solutions for MI&T need to be done on domain-specific meta-models.

- ***capture semantics shared by source and target meta-models***

  The semantics conceptually shared by source and target domain-specific models is the rationale and the main resource for model integration, model transformation, multi-view environments, and tool integration. The shared semantics leads to building up interconnection relationships between the source and target models, and guiding their coordination (including structural, behavioral, and functional) over time. Shared semantics needs to be captured and well-formed to effectively guide MI&T.

  In most existing model transformation (ATLAS Transformation, 2006; XSLT Transformation, 2001; Csertan et al, 2002) and integration technologies (Sanchez et al, 2008; Ramos et al, 2007), users normally come up with an ad-hoc understanding of the shared semantics, then use it implicitly to guide the construction of rules and templates for model integration, transformation, and multi-view environment. Explicit, well-structured shared semantics is needed to replace the ad-hoc implicit understanding. An effective approach to capture shared semantics must 1) capture appropriate atomic units of the shared semantics; 2) associate atomic units and allow them to effectively communicate with each other; 3) support efficient communication between the captured shared semantics and the involved domain-specific models.

- ***create Interconnection Relationships between source and target modeling elements***

  Explicitly or implicitly, MI&T establishes Interconnection Relationships between source and target models. Interconnection Relationships define what and how to interconnect models that conceptually

share semantics. What-to-interconnect defines the interested source model element(s) with the target model element(s). How-to-interconnect defines the constraints that the interested model elements must satisfy before they can be interconnected.

The Interconnection Relationships represent the same meaning in both model integration and transformation, in terms of they both extend the applicable scope of a model; need to maintain the rational of the integration or transformation; record semantics maintained and lost during the integration or transformation; track the evolvement of modeling elements; and synchronize the involved models.

- ***consistency and consistency check***

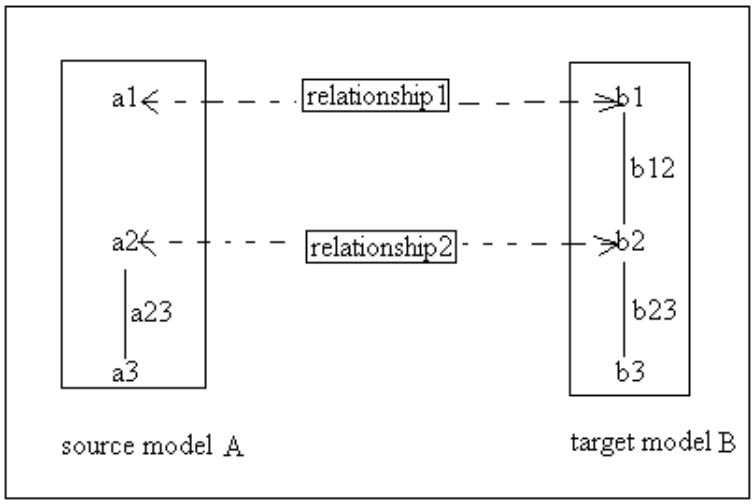


Figure 8.1. Consistency during MI&T

Source and target meta-models have different abstract syntax and semantics. MI&T may cause semantic inconsistencies. Figure 8.1 illustrates a sample situation where semantic inconsistencies may occur. Here, instance models A and B have different domain-specific meta-models. The source model A contains entities: a1, a2, and a3; and one association: a23. The target model B contains three entities: b1, b2, and b3; and two association b12 and b23. When Interconnection Relationships are established between a1 and b1, and a2 and b2, the interaction between b1 and b2 (through association b12) may imply a "hidden" association between a1 and a2 in model A. If the interaction (through associations) between a1 and a2 is not allowed by the abstract syntax of model A, the

"hidden" association will cause semantic inconsistency to model A after its interconnecting to model B (via integration with or transformation). The semantic inconsistencies need to be detected and recorded to give modelers cognitive support when making decisions during MI&T.

- ***maintain traceability without a low level centralized data repository***

Traditional software modeling tools (e.g. Rose, ArgoUML) maintain traceability by using low level data repositories, and programming framework and mechanisms. A high-level support must provide a flexible mechanism to maintain traceability of MI&T of ever-changing interested models.

- ***behavior synchronization across the models***

When models are interconnected (via either model integration or transformation), their behaviors need to be synchronized. What and how to synchronize need to be decided. What-to-synchronize defines what model parts and their behaviors need to be synchronized across the models. How-to-synchronize defines the response events in order to maintain the validity of the Interconnection Relationships over time. Behaviors triggered by MI&T need to be recorded and well-formed to provide guidelines for behavior synchronization during MI&T.

- **explore the potential of model transformation programs and improve their maintainability**

Model transformation programs are well-structured information. They should be able to support the main concerns involved in MI&T, including semantics maintenance, semantic consistency check, traceability maintenance, and behavior synchronization. Model transformation programs are hard to maintain. Most of them are just a list of ad-hoc operations to transform models. The readability and maintainability of transformation programs need to be improved before they can help to solve the main MI&T issues.

- ***analysis and design support for MI&T***

A high level support for MI&T must allow users to analyze and design the intended MI&T incrementally. It must separate rational from operational code; allow users to reason about if the intended MI&T maximally match their mindset; generate code to support traceability maintenance and behavior synchronization; and detect, record semantic consistency.

8.4 MaramaCRelation overview

The MaramaCRelation approach is designed to interconnect domain-specific models. The MaramaCRelation approach treats both model integration and model transformation as the same in terms of their: maintaining the rational of the integration or transformation; recording semantics maintained and lost during the integration or transformation; tracking the evolvement of modeling elements; and maintaining synchronization across the involved models.

The MaramaCRelation approach is consisted of two main contents: The CRelation (read as "crea-lation") model and the MaramaCRelation prototype tool. The CRelation model is the core part of the MaramaCRelation approach, and its lifecycle is illustrated in Figure 8.2. This model captures the semantics conceptually shared by a source and a target domain-specific meta-model (1). The Crelation model can effectively communicate with the involved source and target meta-models (2). The CRelation model leverages third party knowledge to define selection constraints and behaviour synchronization information (3). The CRelation model raises MI&T to high abstraction level, permitting analysis and design of MI&T programs, and detecting semantic inconsistency (4).

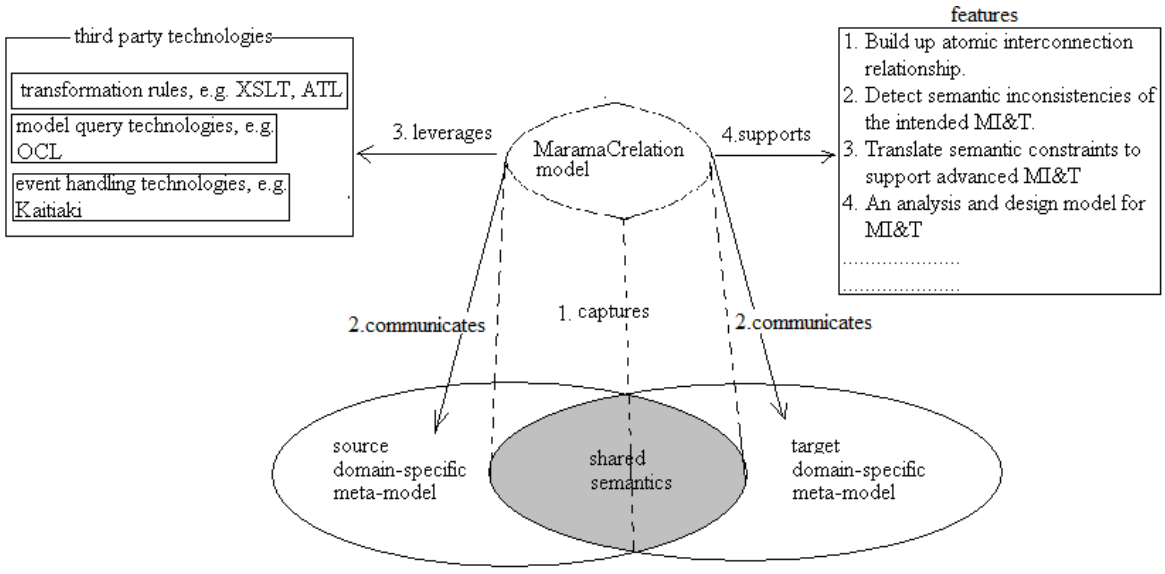


Figure 8.2. CRelation model lifecycle

Figure 8.3 shows how a sample CRelation model interconnects the MaramaMTE meta-model and EJB-extended UML meta-model. The MaramaMTE software architecture meta-model in Figure 8.3(a)

contains abstraction entities (green rectangles) and associations (pink round rectangles) modelling software architecture of web applications. The main modelling types in the MaramaMTE meta-model include: *ApplicationClient*, *ApplicationServer*, *RemoteObject, Request, Service, Database, and DatabaseTable.* The main meta-associations are: *ServerObject, ServerDatabase, ClientRequest, ClientServer, DatabaseTables, and ObjectService*. In Figure 8.3, the MaramaMTE software architecture meta-model is used as the source meta-model to be interconnected. The target meta-model is an extended UML meta-model for EJB (EJBUML meta-model) illustrated in Figure 8.3(b). This EJBUML meta-model contains basic EJB concepts, including: *AppServerHome* representing application server home, *AppServerApp* representing server application, *EJBHome* representing EJB home, *EJBInterface* representing EJB Interface, *EJBBean* representing EJB Bean. The EJBUML meta-model is a proof-of-concept UML profile, and is similar to the well-documented UML profile for EJB (Greenfield, 2001).

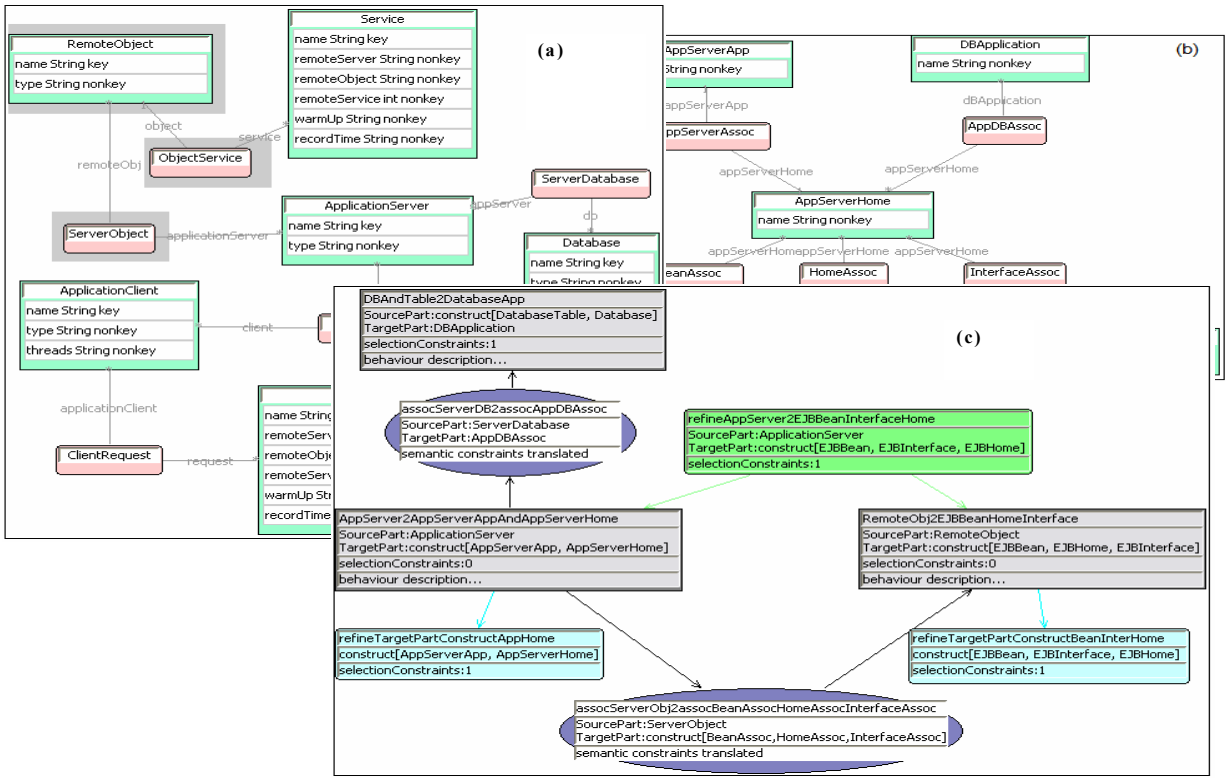


Figure 8.3. (a) MaramaMTE meta-model; (b) EJBUML meta-model; (c) MaramaMTE-EJBUML MaramaCRelation model

Figure 8.3(c) is the CRelation model that interconnects the MaramaMTE and EJBUML meta-models. The CRelation model captures the shared semantics by using four abstraction entities: *StructureMapping*, *StructureRefinement*, *SelectionRefinement* and *SemanticAssociation*, and three abstraction associations connecting them. The semantics of the CRelation model will be described in detail in Chapter 9. The CRelation model (Figure 8.3(c)) consists of three StructureMappings (grey rectangles), one StructureRefinement (green rectangle, centre top) two SelectionRefinements (cyan rectangles) and two SemanticAssociations (purple ovals).

8.5 Main features of the MaramaCRelation approach

The main features of the MaramaCRelation approach is highlighted here, and will be explained in detail in Chapter 9:

- The MaramaCRelation approach explicitly captures the semantics shared by the source and target models.
- The MaramaCRelation approach can interconnect self-evolved domain-specific models, as well as the models that are transformed from one to another.
- The MaramaCRelation approach does not define its own model query language, but uses third party model query languages (e.g. OCL) to define interconnection selection constraints
- Most model transformation approaches treat a model as a group of individual constructs when constructing transformation rules and templates. Associations between those rules and templates are implicit and ignored. The MaramaCRelation approach treats a model as a model instead of a group of individual constructs, and explicitly represents the information that is usually implied but discarded in traditional model transformation technologies.
- The MaramaCRelation approach supports traceability through a "search and relate" mechanism. It generates search conditions from the CRelation model to search source and target model parts that meet the selection constraints (mapping constraints).
- MI&T may cause semantic inconsistencies between source and target models. The MaramaCRelation approach detects semantic inconsistencies in the CRelation model.
- The MaramaCRelation approach generates high-level descriptions for the events triggered by the MI&T. The generated behavior descriptions are guidelines for users to construct behavior synchronization mechanisms.

- The MaramaCRelation approach can be used as an analysis and design model for MI&T. It breaks down usually monolithic model integration and transformation activity, explicitly represents the associations between usually isolated MI&T rules and templates, and separates *what* to transform and integrate from *how* to do transform and integrate. Although most MI&T approaches support mechanisms for reusing and refactoring transformation scripts (ATLAS Transformation, 2006; XSLT Transformation, 2001; Csertan et al, 2002) , they do not support users to analyze and design integration or transformation.

8.6 Summary

MI&T have been extensively researched and become core technologies in the paradigm of Model Driven Engineering. MI&T are closely related to research areas such as model merging, multi-view environments, and model refinement. Technologies have been developed to solve the operational tasks involved in MI&T, including model and tool integration, model transformation, semantic maintenance, traceability, behaviour and model synchronization, and semantic consistency check. This chapter reviews the problems involved in the research of MI&T caused by the operational level focuses, generalizes the motivations for the research of the MaramaCRelation, and introduces the main features of MaramaCRelation. The MaramaCRelation approach is intended to provide analysis and design support for MI&T, which, in turn, helps to review the operational issues involved in MI&T and provide better solutions.

Chapter 9 - The CRelation Model

As is introduced in Chapter 8, the MaramaCRelation approach consists of the CRelation model and the supporting MaramaCRelation tool. This chapter introduces the abstract syntax of the CRelation model and explains the semantics of its abstraction entities and associations in natural language; demonstrates how the CRelation model entities and associations provide the features highlighted in chapter 8; and introduces the process to construct a CRelation model.

9.1 Terms and concepts

The terms and concepts used in the CRelation model are designed to be maximally compatible with the similar concepts in the related research areas. The terms and concepts, which are corner stones of the model, are explained as follows:

- ***the CRelation model interconnects diagrammatic domain-specific meta-models***

A model can mean a diagrammatic model, formal mathematical model, source code, and so on. At this stage, the CRelation model targets diagrammatic domain-specific meta-models.

- ***domain-specific meta-model entities, associations, and constructs***

A diagrammatic meta-model defines its abstract syntax through abstraction entities, associations, and how they will be constructed in instance models. Instance models of a meta-model are typed by the meta-model abstractions (also called meta-elements).

In the CRelation model, *construct* is a reserved concept, and represents a group of meta-elements (including entities, associations, or both) that form a sensible subset of a domain-specific meta-model.

- ***diagrammatic domain-specific meta-model semantics, and semantic constraints***

The CRelation model assumes a domain-specific meta-model consists of semantics and semantic constraints, which are respectively similar to semantics and OCL constraints of the UML meta-model (UML, 1996; MDT, 2008). Semantic constraints of a meta-model are represented by well-formed rules (e.g. OCL constraints). They define a set of invariants of instances of meta-elements, which have to be satisfied for the abstract syntax elements to be meaningful. Semantics define the meaning of abstract syntax elements that are well formed by fulfilling their semantic constraints. Semantics are often described in natural language.

- ***shared semantics, Interconnection Relationship, and selection constraints***

The CRelation model captures semantics shared by two meta-models via atomic Interconnection Relationships. An atomic Interconnection Relationship consists of an Interconnection Relationship Unit (IRU) and selection constraints. An IRU is a 2-tuple of source and target meta-elements. Selection constraints are conditions that the 2-tuple elements must satisfy before the Interconnection Relationship is valid. Selection constraints must be well-formed and defined in tool-API-independent languages (e.g. by using OCL, ATL, or tool-API-independent java).

An atomic Interconnection Relationship specifies that when its selection constraints are satisfied; the instances of the source and target meta-elements involved in the IRU represent similar semantics with the different representations in the different (source and target) instance models.

- ***behaviour description***

Behaviour synchronization is the foundation of model synchronization. The source and target model elements of an IRU need to be synchronized in behaviour to maintain the validity of the Interconnection Relationship over time. In the CRelation model, behaviour description is a high level description of behaviours triggered by IRUs. Behaviour description provides structured information to guide users to implement target model response events.

9.2 Running Example: Interconnecting the Pet Store architecture model with the Pet Store design model

The MaramaMTE-styled (MTE) software architecture meta-model and instance model have been introduced in previous chapters. The MTE meta-model (Figure 9.1(a)) contains abstraction entities

(green rectangles) and associations (pink round rectangles) that will be instantiated in instance software architecture model. In this chapter, as a running example, the MTE architecture meta-model is used as the source meta-model to be interconnected.

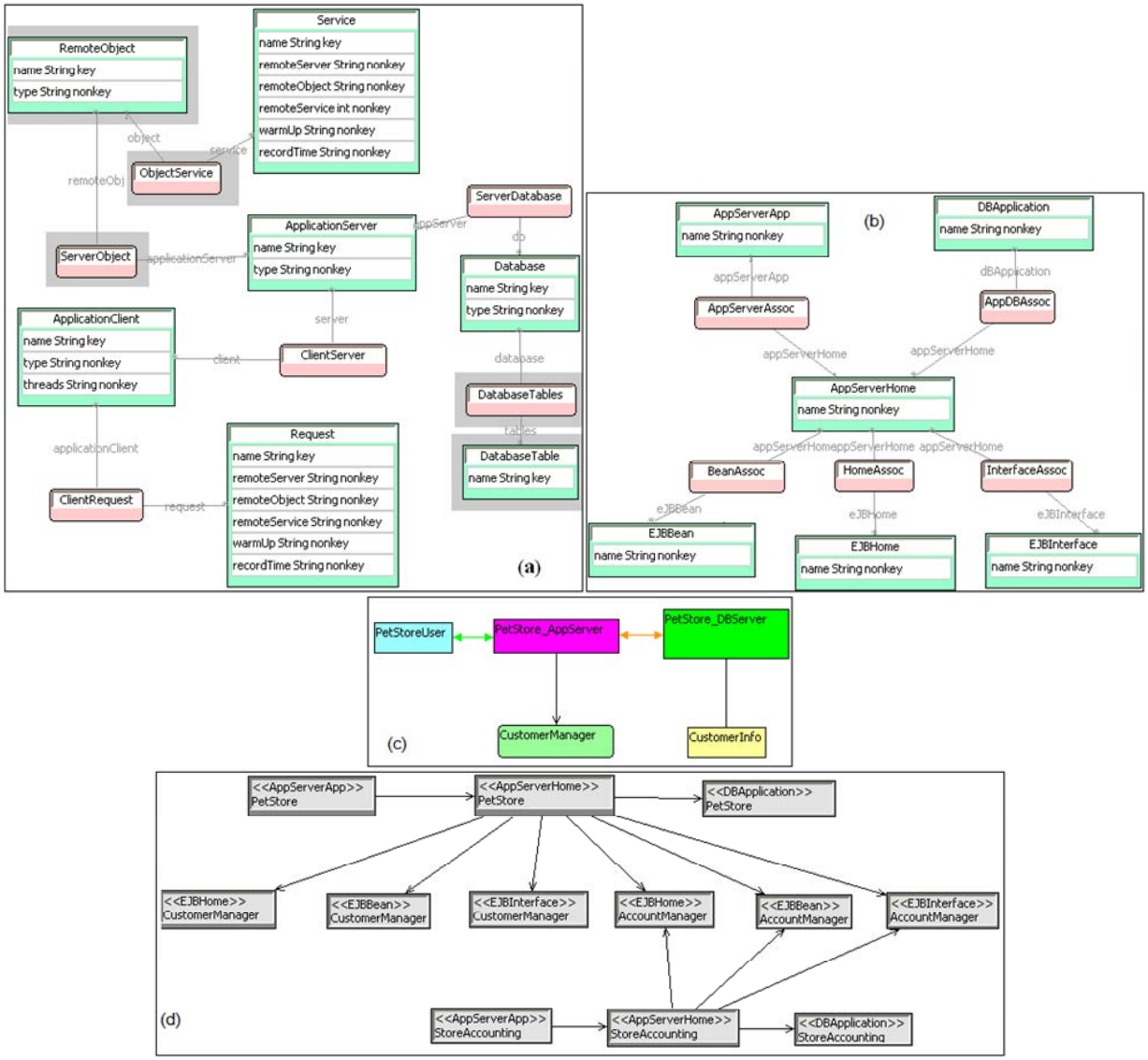


Figure 9.1. (a) MaramaMTE architecture meta-model; (b) EJBUML meta-model; (c) Pet Store MTE-architecture model; (d) Pet Store EJBUML model

Figure 9.1(b) illustrates an extended UML meta-model for EJB (EJBUML meta-model). This extended UML meta-model represents the basic concepts of EJB, including application server home, server

application, EJB home, EJB Interface, EJB Bean. The extended UML meta-model is similar to the formalized UML profile for EJB (Greenfield, 2001) but only focused on a smaller range of EJB concepts. For the running example, the extended UML meta-model is used as the target meta-model to be interconnected.

Figure 9.1(c) illustrates a simple java Pet Store architecture model. The model shows that the "PetStoreUser" (typed as *ApplicationClient*) can access the "PetStore_AppServer" (typed as *ApplicationServer*) to obtain services. The "PetStore_AppServer" hosts the "CustomerManager" (typed as *RemoteObject*) that provides services to clients. The "PetStore_AppServer" can obtain services from the "PetStore_DBServer" (typed as *DatabaseServer*), which, in turn, hosts database table "CustomerInfo". Figure 9.1(d) illustrates a simple Pet Store EJBUML design model. The stereotype of each model entity represents the similar concept as is in the standard java EJB technology. The two sample domain-specific meta-models and their instance models will be used to demonstrate the CRelation model's abstract syntax, semantics, and features through the rest of this chapter.

9.3 The CRelation model

The CRelation model is designed through meta-modelling as shown in Figure 9.2(a). The CRelation meta-model consists of four entities (green rectangles), *StructureMapping*, *SelectionRefinement*, *StructureRefinement*, and *SemanticAssociation*; and three associations (pink round rectangles) that connect those entities. At this stage, the associations are simple connectors (*StructureRefineAssoc* connecting *StructureMapping* and *StructureRefinement*; *StructureSelectionAssoc* connecting *StructureMapping* and *SelectionRefinement*; *StructureAssocAssoc* connecting *StructureMapping* and *SemanticAssociation*) and do not contain much semantics. The CRelation model semantics is carried by the four abstraction entities. The entities and associations provide model types to be instantiated in a CRelation instance model.

Figure 9.2(b) is a sample CRelation model that interconnects the MaramaMTE architecture meta-model with the EJBUML meta-model. Figure 9.2(b) consists of three instances of *StructureMapping* (grey rectangle), two instances of *SelectionRefinement* (cyan round rectangle), one instance of *StructureRefinement* (green round rectangle), and two instances of *SemanticAssociation* (purple oval). The meaning of the CRelation model will be introduced in detail in the rest of the chapter.

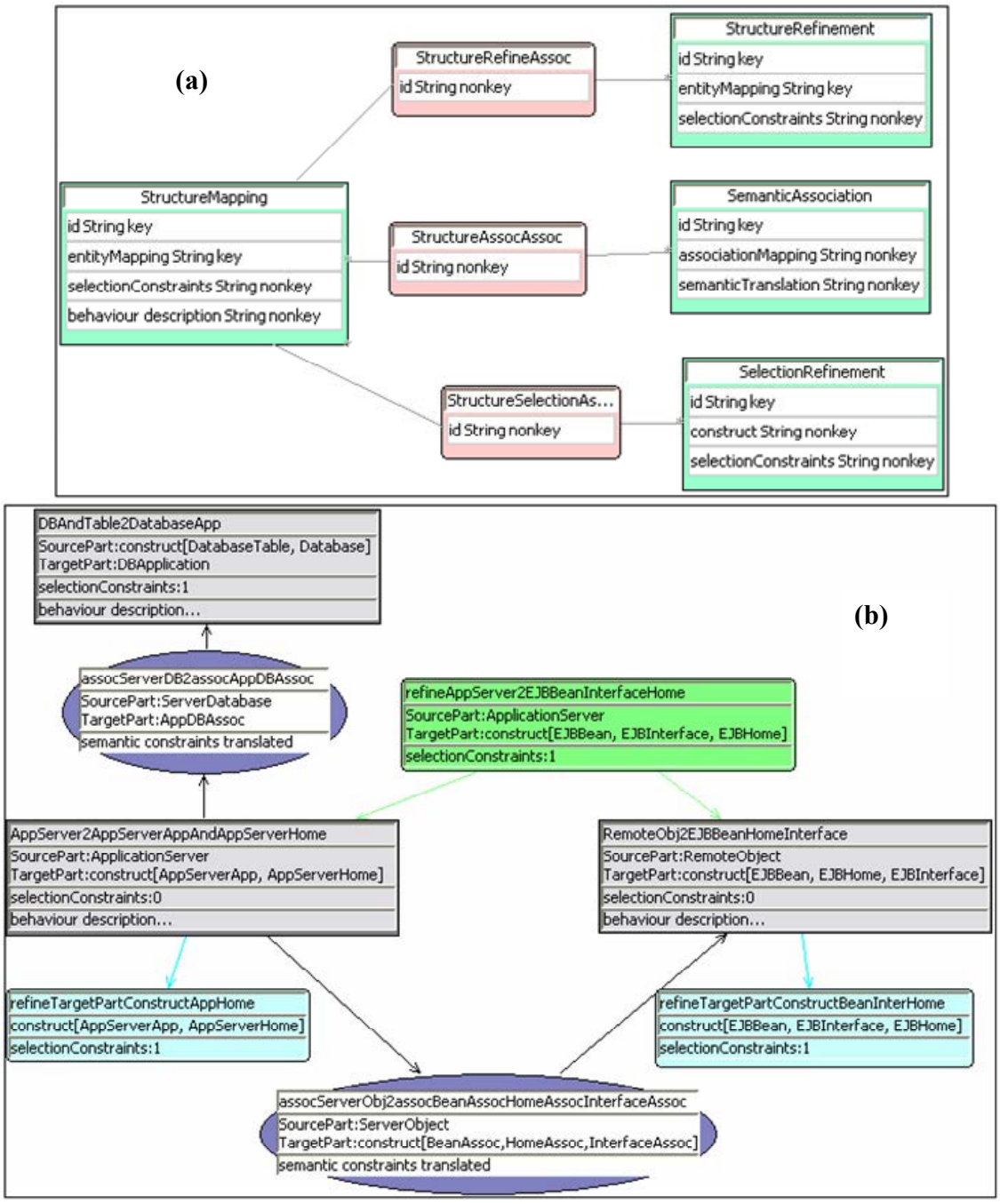


Figure 9.2. (a) CRelation meta-model; (b) a sample CRelation model

9.4 The CRelation model abstract syntax and semantics

A CRelation model is consisted of StructureMappings, SelectionRefinements, StructureRefinements, SemanticAssociations, and associations connecting them. The semantics of the CRelation model is carried by those modelling entities, associations, and their properties. This section introduces the semantics of each modelling entity.

9.4.1 StructureMapping

A StructureMapping represents an atomic Interconnection Relationship between two meta-models. It defines what and how to interconnect interested source and target meta-elements by using four properties: *id, entityMapping, selectionConstraints,* and *behaviourDescription*. Sample properties and values of a StructureMapping ("DBAndTable2DatabaseApp" in Figure 9.2(b)) are illustrated in Figure 9.3(a). The value of "selectionConstraints" is illustrated in Figure 9.3(b) and the value of "behaviourDescription" is illustrated in Figure 9.3(c).

9.4.1.1 *"id" property*

The "id" property distinguishes one StructureMapping from other model elements in the CRelation model. The "id" property value needs to be concise and meaningful. Typical value can be "DBAndTable2DatabaseApp", "AppServer2AppServerAppAndAppServerHome", and "RemoteObj2EJBBeanHomeInterface" (refer to Figure 9.2(b)).

9.4.1.2 *"entityMapping" property*

The "entityMapping" property consists of two parts: source and target, representing the interested source and target meta-elements respectively. Both source and target parts can be a single meta-model element or a construct of meta-model elements. The term *construct* represents a set of meta-elements, and is of the form of "construct [element, element, …]". In Figure 9.3(a), the entityMapping "SourcePart:construct[DatabaseTable,Database] TargetPart:DBApplication" means that the software modeller understands the conceptual semantic similarities between the source meta-model construct (made up of "DatabaseTable" and "Database") and the target meta-model element "DBApplication", and wishes to capture the similarities by establishing an Interconnection Relationship between them.

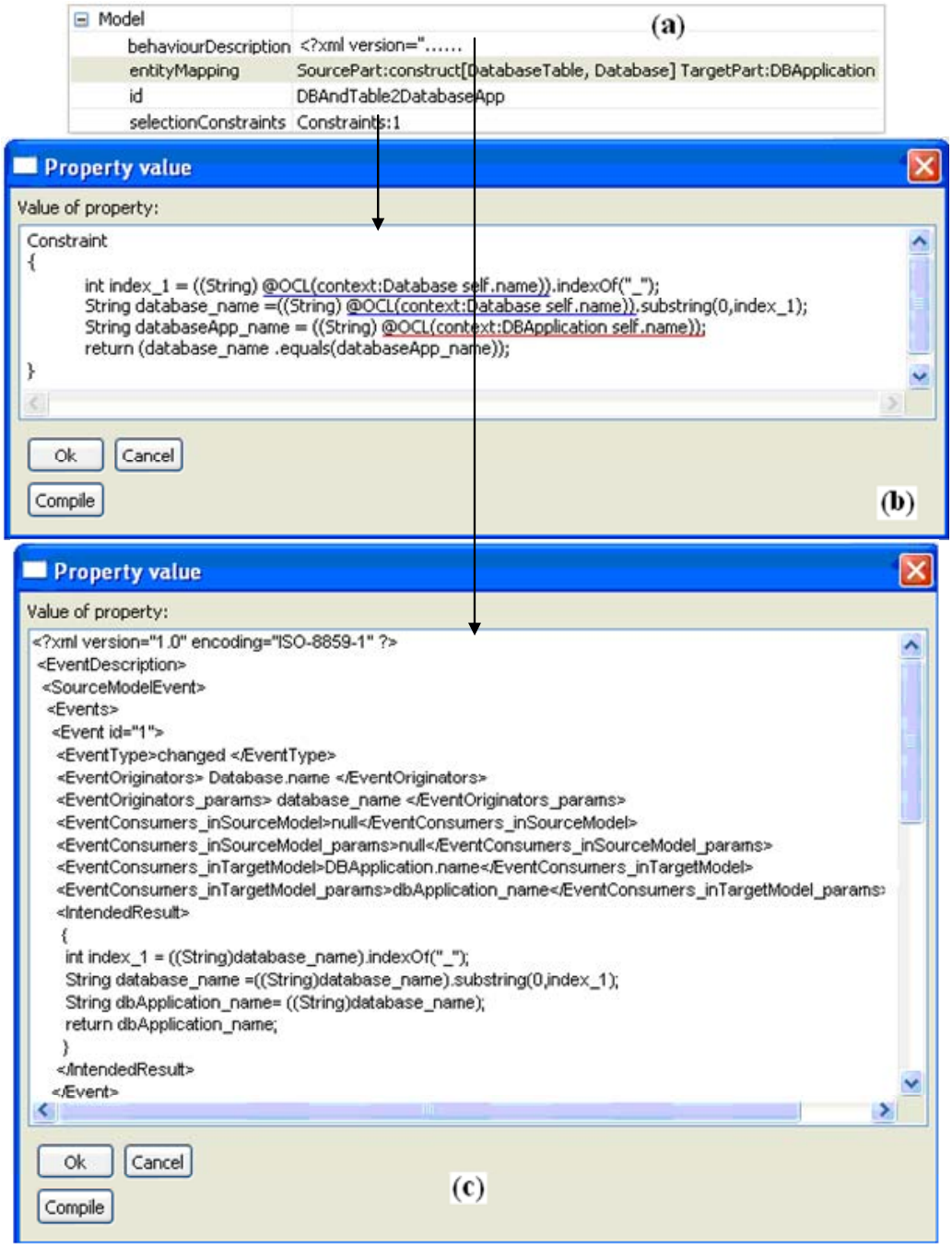


Figure 9.3. (a) sample property sheet of a "StructureMapping"; (b) sample "selectionConstraints" property sheet; (c) sample "behaviourDescription" property sheet

9.4.1.3 *"selectionConstraints" property*

The "selectionConstraints" property represents conditions that need to be satisfied before the Interconnection Relationship defined by the StructureMapping becomes valid. More specifically, when interconnecting instance models, the constraints determine which instances of the StructureMapping source part can be integrated with which instances of the StructureMapping target part. Figure 9.3(b) shows that property "selectionConstraints" of the StructureMapping contains one constraint. The instances of "construct [DatabaseTable, Database]"can be interconnected with the instances of element "DBApplication" only when the constraint is satisfied (return result variable to be "true").

A selection constraint is in the form of "Constraint{ constraint contents }" (refer to Figure 9.3(b)). The constraint contents are a mixture of OCL queries and java code. The CRelation model uses OCL expressions to query information of model elements involved in the StructureMapping, and uses java to construct operations upon the queried information. A selection constraint returns a boolean result through a boolean variable. In a selection constraint, the OCL expressions are in the form of "@OCL(ordinary OCL expressions)".The mixture of OCL and java allows users to construct tool-API-independent complex selection constraints. If the OCL expressions are viewed as special java operations, a selection constraint follows java language syntax. The "Compile" button in Figure 9.3(c) compiles if the constraints are valid. In Figure 9.3(a), the property entry of "selectionConstraints" records the number of constraints involved in the StructureMapping. "Constraints: 1" means that there is only one constraint for this StructureMapping. The "selectionCnstraints" can only constrain elements involved in this StructureMapping. The property represents users' understanding of how the source and target parts share semantics. The constraints can be built up incrementally. The more the constraints, the fewer instances of the meta-model elements from both instance models can be integrated.

9.4.1.4 *"behaviourDescription" property*

The "behaviourDescription" property records the events that can be triggered by the StructureMapping and specifies the response actions that need to be taken to maintain the validity of the Interconnection Relationship. The value of the "behaviourDescription" property is XML-formatted. The "behaviourDescription" property is designed to organize information of the triggered. In the "behaviorDescription" value, the information of the triggered events is automatically generated, and the information of the response events needs to be manually programmed.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://tempuri.org/po.xsd"
xmlns="http://tempuri.org/po.xsd" elementFormDefault="qualified">

  <xs:element name="BehaviorDescription" type="BehaviorDescriptionType"/>
  <xs:complexType name="BehaviorDescriptionType">
    <xs:sequence>
      <xs:element name="SourceModelEvent" type="SourceModelEventType"/>
      <xs:element name="TargetModelEvent" type="TargetModelEventType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="SourceModelEventType">
    <xs:sequence>
      <xs:element name="Events"  type="EventsType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="TargetModelEventType">
      <xs:sequence>
      <xs:element name="Events"  type="EventsType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="EventsType">
    <xs:sequence>
      <xs:element name="Event" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:element name="EventType" type="Concerned-Events"/>
          <xs:element name="EventOriginators" type="SourceElemList"/>
          <xs:element name="EventOriginators_params" minOccurs="0" type="SourceElemValueList"/>
          <xs:element name="EventConsumers_inSourceModel" type="SourceConsumerElemList"/>
          <xs:element name="EventConsumers_inSourceModel_params" minOccurs="0" type="SrcConList"/>
          <xs:element name="EventConsumers_inTargetModel" type="TargetConsumerElemList"/>
          <xs:element name="EventConsumers_inTargetModel_params" minOccurs="0" type="TgtConList"/>
          <xs:element name="IntendedResult" type="xs:string"/>
        </xs:complexType>
        <xs:attribute name="id" type="xs:string"/>
      </xs:element >
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="Concerned-Events">
    <xs:restriction base="xs:string">
      <xs:enumeration value="changed"/>
      <xs:enumeration value="removed"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="SourceElemList">
    <xs:list itemType="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="SourceElemValueList">
    <xs:list itemType="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="SourceConsumerElemList">
    <xs:list itemType="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="SrcConList">
    <xs:list itemType="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="TargetConsumerElemList">
    <xs:list itemType="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="TgtConList">
    <xs:list itemType="xs:string"/>
  </xs:simpleType>
</xs:schema>
```

Figure 9.4. The schema of the behavior synchronization description

Figure 9.4 shows the complete schema of the behaviour description. This consists of "SourceModelEvent" and "TargetModelEvent" (refer to lines 7&8), which respectively organize events that are triggered by source and target parts of the StructureMapping. Both "SourceModelEvent" and "TargetModelEvent" contain "Events", which, in turn, consist of 0 or more events. Each "Event" contains a sequence of the following elements:

- ***EventType:*** Type of the event. So far only two types are concerned in the CRelation model, including "changed" (refer to line 44), and "removed" (refer to line 45). A "changed" event is triggered when the property value of a model element that will influence the selection constraints is changed. A "removed" event is triggered when the source or target part of the Interconnection Relationship is removed.
- ***EventOriginators:*** The model elements that trigger the event.
- ***EventOriginators_params:*** This is a list of parameters matching the list of "EventOriginators". These parameters represent the "EventOriginators" in "*IntendedResult*".
- ***EventConsumers_inSourceModel:*** The source meta-model elements that need to respond to the triggered events.
- ***EventConsumers_inSourceModel_params:*** This is a list of parameters matching the list of "EventConsumers_inSourceModel". These parameters represent the "EventConsumers_inSourceModel" in "*IntendedResult*".
- ***EventConsumers_inTargetModel:*** The target meta-model elements that need to respond to the triggered events.
- ***EventConsumers_inTargetModel_params:*** This is a list of parameters matching the list of "EventConsumers_inTargetModel". These parameters represent the "EventConsumers" in "*IntendedResults*".
- ***IntendedResult:*** IntendedResult is structured information (e.g. functional java code) that describes the result the response event should achieve. In the CRelation model, only two types of IntendedResult are concerned at this stage: 1) to maintain the selection constraints to be true when property values of model elements are changed; 2) to maintain the validity of the Interconnection Relationship when the source or target part of a StructureMapping is removed. In the first situation, response events need to update the event consumers' property value, and the modeller needs to manually calculate the intended updated value. In the second situation, response events need to either remove the event consumers or simply remove the Interconnection Relationships between the

event originators and event consumers. The CRelation model reserves the keyword *InterconnectionRelationship* to represent the Interconnection Relationship between the source and target parts of a StructureMapping.

Figure 9.5 and Figure 9.6 illustrate the complete behaviour description of StructureMapping "DBAndTable2DatabaseApp" of Figure 9.2(b). The StructureMapping has only one selection constraint illustrated in Figure 9.3(b). There are 3 events triggered by the source part of the StructureMapping (Figure 9.5) and two events triggered by the target part of the StructureMapping (Figure 9.6).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
 <EventDescription>
  <SourceModelEvent>
   <Events>
    <Event id="1">
     <EventType>changed </EventType>
     <EventOriginators> Database.name </EventOriginators>
     <EventOriginators_params> database_name </EventOriginators_params>
     <EventConsumers_inSourceModel>null</EventConsumers_inSourceModel>
     <EventConsumers_inSourceModel_params>null</EventConsumers_inSourceModel_params>
     <EventConsumers_inTargetModel>DBApplication.name</EventConsumers_inTargetModel>
     <EventConsumers_inTargetModel_params>dbApplication_name</EventConsumers_inTargetModel_params>
     <IntendedResult>
      {
       int index_1 = ((String)database_name).indexOf("_");
       String database_name =((String)database_name).substring(0,index_1);
       String dbApplication_name= ((String)database_name);
       return dbApplication_name;
      }
     </IntendedResult>
    </Event>
    <Event id="2">
     <EventType> removed </SourceEventType>
     <EventOriginators> Database</EventOriginators>
     <EventConsumers_inSourceModel>DatabaseTable</EventConsumers_inSourceModel>
     <EventConsumers_inTargetModel> DBApplication </EventConsumers_inTargetModel>
     <IntendedResult>
     {InterconnectionRelationship.removed();}
     </IntendedResult>
    </Event>
    <Event id="3">
     <EventType> removed </EventType>
     <EventOriginators> DatabaseTable</EventOriginators>
     <EventConsumers_inSourceModel>Database</EventConsumers_inSourceModel>
     <EventConsumers_inTargetModel> DBApplication </EventConsumers_inTargetModel>
     <IntendedResult>
     {InterconnectionRelationship.removed();}
     </IntendedResult>
    </Event>
   </Events>
```

Figure 9.5. "SourceModelEvent" part of the "behaviourDescription"

In Figure 9.5, event_1 (line 5) describes that when the "name" property value of model element *Database* is changed, the response event needs to be programmed to calculate the intended updated value of the "name" property of the "DBApplication" entity (target part of the StructureMapping). The IntendedResult (response event) of Event_1 (between line 13 and line 20) is based on the selection constraints of the StructureMapping (described in Figure 9.3(b)) and need to be programmed manually by modellers. The EventType of both Event_2 (line 22) and Event_3 (line 31) is "removed". They represent that when Database or DatabaseTable is removed; the target part of the StructureMapping will be influenced and should respond. The IntendedResult of both events use "InterconnectionRelationship.removed()" (an expression reserved by the CRelation model) to specify that if the event originator is removed the Interconnection Relationship between the event originators and event consumers needs to be removed. At this stage, removing an Interconnection Relationship can be achieved by either removing the event consumers or removing the Interconnection Relationship between the event originators and event consumers.

```
1  <TargetModelEvent>
2   <Events>
3    <Event id="4">
4     <EventType>changed </EventType>
5     <EventOriginators> DBApplication.name </EventOriginators>
6     <EventOriginators_params>dbApplication_name </EventOriginators_params>
7     <EventConsumers_inSourceModel>Database.name</EventConsumers_inSourceModel>
8     <EventConsumers_inSourceModel_params>database_name</EventConsumers_inSourceModel>
9     <EventConsumers_inTargetModel>null</EventConsumers_inTargetModel>
10    <EventConsumers_inTargetModel_params>null</EventConsumers_inTargetModel_params>
11    <IntendedResult>
12     {
13      int index_1 = ((String)database_name).indexOf("_");
14     String database_name =((String)database_name).substring(0,index_1);
15     String database_suffix =((String)database_name).substring(index_1);
16     database_name = dbApplication_name + database_suffix;
17     return database_name;
18     }
19    </IntendedResult>
20   </Event>
21   <Event id="5">
22    <SourceEventType>removed </SourceEventType>
23    <EventSource>DBApplication</EventSource>
24    <EventTargets_inSourceModel> null</EventTargets_inSourceModel>
25    <EventTargets_inTargetModel>construct[DatabaseTable,Database]</EventTargets_inTargetModel>
26    <IntendedResult>
27     {
28      InterconnectionRelationship.removed();
29     }
30    </IntendedResult>
31   </Event>
32  <TargetModelEvent>
```

Figure 9.6. "TargetModelEvent" part of the "behaviourDescription"

Figure 9.6 organizes the events triggered by the target part of the StructureMapping. The "changed" event (line 4, Figure 9.6) represents that the change of DBApplication "name" property value requires appropriate response events from the source part of the StructureMapping. The "removed" event (line 22, Figure 9.6) represents when the DBApplication is removed, this Interconnection Relationship also needs to be removed.

As is shown in Figure 9.5 and Figure 9.6, the "behaviourDescription" of a StructureMapping in the CRelation model is based on the meta-elements and the selection constraints of the StructureMapping. The "behaviourDescription" provides bi-directional specifications for behaviour synchronization. It is a structured high-level specification, which helps users to organize and implement bi-directional incremental behaviour synchronization for MI&T. Chapter 10 will explain how to generate "behaviourDescription" for a StructureMapping.

9.4.1.5 *Sample use of StructureMapping*

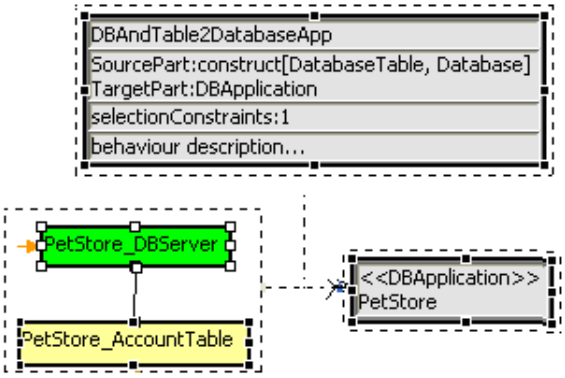


Figure 9.7. Example usage of StructureMapping

The information defined in the StructureMappings in the sample CRelation model (Figure 9.2(b)) guides the interconnection between the Pet Store architecture model (Figure 9.1(c)) and the Pet Store design model (Figure 9.1(d)). Figure 9.7 shows StructureMapping DBAndTable2DatabaseApp interconnects Pet Store architecture model with Pet Store design model, where the instances of MaramaMTE Database (PetStore_DBServer) and DatabaseTable (PetStore_AccountTable) are interconnected to the instance of EJBUML DBApplication (PetStore).

9.4.2 SelectionRefinement

The selection constraints of a StructureMapping can be complicated. A selection constraint may apply across source and target parts or just on either source or target part of a StructureMapping. A SelectionRefinement is designed to specify selection constraints on a construct part (either source or target part) of a StructureMapping. A SelectionRefinement may have one or more parent StructureMappings in the CRelation model. Selection constraints defined in a SelectionRefinement apply to its parent StructureMapping(s). The CRelation model allows a StructureMapping to provide its own selection constraints as well as refine complex selection constraints to one or more SelectionRefinements.

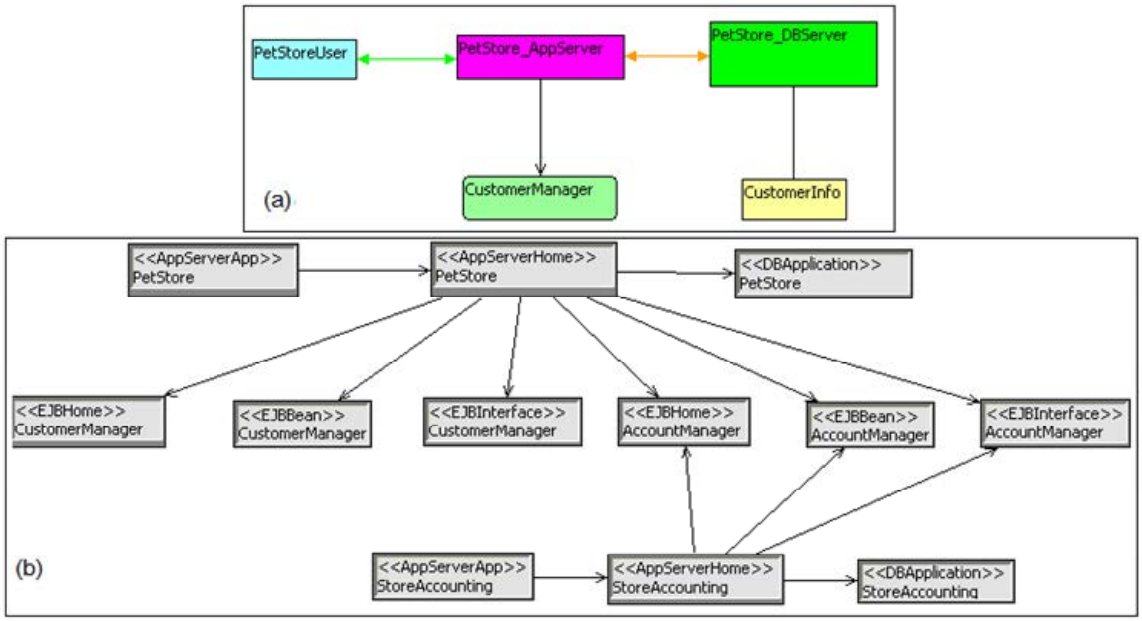


Figure 9.8. (a) a sample Pet Store MaramaMTE model; (b) a sample Pet Store EJBUML model

The selection constraints involved in single-model normally derive from the semantic constraints of that model. For example, the Pet Store EJBUML model (Figure 9.8(b)) requires that a valid set of EJBHome, EJBBean, and EJBInterface must follow certain naming conventions, e.g. if there is a "CustomerManager" EJBInterface, there must be a corresponding "CustomerManager" EJBBean and a "CustomerManger" EJBHome. When software modellers want to integrate the "CustomerManager" (a

remote object in the Pet Store architecture model in Figure 9.8(a)) with a set of EJBHome, EJBBean, and EJBInterface (in EJBUML design model in Figure 9.8(b)), it can only integrate with either a set of "CustomerManager" (EJBHome), "CustomerManager" (EJBBean), and "CustomerManager" (EJBInterface), or a set of "AccountManager"(EJBHome), "AccountManager"(EJBBean), and "AccountManager"(EJBInterface). A set of "CustomerManager" (EJBBean), AccountManager"(EJBHome), and "CustomerManager"(EJBinterface) will not meet the naming convention required in the EJB UML meta-model.

A SelectionRefinement uses three properties to specify its features; and they are: *id*, *construct*, and *selectionConstraints*. Figure 9.9(a) is the property sheet of a SelectionRefinement, and Figure 9.9(b) illustrates the selection constraint of the SelectionRefinement.

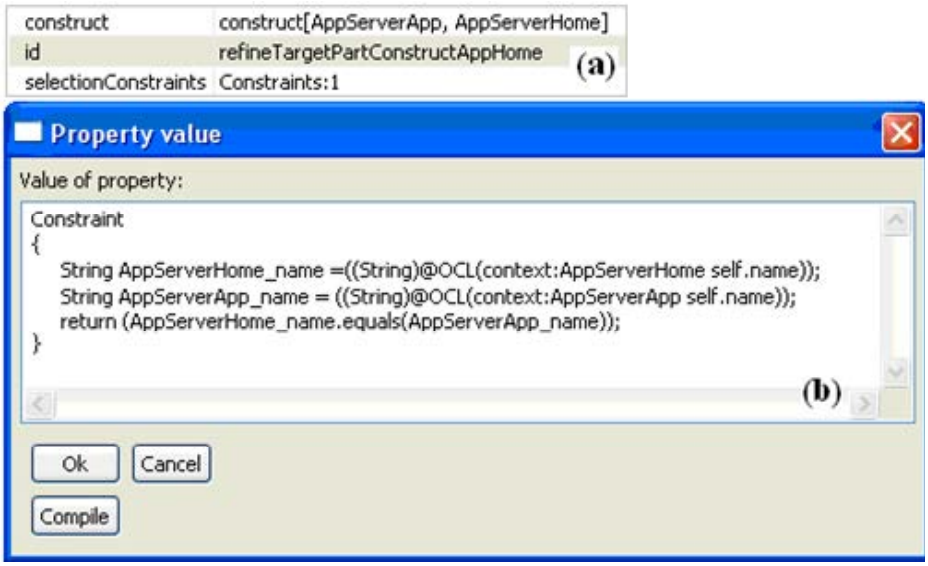


Figure 9.9. Property sheet and sample property value of a SelectionRefinement

9.4.2.1 *"id" property*

The "id" property identifies a SelectionRefinement from other model elements. Value of "id" property can be any concise and meaningful textual value. Example value can be "refineTargetPartConstructAppHome" and "refineTargetPartConstructBeanInterHome" (the two cyan shapes in Figure 9.2(b)).

9.4.2.2 *"construct" property*

The "construct" property value consists of only one part: either the source part or the target part of its parent StructureMappings. The interested part can only be a construct, because only a construct need the single-model selection constraints.

9.4.2.3 *"selectionConstraints" property*

The "selectionConstraints" property represents conditions that need to be satisfied during model interconnection. The selectionConstraints of a SelectionRefinement are similar to the selectionConstraints of a StructureMapping; but can only use information of the model elements involved in this SelectionRefinement. Figure 9.9(b) shows one selection constraint of the SelectionRefinement "refineTargetPartConstructAppHome" (refer to Figure 9.2(b)). The selection refinement provides extra selection constraints for its parent "AppServer2AppServerAppAndAppServerHome" (refer to Figure 9.2(b)). At this stage, the *selectionConstraints* brought by a SelectionRefinement do not contribute to the *behaviorDescription* of its parent StructureMappings"; but their influence on the *behaviorDescription* of the parent StructureMappings will be researched in the future.

9.4.2.4 *Sample use of SelectionRefinement*

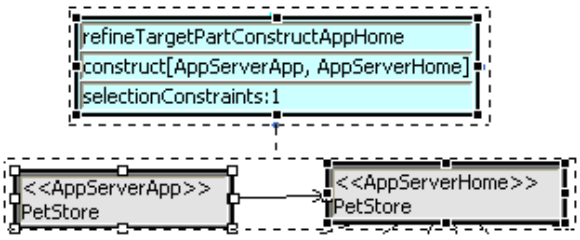


Figure 9.10. Example usage of SelectionRefinement

The information defined in the SelectionRefinements in the sample CRelation model (Figure 9.2(b)) guides the interconnection between the Pet Store architecture model (Figure 9.1(c)) and the Pet Store design model (Figure 9.1(d)). Figure 9.10 shows that the SelectionRefinement "refineTargetPartConstructAppHome" specifies the naming convention between an instance of

AppServerApp (PetStore) and an instance of AppServerHome (PetStore) (part of their names must be the same).

9.4.3 StructureRefinement

The CRelation model uses a StructureRefinement to specify selection constraints between the source part of one StructureMapping and the target part of another StructureMapping. A StructureRefinement specifies second order selection constraints between two StructureMappings. A StructureRefinement uses three properties, *id*, *entityMapping*, and *selectionConstraints* to specify its characters. Figure 9.11 shows a property sheet of a StructureRefinement.

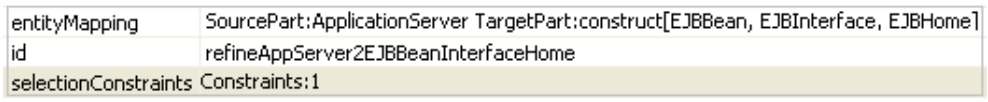


Figure 9.11. Properties and values of an example StructureRefinement

9.4.3.1 *"id" property*

The "id" property distinguishes one StructureRefinement from other model elements. Value of "id" property can be anything concise and meaningful. Typical value can be "refineAppServer2EJBBeanInterfaceHome" (refer to Figure 9.2(b)).

9.4.3.2 *"entityMapping" property*

The "entityMapping" property value consists of two parts: source meta-model elements and target meta-model elements. The source and target parts are derived from its parent StructureMappings, and can be either a single meta-model element or a construct. A sample "entityMapping" value is illustrated in Figure 9.11, where the source part is "ApplicationServer", and the target part is "construct[EJBBean, EJBHome, EJBInterface]".

9.4.3.3 *"selectionConstraints"*

The "selectionConstraints" property represents conditions that need to be satisfied. The selectionConstraints of a StructureRefinement are similar to the selectionConstraints of a StructureMapping; and can only use the information of the model elements involved in this StructureRefinement. Figure 9.12 illustrates the selection constraint brought by StructureRefinement

"refineAppServer2EJBBeanInterfaceHome" (refer to Figure 9.2(b)). In Figure 9.12, this selection constraint finds out the RemoteObjects associated with the ApplicationServer (via navigation to the association end of ServerObject, refer to the MaramaMTE meta-model in Figure 9.1(a)), and requires that the ApplicationServer is associated with the *RemoteObject* that is mapped to the target model construct containing the EJBHome and has the same *name* value of the EJBHome. At this stage, the *selectionConstraints* brought by a StructureRefinement do not contribute to the *behaviorDescription* of its parent StructureMappings; but their influence on the *behaviorDescription* of the parent StructureMappings will be researched in the future.

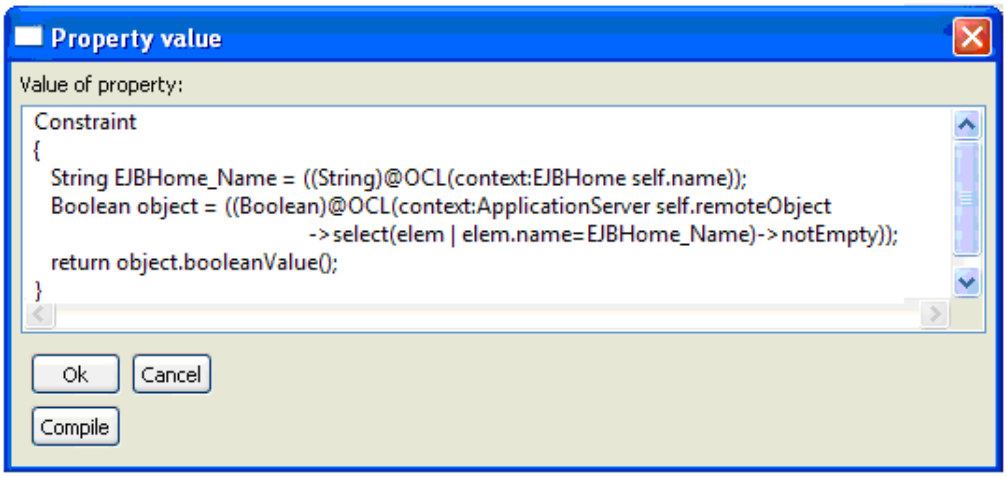


Figure 9.12. Sample value of *selectionConstraints* of a StructureRefinement

9.4.3.4 *Sample use of StructureRefinement*

The information defined in StructureRefinement is used to guide the interconnection between the Pet Store architecture model and design model. Figure 9.13 shows that the "refineAppServer2EJBeanInterfaceHome" StructureRefinement qualifies its two parent StructureMappings by constraining the particular RemoteObject that the PetStore_AppServer associates with (in this case "CustomerManager") to be the same as the RemoteObject (also in architecture model) that is mapped to the EJB CustomerManager elements (in design model).

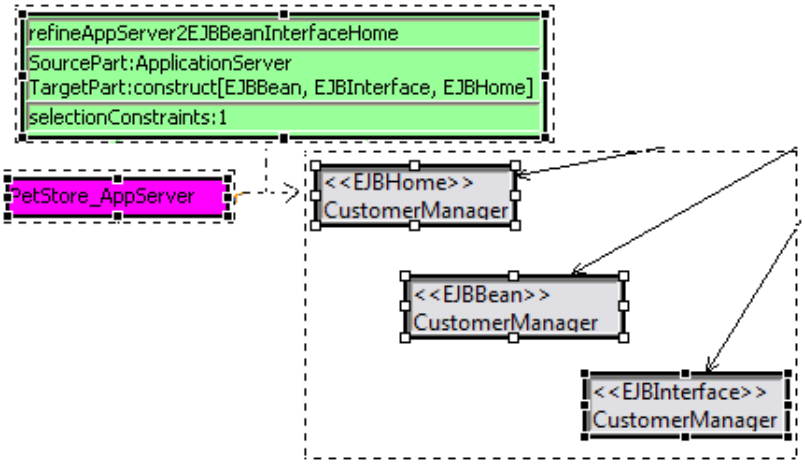


Figure 9.13. Example usage of StructureRefinement

9.4.4 Brief summary of StructureMapping, SelectionRefinement, and StructureRefinement

Until now, StructureMappings capture what semantics between two models are shared - by *entityMapping*, how the semantics should be shared – by *selectionConstraints*, and the events that can be triggered by the shared semantics – by *behaviorDescription*. SelectionRefinement and StructureRefinement refine selection constraints of StructureMappings. The selection constraints are categorized by the involved meta-elements.

But the StructureMappings are isolated. Two individual StructureMappings may involve other model information, such as the associations between the source parts and the associations between the target parts of the two StructureMappings. The involved information is implicit and normally ignored during MI&T (think about the isolated ATL rules and queries, and the transformation information they imply and ignore). The ignored information may relate to semantic inconsistencies caused by MI&T. The CRelation model explicitly represents the implied information by using SemanticAssociation entities.

9.4.5 SemanticAssociation

A SemanticAssociation specifies an association between two StructureMappings. A SemanticAssociation explicitly represents the source meta-model and target meta-model associations that are involved and maintained during MI&T. A SemanticAssociation uses three properties to define its semantics, including *id, associationMapping*, and *semanticTranslation*.

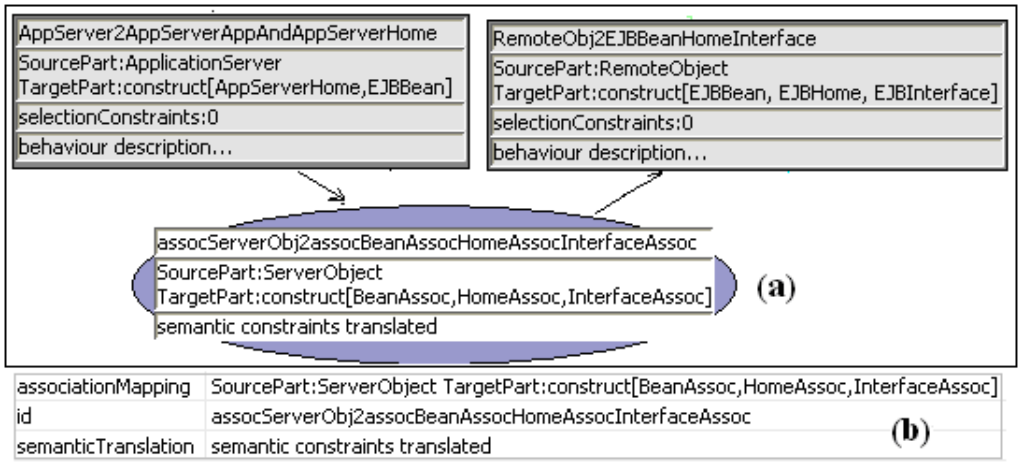


Figure 9.14. (a) a SemanticAssociation associates two StructureMappings ; (b) property sheet of a sample SemanticAssociation

9.4.5.1 *"id" property*

The "id" property distinguishes one AssociationMapping from other model elements. The value of the "id" property needs to be concise and meaningful, such as assocServerDB2assocAppDBAssoc, and assocServerObj2assocBeanAssocHomeAssocInterface (refer to Figure 9.2(b)).

9.4.5.2 *"associationMapping" property*

The "associationMapping" property value consists of two parts: source and target. The source part represents the directed path connecting source parts of the associated StructureMappings in the source meta-model. The target part represents the directed path connecting target parts of the associated StructureMapping in the target meta-model. Figure 9.14(b) illustrates the property sheet of the SemanticAssociation in Figure 9.14(a).

9.4.5.2.1 The definition for deriving "associationMapping" value

In a SemanticAssociation, the value of the "associationMapping" property is *automatically* derived from the associated StructureMappings according to the abstract syntax of the source and target meta-models. Figure 9.14(b) shows the value of "associationMapping" property of SemanticAssociation assocServerDB2assocAppDBAssoc (in Figure 9.14(a)). The source part of the SemanticAssociation is "ServerObject", which is a directed path between "ApplicationServer" (source part of StructureMapping

AppServer2AppServerAppAndAppServerHome) and “RemoteObject” (source part of StructureMapping RemoteObj2EJBBeanHomeInterface) in the source meta-model (refer to Figure 9.1(a)). The target part of the SemanticAssociation is “construct[BeanAssoc, HomeAssoc, InterfaceAssoc]”, which is a directed path between “construct[AppServerApp, AppServerHome]” (target part of StructureMapping AppServer2AppServerAppAndAppServerHome) and “construct[EJBBean, EJBHome], EJBInterface]”(target part of StructureMapping RemoteObj2EJBBeanHomeInterface) in the target meta-model (refer to Figure 9.1(b)). The source (target) part of a SemanticAssociation may be a single model element or a construct of model elements.

When constructing a SemanticAssociation, the CRelation model needs to determine the paths between two vertices of a model. It is straightforward to determine the paths between two single model elements within a meta-model, but it can be complex to determine paths between two construct vertices. The CRelation model empirically defines that when the two construct vertices are the same, the paths between the vertices become the paths within the construct vertex; as a construct vertex is a valid sub model of a meta-model, the paths within a construct vertex are actually available paths within that sub model.

The CRelation model defines paths between *two different construct vertices* in two steps:

- ***Step 1: Define the path between an element and a construct***
  The path between an element of construct vertex 1 and construct vertex 2 is a construct of the paths between this element and the elements of construct vertex 2. Any paths within construct vertex 1 must be excluded from the construct of paths.
- ***Step 2: Define the paths between two constructs***
  The paths between two construct vertices are a list of paths between one element of one construct and the other construct;

This definition is illustrated in Figure 9.15(a). The construct vertices 1 and 2 represent the source parts of the two associated StructureMappings respectively. There are five entities involved in the two vertices including v1 and v2 of construct vertex 1; and v3, v4, and v5 of construct vertex 2. V1 has two paths between itself and the elements of construct vertex 2, so the path between v1 and construct vertex 2 is “construct[p13, p14]”. V2 has two paths between itself and the elements of construct vertex 2, so the

path between itself and construct vertex 2 is “construct[p24, p25]”. V3 has one path between itself and the elements of construct vertex 1, so the path between itself and construct vertex 1 is “p13”. This process goes on, and the situation in Figure 9.15(a) totally derives 5 paths and they are: “construct[p13, p14]”, “construct[p24, p25]”,”p13”, “construct[p14, p24]”, and “p25”. Applying the definition to the situation in Figure 9.15(b) leads to four paths between the two construct vertices, and they are: “construct[BeanAssoc, HomeAssoc, InterfaceAssoc]”, “BeanAssoc”, “HomeAssoc”, and “InterfaceAssoc”. Once all the paths are automatically derived for both the source and target parts of the SemanticAssociation, modellers can choose a pair of paths that share the closest semantics to set up the value of “associationMapping” property.

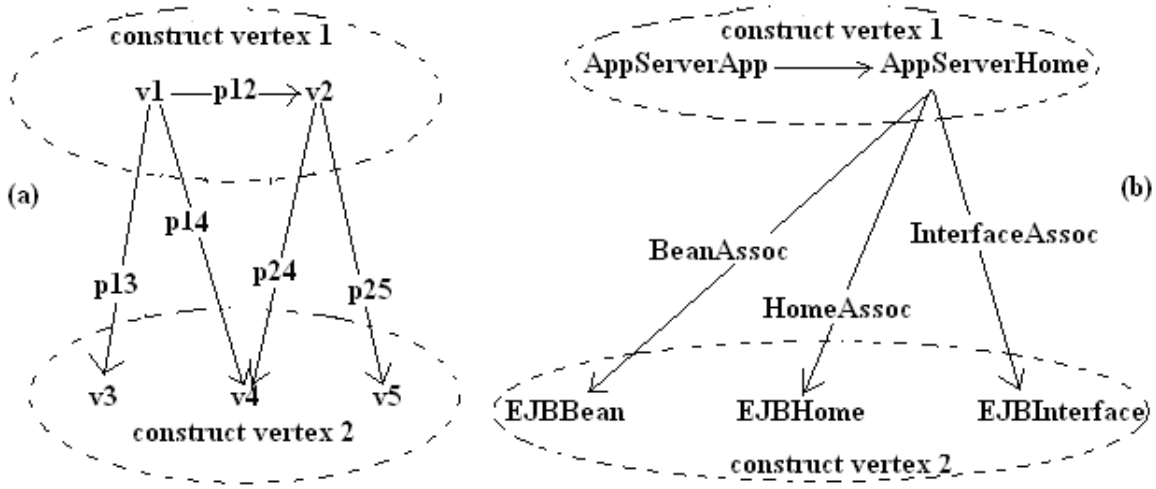


Figure 9.15. Directed paths between source (target) parts of the associated StructureMappings

9.4.5.2.2 Using associationMapping value to analyze and design MI&T

Figure 9.16 illustrates how a SemanticAssociation helps to validate an intended MI&T. Figure 9.16(a) shows a tentative design for MI&T between the MaramaMTE meta-model and the EJBUML meta-model, where two StructureMappings are associated by one SemanticAssociation. According to the definition in section 9.4.5.2.1, there is only one directed path between the source parts of the two intended StructureMappings, and the path is “ServerDatabase” (refer to Figure 9.1(a)). There is only one directed path between the target parts of the two intended StructureMappings, and the path is “AppServerAssoc” (refer to Figure 9.1(b)). So, the derived associationMapping value of the SemanticAssociation can only be “SourcePart:ServerDatabase TargetPart:AppServerAssoc”, which does

not match the understanding that the "ServerDatabase" of the MaramaMTE meta-model (in Figure 9.1(a)) should be mapped to the "AppDBAssoc" of the EJBUML meta-model (in Figure 9.1(b)). The semantic inconsistency means the current CRelation model may not have captured the shared semantics correctly and needs to be redesigned.

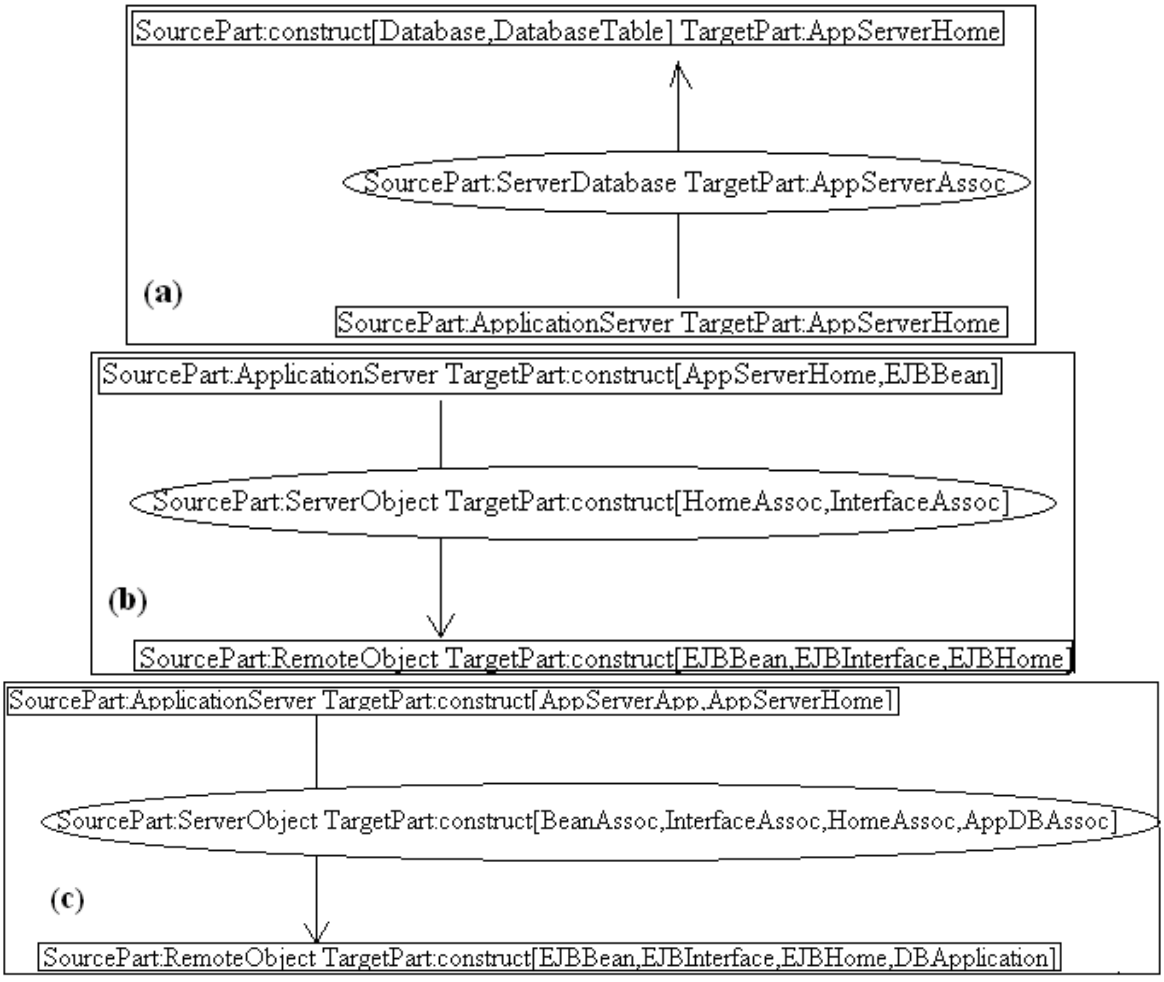


Figure 9.16. Using SemanticAssociation to detect semantic inconsistency during MI&T

Figure 9.16(b) shows the second tentative design for MI&T between the MaramaMTE and EJBUML meta-models. Referring to Figure 9.1(a), there is only one directed path between the source parts of the two intended StructureMappings, and the path is "ServerObject". Referring to Figure 9.1(b), there are three directed paths between the target parts of the two intended StructureMappings, including "construct[InterfaceAssoc, HomeAssoc]", "InterfaceAssoc", and "HomeAssoc". The directed path

"BeanAssoc" is within the top StructureMapping, so it is not a path between the two construct vertices. The derived "associationMapping" value of the SemanticAssociation can be "SourcePart:ServerObject TargetPart:construct[InterfaceAssoc,HomeAssoc]", or "SourcePart:ServerObject TargetPart:InterfaceAssoc}", or "SourcePart:ServerObject TargetPart:HomeAssoc". None of the mappings matches the understanding that the "ServerObject" (in Figure 9.1(a)) should be mapped to the "construct[BeanAssoc, InterfaceAssoc, HomeAssoc]" (in Figure 9.1(b)). The semantic inconsistency means the current CRelation model may not have captured the shared semantics correctly and needs to be redesigned.

Figure 9.16(c) shows the third tentative design between the MaramaMTE and EJBUML meta-models. There is only one directed path between the source parts of the two intended StructureMappings, and the path is "ServerObject" (refer to Figure 9.1(a)). There are five directed path between the target parts of the two intended StructureMappings, and they are: "construct[BeanAssoc, InterfaceAssoc, HomeAssoc, AppDBAssoc]", "BeanAssoc", "InterfaceAssoc", "HomeAssoc", and "AppDBAssoc" (refer to Figure 9.1(b)). So, the derived "associationMapping" value of the SemanticAssociation can be "SourcePart:ServerObject TargetPart:construct[BeanAssoc, InterfaceAssoc, HomeAssoc, AppDBAssoc]", or "SourcePart:ServerObject TargetPart:BeanAssoc", or "SourcePart:ServerObject TargetPart:InterfaceAssoc", or "SourcePart:ServerObject TargetPart:HomeAssoc" or "SourcePart:ServerObject TargetPart:AppDBAssoc". None of the mappings match the understanding that the "ServerObject" (in Figure 9.1(a)) should be mapped to the "construct[BeanAssoc, InterfaceAssoc, HomeAssoc]" (in Figure 9.1(b)). The semantic inconsistency means the current CRelation model may not have captured the shared semantics correctly and needs to be redesigned.

9.4.5.3 *"semanticTranslation" property*

A domain-specific meta-model may contain many semantic constraints (similar to a UML model has OCL constraints). Ideally, all the semantic constraints should be maintained during MI&T. In the CRelation model, the selection constraints have encoded and maintained some of those semantic constraints, where property values of model elements are involved. The CRelation model uses the "semanticTranslation" property of SemanticAssociation to maintain ***Translatable Semantic Constraints***. The translatable model semantic constraints are constraints that act upon source (target) parts of the SemanticAssociation and its parent StructureMappings without involving property values.

Figure 9.17 shows an OCL-formatted translatable semantic constraint of the MaramaMTE meta-model (source meta-model). The constraint (tree root) involves two elements: "RemoteObject" (self) and "ServerObject. "RemoteObject" is the source part of StructureMapping RemoteObj2EJBBeanHomeInterface (in Figure 9.14(a)), and "ServerObject" is the source part of SemanticAssociation assocServerObj2assocBeanAssocHomeAssocInterface (in Figure 9.14(a)). The constraint does not involve the property values of the model elements, so the constraint is translatable. The semantic constraint means "in a MaramaMTE architecture model, each remote object needs to be hosted by one application server". This constraint is sensible as if the application server is removed the remote object must be deleted or re-hosted as well

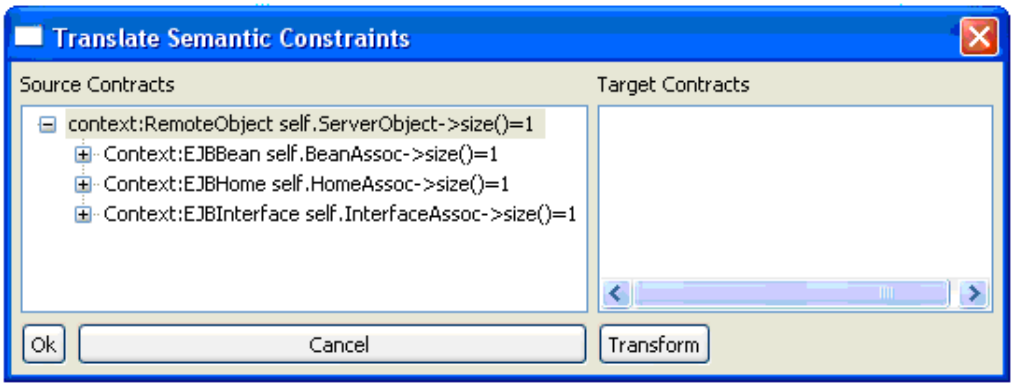


Figure 9.17. Sample semantic constraints and translated semantic constraints

This translatable semantic constraint needs to be translated into sensible target model semantic constraints; and they are the three children of the root semantic constraint. Translated semantic constraint "context:EJBBean self.BeanAssoc->size()=1" (first child of the tree) can be understood as in a EJBUML model, each EJBBean needs to be hosted by one AppServerApp, then when the hosting AppServerApp server is removed the EJBBean must be deleted or re-hosted as well. The translation of semantic constraints helps to maintain more semantics during MI&T. Chapter 10 will explain how to translate the translatable semantic constraints.

9.4.5.4 *Sample use of SemanticAssociation*

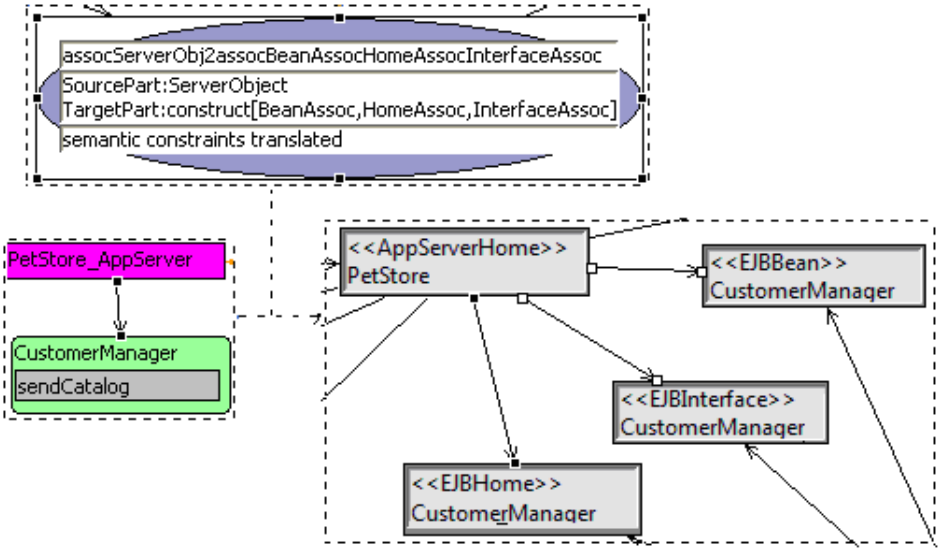


Figure 9.18. Example of using SemanticAssociation

The information defined in SemanticAssociation is used to guide the interconnection between the Pet Store architecture model and the design model. In Figure 9.18, the MaramaMTE ServerObject association (between ApplicationServer and RemoteObject) is being mapped to the BeanAssoc, HomeAssoc and InterfaceAssoc associations of the Pet Store EJBUML model.

9.5 The process to construct a MaramaCRelation model

The process to construct a CRelation model such as the one illustrated in Figure 9.2(b) is as follows:

Step 1: create StructureMappings

Find conceptually similar elements from source and target meta-models and arrange them into a set of StructureMappings.

Step 2: create SemanticAssociation

Create SemanticAssociations between pairs of StructureMappings; the value of the "associationMapping" property of each SemanticAssociation is derived automatically from the associated StructureMappings.

Step 3: create SelectionRefinements

For each StructureMapping, refine the selection constraints in the SelectionRefinement if necessary. Connect the SelectionRefinements to their parent StructureMappings.

Step 4: create StructureRefinement

For each StructureMapping, if there are semantic constraints (from both the source and target meta-models) constraining this StructureMapping and another StructureMapping, it may be necessary to introduce StructureRefinements to specify the second-order selection constraints. Connect the StructureRefinement to both of its parent StructureMappings.

9.6 Summary

The CRelation model is an interconnection model for MI&T. It uses StructureMappings to capture the main concerns of MI&T; SelectionRefinements to refine selection constraints; and StructureRefinement to define second-order selection constraints on StructureMappings. The CRelation model uses SemanticAssociation to explicitly represent the information that is typically hidden and implicit in traditional model integration and transformation technologies. The SemanticAssociation allows users to relate isolated StructureMappings in the context of a broader model, and help users to reason about the design of an intended MI&T. The SemanticAssociation also maintains translatable semantic constraints. The CRelation model improves the traditional model integration and transformation to a higher analysis and design level, which allows modellers to review and organize the existing isolated issues of MI&T, including semantics maintenance and behaviour synchronization. Chapter 10 will show that the CRelation model also helps to maintain flexible traceability mechanism.

Chapter 10 - The MaramaCRelation Tool

The MaramaCrelation tool supports CRelation modelling. The MaramaCrelation tool provides a modelling environment for the CRelation model; supports efficient communication between the CRelation model and the involved source and target domain-specific meta-models; enables automatic derivation of property values; and generates search conditions to maintain traceability during MI&T. This chapter introduces in detail the main functions of the MaramaCRelation tool and discusses the main design and implementation issues of the tool.

10.1 Overview of the MaramaCRelation tool

The MaramaCRelation tool is a built-in Eclipse environment by using the Eclipse tool-built facilities. Figure 10.1 shows the main interface of the tool. The MaramaCRelation tool uses the Eclipse package explorer to organize model projects including CRelation projects, domain-specific meta-models, and domain-specific models; users can open projects, create new projects, and import and export projects in the package explorer (1). The MaramaCRelation tool palette provides access to model editing features for constructing CRelation models (2). Users build a CRelation model in a CRelation model editor (3). The MaramaCRelation tool customizes and uses Eclipse views to support CRelation modelling, including property view and error view (4). The MaramaCRelation tool support effective communication between the CRelation model and its source and target meta-models (5).

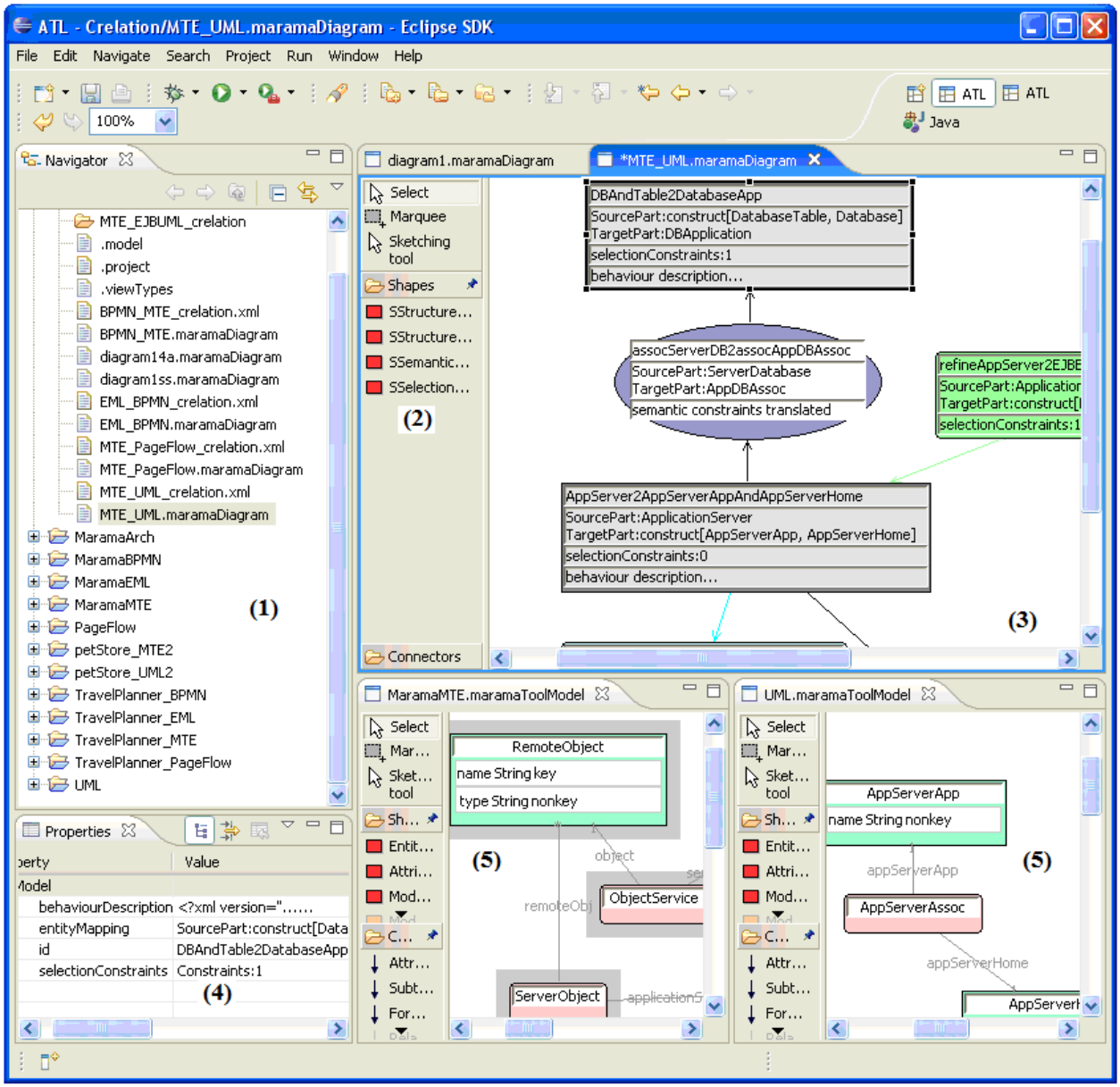


Figure 10.1. The main interface of the MaramaCRelation tool

10.2 The MaramaCRelation tool supporting constructing a StructureMapping

StructureMappings are the main model elements of the CRelation model. The MaramaCRelation tool allows users to efficiently set up values for the properties of a StructureMapping. The “id” property has

a simple textual value that is easy to set up. How to set up the other three properties is explained in this section.

10.2.1 Setting up the value for the entityMapping property

10.2.1.1 *Bringing up the source and target meta-models*

In order to set up the *entityMapping* value of a StructureMapping, the MaramaCRelation tool needs to retrieve data from both source and target meta-models. Figure 10.2 shows how the MaramaCRelation tool loads the source meta-model. In the CRelation editor (1) users use a popup menu to "Load Source MetaModel" (2). Users select the interested source meta-model from a list of available domain-specific meta-models (3), and the selected source meta-model is then loaded and displayed in a new MaramaCRelation editor (4). Users then load the target meta-model in the same way ((5) (6) in Figure 10.3), and the selected target meta-model is loaded and displayed in a new MaramaCRelation editor (7) beside the loaded source meta-model. Once both meta-models are loaded, users can choose to hide them or leave them open. If the two meta-models are left open, the two loaded meta-models (source, target) and the intended CRelation model are visually coordinated. Whenever a StructureMapping is highlighted in the intended CRelation model, the related source and target meta-modelling elements will also be highlighted in both the corresponding source and target meta-models.

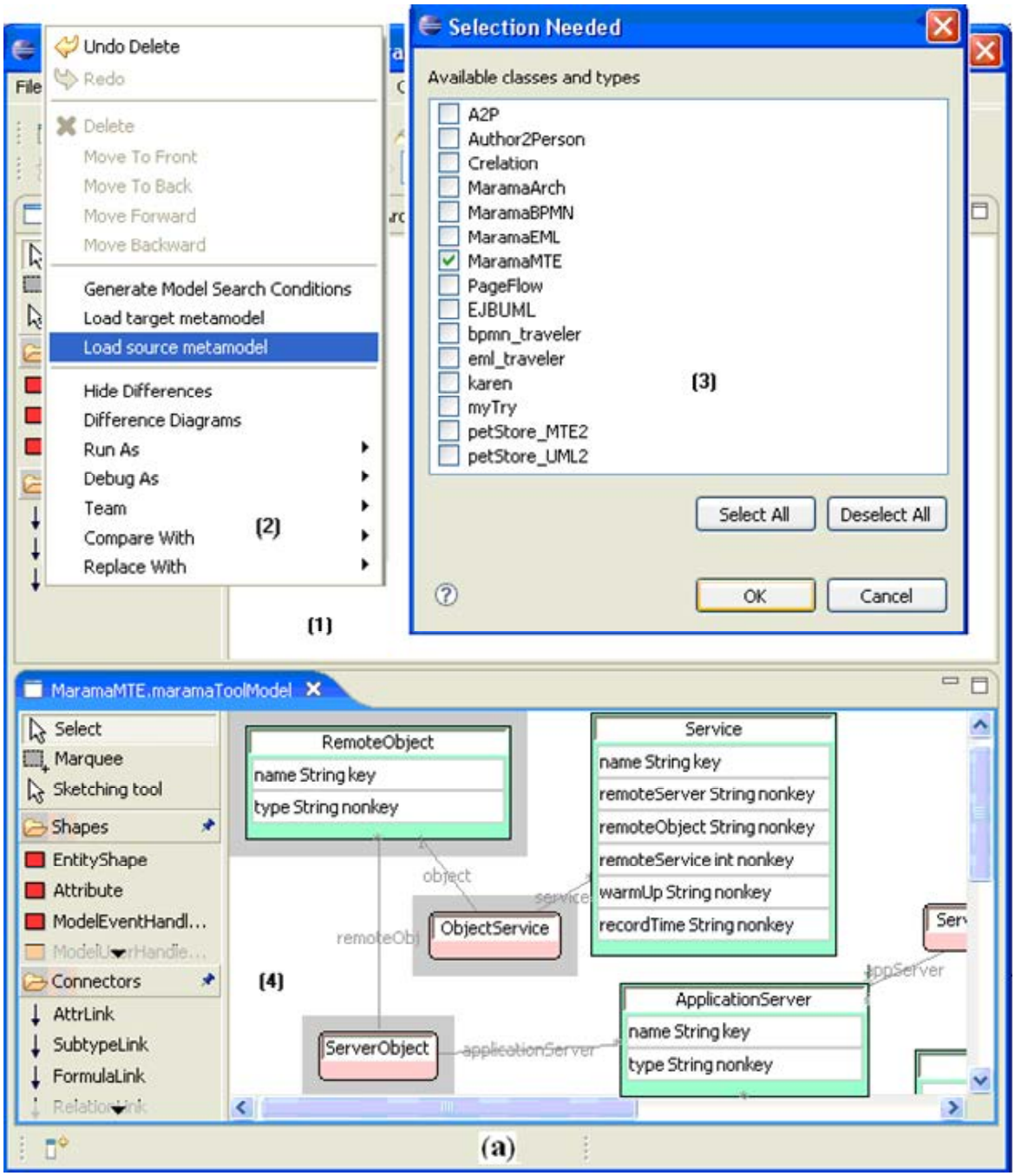


Figure 10.2. Loading the source meta-model

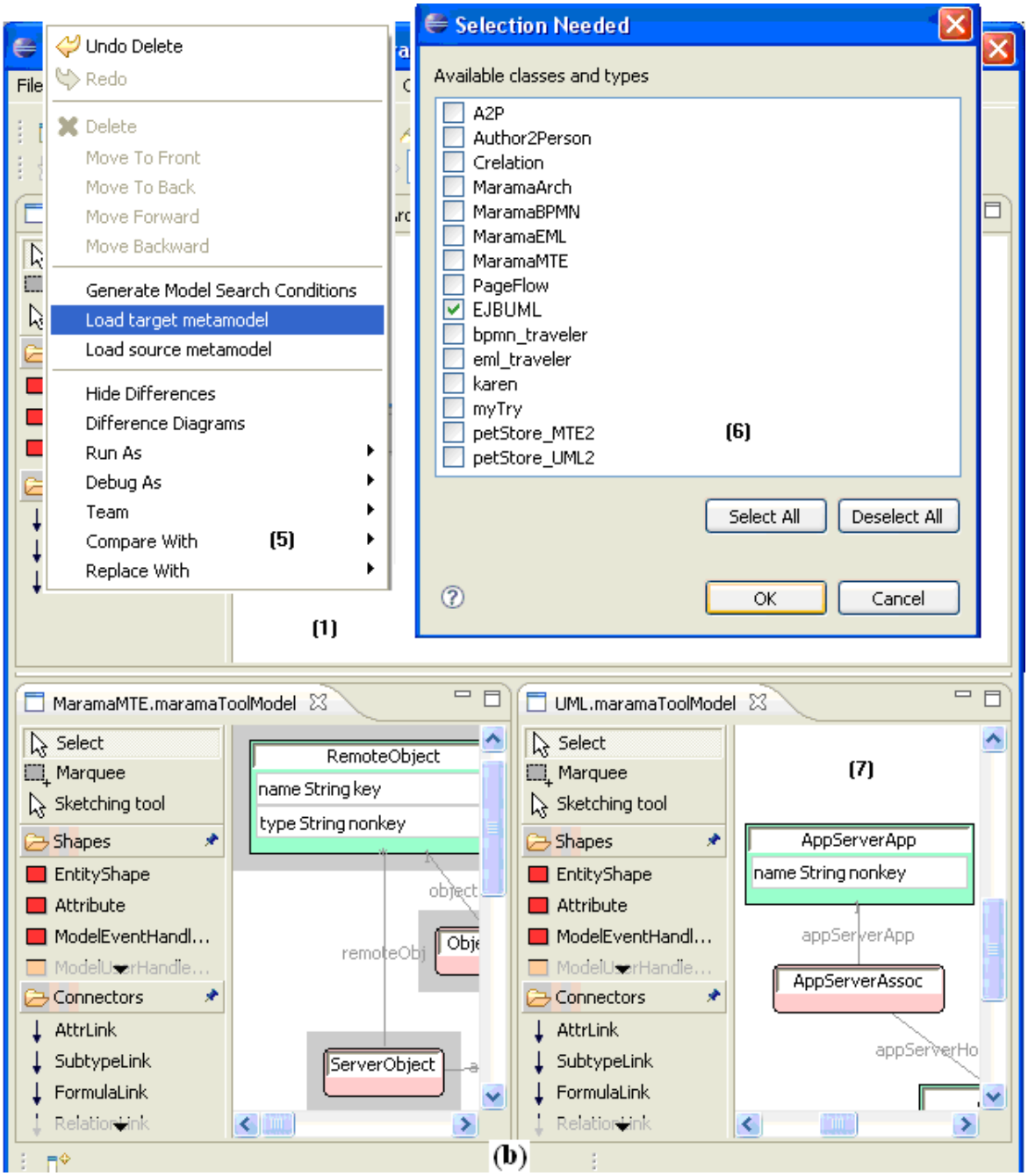


Figure 10.3. Loading the target meta-model

10.2.1.2 *Retrieving available meta-elements and meta-constructs*

StructureMappings can consist of not only single meta-model elements but also constructs of the meta-model elements (meta-constructs). When setting up the *entityMapping* property of a StructureMapping,

the MaramaCRelation tool needs to calculate a complete set of constructs (the complete set of sub-models) of the source and target meta-models, and list them together with the single meta-elements, so users can choose a pair of interested source and target meta-elements or meta-constructs to set up the value of *entityMapping* of the StructureMapping.

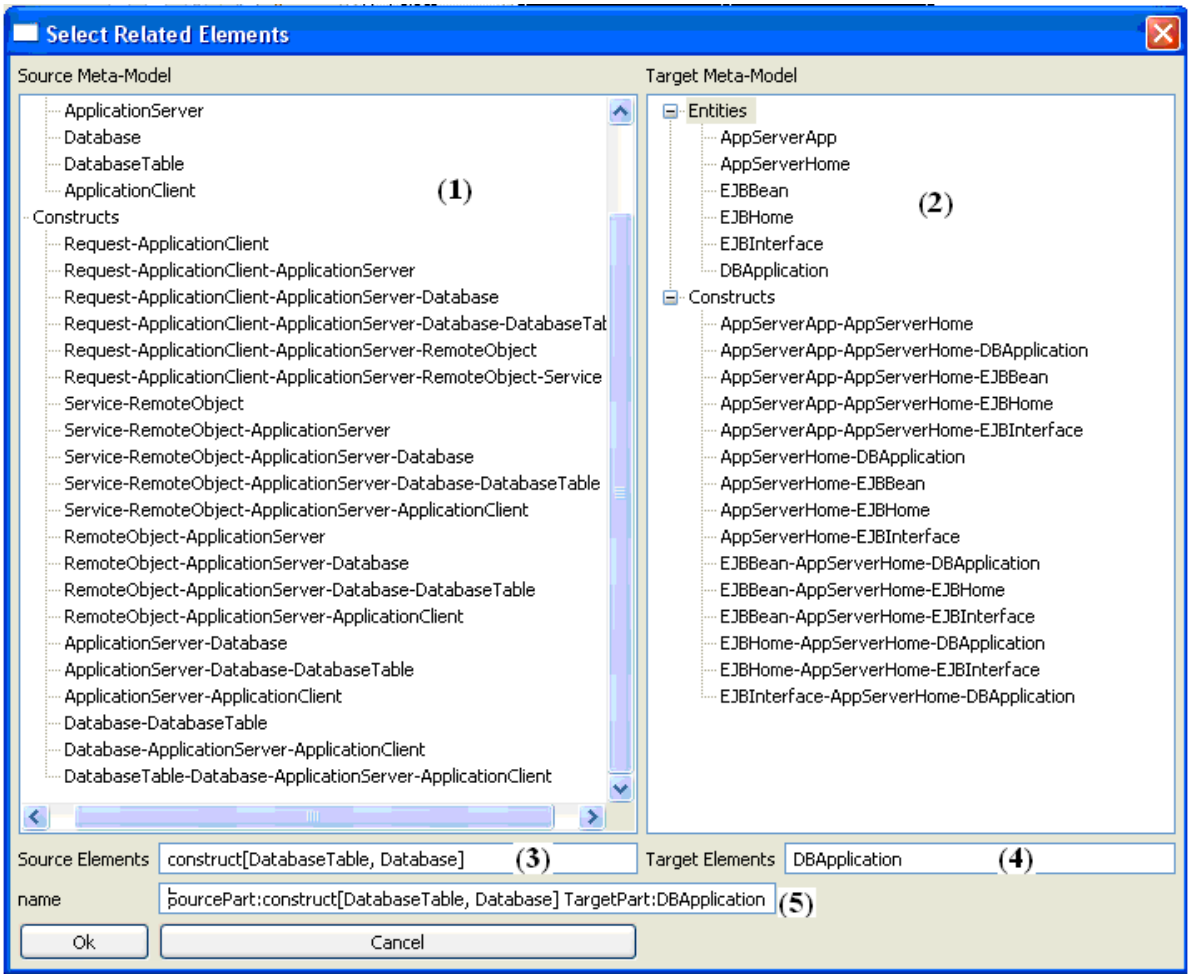


Figure 10.4. Calculating and listing available meta-model elements and constructs of the source and target meta-models

When users click the *entityMapping* property entry in the MaramaCRelation tool property view (an Eclipse property view) to edit the value, the *entityMapping* property sheet is opened as is illustrated in Figure 10.4. The property editor contains two complete lists of the source and target meta-model elements and constructs (1)(2). The user wishes to interconnect the construct [DatabaseTable, Database]

(source meta-construct) from the MaramaMTE meta-model (3) with the meta-element DBApplication (target meta-element) from the EJBUML meta-model (4). The *entityMapping* of the StructureMapping is then set up as "SourcePart:construct [DatabaseTable, Database] TargetPart:DBApplication" (5).

10.2.2 Setting up the value for the selectionConstraints

10.2.2.1 *Defining selection constraints using OCL and java*

The CRelation model defines selection constraints of a StructureMapping, SelectionRefinement, and StructureRefinement by using a combination of OCL and tool-API-independent java code. OCL is used to query property values of the meta-elements, and protects users from having to deal with the MaramaCRelation tool API. The tool-API-independent java code supports complicated operations on the retrieved model data.

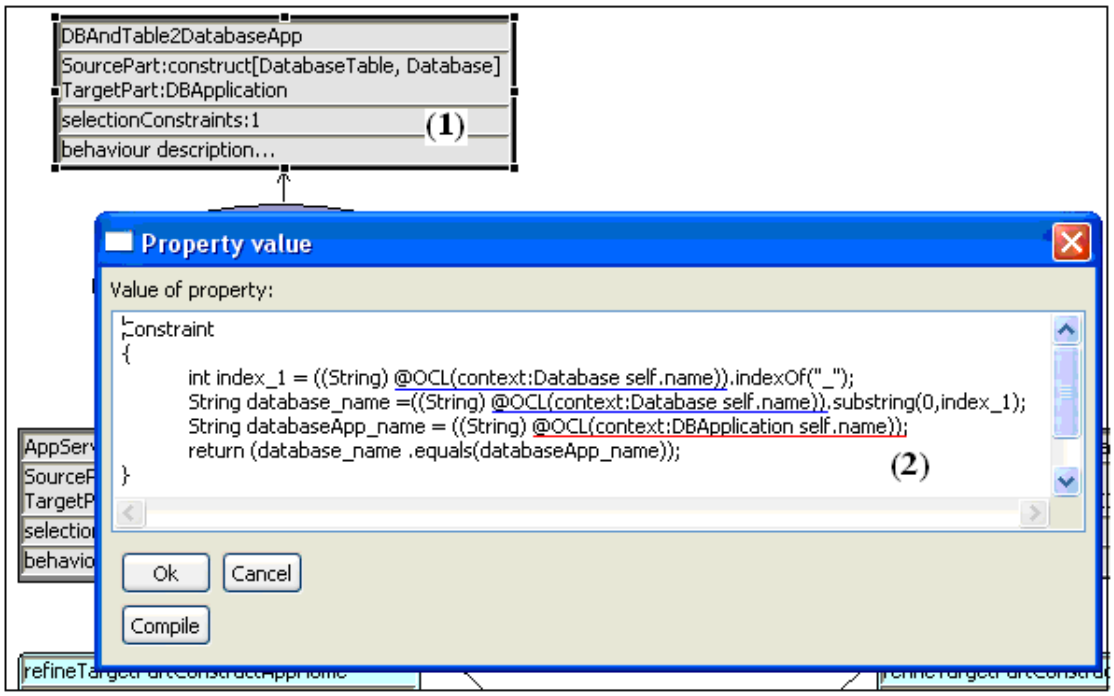


Figure 10.5. Using OCL + java to define StructureMapping selectionConstraints

Figure 10.5 shows the selection constraint of StructureMapping DBAndTable2DatabaseApp (1). Three OCL queries are used in the selection constraint (2). The two queries of "@OCL(context:Database self.name)" (underlined in blue) retrieve the *name* value of the Database in the source meta-model. The one query of "@OCL(context:DBApplication self.name)" (underlined in red) retrieves the *name* value

of DBApplication in the target meta-model. If the OCL queries are viewed as special java expressions, the contents of the selection constraint follow java syntax and can be viewed as a block of functional java code. The MaramaCRelation tool can compile the selection constraints (by using the "compile" button in Figure 10.5, Figure 10.6, and Figure 10.7) to avoid syntax errors of OCL expressions and java code.

The same format applies to the selection constraints of SelectionRefinements and StructureRefinements. Figure 10.6 and Figure 10.7 show the selection constraint of SelectionRefinements refineTargetPartConstructAppHome and refineTargetPartConstructBeanInterHome respectively.

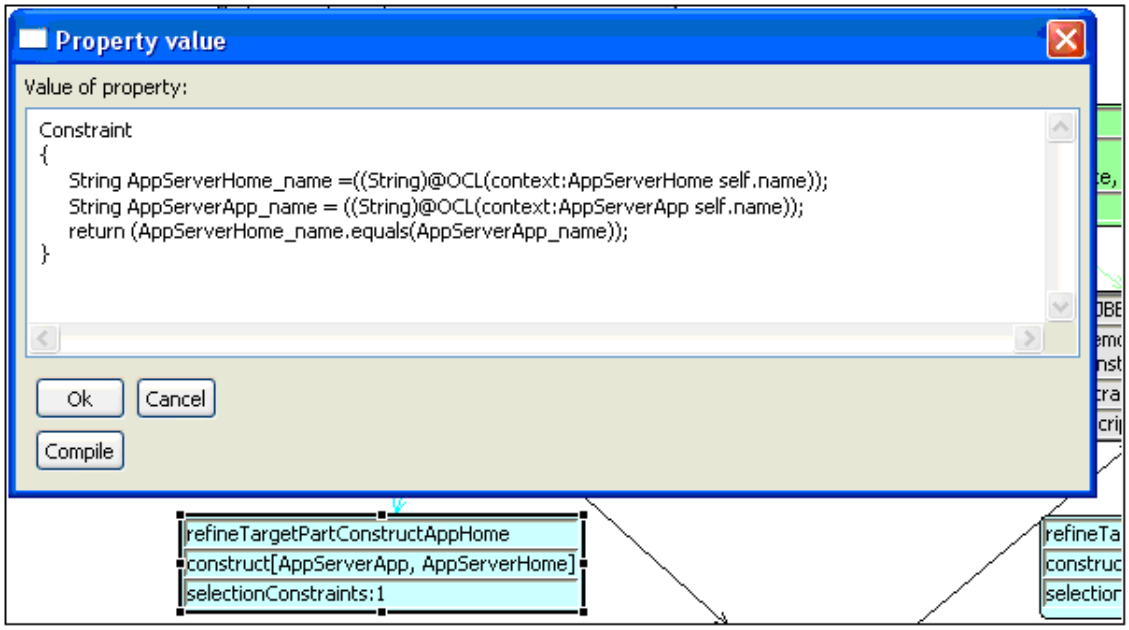


Figure 10.6. Using OCL + java to define selection constraints of a SelectionRefinement

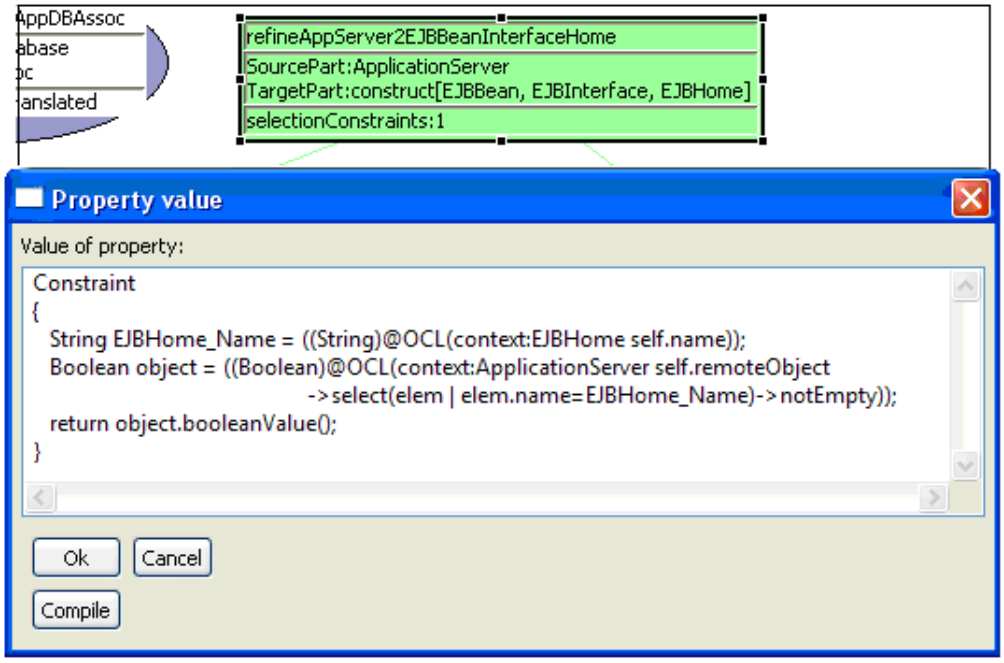


Figure 10.7. Using OCL + java to define selection constraints of StructureRefinement

10.2.2.2 *Defining selection constraints by using ATL*

The MaramaCRelation tool tries to leverage the popular third party technologies to describe the selection constraints; and ATL meets the requirements. At this stage, the MaramaCRelation tool uses ATL matched rules to define selection constraints that involve both source and target meta-elements, and uses ATL queries to define selection constraints that only involve a single model.

ATL matched rules can specify 1) for which kinds of source elements target elements must be generated, and 2) the way the generated target elements have to be initialized. The matched rules can be used by the MaramaCRelation tool to specify selection constraints across the source and target models in a CRelation model.

An ATL query transforms a model to primitive type value. More specifically, ATL queries are used to generate textual outputs (encoded into a string value) from a set of source models, or return a numerical or a Boolean value. ATL queries are used by the MaramaCRelation tool to specify single-model selection constraints in a CRelation model.

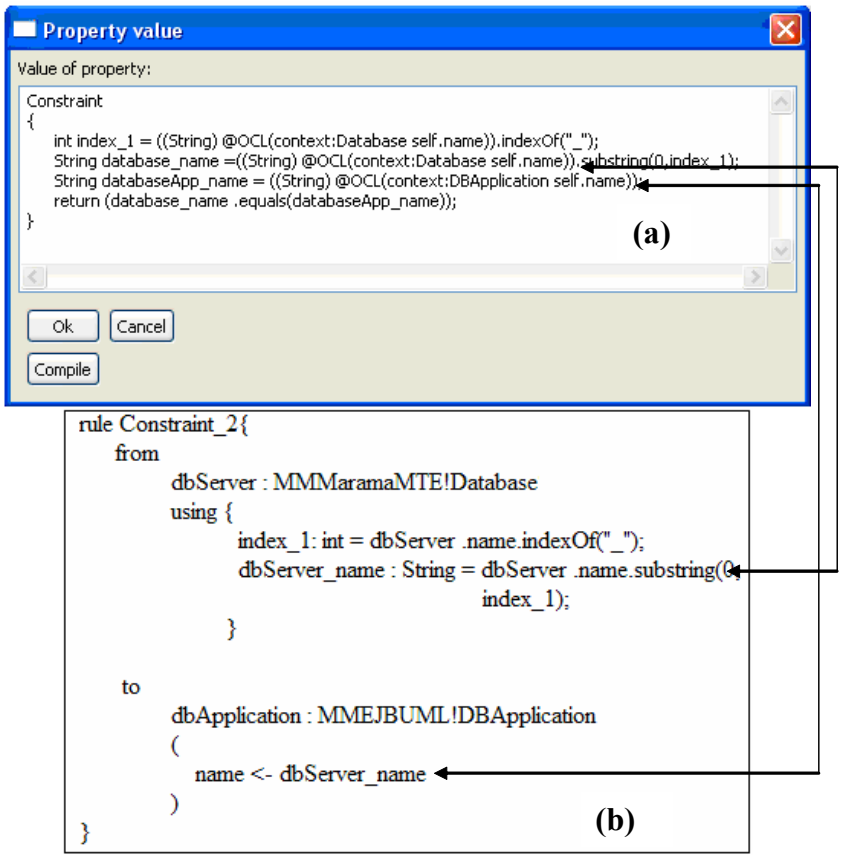


Figure 10.8. Using ATL to define selection constraints

Figure 10.8 illustrates how to define selection constraints by using ATL matched rules. Figure 10.8(a) illustrates a sample selection constraint (defined in java + OCL) of a StructureMapping with the constraint crossing the source and the target meta-models. The ATL matched rule (in Figure 10.8(b)) defines a valid selection constraint that achieves the same result as the constraint in Figure 10.8(a). The matched rule computes a string value "dbServer_name" and uses it to initialize target model element DBApplication (of EJBUML meta-model).

Figure 10.9 shows an example using ATL queries to define CRelation model selection constraints. The selection constraint defined in ATL query in Figure 10.9(b) achieves the same result as the constraint defined in Figure 10.9(a).

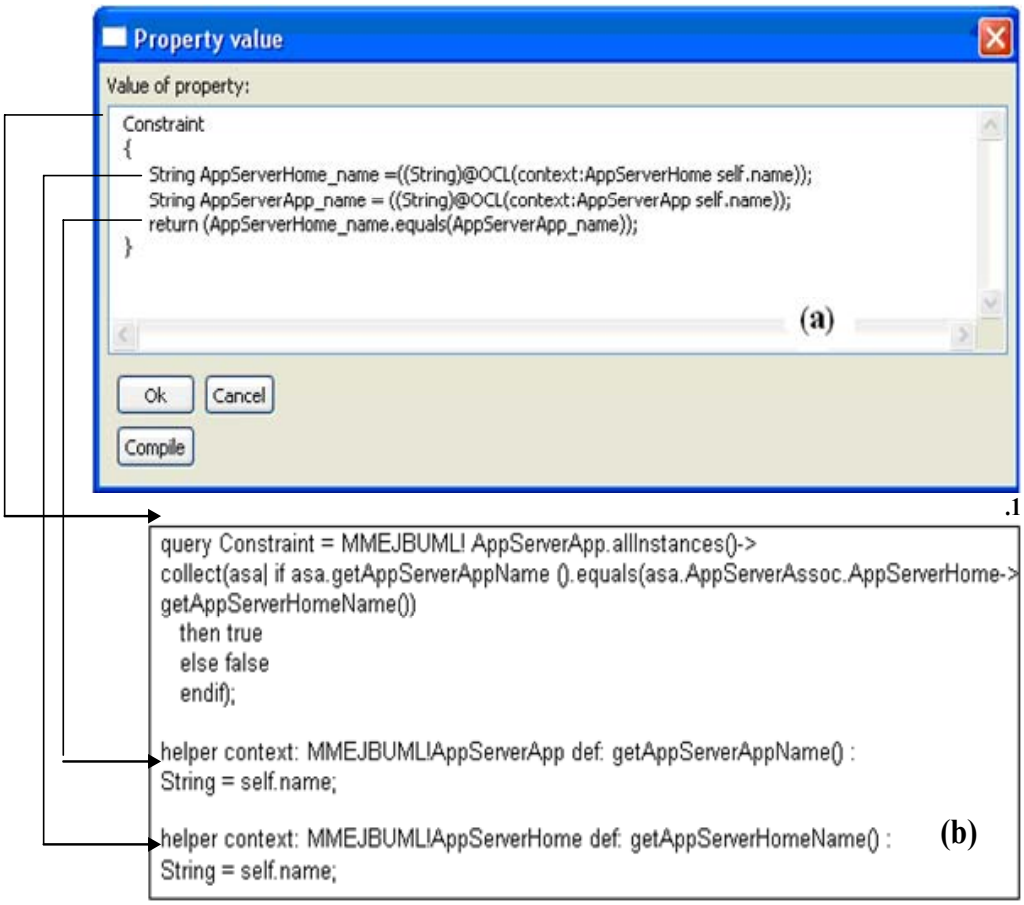


Figure 10.9. More samples of using ATL to define selection constraints

At this stage, the MaramaCRelation tool leverages ATL in defining selection constraints of the CRelation model. The MaramaCRelation tool is yet to support generating functional ATL transformation scripts. A CRelation model shows the potential to generate functional ATL model transformation script, which will be discussed in the future work.

10.2.3 Setting up value for the behaviourDescription of a StructureMapping

The *behaviourDescription* of a StructureMapping is a structured high-level interface for behaviour synchronization during MI&T. The MaramaCRelation tool generates the *behaviourDescription* for a StructureMapping through two steps: 1) rewriting the selection constraints; 2) generating the *behaviourDescription* according to the behaviour description schema (refer to section 9.4.1.4).

10.2.3.1 *Rewriting selection constraints*

```
void rewriteSelectionConstraint()
  {


    for (each variable assignation operation )
    {
      if (the right hand side of the assignation involves involvement
              variables or OCL queries)
      {
        tempInvolvementPart  = construct the involvement part for the
                                  right hand side information;

        if(the variable doesn't contain the tempInvolvementPart)
          {
            newVariable = tempInvolvementVariable +the existing variable;

            replace the existing old variable with the newVariable
                    all through the selection constraint;

          }

        }

      if(the assignation is influenced by involvement variables
                    or OCL queries)
      {
        tempInvolvementPart  = construct the involvement part for
                          the influential involvement variables or
                          OCL queries;

        if(the variable doesn't contain the tempInvolvementPart)
        {
            newVariable = tempInvolvementPart + the existing variable;

            replace the existing old variable with the newVariable
                    all through the selection constraint;
        }

      }
    }
  }
```

Figure 10.10. An empirical algorithm to rewrite an existing selection constraint

The model data (property values) involved in a selection constraint may or may not influence the constraint result. The MaramaCRelation tool uses an empirical algorithm to rewrite the selection constraints to find out what model data (model element property values) will truly influence the constraint results. Figure 10.10 illustrates the empirical algorithm used to rewrite an existing selection constraint. The ***involvement part*** and ***involvement variable*** are two important concepts in the algorithm. The *involvement part* is in the form of "iv@((model element name)_(property name))" where the model element name and property name are case insensitive. The involvement variable is in the form of "involvement part + a valid java variable**"**. Note: the involvement variable is not a valid java variable,

but a way to record the involvement part in a valid java variable. In a selection constraint, each OCL query has a corresponding involvement part. For example, involvement part for query "@OCL(context:Database self.name)" is "iv@(database_name)", and the involvement part for query "@OCL(context:DatabaseTable self.name)" is "iv@(databaseTable_name)". Valid involvement variables are like "iv@(database_name)index_1" and "iv@(databaseTable_name)index_2" with the involvement part "iv@(database_name)" and "iv@(databaseTable_name)" respectively. The algorithm in Figure 10.10 explains how to use the involvement variables to track the model information that is influential on the constraint result. The algorithm is explained through rewriting two sample selection constraints.

Figure 10.11 shows a sample selection constraint with four variable assignations involved. For the assignation on line 3, when executing the algorithm between lines 7 and 22 (refer to Figure 10.10), the "tempInvolvementPart" (refer to line 10, Figure 10.10) for the right hand side of the assignation is "iv@(database_name)". Because the variable "index_1" does not contain this "tempInvolvementPart" (refer to line 13, Figure 10.10), so the "newVariable" (refer to line 15, Figure 10.10) should be "iv@(database_name)index_1". Replace the existing variable "index_1" with the involvement variable "iv@(database_name)index_1" all through the selection constraint in the selection constraint (refer to line 17~18, Figure 10.10). This assignation is not influenced by other involvement variables and OCL queries, so the part of the algorithm between lines 24 and 39 (refer to Figure 10.10) is skipped.

```
1  Constraint
2  {
3    int index_1 = ((String)@OCL(context:Database
4                              self.name)).indexOf("_");
5
6    String database_name = ((String)@OCL(context:Database
7                                       self.name)).substring(0, index_1);
8
9    int index_2 = ((String)@OCL(context:DatabaseTable
10                             self.name)).indexOf("_");
11
12   String databaseTable_name = ((String)@OCL(context:DatabaseTable
13                                            self.name)).substring(0, index_2);
14
15   return (database_name.equals(databaseTable_name));
16 }
```

Figure 10.11. A valid selection constraint

For the assignation in line 6 in Figure 10.11, when executing the algorithm between lines 7 and 22 (refer to Figure 10.10), the "tempInvolvementPart" for the right hand side of the assignation is

“iv@(database_name)”. Because the variable “database_name” of the assignation does not contain this “tempInvolvementPart” (refer to line 13, Figure 10.10), so the “newVariable” (refer to line 15, Figure 10.10) should be “iv@(database_name)database_name”. Replace the existing variable “database_name” with the involvement variable “iv@(database_name)database_name” all through the selection constraint (refer to line 17~18, Figure 10.10). This assignation is not influenced by other involvement variables and OCL queries, so the part of the algorithm between lines 24 and 39(refer to Figure 10.10) is skipped.

For assignation in line 9, when executing the algorithm between lines 7 and 22 (refer to Figure 10.10), the “tempInvolvementPart” for the right hand side of the assignation is “iv@(databaseTable_name)”. Because the variable “index_2” does not contain this “tempInvolvementPart”, so the “newVariable” should be “iv@(databaseTable_name)index_2”. Replace the existing variable “index_2” with the involvement variable “iv@(databaseTable_name)index_2” all through the relationship constraint. This assignation is not influenced by other involvement variables and OCL queries, so the part of the algorithm between lines 24 and 39(refer to Figure 10.10) is skipped.

For assignation in line 12, when executing the algorithm between lines 7 and 22 (refer to Figure 10.10), the “tempInvolvementPart” for the right hand side of the assignation is “iv@(databaseTable_name)”. Because the variable “databaseTable_name” does not contain this “tempInvolvementPart”, so the “newVariable” should be “iv@(databaseTable_name)databaseTable_name”. Replace the existing variable “databaseTable_name” with the involvement variable “iv@(database_name)database_name” all through the selection constraint. This assignation is not influenced by other involvement variables and OCL, so the part of the algorithm between lines 24 and 39(refer to Figure 10.10) is skipped. The rewritten selection constraint is illustrated in Figure 10.12. The involvement parts of all the involvement variables in the return result variable represent the model entities and their properties directly contributing to the selection constraint result. In this case, it is the *name* properties of Database and DatabaseTable that are influential on the constraint result.

```
1  Constraint
2  {
3    int iv@(database_name)index_1 =
4          ((String)@OCL(context:Database self.name)).indexOf("_");
5
6    String iv@(database_name)database_name =
7          ((String)@OCL(context:Database self.name)).substring(0,
8          iv@(database_name)index_1);
9
10   int iv@(databaseTable_name)index_2 =
11         ((String)@OCL(context:DatabaseTable
12         self.name)).indexOf("_");
13
14   String iv@(databaseTable_name)databaseTable_name =
15         ((String)@OCL(context:DatabaseTable
16         self.name)).substring(0,iv@(databaseTable_name)index_2);
17
18   return (iv@(database_name)database_name.equals(
19           iv@(databaseTable_name)databaseTable_name));
20 }
```

Figure 10.12. The rewritten selection constraint

Figure 10.13 illustrates another sample selection constraint. Rewriting this constraint is similar to the rewriting of the selection constraint in Figure 10.11. The variable assignation in line 19 needs more attention. The right hand side of the assignation involves a variable "database_name". This variable should have already been replaced by an involvement variable "iv@(database_name)database_name" due to the rewriting of the assignation in line 3. So, for assignation in line 19, when executing the algorithm between lines 7 and 22 (refer to Figure 10.10), the "tempInvolvementPart" (refer to line 10, Figure 10.10) for the right hand side of the assignation is "iv@(database_name)". The "result" variable (line 19, Figure 10.13) should be rewritten as "iv@(database_name)result". Moreover, the variable "iv@(database_name)result" is also influenced by the *if statement* (line 17) that has the involvement variable "iv@(databaseTable_name)databaseTable_name". When executing the rewriting algorithm between lines 24 and 39 (refer to Figure 10.10), the "tempInvolvementPart" (brought by the variable in the *if statement* in line 17) is "iv@(databaseTable_name)", which is not contained by the existing involvement variable "iv@(database_time)result". A new variable "iv@(databaseTable_name")iv@(database_name)result" needs to be constructed to replace the existing "iv@(database_name)result". The rewritten selection constraint is illustrated in Figure 10.14. In this case, it is the *name* properties of Database and DatabaseTable that are influential on the constraint result.

```
1Constraint
2{
3  String database_name=((String)@OCL(context:Database self.name));
4
5  int index_1=((String)@OCL(context:Database self.name)).indexOf("_");
6
7  database_name=database_name.substring(0, index_1);
8
9  String databaseTable_name=((String)@OCL(context:DatabaseTable self.name));
10
11 int index_2=databaseTable_name.indexOf("_");
12
13 databaseTable_name=databaseTable_name.substring(0, index_2);
14
15 boolean result=false;
16
17 if(databaseTable_name.length()<15)
18
19    result=database_name.equals("helloworld");
20
21 return result;                                                    (c)
22}
```

Figure 10.13. A valid selection constraint

```
1Constraint
2{
3  String iv@(database_name)database_name =
4      ((String)@OCL(context:Database self.name));
5
6  int iv@(database_name)index_1 =
7      ((String)@OCL(context:Database self.name)).indexOf("_");
8
9  iv@(database_name)database_name =
10     iv@(database_name)database_name.substring(0,iv@(database_name)index_1);
11
12 String iv@(databaseTable_name)databaseTable_name =
13     ((String)@OCL(context:DatabaseTable self.name));
14
15 int iv@(databaseTable_name)index_2 =
16     iv@(databaseTable_name)databaseTable_name.indexOf("_");
17
18 iv@(databaseTable_name)databaseTable_name =
19     iv@(databaseTable_name)databaseTable_name.substring(0,
20     iv@(databaseTable_name)index_2);
21
22 boolean iv@(databaseTable_name)iv@(database_name)result =false;
23
24 if(iv@(databaseTable_name)databaseTable_name.length()<15)
25
26   iv@(databaseTable_name)iv@(database_name)result =
27       iv@(database_name)database_name.equals("helloworld");
28                                                                  (d)
29 return iv@(databaseTable_name)iv@(database_name)result;
30}
```

Figure 10.14. The rewritten selection constraint

10.2.3.2 *Generating the behaviourDescription*

Figure 10.15, Figure 10.16, and Figure 10.17 use pseudo java code to encode an empirical algorithm of generating the *behaviourDescription* of a StructureMapping. Figure 10.15 illustrates the main method of generating the *behaviourDescription*. The source model events are generated between lines 4 and 16, and the target model events are generated between lines 19 and 31. When generating source model events of the *behaviourDescription*, the MaramaCRelation tool rewrites the selection constraints to retrieve a list of influential source model entities and their properties (line 6 in Figure 10.15). For each influential meta-model entity and its property, a "changed" event is generated (line 9 in Figure 10.15). After the generation of the "changed" events, the algorithm is ready to generate the "removed" events. The MaramaCRelation tool retrieves all source model entities involved in the StructureMapping (line 12 in Figure 10.15). For each retrieved entity, a "removed" event is generated (line 15 in Figure 10.15). The process is similar when generating target model events of the *behaviourDescription* (lines between 19 and 31 in Figure 10.15).

```
1  main ()
2  {
3    // generating source model events
4    for (each selection constraint)
5    {
6      retrieve_Influential_SourceModel_Entities_And_Properties();
7      for (each retrieved model element and its property)
8
9        generateChangedEvent(modelElementProperty,
10                            selection constraint);
11
12     retrieve_SourceModel_EntitiesOfStructureMapping);
13
14     for each retrieved model element
15       generateRemovedEvent(modelElement);
16   }
17
18   // generating target model events
19   for (each selection constraint)
20   {
21     retrieve_Influential_TargetModel_Entities_And_Properties();
22     for (each retrieved model element and its property)
23
24     generateChangedEvent(modelElementProperty,
25                          selection constraint);
26
27     retrieve_TargetModel_EntitiesOfStructureMapping();
28
29     for each retrieved model element
30         generateRemovedEvent(modelElement);
31   }
32 }
```

Figure 10.15. The algorithm of generating *behaviourDescription* for a StructureMapping

Figure 10.16) shows the algorithm to generate a "changed" event. In a *behaviourDescription*, a "changed" event is described by a group of parameters, *including EventType, EventOriginators, EventOriginators_params, EventConsumers_inSourceModel, EventConsumers_inSourceModel_params, EventConsumers_inTargetModel, EventConsumers_inTargetModel_params, and IntendedResult*. The meaning of those parameters has been explained in section 9.4.1.

```
generateChangedEvent(String modelElementProperty,
                String selection constraint)
{

  EventOriginators = modelElementProperty;

  EventOriginators_Param = assign sensible string parameters
                       to EventOriginators;

  EventConsumers_inSourceModel = other source model element
        properties  influential on the constraint result;

  EventConsumers_inSourceModel_params = assign string parameters
                  to EventConsumters_inSourceModel;

  EventConsumers_inTargetModel = target model element properties
                influential on the constraint result;

  EventConsumers_inTargetModel_params  = assign string parameters
                  to EventConsumers_inTargetModel;

  IntendedResult  = manually-construct information based on
          the logic of the selection constraint;
}
```

Figure 10.16. The algorithm of generating "changed" events for the *behaviourDescription*

Figure 10.17 shows the algorithm to generate a "removed" event. In a *behaviourDescription*, a "removed" event is described by a group of parameters, including *EventOriginators, EventConsumers_inSourceModel, EventConsumers_inTargetModel, and IntendedResult*. The meaning of those parameters has been explained in section 9.4.1.

```
46  generateRemovedEvent(ModelElement modelElement)
47  {
48    if(modelElement belongs to a construct)
49    {
50      EventOriginators = the construct the modelElement involves;
51
52      EventConsumers_inSourceModel = other source model involved
53                              in the construct;
54    }
55
56    else
57    {
58      EventOriginators = modelElement
59      EventOriginators_inSourceModel = null;
60    }
61
62    EventConsumers_inTargetModel = target model elements involved
63                            in the InterEntity;
64
65    IntendedResult   = InterconnectionRelationship.removed();
66  }
```

Figure 10.17. The algorithm of generating "removed" events for the *behaviourDescription*

Figure 10.18 and Figure 10.19 show an example of how to generate "changed" events of the *behaviourDescription* of a StructureMapping. In order to generate the source model "changed" events, the selection constraint of StructureMapping DBAndTable2DatabaseApp (in Figure 10.18) is rewritten (in Figure 10.19) to retrieve influential source meta-model entities and their properties. The MaramaCRelation tool retrieves source meta-model influential element *Database* and target meta-model influential element *DBApplication* whose *name* property change will influence the established Interconnection Relationship. Referring to the algorithm in Figure 10.15 between lines 7 and 10, the change of the *name* property of Database element triggers a "changed" event (event 1, line 5, in Figure 10.20).

The StructureMapping DBAndTable2DatabaseApp (Figure 10.18) consists of "SourcePart:construct[Database, DatabaseTable] TargetPart:DBApplication". According the *behaviourDescription* generation algorithm (refer to line 12 in Figure 10.15), the removal of each of the involved source meta-model element will influence the Interconnection Relationship. The Database element triggers one "removed" event (event 2, line 22, in Figure 10.20), and the DatabaseTable element triggers one "removed" event (event 3, line 31, in Figure 10.11). The "changed" event and the two "removed" events consist of the source meta-model events of the *behaviourDescription* of the StructureMapping.

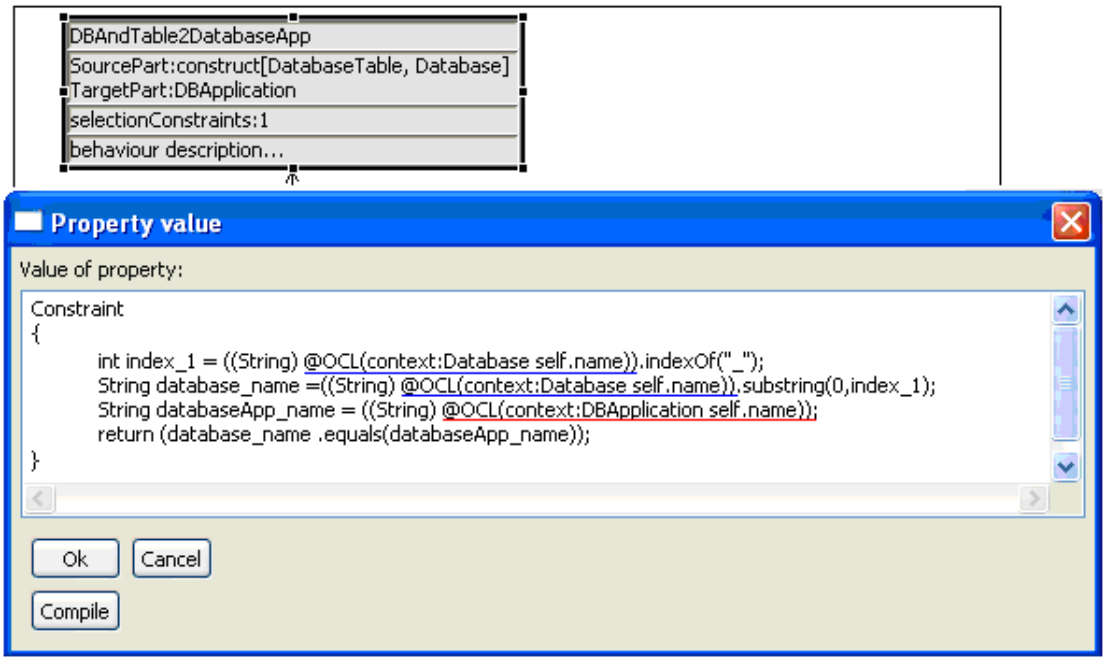


Figure 10.18. A sample StructureMapping and its selection constraint

```
Constraint
{
  int iv@(database_name)index_1 =
        ((String)@OCL(context:Database self.name)).indexOf("_");

  String iv@(database_name)database_name =
        ((String)@OCL(context:Database self.name)).substring(0,
        iv@(database_name)index_1);

  String iv@(DBApplication_name)databaseApp_name =
        ((String)@OCL(context:DBApplication self.name));

  return (iv@(database_name)database_name.equals(
          iv@(DBApplication_name)databaseApp_name));
}
```

Figure 10.19. A rewritten selection constraint

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2   <EventDescription>
3    <SourceModelEvent>
4     <Events>
5      <Event id="1">
6       <EventType>changed </EventType>
7       <EventOriginators> Database.name </EventOriginators>
8       <EventOriginators_params> database_name </EventOriginators_params>
9       <EventConsumers_inSourceModel>null</EventConsumers_inSourceModel>
10      <EventConsumers_inSourceModel_params>null</EventConsumers_inSourceModel_params>
11      <EventConsumers_inTargetModel>DBApplication.name</EventConsumers_inTargetModel>
12      <EventConsumers_inTargetModel_params>dbApplication_name</EventConsumers_inTargetModel_params>
13      <IntendedResult>
14       {
15        int index_1 = ((String)database_name).indexOf("_");
16        String database_name =((String)database_name).substring(0,index_1);
17        String dbApplication_name= ((String)database_name);
18        return dbApplication_name;
19       }
20      </IntendedResult>
21     </Event>
22     <Event id="2">
23      <EventType> removed </SourceEventType>
24      <EventOriginators> Database</EventOriginators>
25      <EventConsumers_inSourceModel>DatabaseTable</EventConsumers_inSourceModel>
26      <EventConsumers_inTargetModel> DBApplication </EventConsumers_inTargetModel>
27      <IntendedResult>
28      {InterconnectionRelationship.removed();}
29      </IntendedResult>
30    </Event>
31     <Event id="3">
32      <EventType> removed </EventType>
33      <EventOriginators> DatabaseTable</EventOriginators>
34      <EventConsumers_inSourceModel>Database</EventConsumers_inSourceModel>
35      <EventConsumers_inTargetModel> DBApplication </EventConsumers_inTargetModel>
36      <IntendedResult>
37      {InterconnectionRelationship.removed();}
38      </IntendedResult>
39     </Event>
40    </Events>
```

Figure 10.20. Generating source model events for the *behaviourDescription*

Figure 10.21 illustrates the target model events triggered by the StructureMapping DBAndTable2DatabaseApp of Figure 10.18. The target meta-model element involved in the StructureMapping is *DBApplication*. In Figure 10.21, event 4 (line 3) records the "changed" event triggered by the change of the *name* property of *DBApplication*. The *DBApplication* element also triggers one "removed" event (events 5, line 21, in Figure 10.21).

```
<TargetModelEvent>
 <Events>
  <Event id="4">
   <EventType>changed </EventType>
   <EventOriginators> DBApplication.name </EventOriginators>
   <EventOriginators_params>dbApplication_name </EventOriginators_params>
   <EventConsumers_inSourceModel>Database.name</EventConsumers_inSourceModel>
   <EventConsumers_inSourceModel_params>database_name</EventConsumers_inSourceModel>
   <EventConsumers_inTargetModel>null</EventConsumers_inTargetModel>
   <EventConsumers_inTargetModel_params>null</EventConsumers_inTargetModel_params>
   <IntendedResult>
    {
     int index_1 = ((String)database_name).indexOf("_");
     String database_name =((String)database_name).substring(0,index_1);
     String database_suffix =((String)database_name).substring(index_1);
     database_name = dbApplication_name + database_suffix;
     return database_name;
    }
   </IntendedResult>
  </Event>
  <Event id="5">
   <SourceEventType>removed </SourceEventType>
   <EventSource>DBApplication</EventSource>
   <EventTargets_inSourceModel> null</EventTargets_inSourceModel>
   <EventTargets_inTargetModel>construct[DatabaseTable,Database]</EventTargets_inTargetModel>
   <IntendedResult>
    {
     InterconnectionRelationship.removed();
    }
   </IntendedResult>
  </Event>
<TargetModelEvent>
```

Figure 10.21. Generating target model events of the behaviourDescription

10.2.4 Brief summary

To summarize, StructureMappings are the main model elements in the CRelation model. MaramaCRelation tool supports all the requirements for establishing a StructureMapping. The MaramaCRelation tool provides easy access to the involved source and target meta-models; supports efficient retrieving of model information from the both meta-models; allows tool-API independent high-level representation of selection constraints, leverages ATL to represent relationship constraints, and partially automatically generates *behaviourDescriptions*.

10.3 The MaramaCRelation tool supporting constructing a SemanticAssociation

SemanticAssociations associate isolated StructureMappings and allow them to communicate with each other. The MaramaCRelation tool allows users to efficiently establish values for the three properties of a SemanticAssociation including *id*, *associationMapping*, and *semanticTranslation*. Setting up the *id*

property is easy, as it has a simple textual value. Establishing the other two properties is explained in this section.

10.3.1 Setting up the value for the *associationMapping* property

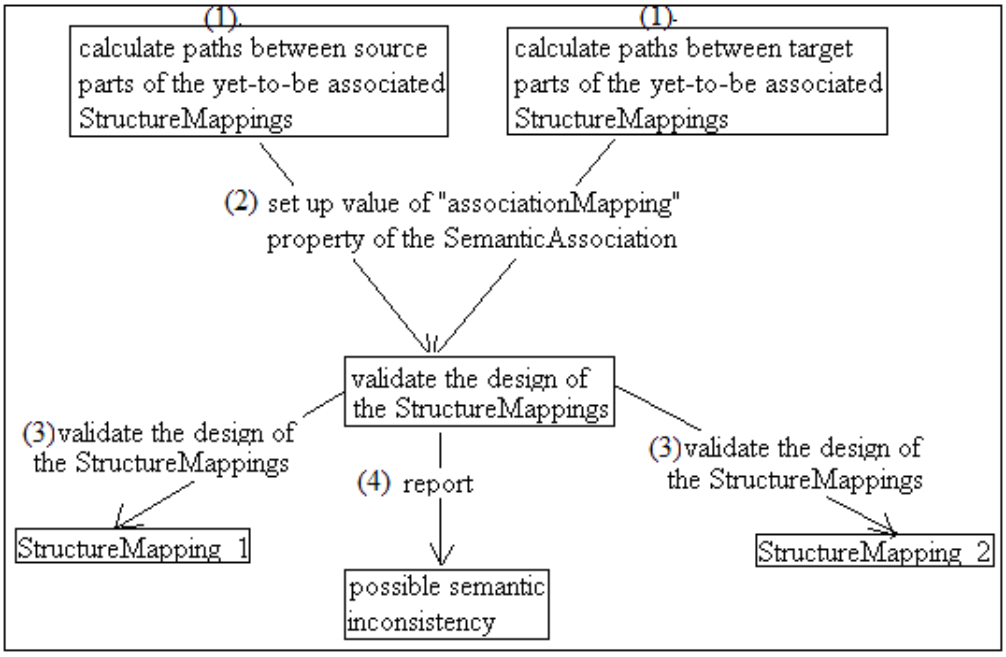


Figure 10.22. How to set up the *associationMapping* value of a SemanticAssociation

Figure 10.22 illustrates the main activities involved in constructing a SemanticAssociation. The MaramaCrelation tool calculates the path(s) between the source parts of the yet-to-be-associated StructureMappings in the source meta-model, as well as the path(s) between the target parts of the StructureMappings in the target meta-model (1). The users then set up the value of the *associationMapping* property by matching a source meta-model path with a target meta-model path (2). The *associationMapping* value can be used to validate the design of the parent StructureMappings (3). The source and target parts of the SemanticAssociation must share similar semantics conceptually; otherwise, it may mean: 1). the parent StructureMappings may not be well designed; or 2). the source and target models can not be interconnected without causing semantic inconsistencies. In the first situation, users need to redesign the parent StructureMappings until the SemanticAssociation captures the sensible semantics shared by the source and target meta-models (3). In the second situation,

SemanticAssociations can explicitly record the unavoidable semantic inconsistencies for the intended MI&T (4).

Section 9.4.5.2.1 introduces the definitions of the associationMapping value for a SemanticAssociation. The MaramaCRelation tool uses that definition to calculate the available source and target paths, and helps users to establish the value of the *associationMapping* property.

Figure 10.23 shows a tentative design of StructureMappings when interconnecting the MaramaMTE meta-model with the EJBUML meta-model. The MaramaCRelation tool derives the available source and target paths for the users to establish the value for the *associationMapping* property of the highlighted SemanticAssociation. Of all the three available target meta-model paths, none of them really matches the source path ServerObject. In the EJBUML meta-model, a correct match for the source path *ServerObject* should be the *construct[BeanAssoc, InterfaceAssoc, HomeAssoc]*. In Figure 10.23, the mismatch between the source and target paths prompts users to redesign the MaramaCRelation model. If the mismatch is unavoidable during the interconnection, the SemanticAssociation records the inconsistency.

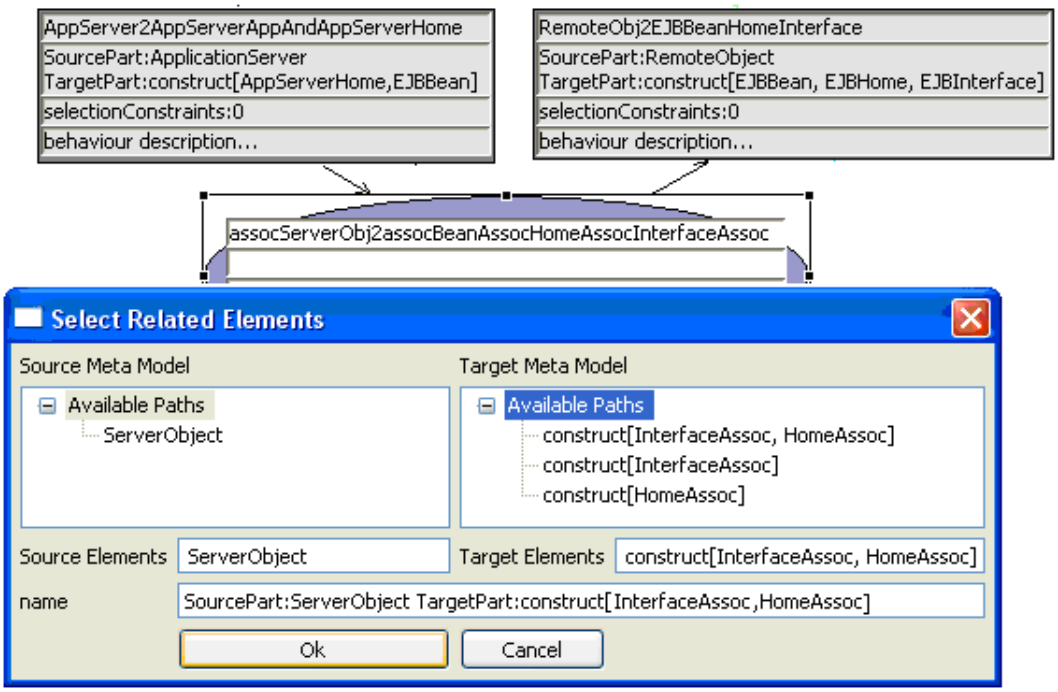


Figure 10.23. An *associationMapping* with inconsistency

Figure 10.24 shows another tentative design of StructureMappings when interconnecting MaramaMTE with EJBUML. Of all the four available target meta-model paths, the construct[BeanAssoc, InterfaceAssoc, HomeAssoc] correctly matches the source path ServerObject.

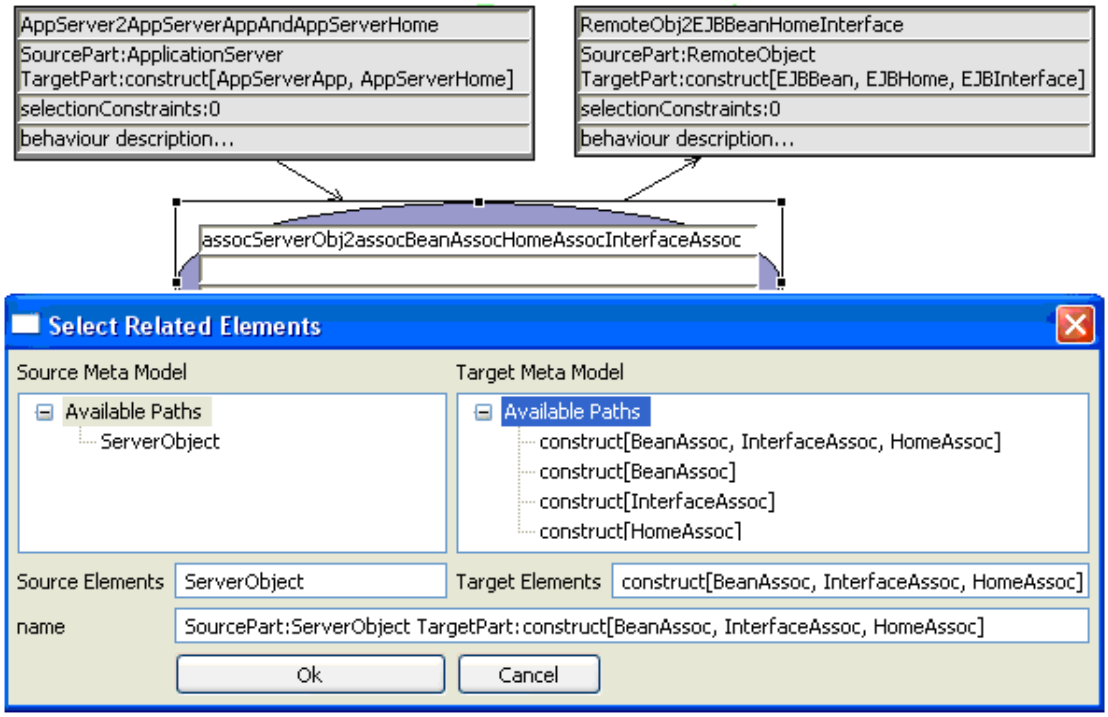


Figure 10.24. An *associationMapping* without inconsistency

10.3.2 Setting up the value for the *semanticTranslation* property

Figure 10.25(a) illustrates how the MaramaCRelation tool sets up the value of the *semanticTranslation* property of a SemanticAssociation. For each SemanticAssociation, the MaramaCRelation tool retrieves the ***translatable semantic constraints*** (defined in section 9.4.5.3) from both the source and target meta-models (1). The retrieved translatable semantic constraints are listed as tree root nodes and ready to be translated (2). The translated semantic constraints are listed as children of the tree root nodes (3).

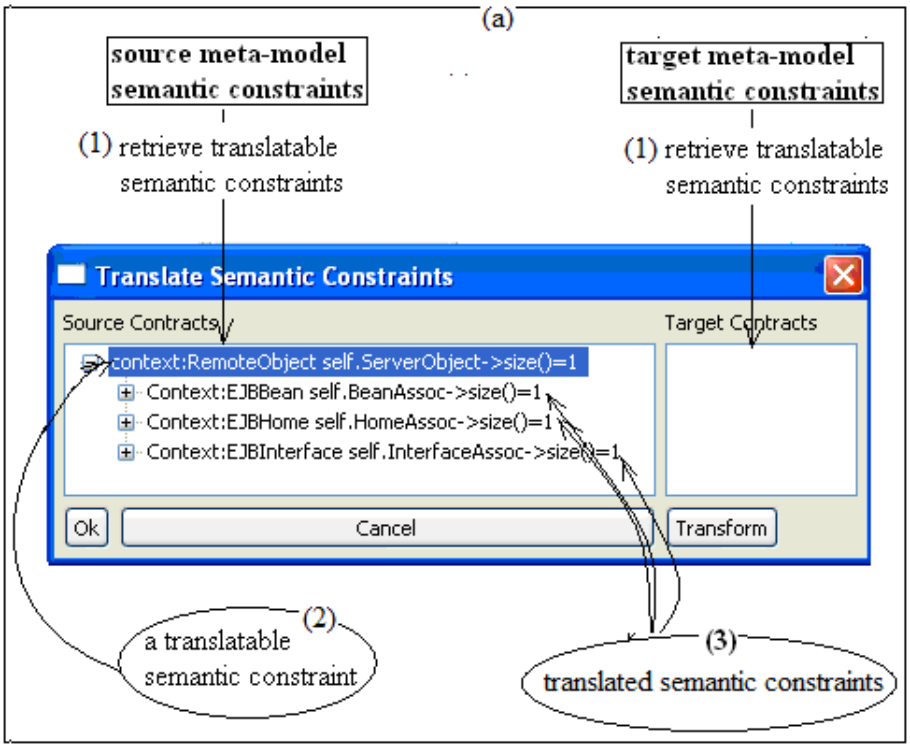


Figure 10.25. Setting up value for the *semanticTranslation* property of a SemanticAssociation

The SemanticAssociation in Figure 10.25 has a translatable semantic constraint *context:RemoteObject self.ServerObject->size()=1*. This translatable semantic constraint is represented as a tree node in Figure 10.25 and needs to be translated into sensible target meta-model semantic constraint(s). The MaramaCRelation tool uses an empirical algorithm to translate semantic constraints. The algorithm, illustrated in Figure 10.26, goes through each source (target) meta-model element involved in a translatable semantic constraint, and translates it into appropriate target (source) meta-model elements.

According to the pink area of the algorithm (line 17 to 24, Figure 10.26), if the source meta-model element in the constraint belongs to a StructureMapping or a SemanticAssociation that has a single-element source part and a single-element target part (lines 17&18, Figure 10.26), replace the source meta-model element with the corresponding target meta-model element (lines 20&21, Figure 10.26) and save the partially translated semantic constraint for the next loop (line 23, Figure 10.26).

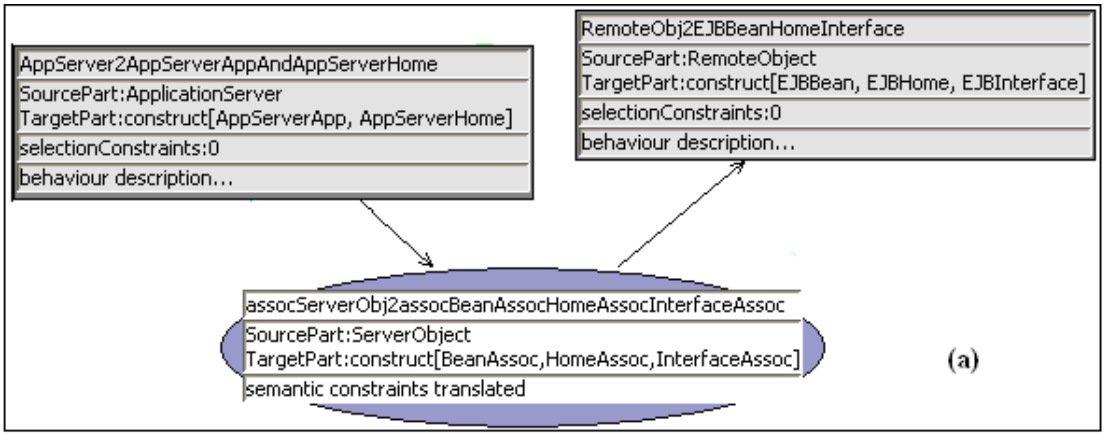
According to the yellow area (line 25 to 35, Figure 10.26), if the source meta-model element in the constraint belongs to a StructureMapping or a SemanticAssociation that has a single-element source part and a construct target part (lines 25&26, Figure 10.26), for each element of the target meta-model construct (line 28, Figure 10.26), use it to replace the source meta-model element (lines 30&31, Figure 10.26) in the constraint and save the partially translated semantic constraint for the next loop (line33, Figure 10.26). As the target part is a construct, which means that multiple temporary translated constraints can be generated and added to the translated result (line 33, Figure 10.26).

According to the blue area (line 36 to 43, Figure 10.26), if the source meta-model element in the constraint belongs to a StructureMapping or a SemanticAssociation that has a construct source part and a single- element target part (lines 36&37, Figure 10.26), replace the source meta-model element in the constraint with the correspondent target meta-model element (lines 39&40, Figure 10.26) and save the partially translated semantic constraint for the next loop (line 42, Figure 10.26).

According to the grey area (line 44 to 54, Figure 10.26), if the source meta-model element in the constraint belongs to a StructureMapping or a SemanticAssociation that has a construct source part and a construct target part (lines 44&45, Figure 10.26), for each element of the target meta-model construct (line 47, Figure 10.26), use it to replace the source meta-model element (lines 49&50, Figure 10.26) in the constraint and save the partially translated semantic constraint for the next loop (line 52, Figure 10.26). As the target part is a construct, which means that multiple temporary translated constraints can be generated and added to the translated result. When all the source meta-model elements in the translatable constraint are translated into target meta-model elements, the translation process is finished (red area, line 8 to 12, Figure 10.26).

```
Vector tempTranslatedConstraints = new Vector();
Vector translatedConstraints = new Vector();
tempTranslatedConstraint = null;

void translateSemanticConstraints(a-semantic-constraint)
{
  if(a-semantic-constraint doesn't contain any source meta-element)
  {
      translatedConstraints.add(a-semantic-constraint);
      return;
  }

  for (each source meta-model element in the translatable
                                        semantic constraint)
  {
    if(the StructureMapping/SemanticAssociation consists of
        single-element source part && single-element target part)
    {
      tempTranslatedConstraint = replace the source meta-model element
                                    with the target meta-model element;

      tempTranslatedConstraints.add(tempTranslatedConstraint);
    }
    if(the StructureMapping/SemanticAssociation consists of
        single-element source part && construct target part)
    {
      for (each element of the target meta-model construct)
      {
        tempTranslatedConstraint = replace the source meta-model element
                      with the element of the target meta-model construct;

        tempTranslatedConstraints.add(tempTranslatedConstraint);
      }
    }
    if(the StructureMapping/SemanticAssociation consists of
        construct source part && single-element target part)
    {
      tempTranslatedConstraint = replace the source meta-model element
                                    with the target meta-model element;

      tempTranslatedConstraints.add(tempTranslatedConstraint);
    }
    if(the StructureMapping/SemanticAssociation consists of
        construct source part && construct target part)
    {
      for (each element of the target meta-model construct)
      {
        tempTranslatedConstraint = replace the source element
                with the element of the target meta-model construct;

        tempTranslatedConstraints.add(tempTranslatedConstraint);
      }
    }
    while (tempTranslatedConstraints.hasMoreElements())
    {
      tempTranslatedConstraint = translatedConstraints.nextElement();

      translatedSemanticConstraints (tempTranslatedConstraint);
    }
  }
}
```

Figure 10.26. The empirical algorithm to translate translatable semantic constraints



context:RemoteObject self.ServerObject->size()=1

1. context:EJBBean self.BeanAssoc->size()=1
2. context:EJBBean self.HomeAssoc->size()=1
3. context:EJBBean self.InterfaceAssoc->size()=1
4. context:EJBHome self.BeanAssoc->size()=1
5. context:EJBHome self.HomeAssoc->size()=1
6. context:EJBHome self.InterfaceAssoc->size()=1
7. context:EJBInterface self.BeanAssoc->size()=1
8. context:EJBInterface self.HomeAssoc->size()=1
9. context:EJBInterface self.InterfaceAssoc->size()=1

(b)

Figure 10.27. Translating semantic constraints

Figure 10.27(b) shows a sample semantic constraint of the MaramaMTE meta-model (top, highlighted). The semantic constraint is translated into 9 semantic constraints of the EJBUML meta-model according to the translation algorithm. The MaramaCRelation tool then checks if the translated constraints are sensible for the target meta-model, and remove the insensible ones. For example, the translated constraint 2 in Figure 10.27(b) is insensible, because in the EJBUML meta-model, it is not allowed for an EJBBean to have an association of HomeAssoc. In Figure 10.27(b) only constraints 1, 5, and 9 are the sensible translated constraints for the target meta-model.

10.3.3 Brief summary

SemanticAssociations makes explicit information that is used to be implicit and hidden in MI&T. The SemanticAssociation associates isolated StructureMappings, which makes it possible for isolated StructureMappings to communicate with each other. The MaramaCRelation tool supports the modelling of a SemanticAssociation. It automatically derives value for the *associationMapping* property and translates source (target) model semantic constraints into target (source) model semantic constraints.

10.4 Supporting analysis and design of model transformation

The CRelation model can be used to analyze and design model transformation. Most existing transformation technologies such as ATL, XSLT, are template-based. Users construct transformation templates driven by ad-hoc goals. Although these technologies allow users to refactor the templates to improve the structure and reduce the repetitive information, the refactoring is implementation/operation level support, and is normally viewed as secondary to the functions of templates.

The current state of transformation is similar to the development of Object Oriented (OO) software prior to the introduction of OO analysis and design formalisms. Without OO analysis and design, OO developers can still develop OO software. With the support of an OO environment such as a Java virtual machine, the OO software can achieve functional goals. However, ad-hoc OO software is largely based on the software developers' experience and lacks support for purposely designed performance, adaptability, maintainability, and so on.

Figure 10.28(a) represents part of a sample ATL script to transform a MaramaMTE model to an EJBUML model. The figure only shows the matched rules of the ATL script, and does not show the contents of ATL queries and helpers. Figure 10.28(b) is the sample MaramaMTE – EJBUML CRelation model. The CRelation model analyzes the ATL script and in the following aspects:

1) *Decomposition of transformation scripts*

   The CRelation model breaks down the monolithic model transformation script into its interconnection units—the StructureMappings. Each ATL matched rule is represented by a StructureMapping. The *from* and *to* elements of a matched rule respectively correspond with the

source part and the *target part* of a StructureMapping. The *using* section and the *initialization section* of a matched rule are represented by the *selectionConstraints* of the StructureMapping.

2) *Association of isolated rules and templates*

The CRelation model uses SemanticAssociations to explicitly represent the associations between rules and templates. In Figure 10.28(a), the transformation is consisted of isolated matched rules, and the associations between those rules are nowhere to see. The CRelation model associates StructureMappings, and allows transformation rules and templates to communicate with each other in the context of a broader model.

3) *Define What and How to transform*

Traditional model transformation technologies focus on operations and do not analyze an intended transformation. For example, none of ATL, XSLT, and VARTIA explicitly analyzes what to transform. The CRelation model explicitly identifies what to transform from how to transform. Users can first establish the CRelation model elements to analyze: if the intended transformation matches the users' mindset; conforms to the semantics of the source and target meta-models; causes the least unwanted information; and maintains the translatable semantic constraints. Once the interconnection units and their associations are well analyzed, users can define how to realize the transformation by establishing selection constraints, generating behaviour descriptions, and traceability.

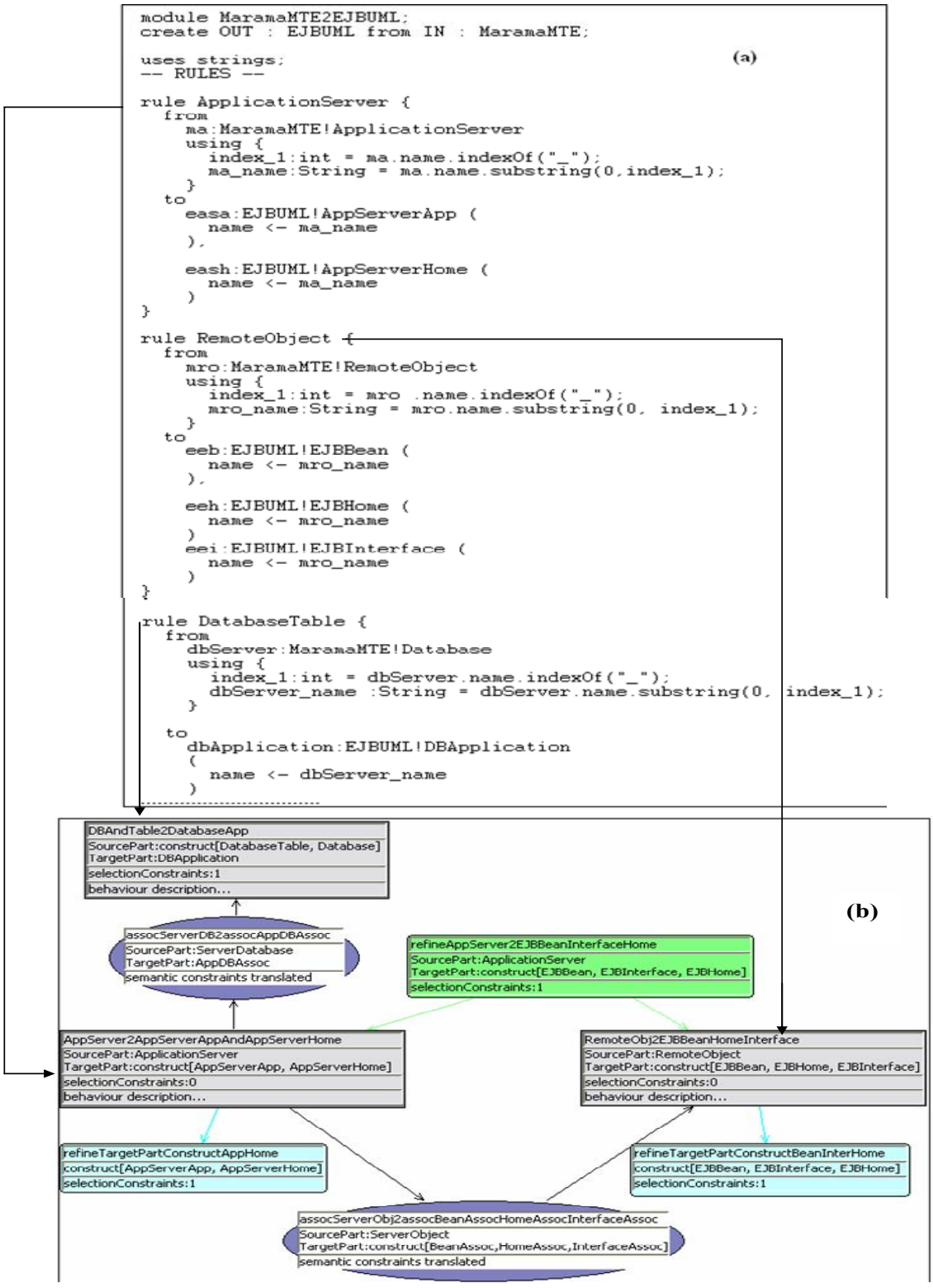


Figure 10.28. Using the Crelation model to analyze and design ATL scripts

4) *A visual presentation of the main concerned problems of transformation*

The CRelation model is different from graph-based model transformation technologies such as VIATRA. Like text-based transformation technologies (e.g. ATL, XSLT), the existing graph-based transformation technologies are concerned with how to create the target model (a new graph) out of the source model (an original graph). At a higher level, the CRelation model abstracts and visually presents the main concerned issues involved in MI&T. The CRelation model explicitly defines things that used to be implicit, such as what and how to integrate and transform; the representation of traditionally implied but ignored transformation information; and the maintenance of the translatable semantic constraints. The CRelation model makes it possible to automate issues such as behaviour description generation, flexible traceability mechanism (will be discussed in Section 10.5), and semantic inconsistency detection.

10.5 The MaramaCRelation tool supporting traceability

The MaramaCRelation tool not only supports CRelation modelling, but also generates *search conditions* from a CRelation model to establish traceability between the interconnected models. Figure 10.29 illustrates the main steps in establishing traceability between instance models of the interconnected meta-models. The users load the source (e.g. Pet Store MaramaMTE model) and target models (e.g. Pet Store EJBUML model) (1)(2). The users choose an appropriate CRelation model that interconnects the meta-models of the source and target models (3). Users choose the interested part of the source model to be interconnected with the target model (4). The MaramaCrelation tool uses the search conditions (5) to compute what target modelling elements would meet the selection constraints and become candidates the source elements can interconnect with (6). Each source model element obtains a set of eligible candidates. Users assign a candidate to the source element (7). The assignation continues until values have been assigned to all the interested source model elements (8). The source model is then interconnected with the target model (9). The interconnected parts of the source and target models are isomorphic, and the interconnected elements meet the selection constraints defined in the used CRelation model.

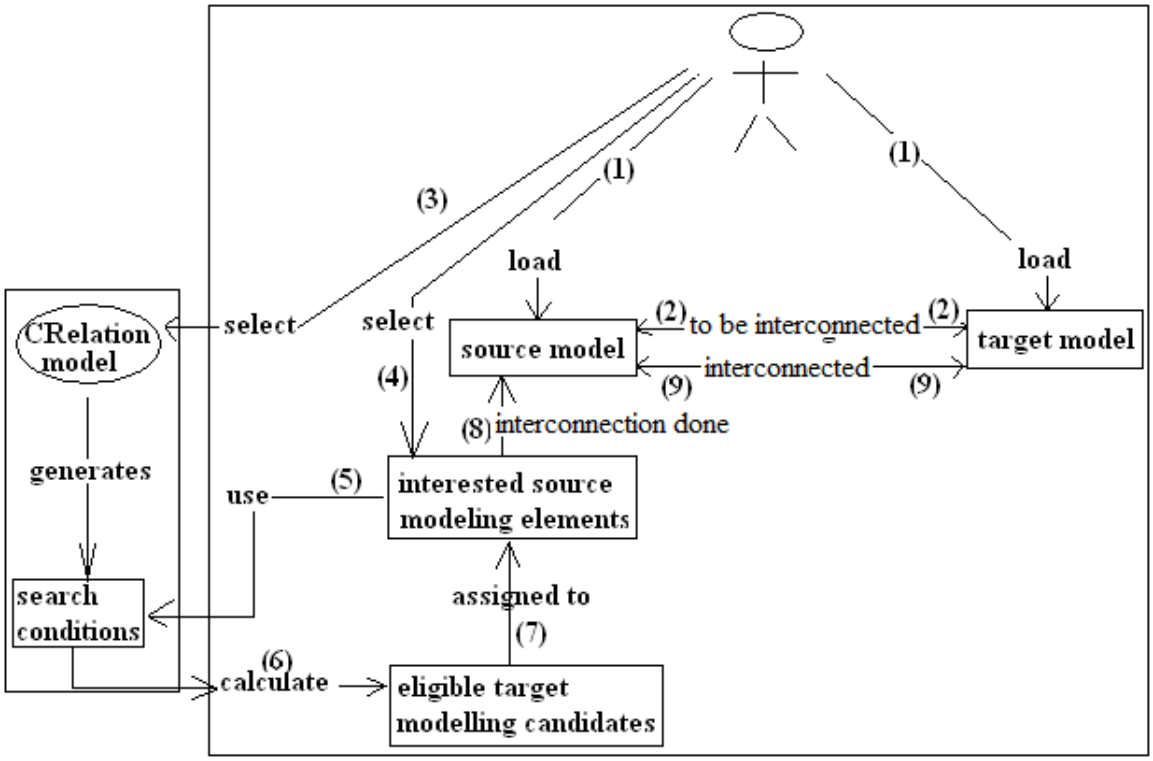


Figure 10.29. Setting up traceability between two models using the MaramaCrelation tool

10.5.1 Search conditions and search interfaces

Search conditions are the core part of the MaramaCRelation tool traceability mechanism. The goal of search conditions is that given a certain source model element (an element or a construct), the search conditions help to find out valid interconnection candidates in the target model. The search conditions are a set of java classes generated from a CRelation model, which provide isomorphic functions for the involved source and target models. The search conditions are MaramaCRelation tool-API-dependent, but their functions can be abstracted to three API-independent search interfaces, and they are: the StructureMapping interface, the SemanticAssociation interface, and the StructureRefinement interface:

- ***the StructureMapping interface***

Figure 10.30 shows the main methods of the StructureMapping search interface.

```
1  public interface StructureMapping
2  {
3    public static Vector getTargetCandidates(Shape[] sourceShapes,
4                                             Diagram targetModelDiagram);
5
6    private static boolean checkConstraint(Shape[]  sourceShapes,
7                                           Shape[] targetShapes);
8
9    public static Boolean isSourceOfTheInterElement(Object sourceElement);
10 }
```

Figure 10.30. Methods of the StructureMapping search interface

Method getTargetCandidates takes two parameters sourceShapes and targetModelDiagram. The sourceShapes parameter represents an array of selected source model shapes, which represents the instances of the source part of this StructureMapping. The targetModelDiagram represents the target model diagram. This method finds out a vector of target model candidates for the source model elements (represented by the sourceShapes). The candidates must be the instances of the target part of the StructureMapping and meet the selection constraints upon them.

Method checkConstraint encodes one selection constraint of the StructureMapping. Parameters of the method represent the source shapes and the possible target model candidate shapes. This method checks if the source shapes and the possible target model candidate elements meet the selection constraint. If the target model candidate shapes make the return result to be true, they are qualified candidates for the source elements. If a StructureMapping contains multiple selection constraints, there should be more methods in the form of checkConstraint1, checkConstraint2, and so on. The checkConstraint methods are called by other methods (e.g. method getTargetCandidates) to find out qualified target model candidates for the source model elements.

Method isSourceOfTheInterElement checks if the selected source model element is or part of an instance of the source part of the StructureMapping.

• ***the SemanticAssociation interface***

Figure 10.31 shows the main methods of the SemanticAssociation search interface. All the methods have similar goals to the same-named methods in the StructureMapping search interface. As the SemanticAssociation does not contain selection constraints, so it does not contain a checkConstraint

method. The method getTargetType returns the abstraction type of the target meta-model element of the SemanticAssociation.

```
1
2  public interface SemanticAssociation
3  {
4    public static Vector getTargetCandidates(Diagram targetModelDiagram);
5
6    public static String getTargetType();
7
8    public static boolean isSourceOfTheInterElement(Object sourceElement,
9                                                    Object sourceModel);
10 }
```

Figure 10.31. Methods of the SemanticAssociation Interface

- ***the StructureRefinement Interface***

```
1  public interface StructureRefinement
2  {
3    public static Object updateIE2TgtCandidateByIE1TgtCandidate(
4                              Object tgtCandidateInInterEntity_1,
5                              Vector tgtCandidateInInterEntity_2)
6
7    private static boolean checkInterConstraint(Shape[] sourceShapes,
8                                                Shape[] targetShapes);
9
10   public static Boolean isSourceOfTheInterElement(Object sourceElement,
11                                                   Object sourceModel);
12 }
```

Figure 10.32. Methods of the StructureRefinement Interface

Figure 10.32 shows the *main methods* of the StructureRefinement search interface. All the methods have similar goals to the same-named methods in the StructureMapping search interface. The updateIE2TgtCandidateByIE1TgtCandidate method (shortened as update method) needs more explanations. A StructureRefinement constrains two parent StructureMappings (for example, StructureMapping-1 and StructureMapping-2) and puts second order constraints on the target candidate elements that have passed the search conditions brought up by the two StructureMappings. The target candidates of StructureMapping-1 are independent from the target candidates of StructureMapping-2. The update method is used to find out what target candidate of StructureMapping-2 is associated to what target candidate of StructureMapping-1, and if they meet (through method checkInterConstraint) the second order selection constraints brought by the StructureRefinement.

10.5.2 Sample MaramaCRelation search conditions

In a CRelation model**,** each StructureMapping, SemanticAssociation, and StructureRefinement generates a java class that implements the StructureMapping, SemanticAssociation, and StructureRefinement search interface respectively**.** Figure 10.33(a) shows the package of search conditions generated from the MaramaMTE-EJBUML CRelation model (refer to Figure 9.2(b)). Each of the three StructureMappings, two SemanticAssociations, and one StructureRefinement generates one java class that implements its own search interface. The SelectionRefinements in a CRelation model do not generate separated java classes but contribute to the constraints of their parent StructureMappings. The CRelation model also generates helper classes (e.g. "InterModel" class). The generation of the helper classes is very implementation-focused, and is not explained here. A sample getTargetCandidates method of StructureMapping_2.java (generating from StructureMapping AppServer2AppServerAppAndAppServerHome in the MaramaMTE-EJBUML CRelation model) is displayed in Figure 10.33(b); the checkConstraint0 method is shown in detail in Figure 10.33(c); and the isSourceOfTheInterElement method is shown in detail in Figure 10.33(d).

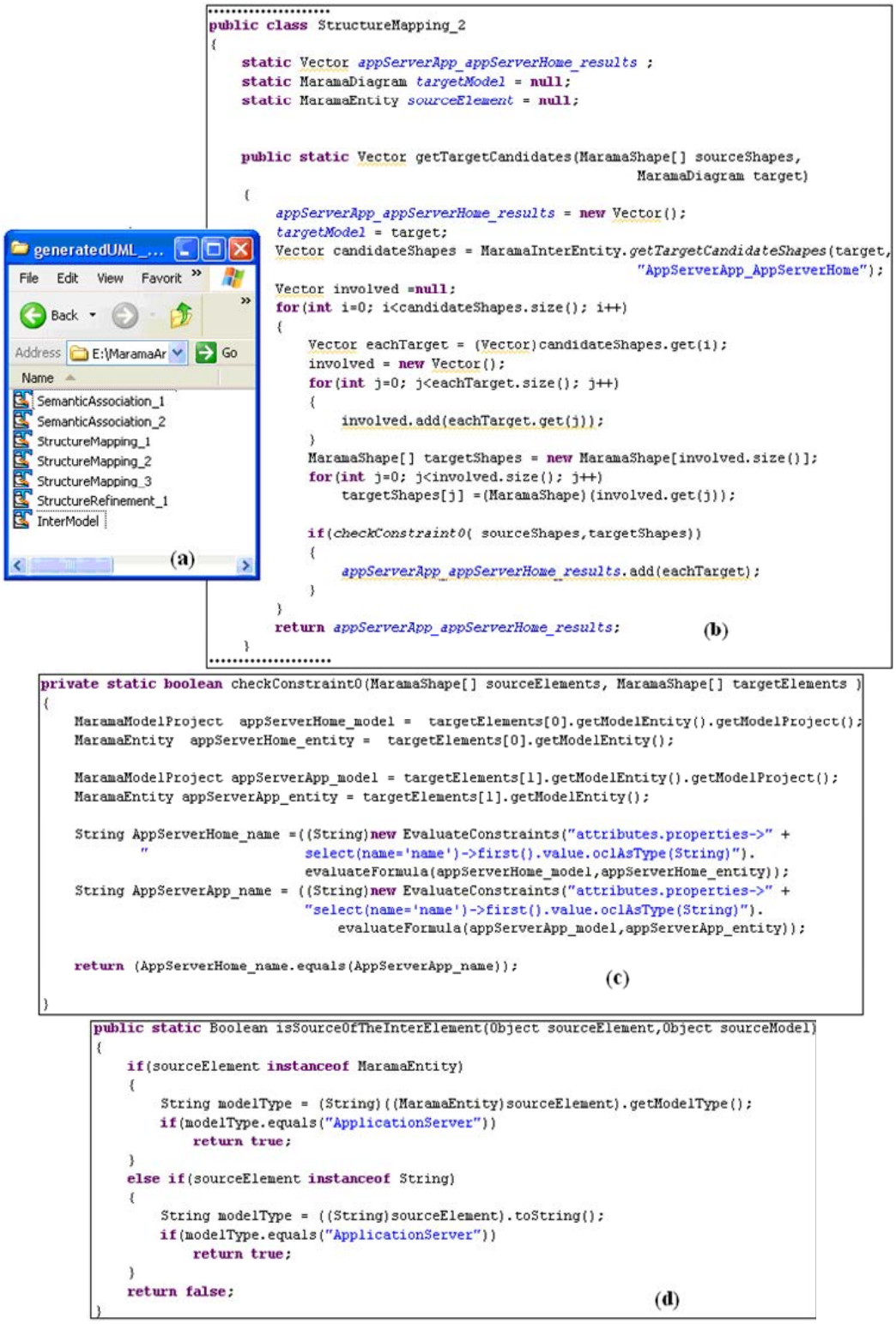


Figure 10.33. Sample search conditions

10.5.3 The algorithm of the interconnecting process

When interconnecting two models (e.g. Pet Store MaramaMTE architecture model and Pet Store EJB-UML design model) and establishing the traceability between them, a CRelation model is, in fact, the isomorphism between the two models; and StructureMappings, StructureRefinements, and SemanticAssociations define the functions of the isomorphism.

The algorithm of the interconnecting process is illustrated in Figure 10.34. For each source model element, the MaramaCRelation tool retrieves and lists all the valid target model candidates that meet the conditions defined by the involved StructureMappings and StructureRefinements (lines between 1 and 16, in Figure 10.34).

```
1  for(each source model element)
2  {
3    if(the element involves in a StructureMapping)
4    {
5      find out all target model candidates;
6      remove the target model candidates that does not
7                        meet the selection constraints of
8                             the involved StructureMapping;
9      }
10   if(the target model candidates involves in a StructureRefinement)
11   {
12     remove the target model candidates that does not
13                            meet the selection constraints of
14                            the involved StructureRefinement;
15     }
16 }
17
18 for(each source model element)
19 {
20   assigning a target model candidate to the source model element;
21   update the candidate lists for the yet-to-be-assigned
22                                    source model elements;
23 }
```

Figure 10.34. The algorithm of the interconnecting process

Users start to interconnect the two models by assigning a target model candidate to a source model element. As the assigning process going, the candidate lists of the yet-to-be-interconnected source model elements need to be updated (line 21, Figure 10.34). Each updated candidate list of a source model element must make sure that for each candidate element (e.g. list_A_element_d) in the list, there must be at least one candidate element in the candidate list of each other yet-to-be-interconnected source model elements (e.g. list_B_element_w, list_C_element_d, list_D_element_c……), that list_A_element_d, list_B_element_w, list_C_element_d, list_D_element_c ……, and the already interconnected target

model elements become isomorphic to the corresponding source model part (lines between 18 and 23 in Figure 10.34). Once all the source model elements have been assigned values, the interconnection is done; and the traceability between the two models is established.

10.5.4 Interconnecting source and target models

Figure 10.35 illustrates how users interconnect two self-evolved domain-specific models (the Pet Store MaramaMTE model and the Pet Store EJBUML model) in the MaramaCRelation tool. A popup menu item opens up the wizard for users to prepare the model interconnection (1). Users need to choose the source and target models in a wizard page (2), and import the CRelation model that interconnects MaramaMTE meta-model and EJBUML meta-model(3). The imported CRelation model represents the path to the search conditions.

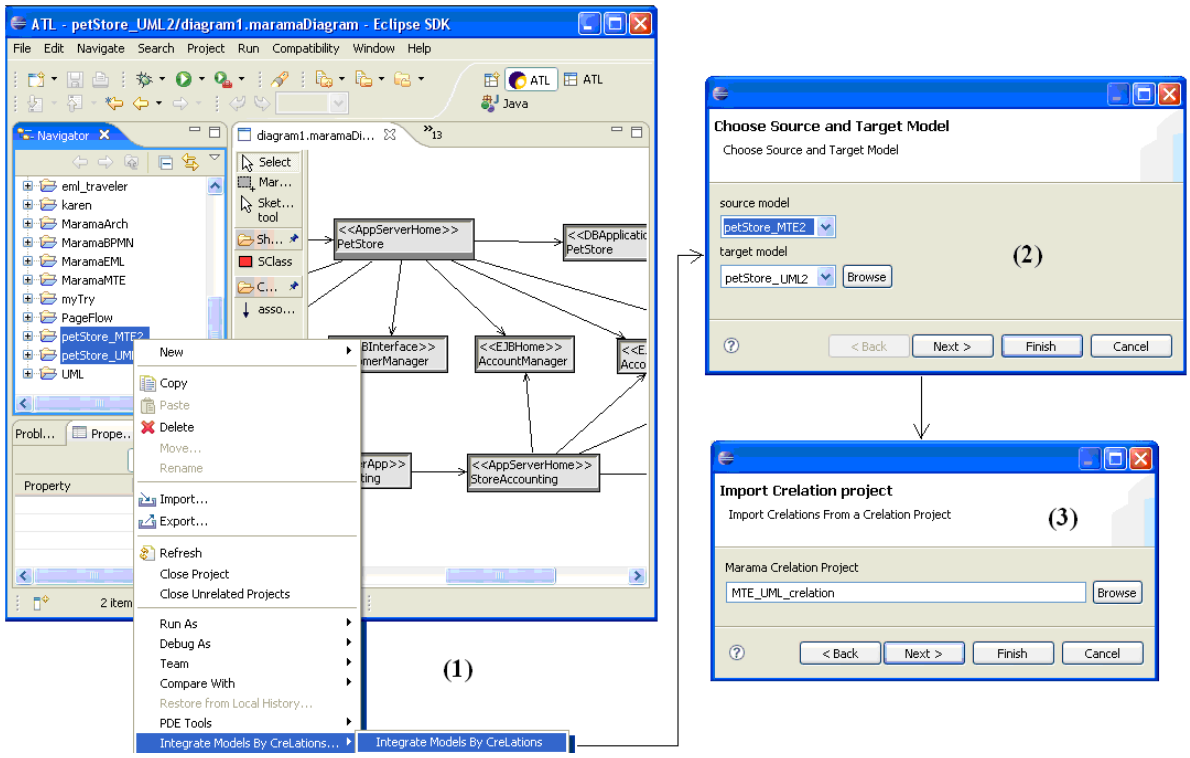


Figure 10.35. Selecting a CRelation model to interconnect Pet Store MaramaMTE model and Pet Store EJBUML model

Once the CRelation model is imported, users start to interconnect the two models. In Figure 10.36, users are interested in interconnecting part of the Pet Store MaramaMTE model (a) with the Pet Store

EJBUML model (b). Each involved source model element has a set of interconnection candidates from the target model (c). Users need to assign a target model element or construct from the candidate list to the source model element. Once values have been assigned to all the interested source model elements, the interconnection process is finished.

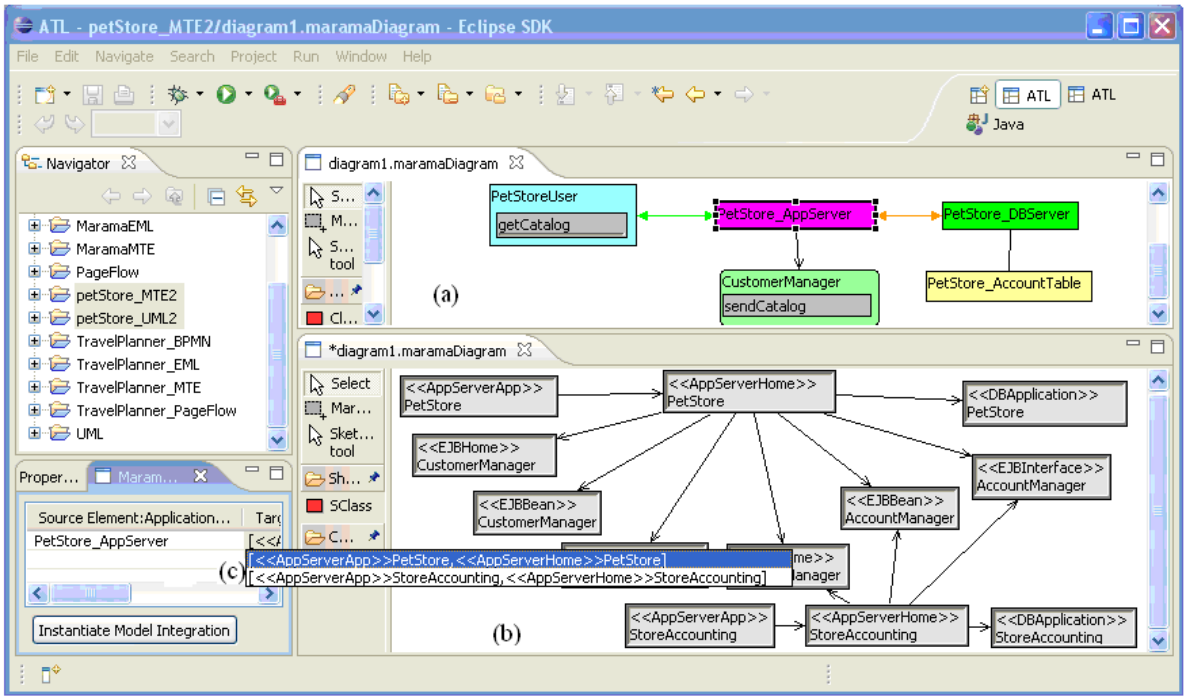


Figure 10.36. Assigning target model element or construct to the source model element

PetStore MaramaMTE Model	PetStore EJBUML Model
PetStore_AppServer	<<AppServerApp>>PetStore,<<AppServerHome>> PetStore
CustomerManager	<<EJBHome>>CustomerManager,<<EJBBean>> CustomerManager,<<EJBInterface>> CustomerManager
PetStore_DBServer,PetStore_DBTable	<<DBApplication>>PetStore

Table 10.1. Interconnected elements between the Pet Store MaramaMTE model and the Pet Store EJBUML model

Table 10.1 shows the results of the model interconnection. The interconnected elements meet all the constraints defined in the chosen CRelation model. Each pair of interconnected source and target model

elements is the instance of the source and target parts of a StructureMapping, and meets all the selection constraints of the StructureMapping.

10.6 Design and Implementation

10.6.1 Building the MaramaCRelation tool from the Marama meta-tool

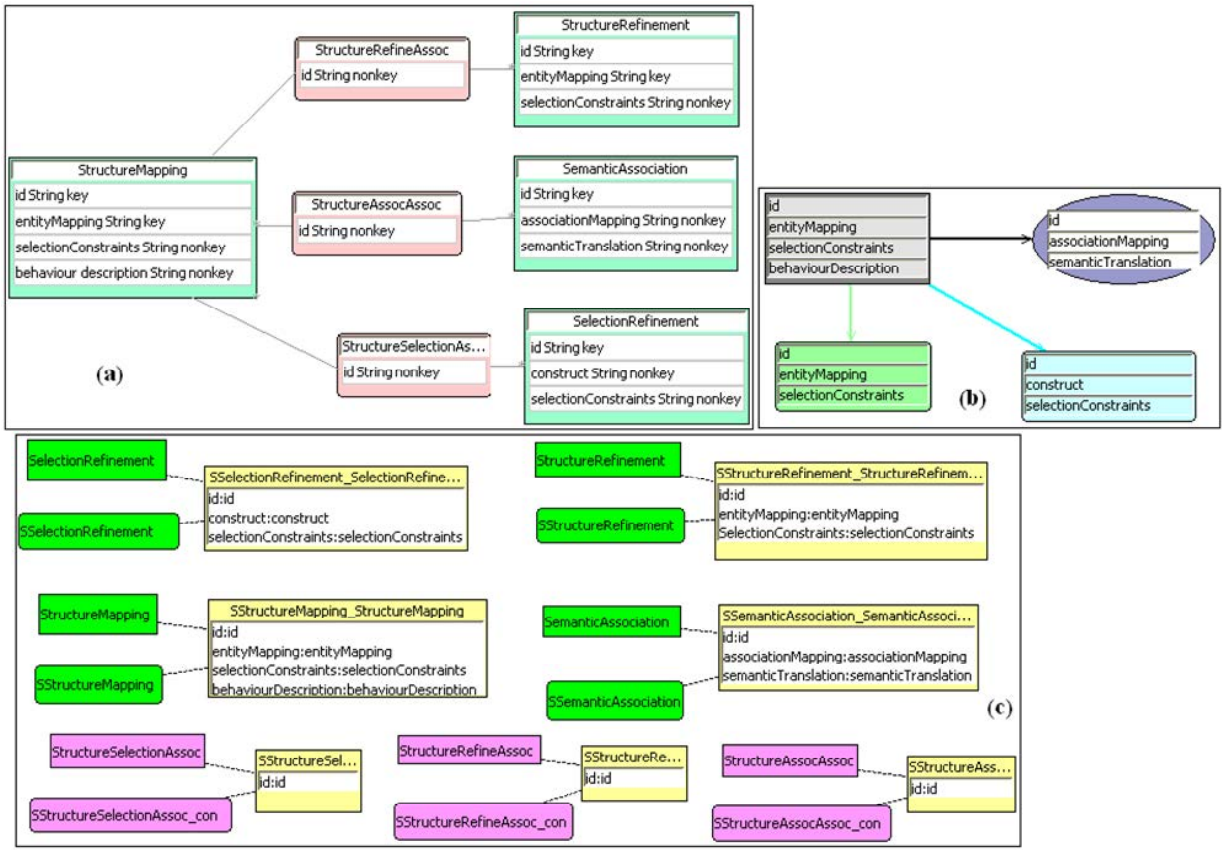


Figure 10.37. Building the MaramaCRelation tool by using the Marama meta-tool

The MaramaCRelation tool is not implemented from the scratch, but built by using the Marama Meta-tool. Users design entities and associations of the MaramaCRelation tool in the *tool definer* of the Marama meta-tool (Figure 10.37(a)). The MaramaCRelation tool provides the CRelation modeller with four entities: StructureMapping, SelectionRefinement, StructureRefinement, and SemanticAssociation; and three associations: StructureRefineAssoc, and StructureSelectionAssoc, and StructureAssocAssoc. Users define a visual representation (shape) for each modelling entity and association in the *shape designer* of the Marama meta-tool (Figure 10.37(b)). Finally, users match each model element (defined

in the “tool definer”) to a visual representation (defined in the view type definer) (Figure 10.37(c)), and the basic modeller is built up. The tool developers develop the backend functions to complete the tool.

10.6.2 Generating search conditions

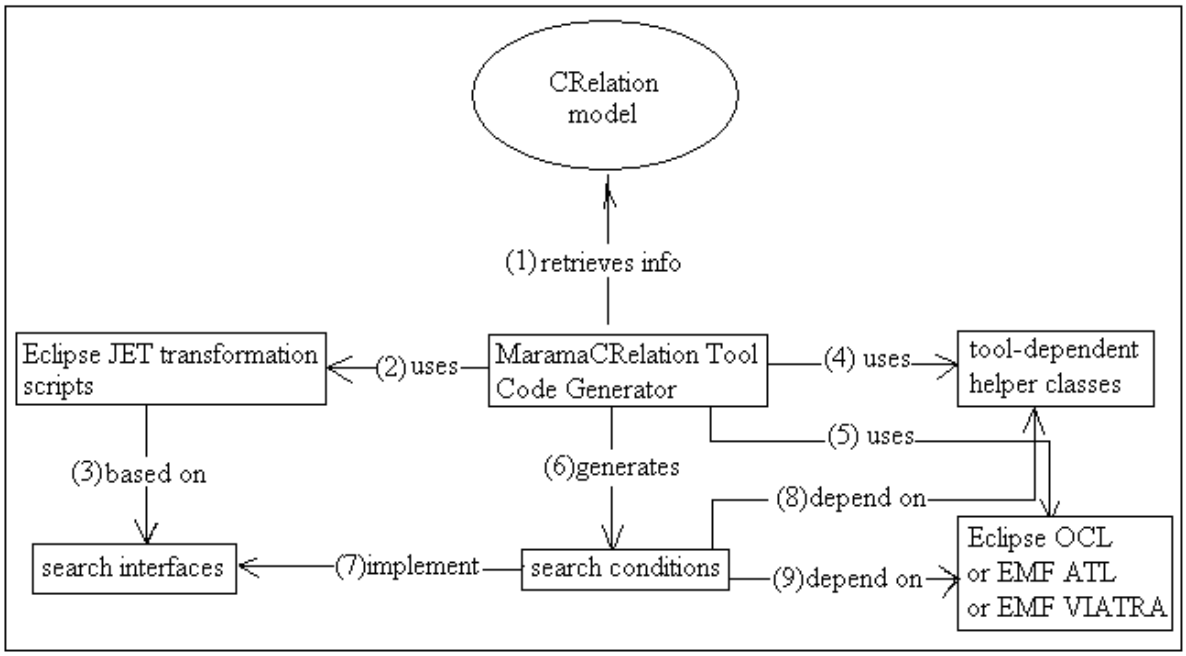


Figure 10.38. The process of generating search conditions

Figure 10.38 illustrates the process of search condition generation in the MaramaCRelation tool. The *MaramaCRelation Tool Code Generator* is the core part. It retrieves modelling information from a CRelation model (1); and uses a set of pre-built Eclipse JET transformation scripts (2), the Eclipse OCL package (4), and tool-dependent helper classes (5) to generate search conditions (6). The *Eclipse JET transformation scripts* are based on pre-designed search interfaces (3), and help to generate the concrete java implementation of the search interfaces. The search conditions depend on tool-dependent helper classes (8) and Eclipse OCL (9).

10.7 Summary

The MaramaCRelation tool provides highly automated and effective support for the CRelation modelling. When establishing a StructureMapping, the tool calculates available constructs of the source and target meta-models; visualizes the coordination between the StructureMapping and the source and target meta-models; and automatically generates the high-level synchronization behaviour description.

When setting up a SemanticAssociation, the tool calculates paths between the source and target parts of the associated StructureMappings. The tool also automatically finds out the translatable semantic constraints and translates them into sensible target model semantic constraints. The tool generates java-based search conditions to establish traceability between two self-evolved models.

The MaramaCRelation tool is built by using Marama meta-tool. The MaramaCRelation tool compiles selection constraints and generates java-based search conditions. Based on the search conditions, the MaramaCRelation tool exercises pattern search to find out valid interconnection candidates in the target model. With the assigning of a target model candidate to the corresponding source model element, the MaramaCRelation tool can validate the candidate lists for the yet-to-be-interconnected source model elements. The interconnected source and target models maintain traceability.

Chapter 11 - Case Study - Using the CRelation Model

This chapter uses two case studies: online Pet Store web application and online Travel Planner to demonstrate how the various domains-specific models can be interconnected by the CRelation model. In this chapter, the Pet Store project will be modelled in MaramaMTE and FormChart; and the two models will be interconnected through a CRelation model. The Travel Planner project will be modelled in MaramaEML (Li et al, 2007), BPMN (BPMN, 2004), MaramaMTE, and Form Chart; and three CRelation models will be used to interconnect the four domain-specific models. For each case study, the interconnected domain-specific models construct a multi-view environment with the maintained traceability, view synchronization, and behaviour synchronization across the interconnected models.

11.1 Case study 1: Interconnecting the Pet Store MaramaMTE model with the Pet Store EJBUML model

The Pet Store case has been used as a running example through chapters 8, 9, and 10. Those chapters introduce the features of the MaramaCRelation approach through interconnecting the Pet Store MaramaMTE and EJBUML models. Here, the Pet Store example is used as a case study to review the features and strength of the MaramaCRelation approach.

11.1.1 The MaramaMTE-EJBUML CRelation model and its entities

Figure 11.1 is the MaramaMTE-EJBUML CRelation model that captures the shared semantics between the MaramaMTE meta-model (refer to section 8.4) and the EJBUML meta-model (refer to section 8.4). The CRelation model uses StructureMappings, SelectionRefinements, and StructureRefinements to categorize the selection constraints. The constraints of a CRelation model entity can only use the information of the meta-model elements involved in the model entity.

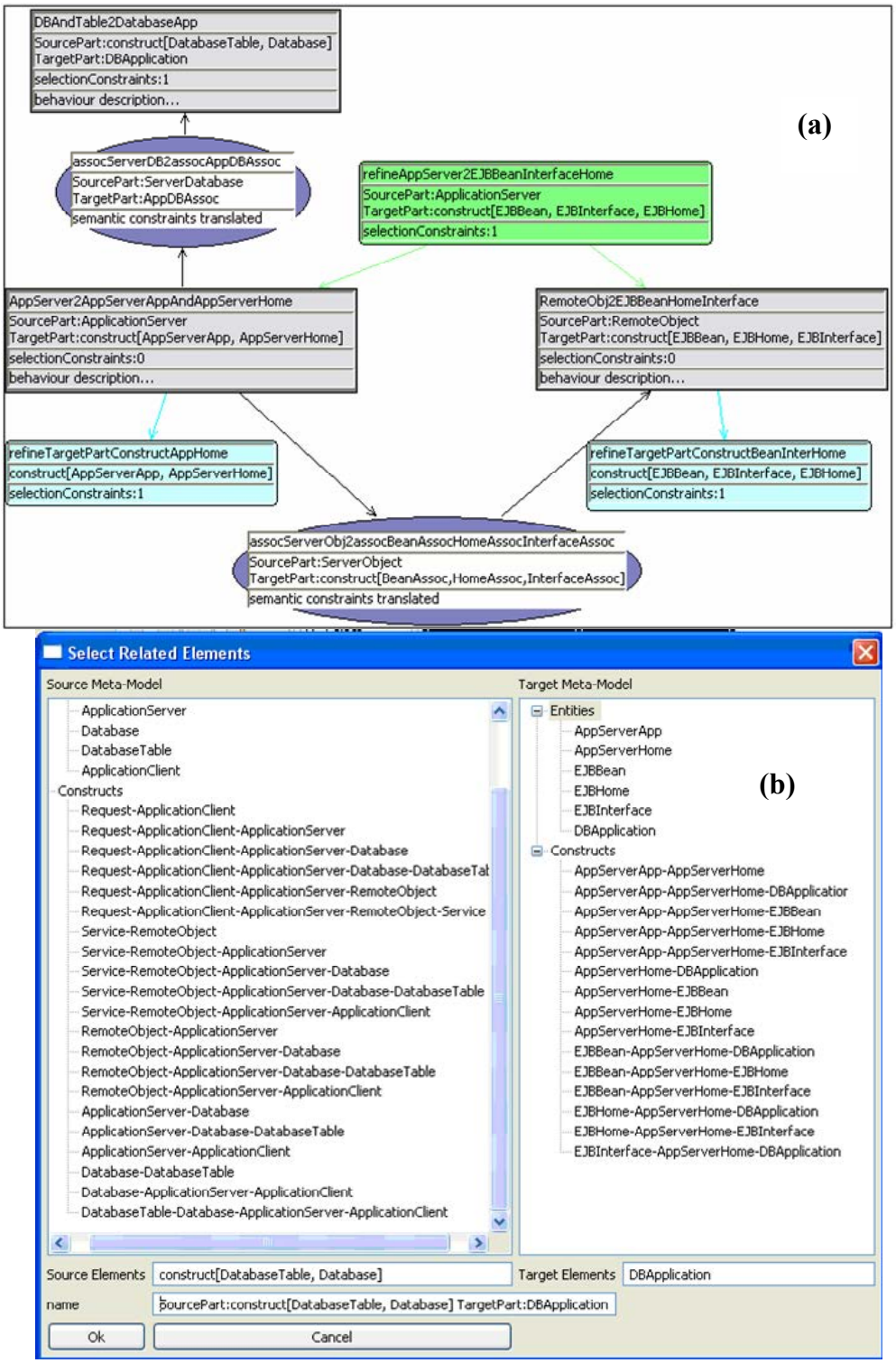


Figure 11.1. (a) the MaramaMTE-EJBUML CRelation model; (b) the sample *entityMapping* property sheet of StructureMapping DBandTable2DatabaseApp

In Figure 11.1(a), each of the three StructureMappings (grey rectangles) means that under certain selection constraints the source and target parts of the StructureMapping represent the similar semantics with the different representations in the source and target models. Interconnection Relationships can be established between the eligible source and target meta-model elements and instance model elements. The traceability can be established and maintained across the interconnected instance models. In Figure 11.1(b), available meta-model entities and sensible constructs are calculated automatically and listed in the *entityMapping* property sheet of a StructureMapping, so users can choose a pair of semantically similar source and target meta-model elements to set up the property value.

A SelectionRefinement (cyan rectangles in Figure 11.1(a)) specifies selection constraints on a construct. For example, SelectionRefinement "refineTargetPartConstructAppHome" in Figure 11.2(a) specifies constraints on the construct[AppServerApp, AppServerHome] of the parent StructureMapping "AppServer2AppServerAppAndAppServerHome" (refer to Figure 11.1(a)). More specifically, this constraint requires that the "name" value of an AppServerApp and the "name" value of an AppServerHome must meet the constraint defined in Figure 11.2(b) before the Interconnection Relationship specified by its parent StructureMapping can be established.

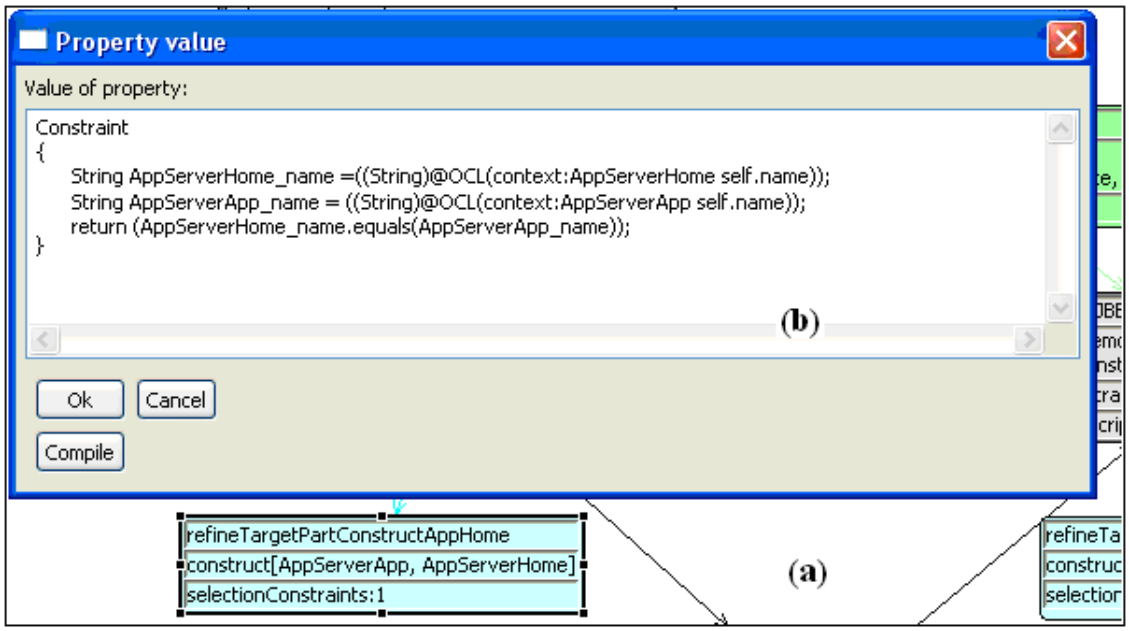


Figure 11.2. The *selectionConstraints* property sheet of SelectionRefinement refineTargetPartConstructAppHome

A StructureRefinement (green rectangles in Figure 11.1(a)) allows users to specify second order selection constraints on its two parent StructureMappings. It puts extra selection constraints between source part of one parent StructureMapping and the target part of the other parent StructureMapping. For example, in Figure 11.3(a), the StructureRefinement refineAppServer2EJBBeanInterfaceHome specifies the second order selection constraints between the source meta-model element ApplicationServer and the target meta-model elements construct[EJBBean, EJBHome, EJBInterface]. In Figure 11.3(b), this selection constraint finds out the RemoteObjects associated with the ApplicationServer (via navigation to the association end of ServerObject, refer to the MaramaMTE meta-model in Figure 9.1(a)), and requires that the ApplicationServer is associated with the *RemoteObject* that is mapped to the target model construct containing the EJBHome and has the same *name* value of the EJBHome, before the Interconnection Relationships specified by the two parent StructureMappings can be established.

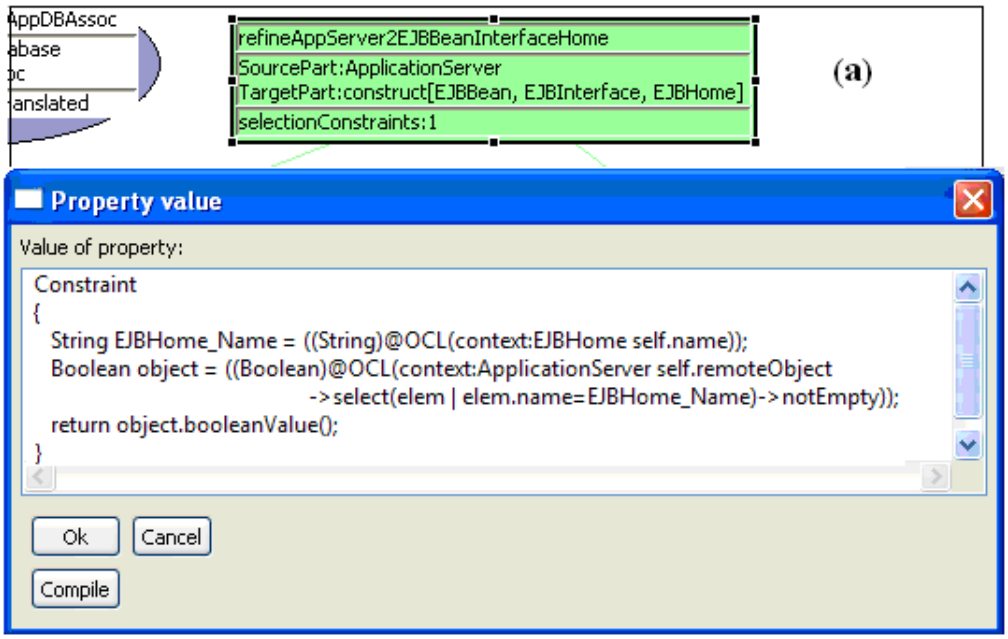


Figure 11.3. The *selectionConstraints* property sheet of StructureRefinement refineAppServer2EJBBeanInterfaceHome

A StructureMapping may trigger events that will influence the established Interconnection Relationship. The typical events include the change of property value and the removal of the interconnected elements.

The MaramaCRelation tool retrieves possible events and lists them in the property sheet of *behaviourDescription*. Figure 11.4 shows a sample StructureMapping (a), its selection constraint (b), and part of the events triggered by the StructureMapping (c). The structure of a *behaviourDescription* and how to generate a *behaviourDescription* have been explained in chapters 9 and 10 respectively.

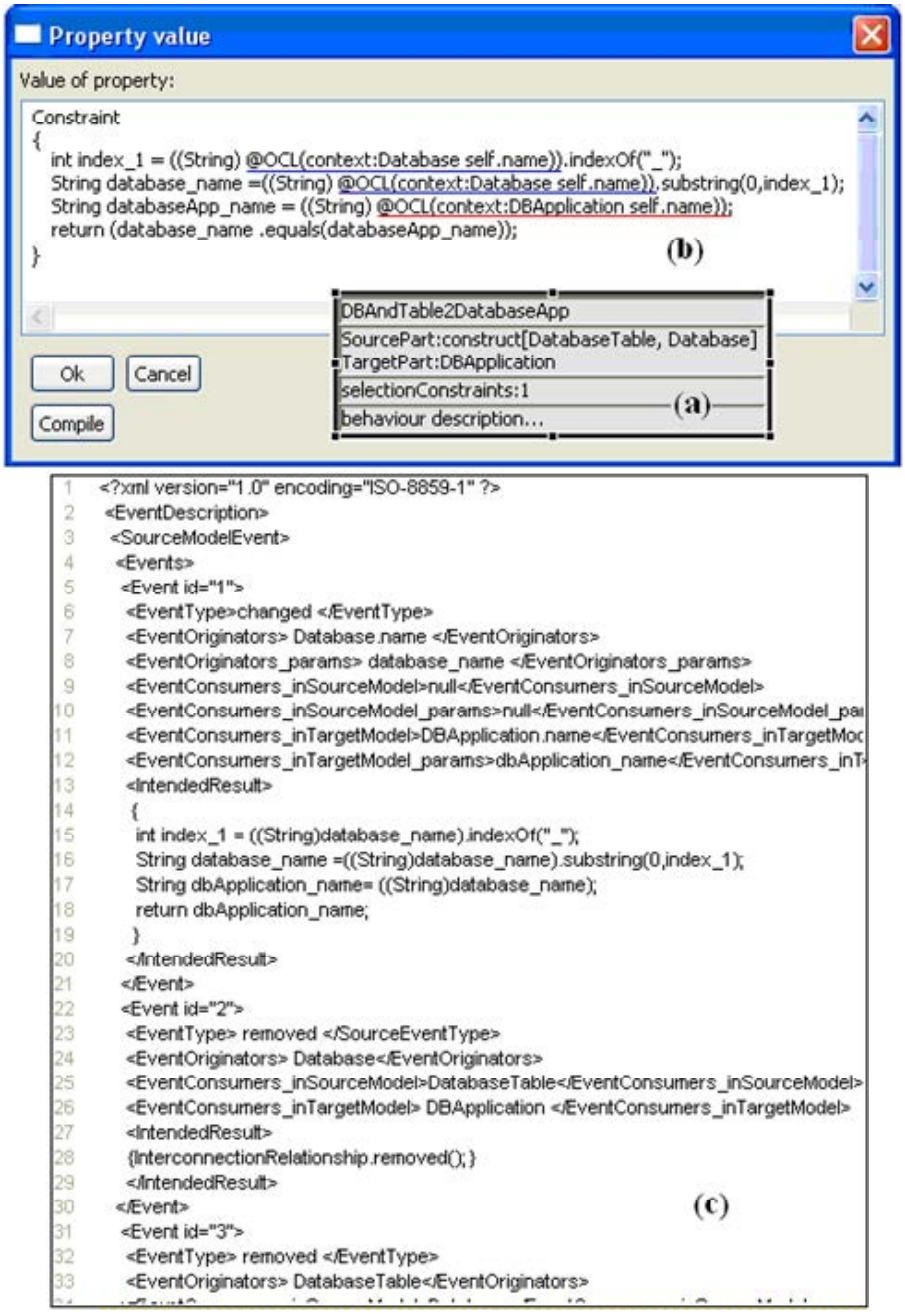


Figure 11.4. The *behaviourDescription* property sheet of StructureMapping DBAndTable2DatabaseApp

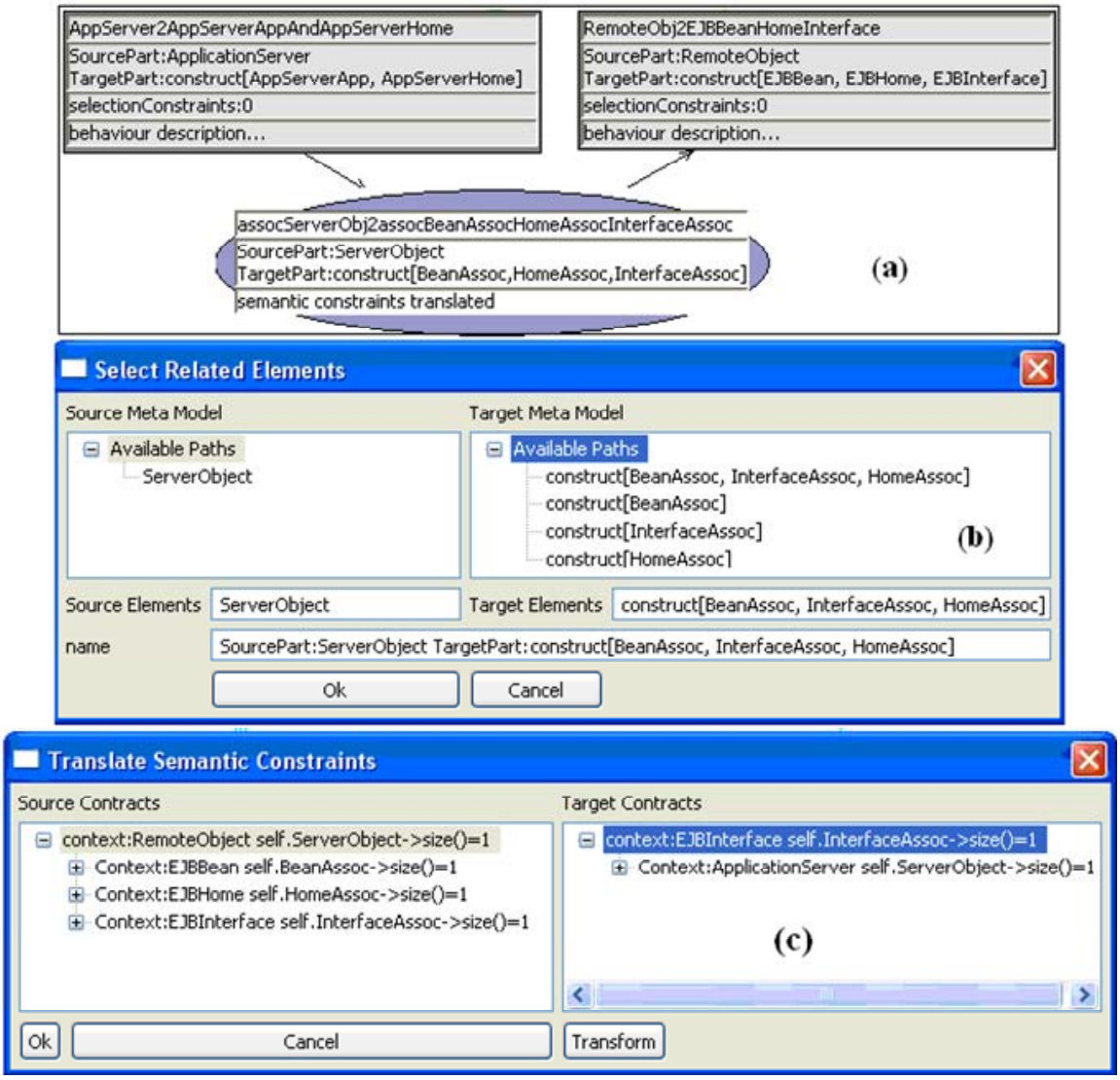


Figure 11.5. The *associationMapping* and *semanticTransformation* property sheets of SemanticAssociation assocServerObj2assocBeanAssocHomeAssocInterfaceAssoc

A SemanticAssociation represents what source and target meta-model associations can be maintained during interconnecting. In Figure 11.5(a), when the two StructureMappings are established, it is implied that there are source and target meta-model associations involved, and their mapping should be detected and explicitly represented. Figure 11.5(a) illustrates a sample SemanticAssociation associating two StructureMappings. Figure 11.5(b) illustrates that the available source and target meta-model paths are

automatically calculated and listed to allow users to set up the value of *associationMapping*. The source part is ServerObject – a path between the source parts of the two associated StructureMappings in the source meta-model. The target part is construct [BeanAssoc, InterfaceAssoc, HomeAssoc] – a path between the target parts of the two associated StructureMappings in the target meta-model.

Figure 11.5(c) illustrates that a source meta-model translatable semantic constraints is retrieved, translated, and listed in the property sheet of the *semanticTranslation* of the SemanticAssociation. The source meta-model semantic constraint in Figure 11.5(c) is sensible as if an ApplicationServer is removed its RemoteObjects must be deleted or re-hosted as well. This semantic constraint is translated into sensible EJBUML meta-model semantic constraints, which specify that each of EJBBean, EJBHome, and EJBInterface needs to be hosted by one AppServerApp, i.e. when the hosting AppServerApp server is removed the EJBBean, EJBHome, and EJBInterface must be deleted or re-hosted as well. Similarly, a target meta-model translatable semantic constraint (tree root, right column) can be translated into a sensible source meta-model semantic constraint (the child of the root, right column).

11.1.2 Generating search conditions and behavior synchronization coordinator

The MaramaMTE-EJBUML CRelation model not only records the rational behind the interconnection of the MaramaMTE and EJBUM meta-models, but also generates search conditions to establish traceability and synchronize views and behaviors across architecture models and EJBUML design models.

Each StructureMapping, StructureRefinement, and SemanticAssociation of a CRelation model generates one java class. The generated java classes are search conditions, which help to find out qualified target elements for each interested source model element by using the selection constraints specified in the CRelation model. Figure 11.6(a) shows the package of java files generated from the MaramaMTE-EJBUML CRelation model, including three StructureMapping classes, and two SemanticAssociation classes, and one StructureRefinement class. Figure 11.6(b) shows part of the java file (generated from the StructureMapping DBAndTable2DBApplication (refer to Figure 11.4(a)). The java file implements methods getTargetCandidates, getTargetType, isSourceModelElements (refer to section 10.5.1), and so on. Figure 11.6(c) illustrates the implementation of a sample selection constraint of the

StructureMapping. When generating the java files, the OCL queries are translated into tool-API-dependent (Marama meta-tool) code with the help of the Eclipse OCL framework.

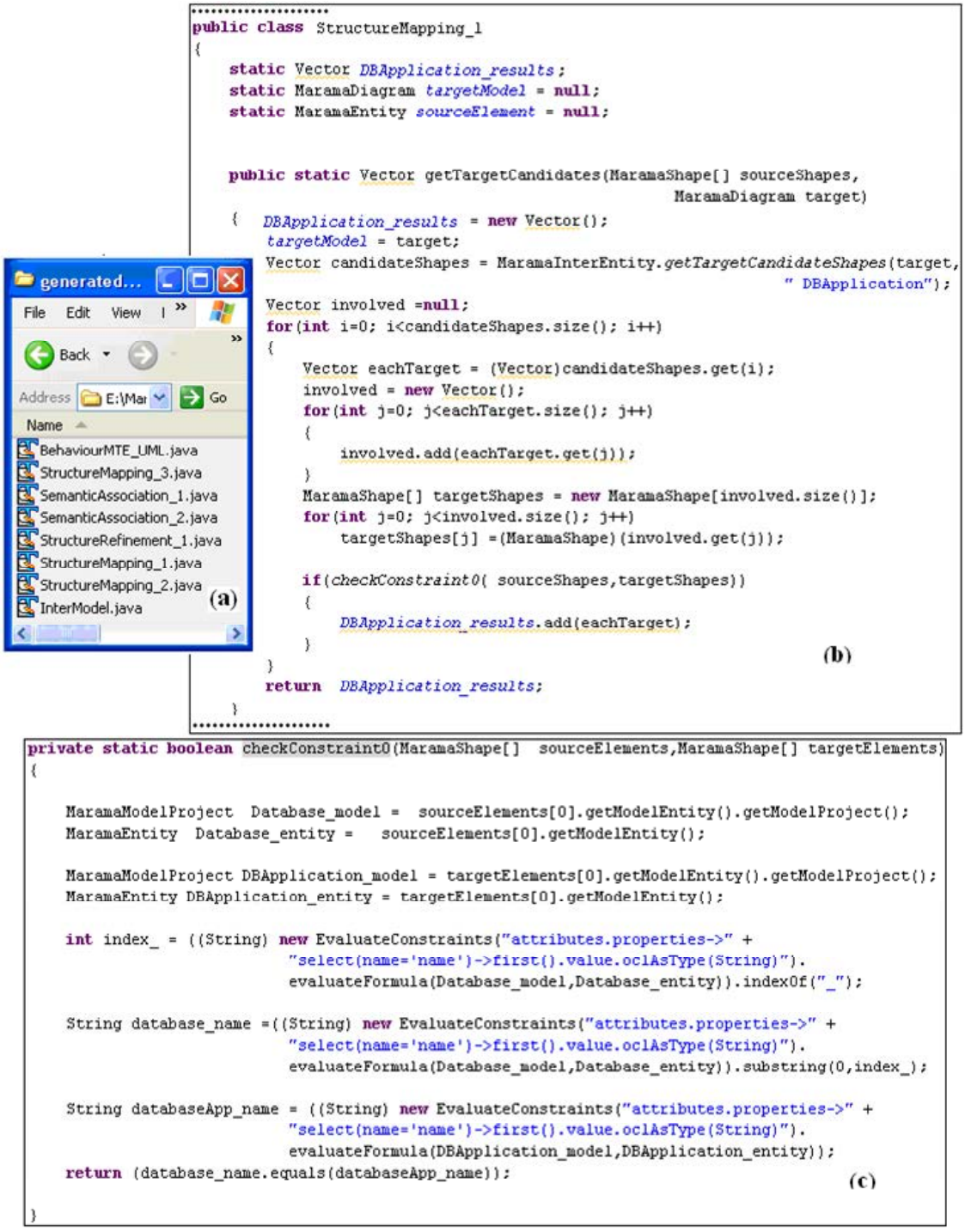


Figure 11.6. Java search conditions generated from the MaramaMTE-EJBUML CRelation model

Building full-blown behaviour synchronization mechanisms is not intended in the MaramaCRelation approach. The well-structured behaviour descriptions of the CRelation model are planned to feed third party technologies (e.g. event modelling technologies such as Kaitiaki (Liu et al, 2007)). However, the MaramaCRelation tool still provides a simple java behavior synchronizer to showcase the use of the behaviour description information. The java synchronizer is a java class generated from the CRelation model. It helps to maintain behavior synchronization across models by passing around simple synchronization messages. Figure 11.7 shows part of the generated BehaviorMTE_UML.java. This synchronizer processes the events triggered by the three StructureMappings of the MaramaMTE-EJBUML CRelation model (refer to Figure 11.1(a)). For example, lines 7, 9, and 11 process three events recorded in the *behaviorDescription* of Figure 11.4(c). The synchronizer generates messages based on the intended results of the *behaviorDescription*; and passes the synchronization messages to the target model.

```
1  public class BehaviourMTE_UML implements Adapter
2  {
3
4    public void notifyChanged(Notification notification)
5    {
6      ................
7      processEvent_1_StructureMapping_1(notification);
8
9      processEvent_2_StructureMapping_1(notification);
10
11     processEvent_3_StructureMapping_1(notification);
12
13     processEvent_1_StructureMapping_2(notification);
14
15     processEvent_1_StructureMapping_3(notification);
16
17     ................
18   }
19   ................
20 }
```

Figure 11.7. The synchronizer generated from the MaramaMTE-EJBUML CRelation model

11.1.3 Interconnecting process

Figure 11.8 and Figure 11.9 show how to interconnect the Pet Store MaramaMTE model with the Pet Store EJBUML model. Figure 11.8 shows how to assign a target model candidate to a source model element. Two constructs of target model elements (2) are qualified candidates for the source model element *CustomerManager* (1). Users can assign a target model candidate to the source model element. The assignation must be done on other interested source model elements. Once the interconnecting

process (refer to section 10.5) is completed, the traceability and behaviour synchronisation between the two models are established. If the change of property values or the removal of model elements happens to the source model, the interconnected target model element(s) will receive suggestive messages to take actions to maintain the validity of the Interconnection Relationships.

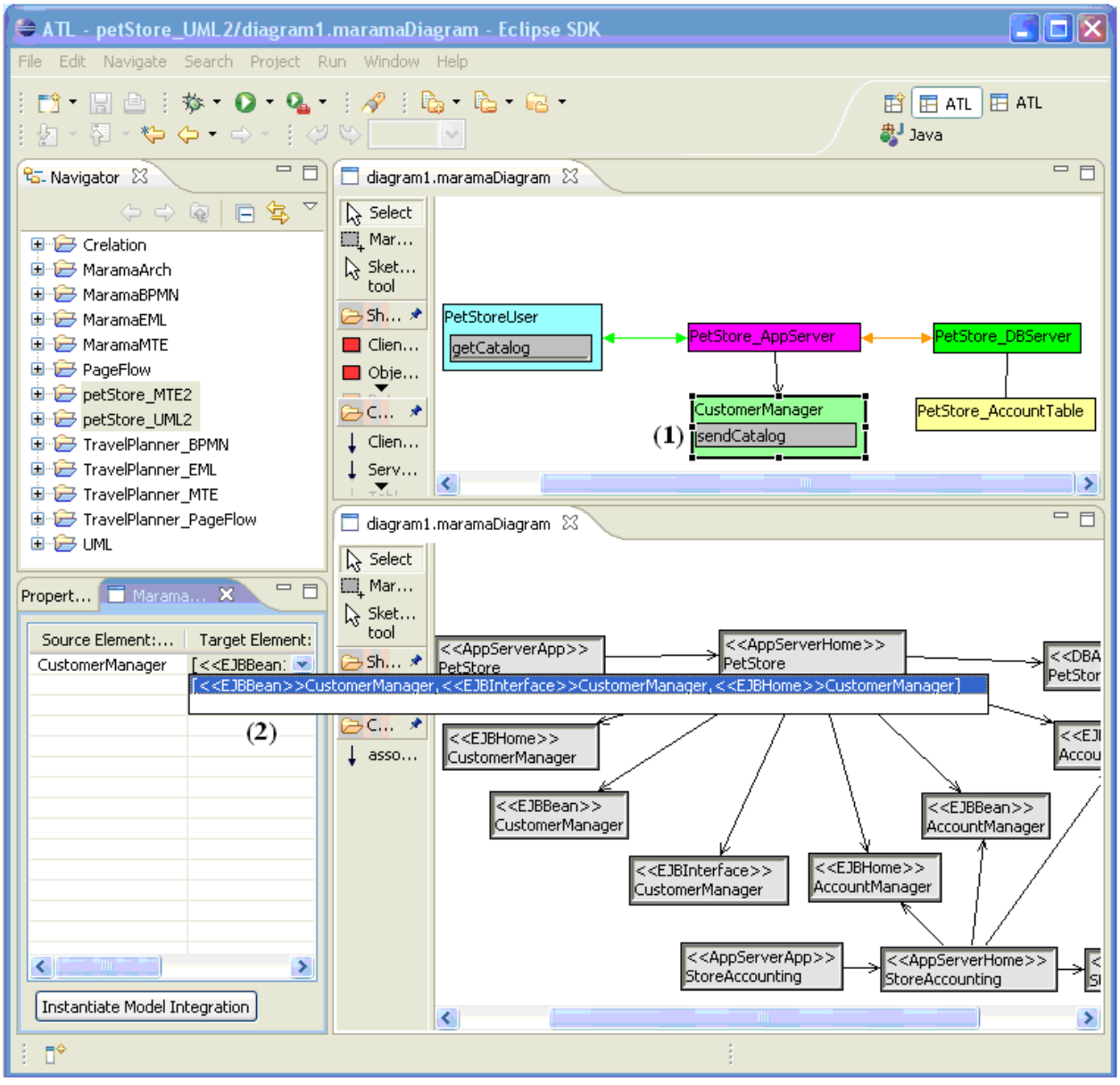


Figure 11.8. Interconnecting the Pet Store MaramaMTE model with the Pet Store EJBUML model

In Figure 11.9, the removal of the *CustomerManager* in the source model (disappeared from (1)) will definitely influence the validity of its Interconnection Relationship with its correspondent target model elements. When the *CustomerManager* is removed, the interconnected target model elements receive synchronization message "<<EJBBean>> CustomerManager, <<EJBHome>> CustomerManager, <<EJBInterface>> CustomerManager's Intercon relationship should be removed" (2). So far, the MaramaCRelation approach has shown the potential in providing structured support for maintaining behaviour synchronization across different domain-specific models, which will be explored in the future.

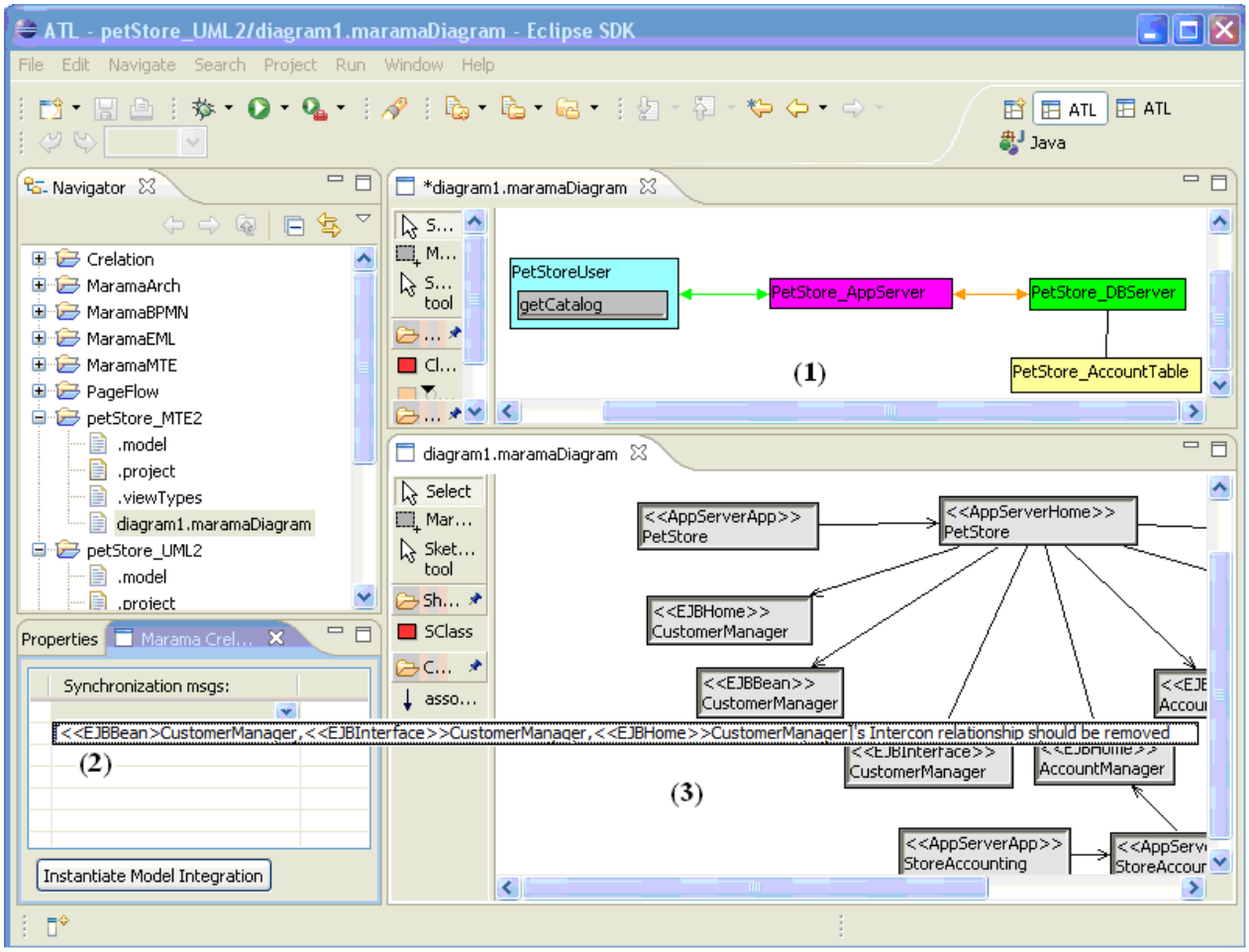


Figure 11.9. Using java synchronizer to synchronize model behaviours

11.1.4 Brief Summary

The interconnection of the Pet Store MaramaMTE model and Pet Store EJBUML design model has been used as example usage to introduce the CRelation model and the MaramaCRelation tool in chapters 8, 9, and 10. Here, this case study reviews the main features of the CRelation model and the MaramaCRelation tool. It demonstrates how the StructureMappings capture the main concerns of an intended interconnection, and how the SemanticAssociations explicitly represent the association information that is often implied but ignored in the traditional model transformation technologies. The case study shows how the MaramaMTE-EJBUML CRelation model can generate search conditions and synchronizer to maintain the traceability and behaviour synchronization across the interconnected models.

11.2 Case study 2: Interconnecting the Travel Planner EML model, the Travel Planner BPMN model, the Travel MaramaMTE model, and the Travel Planner Form Chart model

An online Travel Planner is a travel planning application that provides travel item search (flights, cars, hotel rooms etc), booking, payment, event scheduling and itinerary management. Various domain-specific models may be used to address concerns for different purposes at different stages of the development of the Travel Planner system. Users may use BPMN (BPMN, 2004) to model business processes, EML (Li et al, 2007) to model business service trees; MaramaMTE to model tiered software architecture; and Form Chart to model client behavior. Users use certain domain knowledge to address specific issues individually, but also want to combine the strength of all models. In this case study, three CRelation models are used to combine the strength of four different domain-specific modeling technologies when modeling the complicated Travel Planner project.

11.2.1 Interconnecting the Travel Planner EML model with the Travel Planner BPMN model

Enterprise Modeling Language (EML) (Li et al, 2007) specification provides a graphical notation for expressing business processes in a Tree Overlay based diagram.

BPMN specification creates graphical models of business process operations. BPMN graphical notations are based on a flowcharting technique. A Business Process Model is a network of graphical objects consisting of activities (i.e., work) and the flow controls that define their order of performance (BPMN,

2004). EML and BPMN are two different models to specify business processes with different modeling angles, but they share a lot of similarities. An EML-BPMN CRelation model is needed to capture the similarities and specify the rational for the interconnection of the two meta-models.

11.2.1.1 *The EML meta-model*

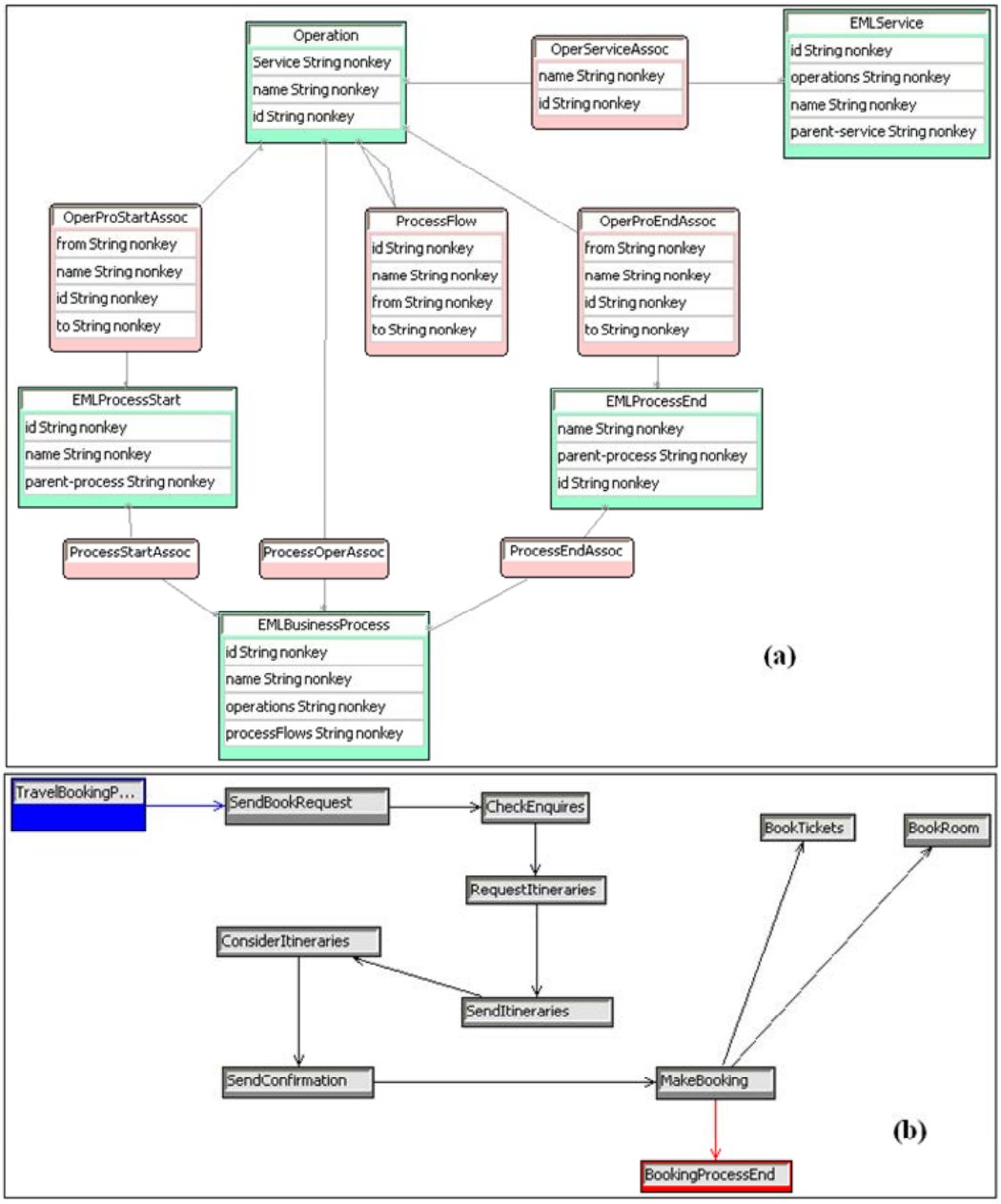


Figure 11.10. (a): the MOF-based EML meta-model (b): the EML Travel Planner model

The main features of EML include: 1) modeling complex business architectures as service trees; 2) modeling business processes as process overlay sequences on the service trees (Li et al, 2007). EML does not have a diagrammatic meta-model, but it is easy to generalize one from the explicit description of the EML syntax. Figure 11.10(a) presents a part of the generalized MOF-based EML meta-model. This part of EML meta-model is focused on the business process overlay in the EML service tree, but ignores the abstractions for the service tree itself.

In the EML meta-model, the five entities (green rectangles) are used to model an EML business process, and they are: EMLProcessStart, Operation, EMLProcessEnd, EMLService, and EMLBusinessProcess. To simplify issues, each of the entities contains only small part of the properties in the original EML specification.

A simple EML model in Figure 11.10(b) explains how the meta-model elements are used to model a simple travel planning business process. Figure 11.10(b) does not show the original EML business service tree, but focuses on the EML business process overlay. The "TravelBookingProcess" (blue rectangle, typed as EMLProcessStart) starts the business process and goes through a sequential Operations of *SendBookRequest*, *CheckEnquires, RequestItineraries, ConsidersItineraries*, and *SendConfirmInformation.* The Operation *MakeBooking* requires two concurrent Operations of *BookTickets* and *BookRoom.* The process ends with the *BookingProcessEnd* (red rectangle, typed as EMLProcessEnd).

11.2.1.2 *The BPMN meta-model*

Alternative BPMN meta-models have been generalized from the explicit description of BPMN syntax ((BPMN, 2004)). Figure 11.11(a) illustrates a partial MOF-based BPMN meta-model. The five entities (green rectangles) are used to model a complete BPMN business process, including EventStart, Activity, EventEnd, Pool, and BPMNProcess. To simplify issues, in Figure 11.11(a), each of the entities contains only small part of the properties in the original BPMN specification.

A simple BPMN model in Figure 11.11(b) explains how the meta-model elements are used to model a simple travel planning business process. The "TravelBookingProcess" (typed as EventStart) starts the

business process and goes through sequential Activities of *Send BookRequest*, *CheckEnquires, RequestItineraries, ConsidersItineraries*, and *SendConfirmInformation"*. The Activity *"MakesBooking"* requires two concurrent Activities of *BookTickets* and *BookRoom*. The process ends with "BookingProcessEnd" (typed as EventEnd).

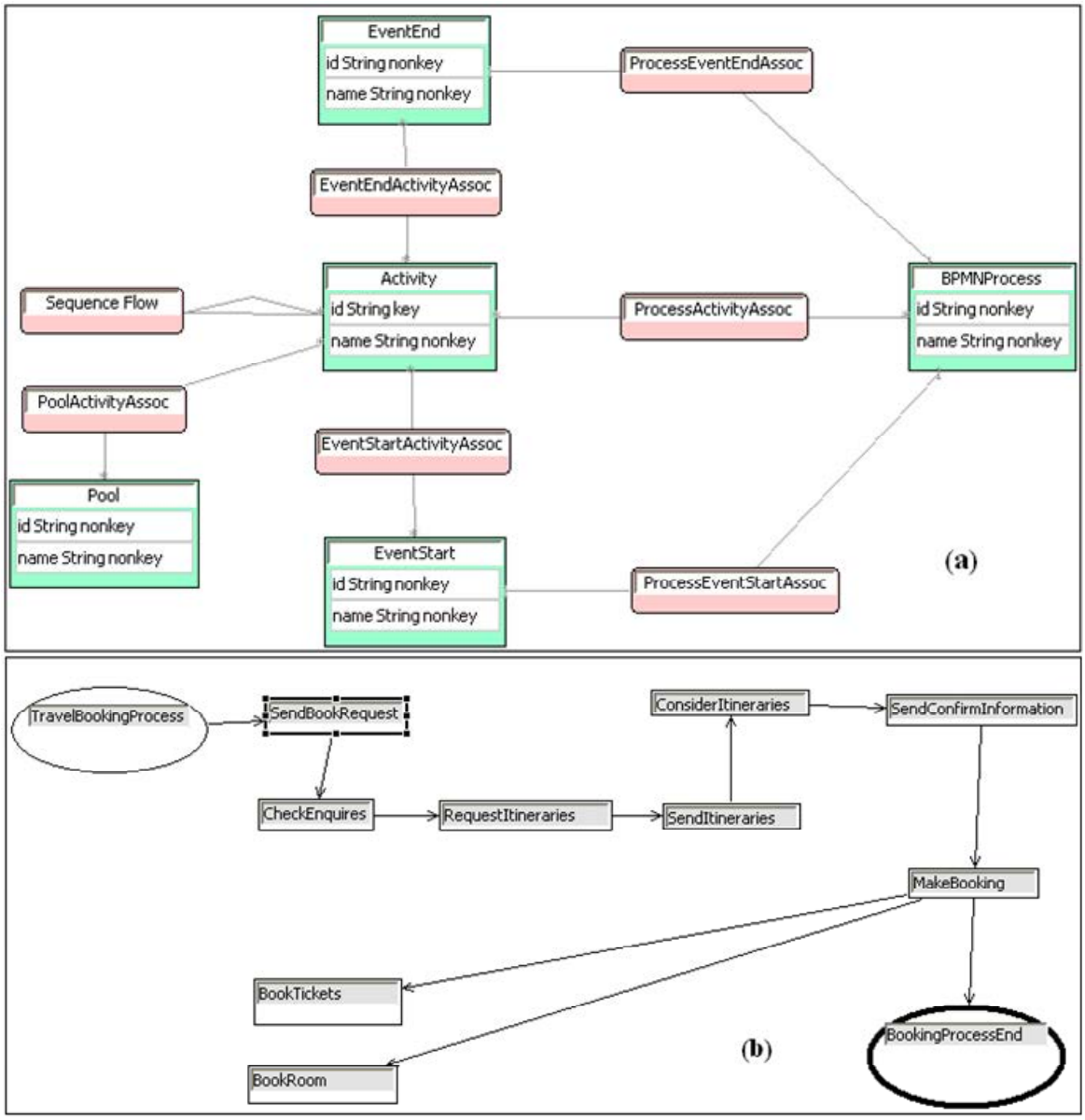


Figure 11.11. (a): the MOF-based BPMN meta-model; (b): the BPMN Travel Planner model

11.2.1.3 *The EML-BPMN CRelation model*

The EML-BPMN CRelation model (Figure 11.12(a)) is developed to capture the semantics shared by the EML meta-model (specifically the part of business process overlay of EML) and the interested part of BPMN meta-model. The five StructureMappings specify the main similarities between EML and BPMN. The highlighted StructureMapping "processEnd2eventEnd" represents the EML's EMLProcessEnd abstraction type is similar to the BPMN's EventEnd abstraction type, and the source and target elements can be interconnected and viewed as the same semantics represented differently in the different models. The SemanticAssociations explicitly represent the source and target model associations that are implied when the associated StructureMappings are set up.

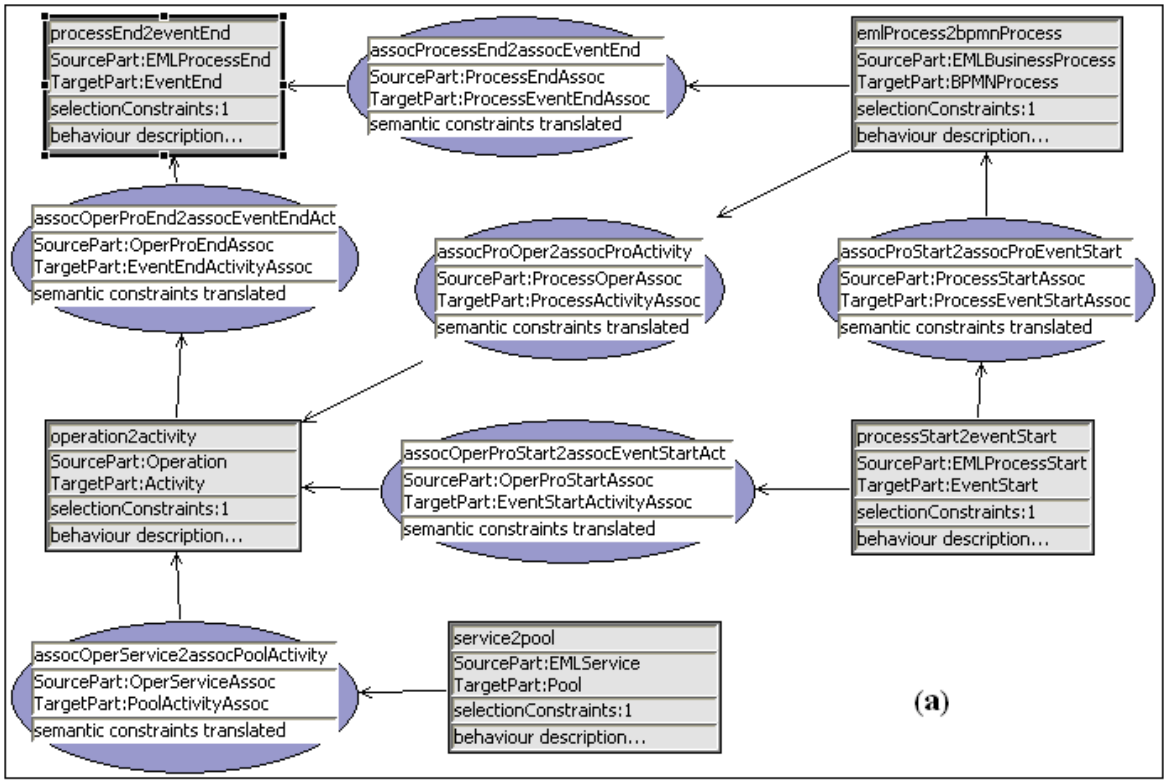


Figure 11.12. The EML-BPMN CRelation model

The *selectionConstraints* property of StructureMapping "processEnd2eventEnd" (the highlighted element in Figure 11.12(a)) is illustrated in Figure 11.13(b). The constraint requires that an EMLProcessEnd (in an EML model, e.g. the Travel Planner EML model) and an EventEnd (in a BPMN model, e.g. the Travel Planner BPMN model) must have the same *name* value before they can be

interconnected. Figure 11.13(c) shows the *behaviorDescription* property triggered by the StructureMapping "processEnd2eventEnd". The StructureMapping triggers two source model events: 1) the change of the *name* value of the EMLProcessEnd; 2) the removal of the EMLProcessEnd. The information of the triggered events is organized in the *behaviorDescription* and will help to generate a java behavior synchronizer.

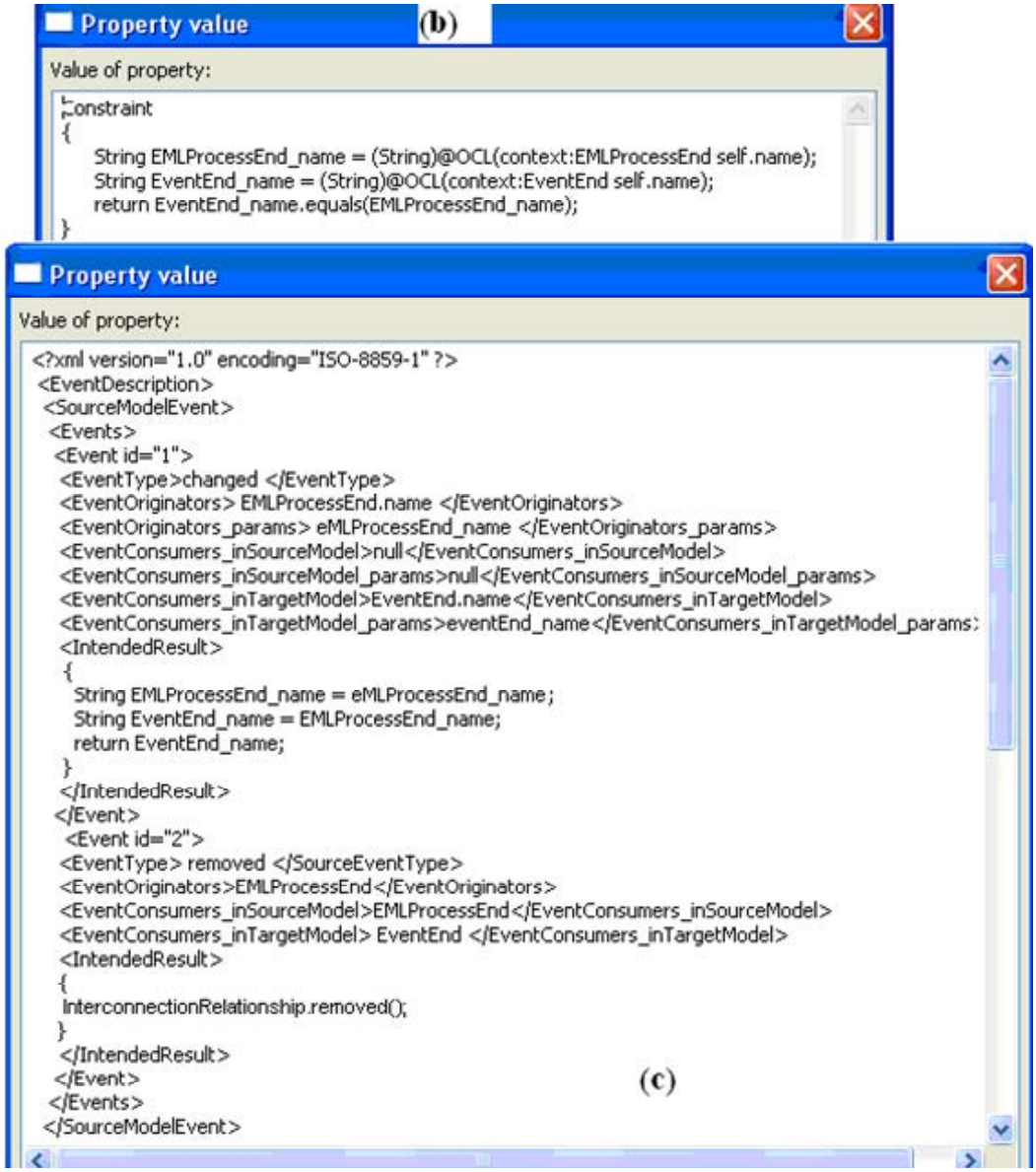


Figure 11.13. The *selectionConstraints* and the partially generated *behaviorDescription* property sheets of StructureMapping "processEnd2eventEnd"

Figure 11.14 illustrates the *semanticTranslation* property sheet of SemanticAssociation "assocOperProEnd2asswocEventEndAct". The source model semantic constraint in Figure 11.14 (tree root, left column) means an EMLProcessEnd must be connected to an Operation. This semantic constraint is translated into the sensible BPMN model semantic constraint (child of the tree root, left column), which specifies that, in a BPMN model, an EventEnd must be connected to an Activity. Similarly, a target meta-model translatable semantic constraint (tree root, right column) can be translated into a sensible source meta-model semantic constraint (the child of the root, right column).

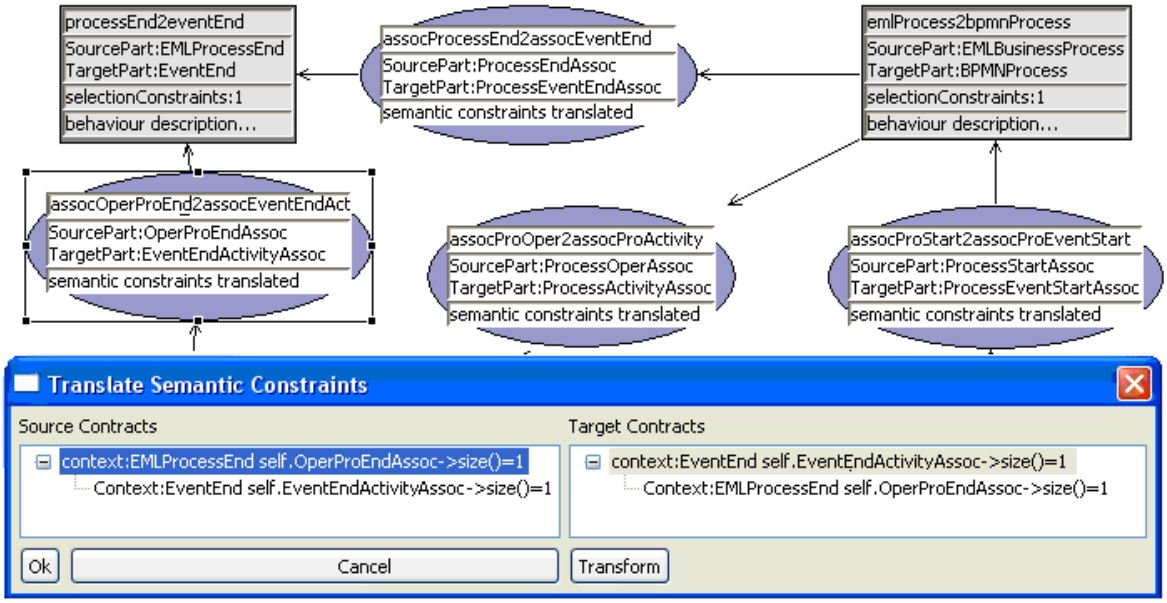


Figure 11.14. The *semanticTranslation* property sheet of SemanticAssociation "assocOperProEnd2assocEventEndAct"

11.2.1.4 *Generating search conditions and the behavior synchronizer*

The EML-BPMN CRelation model records the rational behind the interconnection of the EML and the BPMN meta-models. The model can also generate search conditions to help to establish traceability, and synchronize views and behaviors across the models.

Figure 11.15(a) shows the folder of java files generated from the EML-BPMN CRelation model, including 5 StructureMapping classes and 6 SemanticAssociation classes. Figure 11.15(b) shows part of the generated StructureMapping_1.java file (generated from the StructureMapping

"processEnd2eventEnd" in Figure 11.12(a)). The java file implements methods getTargetCandidates, getTargetType, isSourceModelElements (refer to section 10.5.1), and so on. The selection constraints of the StructureMappings go to the checkConstraint methods in their respective classes. Figure 11.15(c) illustrates the implementation of a sample selection constraint (refer to Figure 11.13(b)). When generating the java files, the OCL queries are translated into tool-API-dependent (Marama meta-tool-dependent) code with the help of the Eclipse OCL framework.

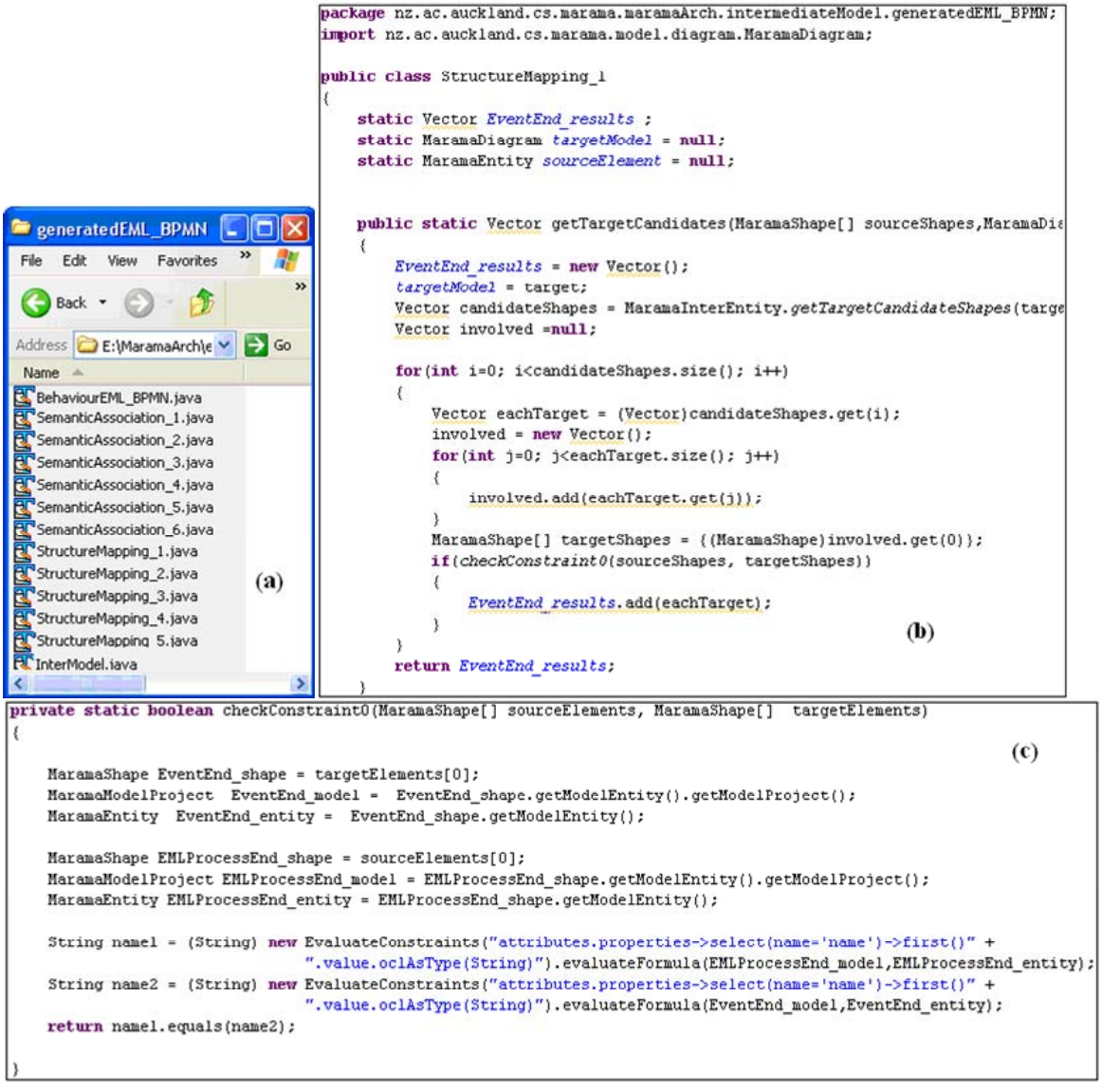


Figure 11.15. Java search conditions generated from the EML-BPMN CRelation model

A java behavior synchronizer (the BehaviourEML_BPMN.java in Figure 11.15(a)) is also generated. Figure 11.16 shows part of the generated BehaviorEML_BPMN.java. This synchronizer processes the events of 5 StructureMappings of the EML-BPMN CRelation model (refer to Figure 11.12(a)). For example, lines 9 and 11 process two events recorded in the *behaviorDescription* of Figure 11.13(b). The synchronization messages are generated based on the intended results the tool users manually program inside the *behaviorDescription*.

```
1  ..................
2   public class BehaviourEML_BPMN implements Adapter
3   {
4
5    public void notifyChanged(Notification notification)
6    {
7
8     ........................
9      processEvent_1_StructureMapping_1(notification);
10     processEvent_2_StructureMapping_1(notification);
11
12     processEvent_1_StructureMapping_2(notification);
13     processEvent_2_StructureMapping_2(notification);
14
15     processEvent_1_StructureMapping_3(notification);
16     processEvent_2_StructureMapping_3(notification);
17
18     processEvent_1_StructureMapping_4(notification);
19     processEvent_2_StructureMapping_4(notification);
20
21     processEvent_1_StructureMapping_5(notification);
22     processEvent_2_StructureMapping_5(notification);
23      ......................
24    }
25    ..................
26  }
```

Figure 11.16. The synchronizer generated from the EML-BPMN CRelation model

11.2.1.5 *Interconnecting process*

Figure 11.17 shows how to interconnect the Travel Booking Process EMLmodel (Figure 11.10(b)) and the Travel Booking Process BPMN model (Figure 11.11(b)). In Figure 11.17, the "SendBookRequest" (1) (typed as Operation of EML) has only one qualified target model candidate (2) and is then interconnected with the candidate "SendBookRequest" (3) (typed as Activity of BPMN). Once the interconnecting process is completed, the traceability and behaviour synchronisation between the two models are established. If the change of property values or the removal of model elements happens to the source model, the interconnected target model elements will receive suggestive messages to take actions to maintain the validity of the Interconnection Relationships

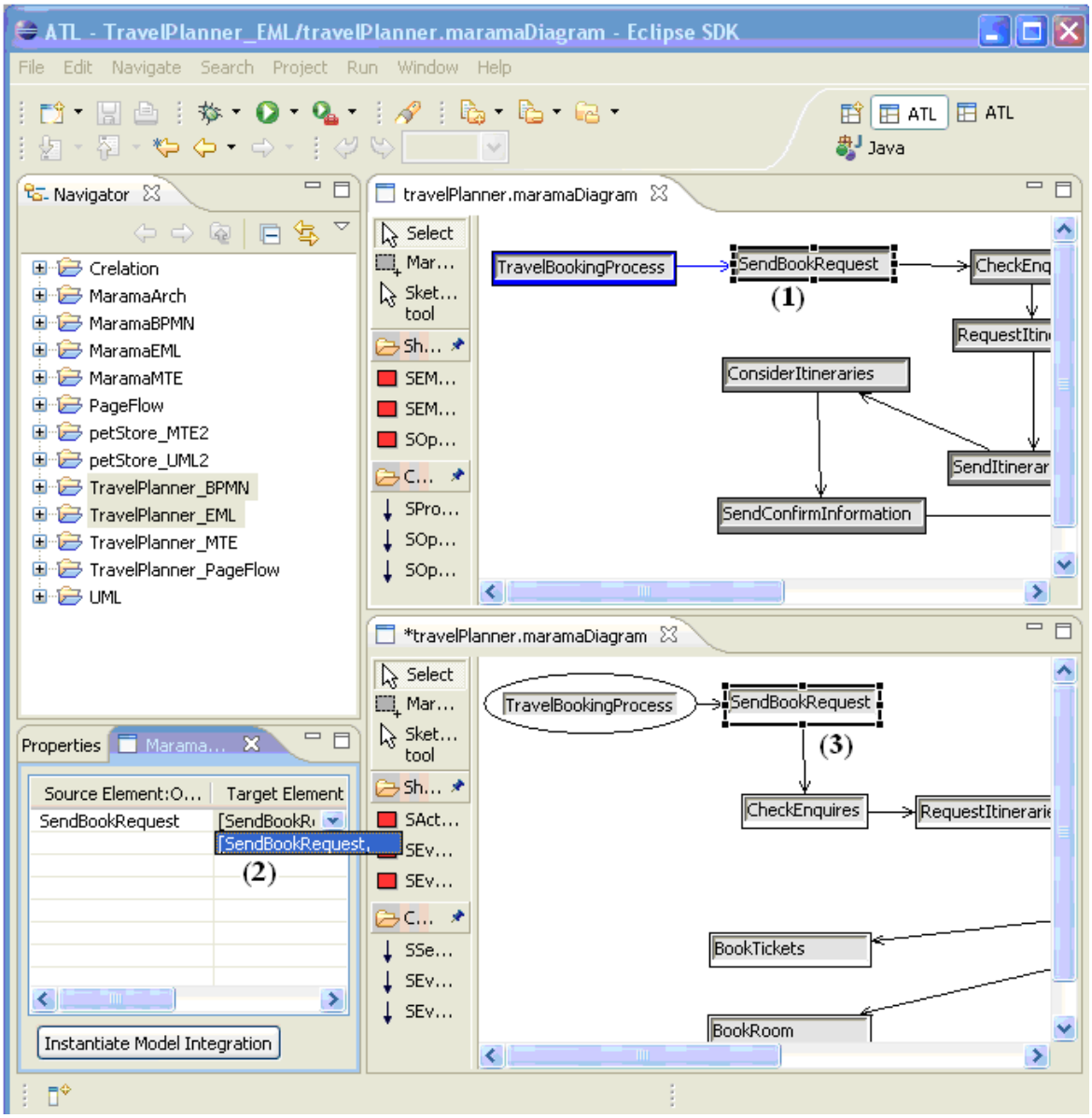


Figure 11.17. Interconnecting the EML Travel Planner model with the BPMN Travel Planner model

Figure 11.18 shows the change of source model element "SendBookRequest" will influence the validity of the Interconnection Relationship between the two models. When the name of "SendBookRequest" is changed to "SendBookRequest_2" (1), the interconnected target model element receives synchronization message "SendBookRequest.name changed to SendBookRequest_2" (2). In order to maintain the

validity of the Interconnection Relationship, the name of the “SendBookRequest” Activity in the target model should also be changed to “SendBookRequest_2” (3). This example has again shown the potential in providing structured support for maintaining behaviour synchronization across different domain-specific models.

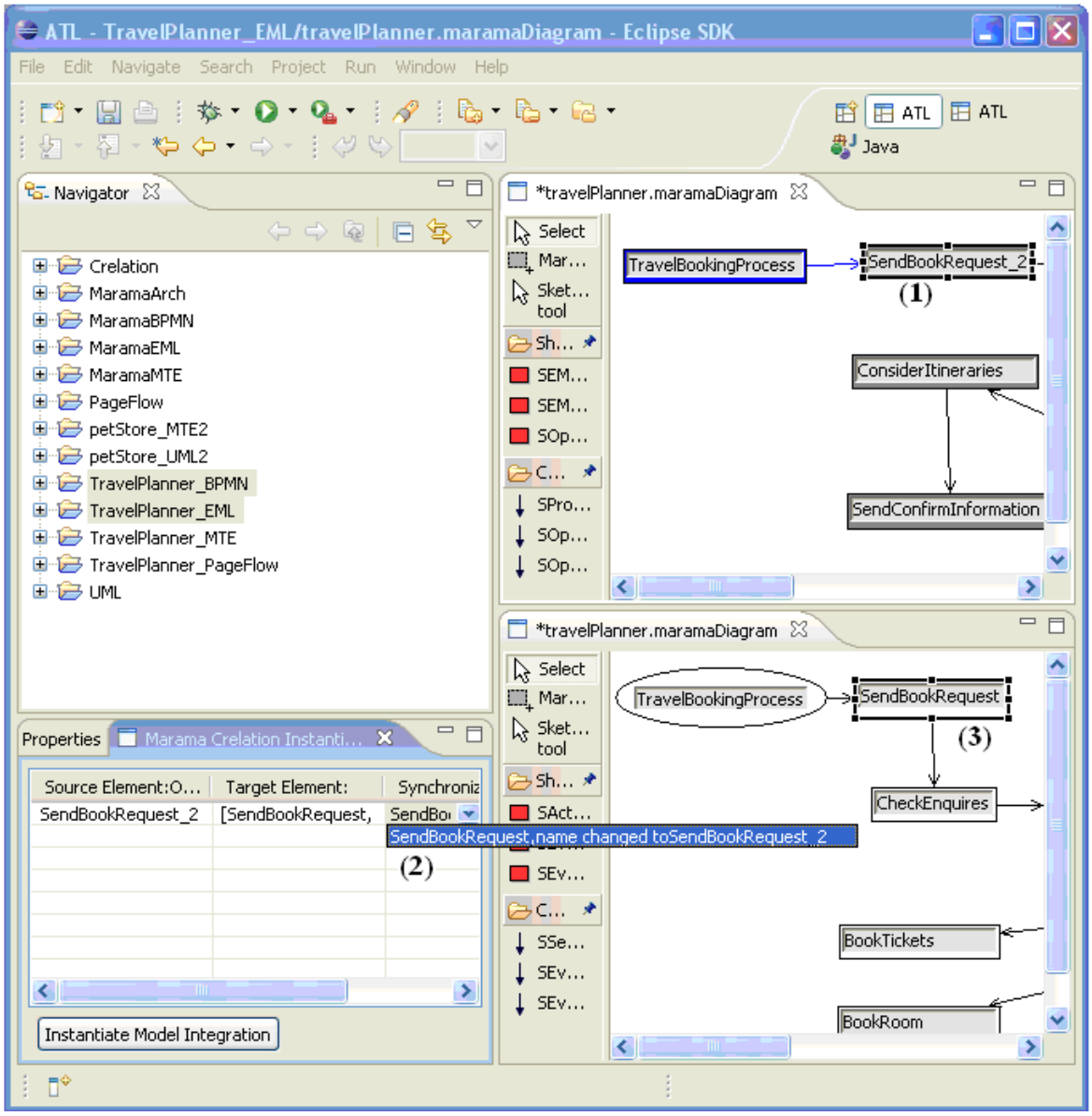


Figure 11.18. Behaviour synchronization between the interconnected models

11.2.2 Interconnecting the Travel Planner BPMN model with the Travel Planner MaramaMTE model

The CRelation model can not only interconnect two similar domain-specific meta-models like EML and BPMN, but also work for meta-models that are not seemingly similar. For example, the BPMN model works at business analysis level, while the MaramaMTE architecture model works at a lower level and closer to software system design and implementation. For these not seemingly close meta-models, it is especially important to use the CRelation model to specify the rational behind the intended interconnection.

11.2.2.1 *The BPMN-MaramaMTE CRelation model and its entities*

Both the MaramaMTE and BPMN meta-models have been introduced before (refer to section 8.4 and section 11.2.1.2). Figure 11.19(a) illustrates the BPMN-MaramaMTE CRelation model. The StructureMapping "eventStart2appClientRequest" (3) specifies that the EventStart of BPMN can be interconnected with the construct of ApplicationClient and Request of MaramaMTE. The Activity of BPMN can either be interconnected to the construct of ApplicationServer, RemoteObject, and Service of MaramaMTE (via StructureMapping "activity2serverObjService" (7)), or the construct of ApplicationClient and Request of MaramaMTE (via StructureMapping "activity2appClientRequest" (4)). The EventEnd of BPMN shares the similar semantics with the construct of ApplicationServer, RemoteObject, and Service of MaramaMTE (via StructureMapping "eventEnd2ServerObjService" (5)). SemanticAssociations "assocSeqFlow2clientRequest" (1) and "assocSeqFlow2assocObjService" (2) are slightly different from other "ordinary" SemanticAssociations, because they associate the same StructureMappings respectively. For example, SemanticAssociation "assocSeqFlow2clientRequest" (1) associates StructureMapping "activity2appClientRequest" with itself.

Figure 11.19(b) presents the *selectionConstraints* property of StructureMapping "eventStart2appClientRequest" ((3), Figure 11.19(a)). The constraint specifies that the *name* of an EventStart and the *name* of a Request must be the same before the EventStart and the Request can be interconnected. This StructureMapping also gets a selection constraint from its SelectionRefinement "refineAppclientRequest" ((6), Figure 11.19(a)) whose selection constraint is shown in Figure 11.19(c). The constraint of the SelectionRefinement means that in the construct of ApplicationClient and Request,

the two elements must be associated through a ClientRequest association (refer to the MaramaMTE meta-model in Figure 9.1(a)).

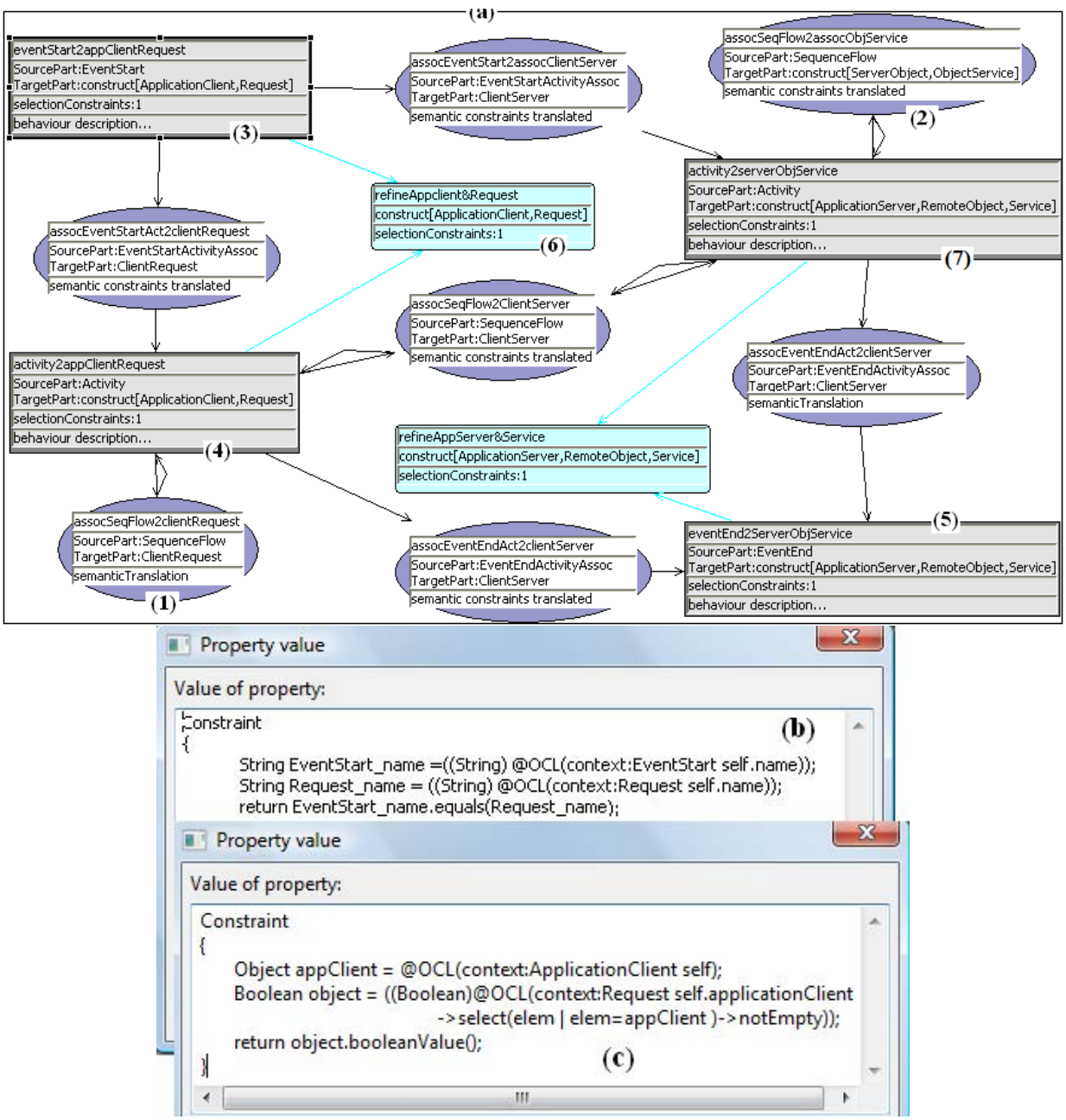


Figure 11.19. (a): the BPMN-MaramaMTE CRelation model; (b) the *selectionConstraints* of StructureMapping "eventStart2appClientRequest"; (c) the *selectionConstraints* of SelectionRefinement "refineAppclient&Request"

The events triggered by StructureMapping "eventStart2appClientRequest" ((3), Figure 11.19(a))) are organized in the *behaviorDescription* illustrated in Figure 11.20. In the *behaviorDescription*, the StructureMapping triggers two source model events that will influence the established Interconnection Relationship: 1) the change of the name of the EventStart element; 2) the removal of the EventStart element. At this stage, the *selectionConstraints* brought by the SelectionRefinement "refineAppclient&Request" do not contribute to the *behaviorDescription* of the parent StructureMapping "eventStart2appClientRequest"; but their influence on the *behaviorDescription* of the parent StructureMappings will be researched in the future. The information of the triggered events is organized in the *behaviorDescription* and will help to generate a java behavior synchronizer.

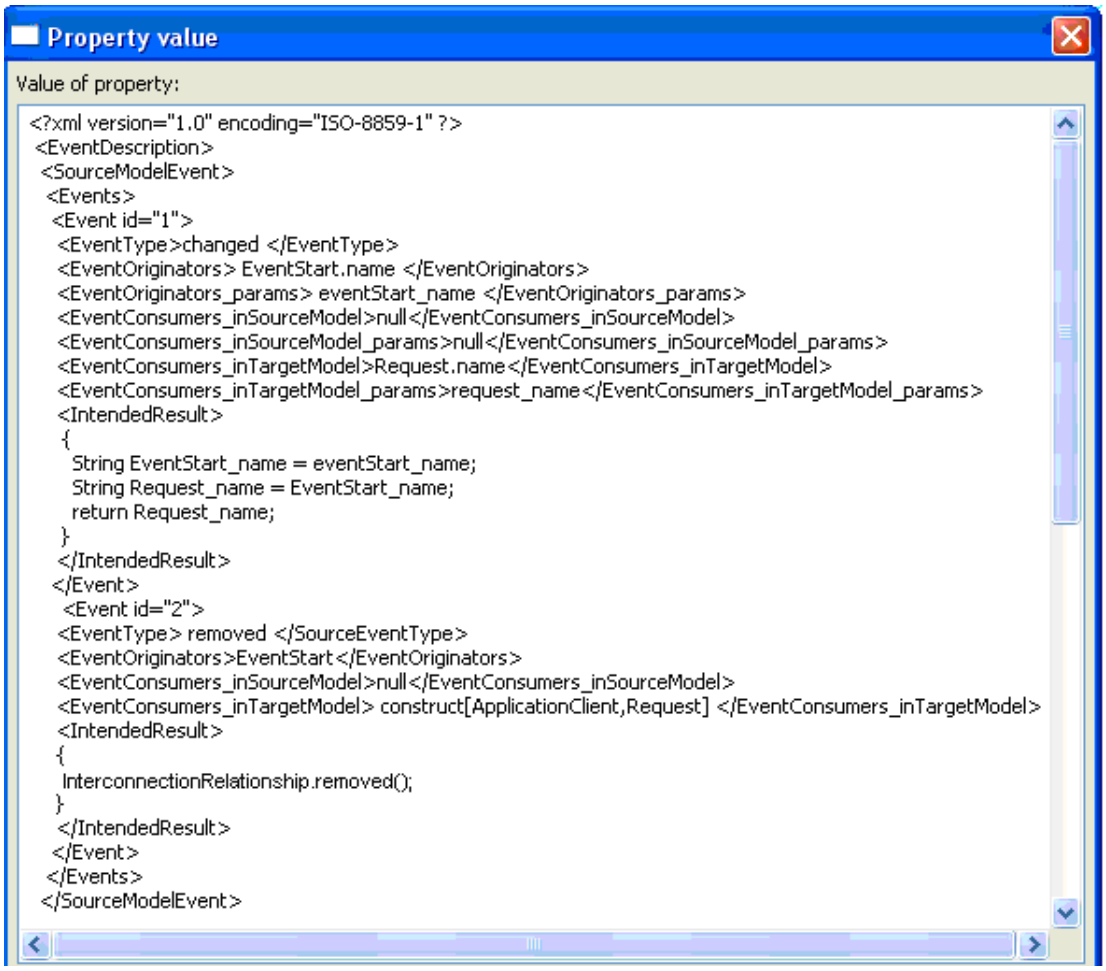


Figure 11.20. The *behaviorDescription* of StructureMapping "eventStart2appClientRequest"

Figure 11.21 illustrates the *semanticTranslation* property sheet of SemanticAssociation "assocEventStart2assocClientServer". The source meta-model semantic constraint (tree root, left column) means that in a BPMN model, an EventStart must be connected to an Activity. This semantic constraint is translated into a sensible MaramaMTE meta-model semantic constraint that specifies that, in a MaramaMTE model, an ApplicationClient must be connected to an ApplicationServer (child, left column). Similarly, a target meta-model translatable semantic constraint (tree root, right column) can be translated into a sensible source meta-model semantic constraint (the child of the root, right column).

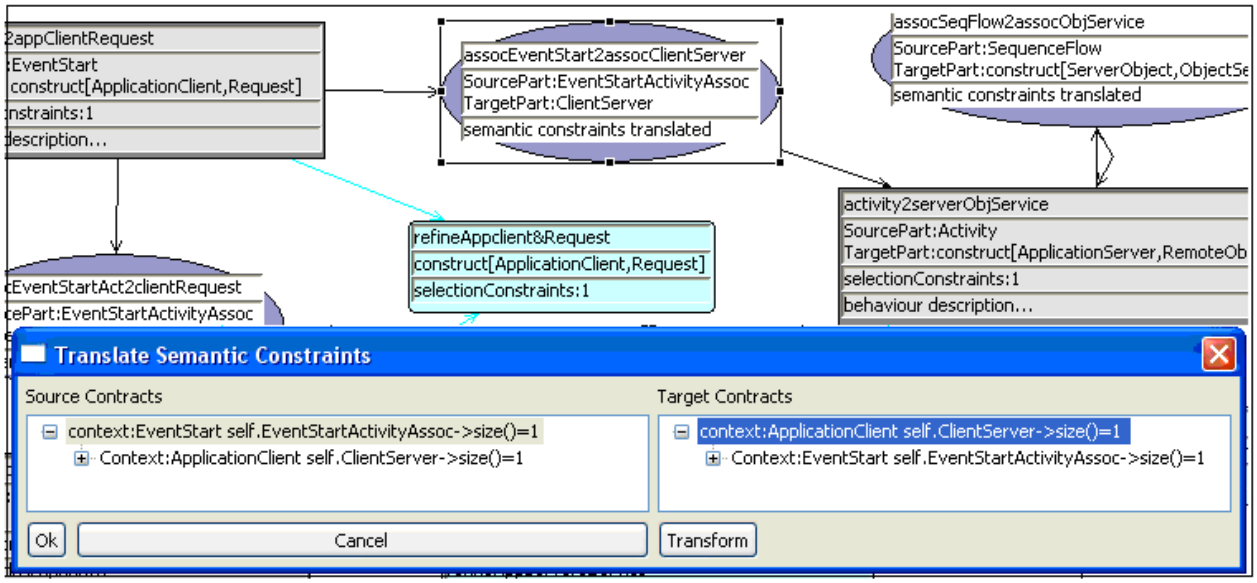


Figure 11.21. The *semanticTranslation* property sheet of SemanticAssociation "assocEventStart2assocClientServer"

11.2.2.2 *Generating search conditions and the behavior synchronizer*

The BPMN-MaramaMTE CRelation model generates a package of java search conditions to maintain traceability between BPMN and MaramaMTE models. Figure 11.22(a) illustrates the generated search conditions. The four StructureMappings and eight SemanticAssociations in the BPMN-MaramaMTE CRelation model generate 12 java classes. The generated java classes are search conditions to help to establish the traceability, and synchronize views and behaviors across the models.

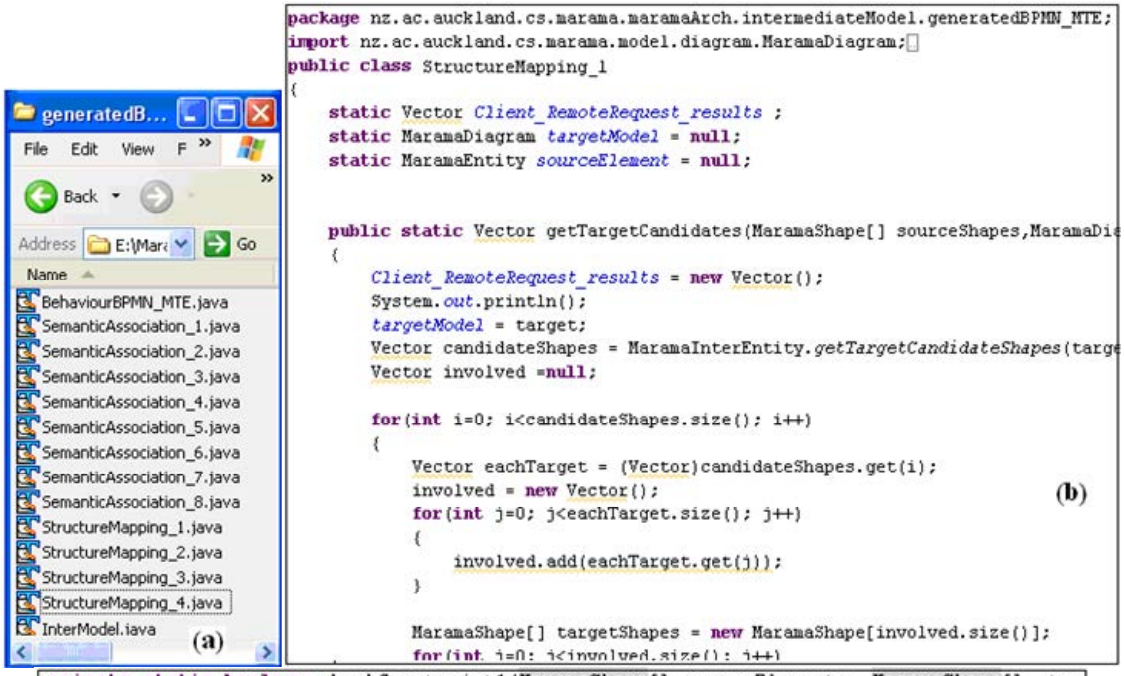


Figure 11.22. Java search conditions generated from the BPMN-MaramaMTE CRelation model

Figure 11.22(b) shows part of the generated StructureMapping_1.java (corresponding with StructureMapping "eventStart2appClientRequest" in Figure 11.19(a)). The important part of the java file is the checkConstraint0 and checkConstraint1 methods (Figure 11.22(c)). The method checkConstraint0 implements the selection constraint of Figure 11.19(c); and the method checkConstraint1 implements the selection constraint of Figure 11.19(b). The two selection constraints help to find out the source and target model elements that can be interconnected.

A java behavior synchronizer (the BehaviourBPMN_MTE.java in Figure 11.22(a)) is also generated. Figure 11.23 shows part of the generated BehaviorBPMN_MTE.java. This synchronizer processes the events of the four StructureMappings of the BPMN-MaramaMTE CRelation model (refer to Figure 11.19(a)). Lines 10 and 11 process two events recorded in the *behaviorDescription* of Figure 11.20. The synchronization messages are generated based on the intended results the tool users manually program inside the *behaviorDescription*.

```
2
3   public class BehaviourBPMN_MTE implements Adapter
4   {
5
6    public void notifyChanged(Notification notification)
7    {
8
9       ....................
10      processEvent_1_StructureMapping_1(notification);
11      processEvent_2_StructureMapping_1(notification);
12
13      processEvent_1_StructureMapping_2(notification);
14      processEvent_2_StructureMapping_2(notification);
15
16      processEvent_1_StructureMapping_3(notification);
17      processEvent_2_StructureMapping_3(notification);
18
19      processEvent_1_StructureMapping_4(notification);
20      processEvent_2_StructureMapping_4(notification);
21       ....................
22     }
23     ...................
24  }
```

Figure 11.23. The synchronizer generated from the BPMN-MaramaMTE CRelation model

11.2.2.3 *Interconnecting process*

Figure 11.24 shows how to interconnect the Travel Planner BPMN model with the Travel Planner MaramaMTE model. In Figure 11.24, Activity "SendBookRequest" (1) has only one qualified target model candidate (2) and is then interconnected with the candidate "construct[Client,

SendBookRequest]" (3). Once each interested source model element is assigned a target model candidate, the traceability and behaviour synchronisation between the models are established. If the change of property values or the removal of model elements happens to the source model, the interconnected target model element(s) will receive suggestive messages to take actions to maintain the validity of the Interconnection Relationships

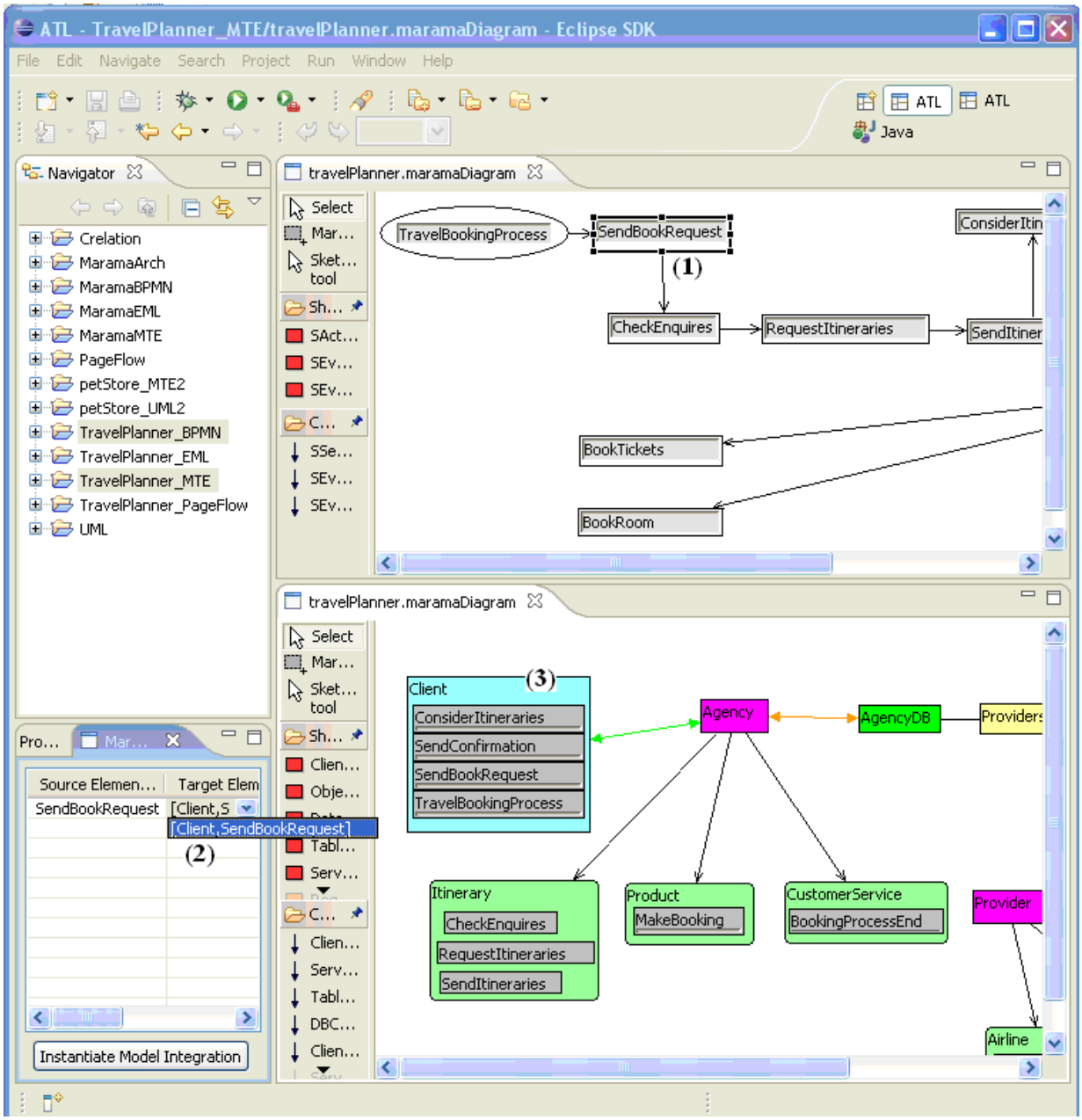


Figure 11.24. Interconnecting the Travel Planner BPMN model with the Travel Planner MaramaMTE model

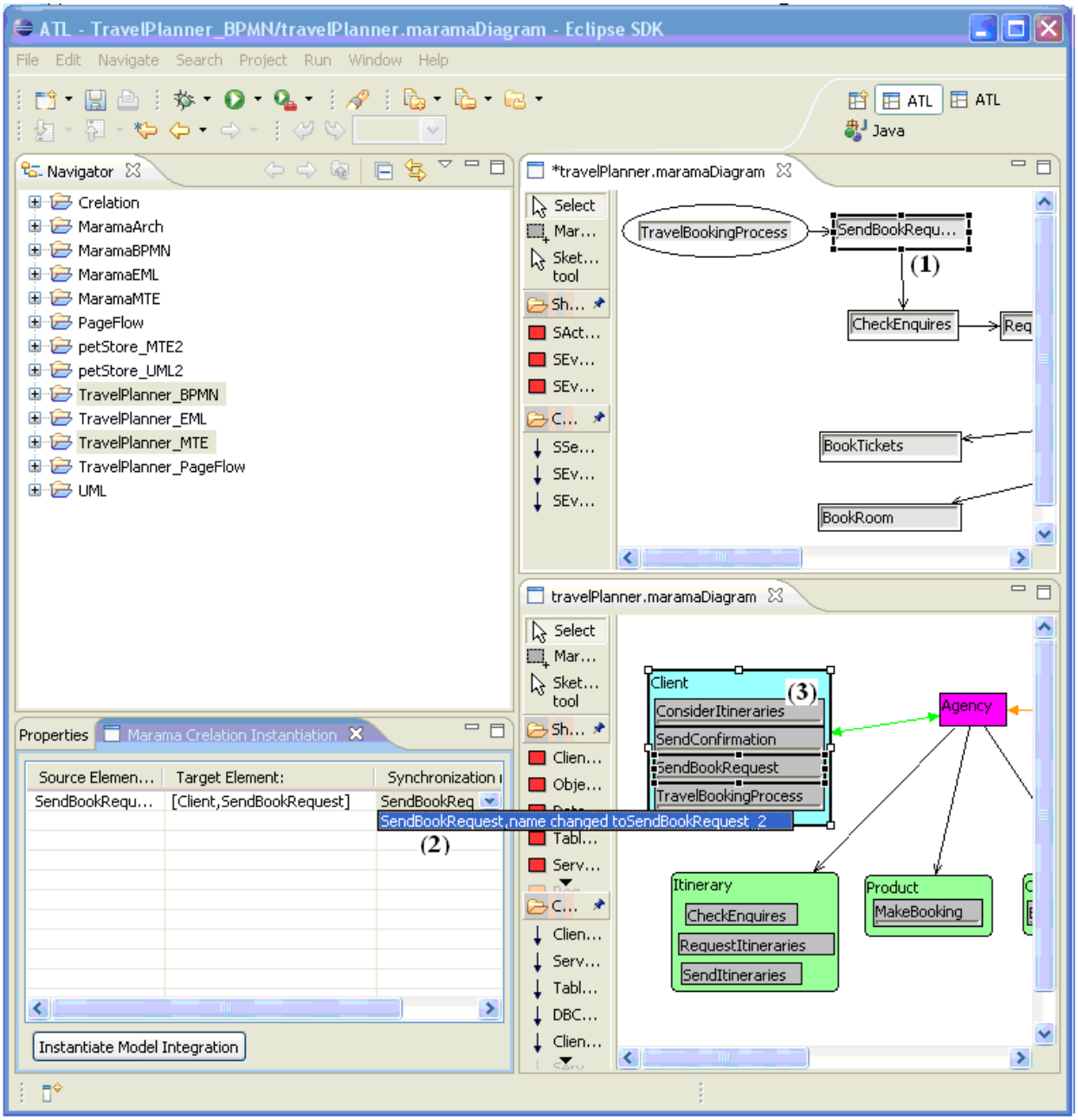


Figure 11.25. Synchronization between the Travel Planner BPMN and the Travel Planner MaramaMTE models

In Figure 11.25, when the "SendBookRequest" in BPMN is changed to "SendBookRequest _2" (1), the behaviour synchronizer sends a message to the target model "SendBookRequest" (2) to remind it that

the source model element is changed, which will influence the Interconnection Relationships between the models, and requires the appropriate response in the target model. More specifically, the "SendBookRequest" in the target model should be changed to "SendBookRequest_2" (3).

11.2.3 Interconnecting the Travel Planner MaramaMTE model with the Travel Planner Form Chart model

Both the MaramaMTE and Form Chart meta-models can be found in chapter 6. The MaramMTE architecture model is focused on the server side architecture design of web applications, while the Form Chart model analyzes the behaviour of web users. The shared semantics of the two meta-models is hard to capture, and needs to be specified clearly.

11.2.3.1 *The MaramaMTE-FormChart CRelation model and its entities*

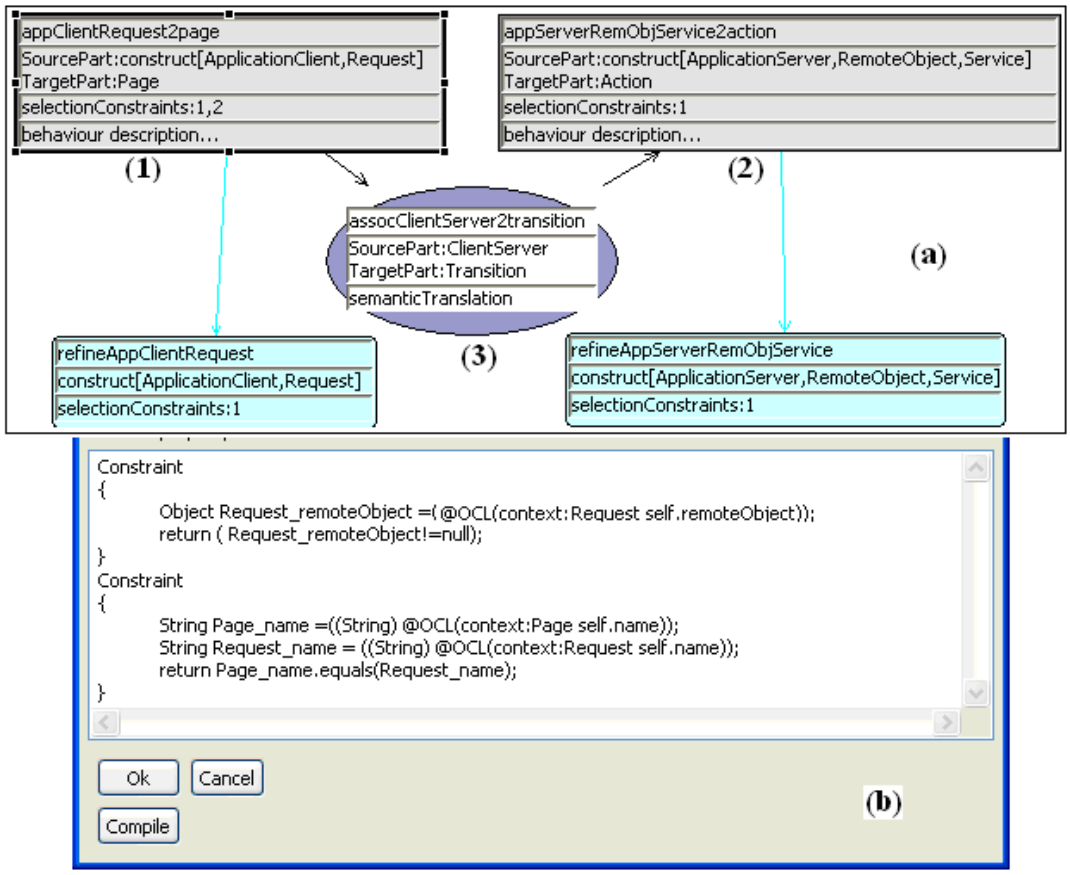


Figure 11.26. (a) the MaramaMTE-FormChart CRelation model; (b) the *selectionConstraints* property sheet of StructureMapping "appClientRequest2page"

Figure 11.26(a) illustrates the MaramaMTE-FormChart CRelation model. The StructureMapping "appClientRequest2page" (1) represents that the construct of ApplicationClient and Request of MaramaMTE can be interconnected with the Page of Form Chart. The StructureMapping "appServerRemObjService2action" (2) defines that a construct of ApplicationServer, RemoteObject, and Service of MaramaMTE can be interconnected with the Action of Form Chart. The SemanticAssociation "assocClientServer2transition" (3) explicitly specifies the associations that are involved in the interconnection. Figure 11.26(b) shows two selection constraints of StructureMapping "appClientRequest2page". The first constraint specifies that the *remoteObject* property value of the Request in the MaramaMTE model must not be null. The second constraint specifies the *name* value of the Request in the MaramaMTE model and the *name* value of the Action in the Form Chart model must be the same.

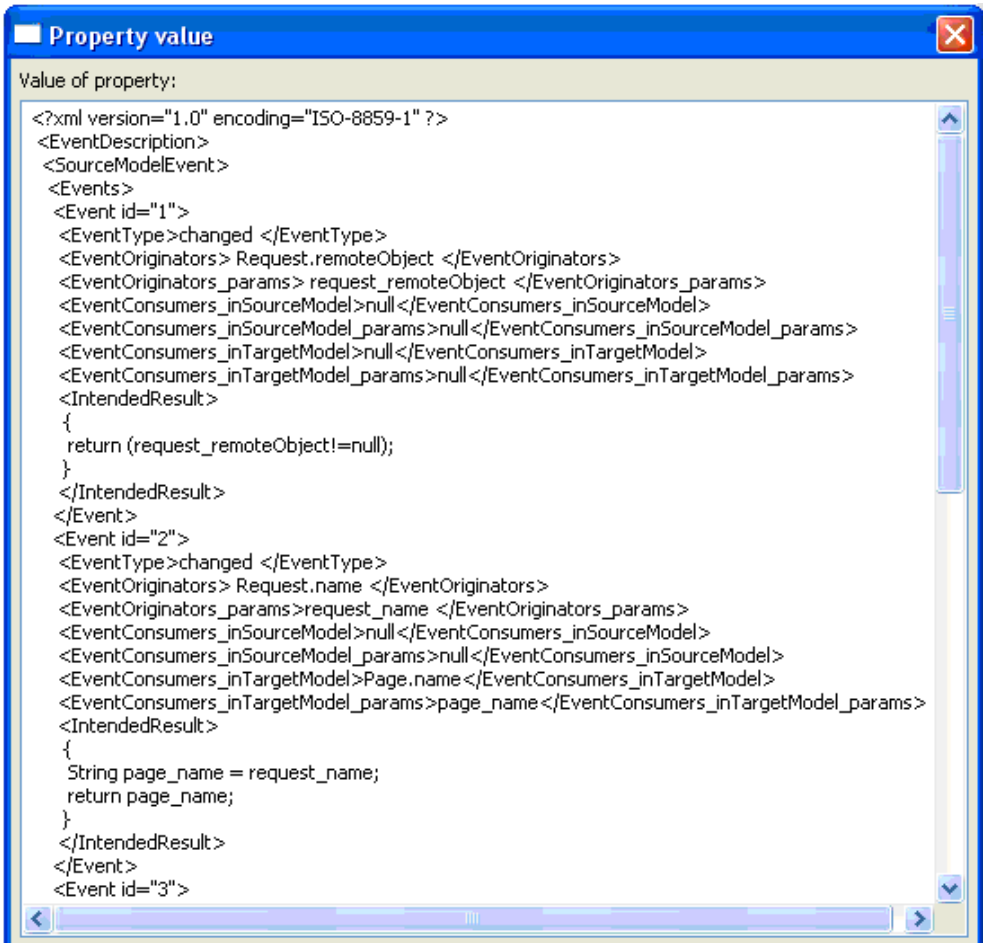


Figure 11.27. The *behaviorDescription* of StructureMapping "appClientRequest2page"

The *behaviorDescription* in Figure 11.27 organizes the events triggered by the StructureMapping "appClientRequest2page" ((1), Figure 11.26(a)). The StructureMapping triggers four source model events: 1) the change of the *remoteObject* property value of the Request (event 1 in Figure 11.27); 2) the change of the *name* property value of the Request (event 2 in Figure 11.27); 3) the removal of the ApplicationClient (not shown in Figure 11.27); 4) the removal of the Request (not shown in Figure 11.27). The information of the triggered events is organized in the *behaviorDescription* and will help to generate the java behavior synchronizer.

Figure 11.28 illustrates the *semanticTranslation* property sheet of SemanticAssociation "assocClientServer2transition". The source model semantic constraint (tree root, left column) means in a MaramaMTE model, an ApplicationClient must be connected to at least one ApplicationServer. This semantic constraint is translated into sensible Form Chart model semantic constraint, which specifies that, in a Form Chart model, a Page must be connected to at least one Transition (child, left column). Similarly, a target meta-model translatable semantic constraint (tree root, right column) can be translated into a sensible source meta-model semantic constraint (the child of the root, right column).

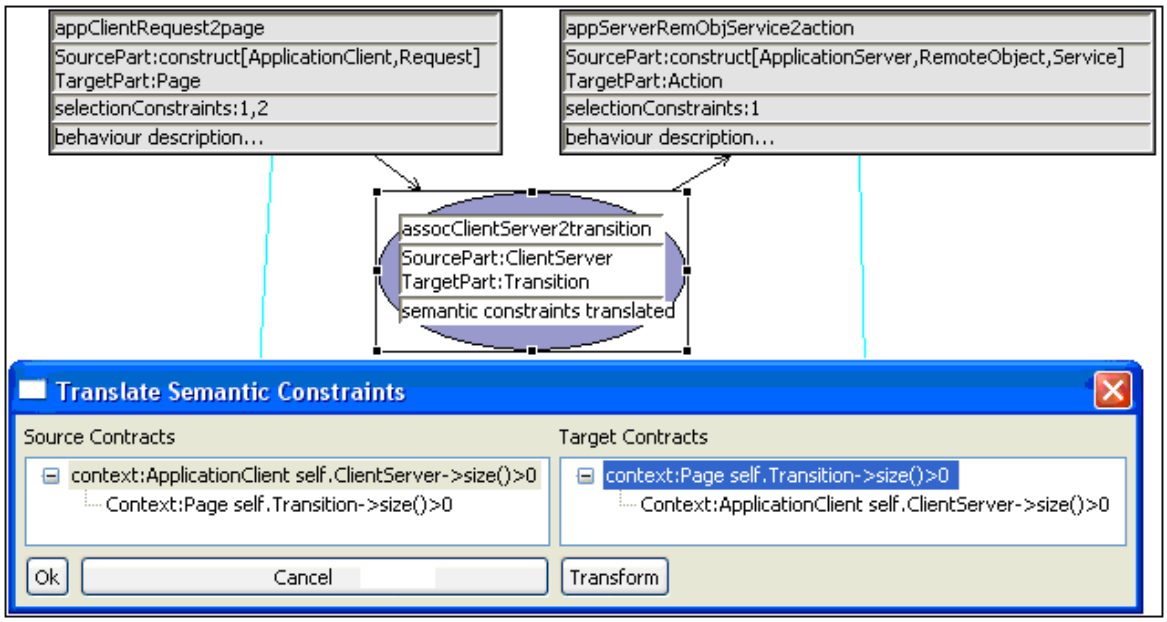
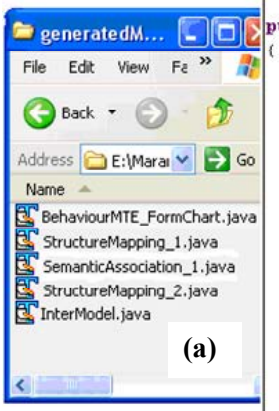


Figure 11.28. The *semanticConstraints* property sheet of SemanticAssociation "assocClientServer2transition"

11.2.3.2 *Generating search conditions and the behavior synchronizer*




```
package nz.ac.auckland.cs.marama.maramaArch.intermediateModel.gen
import nz.ac.auckland.cs.marama.model.diagram.MaramaDiagram;

public class StructureMapping_1
{
    static Vector  Page_results ;
    static MaramaDiagram targetModel = null;
    static MaramaEntity sourceElement = null;


    public static Vector getTargetCandidates(MaramaShape[] source
    {
        Page_results = new Vector();
        targetModel = target;
        Vector candidateShapes = MaramaInterEntity.getTargetCandi
        Vector involved =null;

        for(int i=0; i<candidateShapes.size(); i++)
        {
            Vector eachTarget = (Vector)candidateShapes.get(i);
            involved = new Vector();
            for(int j=0; j<eachTarget.size(); j++)
            {
                involved.add(eachTarget.get(j));
            }

            MaramaShape[] targetShapes = new MaramaShape[involved
            for(int j=0; j<involved.size(); j++)
```

(b)

```
private static boolean checkConstraint0(MaramaShape[] sourceElements)
{
    MaramaShape Client_shape = sourceElements[0];
    MaramaModelProject  Client_model =  Client_shape.getModelEntity().getModelProjec
    MaramaEntity  Client_entity =  Client_shape.getModelEntity();

    MaramaShape Request_shape = sourceElements[1];
    MaramaModelProject Request_model = Request_shape.getModelEntity().getModelProjec
    MaramaEntity Request_entity = Request_shape.getModelEntity();

    String remoteObject = Request_entity.getAttributeValueAsString("remoteObject");
    if(remoteObject !=null)
        return true;
    else
        return false;
}
private static boolean checkConstraint1(MaramaShape[]  sourceElements,MaramaShape[]
{
    MaramaShape Client_shape = sourceElements[0];
    MaramaModelProject  Client_model =  Client_shape.getModelEntity().getModelProjec
    MaramaEntity  Client_entity =  Client_shape.getModelEntity();

    MaramaShape Request_shape = sourceElements[1];
    MaramaModelProject Request_model = Request_shape.getModelEntity().getModelProjec
    MaramaEntity Request_entity = Request_shape.getModelEntity();

    MaramaShape Page_shape = targetElements[0];
    MaramaModelProject  Page_model =  Page_shape.getModelEntity().getModelProject();
```

(c)

Figure 11.29. Java search conditions generated from the MaramaMTE-FormChart CRelation model

The MaramaMTE-FormChart CRelation model generates a package of java search conditions to maintain traceability during interconnection between the MaramaMTE and FormChart models. Figure 11.29(a) illustrates the generated search conditions. The two StructureMappings and one SemanticAssociation in the MaramaMTE-FormChart CRelation model generate three java classes. Figure 11.29(b) shows part of the StructureMapping_1.java (encoding the StructureMapping "appClientRequest2page" in Figure 11.26(a)). The important parts of the java file are the checkConstraint0 and checkConstraint1 methods (Figure 11.29(c)). Method checkConstraint0 translates the first selection constraint of Figure 11.26(b), and method checkConstraint1 translates the second selection constraint of Figure 11.26(b). The two selection constraints help to find out the qualified source and target model elements to interconnect.

```
3       public class BehaviourMTE_PageFlow implements Adapter
4     {
5
6      public void notifyChanged(Notification notification)
7       {
8          ..................
9         processEvent_1_StructureMapping_1(notification);
10
11        processEvent_2_StructureMapping_1(notification);
12
13        processEvent_3_StructureMapping_1(notification);
14
15        processEvent_4_StructureMapping_1(notification);
16
17        processEvent_1_StructureMapping_2(notification);
18
19        processEvent_2_StructureMapping_2(notification);
20
21        processEvent_3_StructureMapping_2(notification);
22
23         ..................
24       }
25     ..................
26  }
```

Figure 11.30. The synchronizer generated from the MaramaMTE-FormChart CRelation model

A java behavior synchronizer (the BehaviourMTE_FormChart.java in Figure 11.29(a)) is also generated from the CRelation model. Figure 11.30 shows part of the synchronizer. When source model events happen, this synchronizer processes the events of the two StructureMappings (refer to Figure 11.26(a)) of the CRelation model. For example, lines 9, 11, process two events recorded in the

behaviorDescription of Figure 11.27. The synchronization messages are generated based on the intended results the tool users manually program inside the *behaviorDescription*.

11.2.3.3 *Interconnecting process*

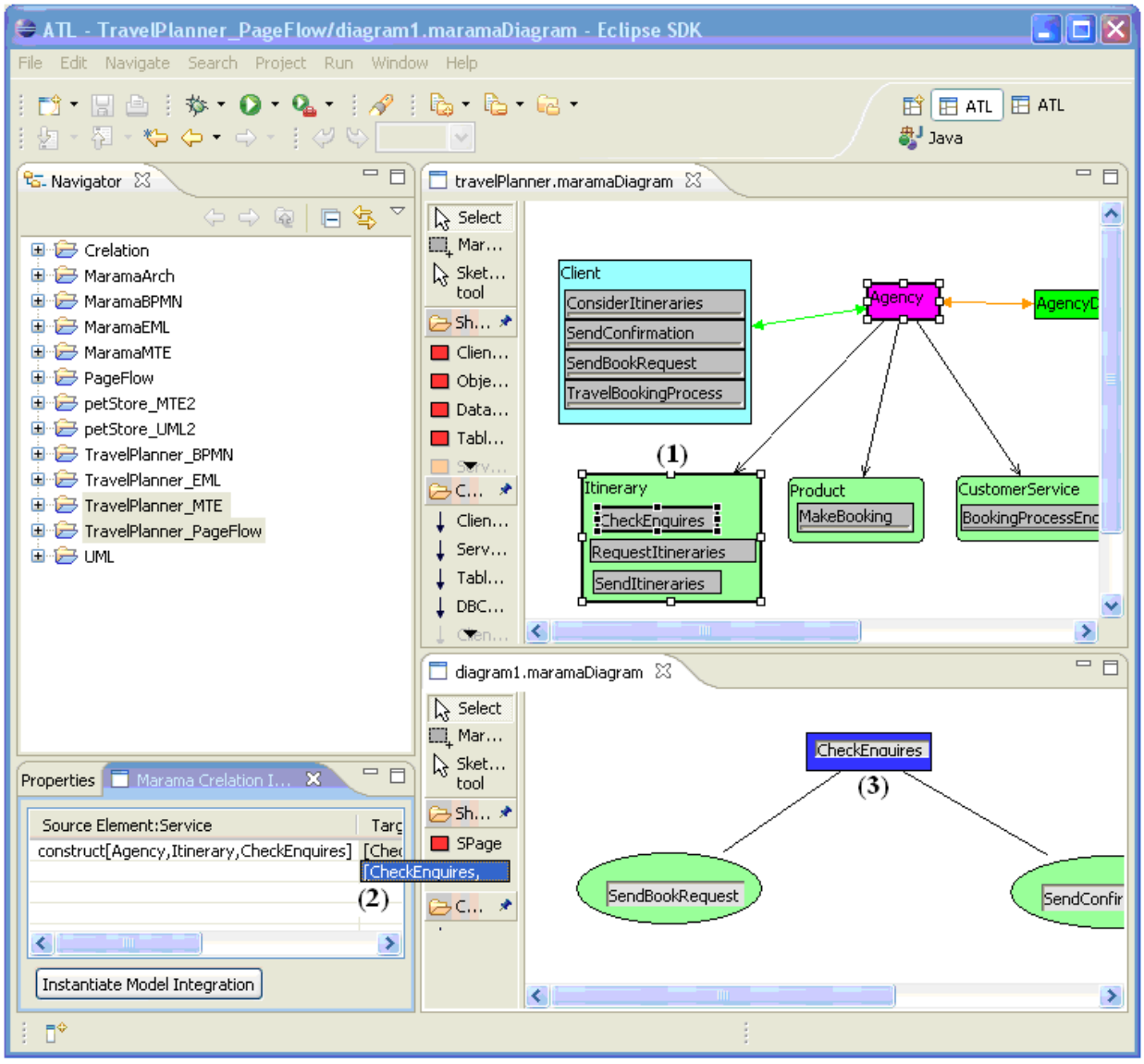


Figure 11.31. Interconnecting the Travel Planner MaramaMTE model with the Travel Planner Form Chart model

Figure 11.31 shows how to interconnect the Travel Planner MaramaMTE model with the Travel Planner Form Chart model. In Figure 11.31, the source model construct of "Agency" (typed as

ApplicationServer), “Itinerary” (typed as RemoteObject), and “CheckEnquires” (typed as Service) (1) has only one qualified target model candidate “CheckEnquires” (typed as Action) (2) and is then interconnected with the candidate “checkEnquires” (3). Once each interested source model element is assigned a target model candidate, the traceability and behaviour synchronisation between the two models are established. If the change of property values or the removal of model elements happens to the source model, the interconnected target model element(s) will receive suggestive messages to take actions to maintain the validity of the Interconnection Relationships

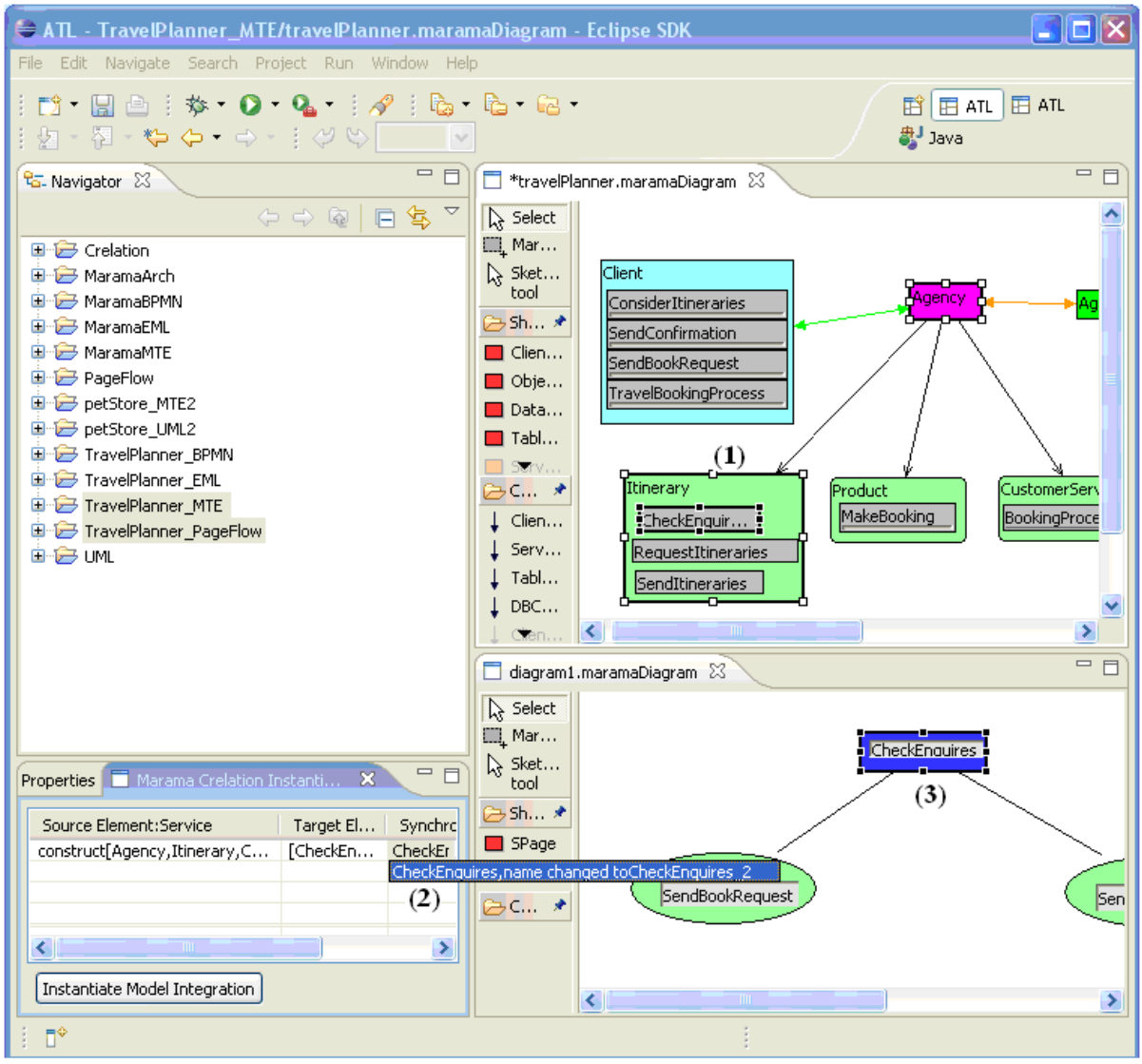


Figure 11.32. Interconnecting the Travel Planner MaramaMTE model and the Travel Planner Form Chart model

In Figure 11.32, when the “CheckEnquires” in MaramaMTE is changed to “CheckEnquires_2”(1), the behaviour synchronizer sends a message to the target model “CheckEnquires” (2) to remind it that the source model element is changed, which will influence the Interconnection Relationship and requires the correspondent action in the target model. More specifically, the “CheckEnquires” Action in the target model should also be changed to “CheckEnquires _2” (3).

11.2.4 The interconnected Travel Planner models

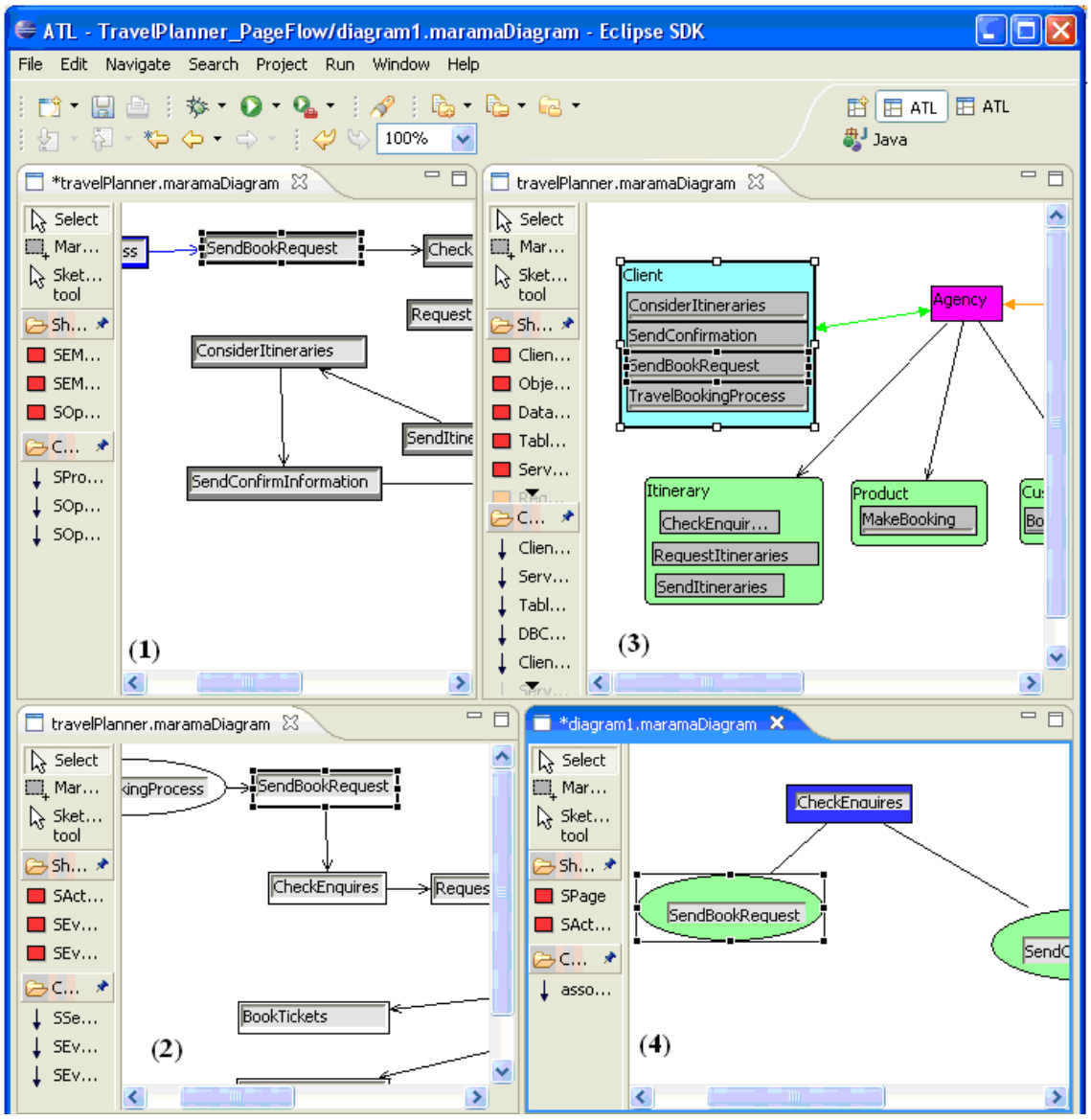


Figure 11.33. Interconnecting the Travel Planner’s EML model, BPMN model, MaramaMTE model, and Form Chart model

Figure 11.33 illustrates the Travel Planner EML model (1), Travel Planner BPMN model (2), Travel Planner MaramaMTE model (3), and Travel Planner Form Chart model. The four models are interconnected through three CRelation models. The four models can be synchronized visually and behaviourally, which allows users to specify, refine, integrate, and transform various concerns of a software project. Figure 11.33 shows that the "SendBookRequest" operation in the EML model shares the similar semantics with 1) the "SendBookRequest" in the BPMN model; 2) the construct of "Client" and "SendBookRequest" in the MaramaMTE model; and 3) the "SendBookRequest" page in the Form Chart model. The interconnected models represent how a model can be refined into, integrated with and transformed to other models.

The interconnection of the four models demonstrates the features of the CRelation model and the MaramaCRelation tool. The involved CRelation models can not only be used to analyze and design MI&T, but also provide high level support for the essential tasks of MI&T including traceability, consistency management, and behaviour synchronization.

11.3 Discussion

In the two case studies, the CRelation models increase the abstraction level of the concerned tasks of MI&T. The used CRelation models provide a structured way to analyze and design important but usually neglected issues, including capturing the shared semantics, and explicitly modeling the associations involved in the intended MI&T. The used CRelation models also organize the usually isolated operational tasks of MI&T, including semantic consistency, traceability, view and behavior synchronization, and traceability.

The CRelation models are correspondence models for various software engineering domain-specific models. The CRelation models specify the rational for the intended MI&T, support incremental construction of selection constraints, and visually categorize the selection constraints by the involved model elements.

The two case studies show that the MaramaCRelation approach has the strength and potential to smoothly interconnect any domain-specific models in terms of analysis and design support, maintenance

of semantic constraints, flexible traceability mechanism, and structured support for maintaining behaviour synchronization.

11.4 Summary

The two case studies demonstrate the strength of the MaramaCRelation approach. The CRelation models capture the rational of an intended MI&T. The CRelation model entities specify Interconnection Relationships between two models; and the Interconnection Relationships can represent transformation relationships, integration relationships, and refinement relationships. The constraints of the intended MI&T are categorized by StructureMappings, SelectionRefinements, and StructureRefinements. SemanticAssociations capture the usually implicit information of associations involved in the intended MI&T. SemanticAssociations also help to maintain translatable semantic constraints. The CRelation models can generate well structured search conditions and behaviour synchronization coordinators to maintain traceability and behaviour synchronization among the interconnected models. Although at its initial stage, the MaramaCRelation research has shown its strength as a structure, high-level support for model integration, model transformation, model refinement, tool integration, and multi-view support environment.

Chapter 12 - The Evaluation of the MaramaCRelation Approach

The MaramaCRelation research identifies a set of problems related to Model Integration and Transformation (MI&T); and provides solutions for those problems. The quality of the MaramaCRelation research depends on: how well the identified problems are understood and accepted; and how efficient and effective the MaramaCRelation approach is to solve the problems. This chapter reports the evaluation of the MaramaCRelation research by the Cognitive Dimension framework (Green et al, 1996); as well as by interviewing a group of experienced Software Engineering tool developers and designers using a questionnaire. The questionnaire contains questions designed against the problems that motivate the research of the MaramaCRelation. The feedback given by the interviewees is analyzed and used to improve the quality of the MaramaCRelation approach.

12.1 Cognitive Dimensions

Cognitive Dimension (Green et al, 1996) investigations have been conducted repetitively during the development of the MaramaCRelation approach. This section reports how the MaramaCRelation approach performs in the main cognitive dimensions.

Abstraction gradient: Figure 12.1 shows the abstraction levels involved in the MaramaCRelation approach. The CRelation model is designed through meta-modeling; and it interconnects domain-specific meta-models. The CRelation model is supported by the MaramaCRelation tool that is constructed using the Marama meta-tools. Thus the MaramaCRelation approach has a high *abstraction gradient,* which may be difficult for tool users that have not much knowledge of meta-modeling. But the MaramaCRelation approach follows the style of the UML meta-modeling, which makes it easy for the experienced tool users to learn the MaramaCRelation approach. We once ran a 45 minute demonstration to a group of experienced tool developers and users. They had no problems to understand the abstraction gradient, and understood the relationships between abstraction levels very well.

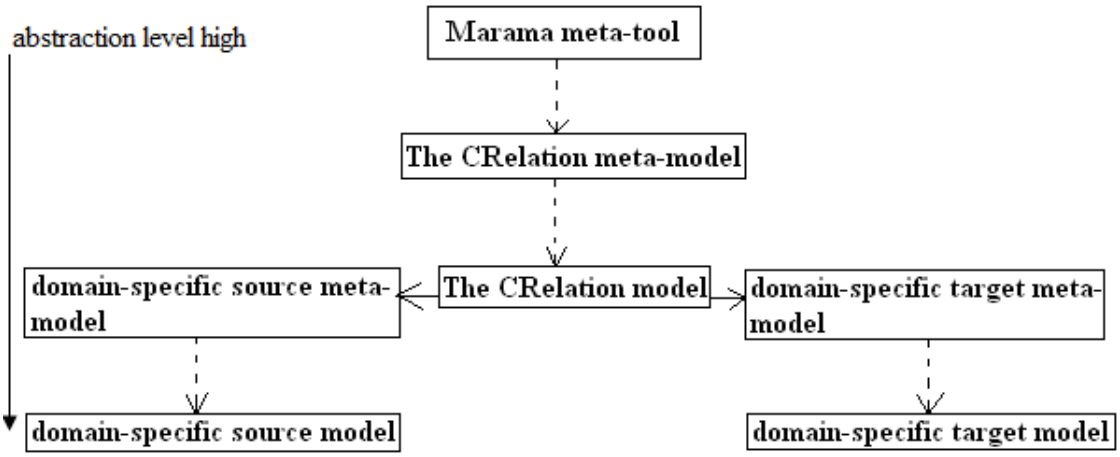


Figure 12.1. The abstraction gradient of the MaramaCRelation approach

Closeness of mapping: The CRelation model notations are close to the problem world – analysis and design of MI&T. A notation of StructureMapping represents: 1) an Interconnection Relationship between the two domain-specific meta-models; 2) the constraints that make the Interconnection Relationship valid; as well as 3) the events triggered by the Interconnection Relationship. The notation of SemanticAssociation represents the association between two Interconnection Relationships. When we ran a demonstration to software modelers, it took some time (around 1 hour) for them to understand the target problems (refer to Section 8.2) of the MaramaCRelation approach. Once they understood the problems, they found that the CRelation model notations are very close to the problem world and very easy to use.

Consistency: The CRelation model is designed for the maximum consistency. After learning the first several notations, software modellers can successfully guess the rest ones. The two main modelling concepts in the CRelation model are the StructureMapping and the SemanticAssociation. Once these two concepts are understood, the SelectionMapping and the StructureRefinement are easy to guess because they have similar properties as the StructureMapping, and they both refine the StructureMapping.

Diffuseness / terseness: The CRelation model is terse. A basic CRelation model can be set up without much manual effort. For example, the *entityMapping* and *behaviorDescription* properties of a StructureMapping can be automatically or partially automatically generated. The *associationMapping*

and *semanticTranslation* properties of a SemanticAssociation can also be automatically generated. The CRelation model can also be verbose. For example, it is a complicated job to set up the *selectionConstraint* property of a StructureMapping, a SelectionRefinement, and a StructureRefinement. The CRelation model does not support effective visual context within the Crelation model at this stage. It is expected that the improved visual context within the CRelation model will mitigate the verbosity of the CRelation model.

Error-proneness: The CRelation model is not error-prone. All the essential modeling information comes from the involved source and target meta-models; and users can not change it. For example, the *entityMapping* property value of a StructureMapping can only be consisted of the source and target meta-model entities and constructs involved in the StructureMapping. For the modeling information that users need to construct manually, such as the *selectionConstraints* of a StructureMapping, the MaramaCRelation tool compiles them to validate the constraints.

Hard mental operations: The MaramaCRelation approach is aimed for solving hard mental operations involved in traditional MI&T technologies, including: recording the rationale of an intended MI&T; capturing the shared semantics; categorizing selection constraints; organizing events triggered by the MI&T; and maintaining traceability by using the shared semantics. The MaramaCRelation approach itself does not involve hard mental operations. Once the users correctly understand the semantics of the CRelation model, they will not need to bear extra mental load to be able to construct a CRelation model.

Hidden dependencies: The CRelation model explicitly represents the dependencies between model entities. For example, the source (target) part of a SemanticAssociation must be a path connecting the source (target) parts of the associated StructureMappings. A SelectionRefinement must refine a construct of its parent StructureMapping(s). The generation of search conditions from the CRelation model is also straightforward, and does not use hidden dependencies that are not shown in the CRelation model.

Juxtaposability: It is essential for the MaramaCRelation tool to support effective communication between the CRelation model and the involved source and target meta-models. The MaramaCRelation tool can juxtapose the CRelation model and the source and target domain-specific meta-models in different editors to help users to understand how the CRelation model communicates with the domain-

specific meta-models. In the future, the Juxtaposability needs to be realized within the CRelation model to provide better visual context of the model.

Premature commitment: The MaramaCRelation approach tries to reduce the premature commitment to the minimum level. When constructing a CRelation model, users must make sure of the correctness and validity of StructureMappings first. Setting up the SemanticAssociations between appropriate StructureMappings gives users a second chance to validate the intended StructureMappings. After the StructureMappings and SemanticAssociations are well designed, users can then design SelectionRefinements and StructureRefinements. Users can modify a CRelation model easily with the automatic support of the MaramaCRelation tool, as most of the properties of the modeling entities are automatically generated.

Progressive evaluation: It is easy to evaluate the progress of a CRelation model. Users can check if the CRelation model can interconnect the interested parts of the source and target models. If not, users can modify or extend the existing CRelation model. As for the generation of search conditions from the CRelation model, users can tighten or loosen the search conditions by putting stronger or weaker selection constraints in the CRelation model.

Role-expressiveness: The CRelation model itself is a simple model containing only 4 entity types and 3 association types. The roles of the four entity types are obvious. The StructureMappings specify the main concerned mappings; the SelectionRefinements and StructureRefinements refine the selection constraints of the StructureMappings; and the SemanticAssociations associate the StructureMappings.

Secondary notation and escape from formalism: The CRelation model notations do not carry extra information by means not related to the CRelation model syntax. The syntax and semantics of the Crelation model are those of StructureMappings, SelectionRefinements, StructureRefinements, SemanticAssociations, and the CRelation associations. The CRelation modeling is supported by the MaramaCRelation tool with well-designed empirical algorithms and mechanisms, which allows software modelers to escape from the formalism.

Viscosity: The CRelation model is at initial stage, and suffers viscosity when interconnecting complicated domain-specific meta-models. The CRelation model is aimed for interconnecting any MOF-based domain-specific meta-models in the environment of the Marama meta-tools; and generating

search conditions for the establishment of traceability. The case studies in Chapter 11 proved the strength of the MaramaCRelation approach; it also prompted viscosity of using the CRelation model to interconnect complicated domain-specific meta-models where the relationships among the entities can be hard to visualize. The simple design of the CRelation model entities is far from enough to capture the complexity of various domain-specific models. The CRelation model needs to be extended to support interconnecting reasonably complicated domain-specific meta-models.

Visibility: The MaramaCRelation tool is an Eclipse plug-in, which provides the tool with good mechanisms to support *visibility and juxtaposability*. It juxtaposes the CRelation model and the source and target meta-models side-by-side to allow simultaneous visualisations of the involved models. However, the visibility within the CRelation model needs to be improved. The textual information (e.g. constructs, the relationship between the *selectionConstraints* and the *behaviorDescription* of a StructureMapping, the relationship between the CRelation model and its generated search conditions) involved in the CRelation model needs to be visualized in the future to improve the visibility of the CRelation model and its related activities.

12.2 Evaluation against the requirements

We have also conducted an informal survey of a small number of experienced Software Engineering meta-modellers to obtain qualitative feedback for the MaramaCRelation approach. A questionnaire is designed to evaluate the MaramaCRelation approach against the problems identified in Section 8.2; and the questions fall into five categories as follows:

Category 1: Do you often come across the problems identified by the MaramaCRelation approach? Are those problems sensible and need to be solved?

Q1: *How do you normally capture the rationale of your Model Integration and Transformation (MI&T) designs?*

Evaluation feedback: Coding and mapping tools with the help of natural language. Rationale is usually captured in low-level code/formulae that specify target model values in terms of source model values. The rationale is usually implicit in the formulae and structure/ordering of the transformation script or

consistency management code. Some mapping tools, such as MaramaTorua (MaramaTorua, 2007), are also used to record the rationale of an intended MI&T.

Evaluation feedback analysis: The evaluation feedback shows that the rationale of an intended MI&T is easy to be lost in the operational code; and there are no well-accepted models and tools to help to explicitly record the rationale of the intended MI&T. The available mapping tools, such as MaramaTorua (MaramaTorua, 2007), visualize the mappings between two domain-specific meta-models; but they are more like visual transformation languages focusing on the operational goals of MI&T without much analysis and design support. In fact, not enough attention has been paid to analyze and design MI&T. The MaramaCRelation approach explicitly records and visualizes the rationale of an intended MI&T; and uses the well-structured rationale (the CRelation model) to provide better solutions for the concerned issues involved in MI&T, including semantic consistency, traceability, and behavior synchronization.

Q2: *Have you ever experienced the need to maintain semantics and semantic constraints when doing MI&T?*

Evaluation feedback: Very often. One of the expert reviewers said: **"**Yes, as this is the whole point in my mind of model integration and/or transformation. Semantically incorrect target models or related models make the integration/transformation of very limited use".

Evaluation feedback analysis: It is important to maintain semantics and semantic constraints during MI&T. The more loss of semantics and semantic constraints, the weaker the bonding between the two models becomes. Most of the existing MI&T technologies (e.g. ATL, XSLT, and Coding) do not help users to analyze and design how to maintain semantics and semantic constraints during MI&T; and users normally treat the maintenance of semantics and semantic constraints as separate operational tasks from the MI&T. The CRelation model explicitly maintains semantics that is conceptually shared by two models. It also maintains *translatable semantic constraints*. The CRelation model is intended to maintain as much semantics and semantic constraints as possible to give tool users and developers chances to control the degree of the bonding between the two models. The CRelation model is a place where users can design the intended MI&T via maintaining semantics and semantic constraints.

Q3: *How easy do you find maintaining model transformation scripts, especially after a long time since you wrote them?*

Evaluation Feedback: Hard. Because they are usually low level code, it is hard to read, understand them after a short time. Also the scripts/code do not lend themselves to easy maintenance due to the mix of constructs in the code i.e. mix of formulae doing transformation./linking vs. extraction of source./target elements esp. in XML based transformations.

Evaluation feedback analysis: It is very hard to maintain, extend, and reuse transformation scripts. The rationale for the scripts is lost in the code; the templates and rules are not well structured; the relationships between the templates and rules are implicit; and the scripts and programs are monolithic. The CRelation model has succeeded in categorizing and visualizing the ATL transformation scripts in Section 10.2.2.2. The CRelation model shows the potential of improving the maintainability of transformation scripts by: explicitly recording the rationale of the transformation; categorizing the templates and rules; explicitly representing the relationships between the templates and rules; and constructing the intended transformation incrementally.

Q4: *How do you currently solve issues such as traceability establishment and view synchronization during MI&T?*

Evaluation feedback: Mainly coding. The implementation of traceability and model synchronization mechanisms are usually based on low-level data repositories, reusable event handlers, and programming framework.

Evaluation feedback analysis: Traceability and behavior synchronization are two main concerned issues during MI&T. The establishment and maintenance of traceability and behavior synchronization is programming-intensive (e.g. Rational Rose, MaramaMTE, MaramaEML). The MaramaCRelation tool generates search conditions from the CRelation model, finds out appropriate target model candidates for interconnection, and establishes traceability between models. Although the MaramaCRelation tool does not provide full-blown behavior synchronization mechanisms, it has shown how the behavior

information of the CRelation model together with the established traceability can support behavior synchronization (case studies, Chapter 11). The MaramaCRelation approach can export the behavior and traceability information to the third party event-modeling environment (e.g. Kaitiaki (Liu et al, 2007)) to construct model-based full-blown behavior synchronization mechanisms.

Category 2: Is the CRelation model easy to understand?

Q5: How hard is it to understand StructureMapping and its properties?

Evaluation feedback: Reasonable. This is the basic relationship between model elements/constructs. An interviewee pointed out that the concept of *construct* is not easy to understand as it is represented textually in the CRelation model. He suggested improving the understandability of the concept of *construct* by using a visual meta-model snippet within the CRelation model to visually link the *construct* to the involved source and target meta-model elements.

Evaluation feedback analysis: The MaramaCRelation tool is at its initial stage. At now, it coordinates the CRelation model with its source and target meta-models. When a StructureMapping is highlighted/chosen in the CRelation model, the involved source and target meta-model elements are highlighted/chosen in the source and target meta-models respectively. Based on the evaluation feedback, the MaramaCRelation tool needs to provide effective visual context within the CRelation model to help users to effectively link the CRelation model elements to the involved source and target meta-model elements.

Q6: How hard is it to understand SelectionRefinement and its properties?

Evaluation feedback: Easy.

Evaluation feedback analysis: The SelectionRefinement refines the selection constraints of StructureMappings. The properties of the SelectionRefinement are similar to the same-named properties of the StructureMapping. The SelectionRefinement helps to maintain the semantic constraints of a domain-specific meta-model (e.g. the naming convention among EJBBean, EJBHome, and EJBInterface in EJBUML meta-model).

*Q7**:** How hard is it to understand StructureRefinement and its properties?*

Evaluation feedback: Reasonable.

Evaluation feedback analysis: The StructureRefinement allows users to model selection constraints between the source part of one StructureMapping and the target part of another StructureMapping. In the CRelation model, the StructureRefinement, together with the StructureMapping and SelectionRefinement, explicitly categorizes selection constraints by the involved model elements. One of the future projects of the MaramaCRelation approach is to visualize ATL and XSLT transformation scripts in the CRelation model where the StructureRefinement, together with the StructureMapping and SelectionRefinement, will be used to categorize transformation templates and rules.

Q8: *How hard is it to understand SemanticAssociation and its properties?*

Evaluation feedback: Hard. Most of the interviewees found this concept one of the harder ones to understand.

Evaluation feedback analysis: When people construct transformation programs and scripts to transform a model from one format to another, they pick up a set of interested source model elements, construct the transformation scripts towards the set of elements, and complete the transformation. In this way, the associations between the templates and rules are implicit. As the associations may indicate the possible semantic inconsistency caused by the transformation, it is important to explicitly represent the implicit but usually ignored associations. The CRelation model uses the SemanticAssociation to explicitly represent the associations between StructureMappings. The SemanticAssociation detects the possible semantic inconsistency brought by the intended StructureMappings, and helps to validate the intended MI&T specified in the StructureMappings (refer to Figure 9.16).

Q9: How do you rate the understandability of the visual notations of the CRelation modeling elements?

Evaluation feedback: Hard to understand. One of the interviewees pointed out that the visual language of the CRelation model is hard to understand at first. He suggested giving a visual indication of the

context of CRelation elements within the CRelation model, so that users can effectively link the CRelation model elements with the corresponding information of the source and target meta-models.

Evaluation feedback analysis: The MaramaCRelation tool coordinates the CRelation model with its source and target meta-models. When a StructureMapping is chosen/highlighted in the CRelation model, the involved source and target meta-model elements are highlighted/chosen in the source and target meta-models respectively. Based on the feedback, the MaramaCRelation tool needs to support visual context within the CRelation model to help users to effectively retrieve the information of the involved source and target meta-model.

There are also other reasons for people to find it hard to understand the visual notations of the CRelation model. The MaramaCRelation approach identifies a set of problems that are normally treated as separate operational tasks related to MI&T. The MaramaCRelation approach tries to provide a central, high-level solution for those separate tasks. It is hard to promote the importance of the identified problems towards the research of MI&T; and it is harder to introduce the solutions (the MaramaCRelation approach) to those problems. After its initial success, the MaramaCRelation approach is now ready to solve well aware problems, including: using the CRelation model to visualize and categorize ATL scripts; using the visualized ATL scripts to construct selection constraints and generate behavior information; and collaborating with third party event modeling environments to construct full-blown behavior synchronization mechanisms. The strength of the MaramaCRelation research will be improved and better demonstraded by solving those well aware problems.

Category 3: Can the CRelation model solve the identified problems? What is the potential?

Q10*: How do you rate the usefulness of the CRelation model for capturing the rationale of MI&T?*

Evaluation feedback: Very useful. The CRelation model provides a model-to-model mapping language with support for capturing not just element correspondences – which visual data mapping tools (e.g MaramaTorua) focus on, but complex constructs, constraints, traceability establishment, and behavior synchronization.

Evaluation feedback analysis: As the rationale of MI&T is the main source of the solutions for all other issues involved in MI&T, it is very useful to have a model like CRelation dedicated to capture the shared semantics (the rationale) during MI&T. The CRelation model is an analysis and design model for MI&T; and is intended to support MI&T the similar way the UML supports Object Oriented Analysis and Object Oriented Design.

Q11: *How do you rate the potential of the MaramaCRelation approach for improving maintainability and readability of model transformation scripts?*

Evaluation feedback: Promising. This would give a quite different way to structure the scripts vs. current approaches in e.g. ATL, XSLT etc.

Evaluation feedback analysis: As the CRelation model visually categorizes the selection constraints, it has the potential to categorize transformation templates and rules. In Section 10.2.2.2, the CRelation model uses ATL scripts to construct selection constraints. The initial success of leveraging the strength of ATL shows huge potential of using the CRelation model to improve the maintainability of transformation scripts by visualizing and categorizing them.

Q12: *Do you think the explicit representation of associations of StructureMappings is helpful to correctly capture shared semantics during MI&T?*

Evaluation feedback: Very useful.

Evaluation feedback analysis: The SemanticAssociation is an unobtrusive checking mechanism to analyze the intended MI&T specified by the StructureMappings. The SemanticAssociation explicitly represents the associations between StructureMappings; helps to validate the intended MI&T specified in StructureMappings; and records possible semantic inconsistency. So far, the similar support has rarely been reported in the existing MI&T technologies. The use of the SemanticAssociation will be better demonstrated by using the MaramaCRelation approach to solve well aware problems such as constructing multi-view environment, generating transformation scripts, extending tools, and refining models.

Q13*: Do you think the explicit representation of associations helpful to record the inconsistent semantics during model integration and transformation?*

Evaluation feedback: Helpful.

Evaluation feedback analysis: The SemanticAssociation makes it possible to detect and record the semantic inconsistencies brought by the StructureMappings. The SemanticAssociation detects inconsistency before the MI&T is realized, and relates the maintenance of semantic consistency with the intended MI&T. For example, in the future, when the CRelation model is extended to be able to generate transformation scripts, the SemanticAssociations will be able to detect and record semantic inconsistencies brought by the transformation scripts.

Q14*: How do you rate the usefulness of visually categorizing selection constraints?*

Evaluation feedback: Useful.

Evaluation feedback analysis: The CRelation model provides a well-structured central place to organize the selection constraints, behavior descriptions, and other important issues involved in MI&T. The CRelation model has used ATL scripts to construct selection constraints. It puts ATL scripts into appropriate CRelation StructureMappings; represents the associations between the ATL snippets via SemanticAssociations; and explicitly visualizes what modeling elements are involved in a template or a rule (because a selection constraint can only use the information of the modeling elements of the StructureMapping, or SelectionRefinement, or StructureRefinement). Instead of digging in the spaghetti of ATL scripts, people will be able to quickly understand the rationale and the structure of the ATL scripts.

Q15*: How do you rate the usefulness of separating what from how when interconnecting models?*

Evaluation feedback: Very useful. This is the thing most lacking in current script/code-based approaches – they get completely mixed up.

Evaluation feedback analysis: The MaramaCRelation approach allows people to analyze and design the intended MI&T before putting much effort to realize them. Separating what from how to interconnect models clarifies hard mental operations of software modelers, and improves the maintainability, manageability of the intended MI&T.

Category 4: Is the MaramaCRelation tool easy to use?

Q16: *How do you rate the automatic support that the MaramaCRelation tool provides?*

Evaluation feedback: Good. All the interviewee suggested that supporting visual context within the CRelation model would further improve the quality of the MaramaCRelation tool.

Evaluation feedback analysis: The MaramaCRelation approach is intended to analyze and design MI&T effectively and efficiently. The MaramaCRelation tool automatically calculates a list of available meta-model elements and constructs; calculates the available paths between two constructs of a meta-model; partially automatically generates behavior descriptions; automatically generates traceability search conditions; and finds out eligible target model candidates for a source model element. More automatic support is wanted to provide visual context within the CRelation model to reduce the learning curve and improve the productivity of the CRelation modeling.

Q17: *How do you rate the traceability mechanism of the MaramaCRelation approach?*

Evaluation feedback: Good. This looks a good advance and uses the inter-model relationships well.

Evaluation feedback analysis: The CRelation model is a high level central place to review usually isolated tasks in MI&T. The model specifies the Interconnection Relationships between two domain-specific meta-models; and uses the Interconnection Relationships to guide the maintenance of traceability. The MaramaCRelation approach can establish traceability between any domain-specific models. Instead of turning to coding, people can: create a CRelation model; generate search conditions from it; and find out eligible target model elements for the interested source model elements. The CRelation model relates traceability mechanism with other operational tasks of MI&T. For example,

when the CRelation model uses ATL transformation scripts as selection constraints, it can use the ATL scripts to generate search conditions and establish traceability between the source and target models.

***Q18**: How do you rate the usefulness of the behavior information of StructureMappings?*

Evaluation feedback: Useful. All interviewees suggested exporting the well-structured behavior information to the third party event-modeling environments to visually construct behavior synchronization mechanisms for MI&T.

Evaluation feedback analysis: The CRelation model is a high level model. The MaramaCRelation approach is aimed for leveraging third party technologies to solve problems involved in MI&T. Behavior synchronization is an important issue involved in MI&T. The behavior descriptions generated from the CRelation model can be used as well-structured functional requirements to feed the existing visual event-handling environment to visually construct behavior synchronization mechanisms for MI&T.

Category 5: How to use the MaramaCRelation approach to improve software engineering?

***Q19:** How do you rate the potential of using the MaramaCRelation approach to decompose the monolithic, programming-intensive approach of constructing multi-view environments?*

Evaluation feedback: Very promising.

Evaluation feedback analysis: Traditionally, the construction of multi-view environments is based on programming framework and very programming intensive (e.g. Rational Rose, ArgoUML). The main tasks in constructing a multi-view environment include traceability maintenance, and model and behavior synchronization. The traditional multi-view environments use low-level data repositories to maintain traceability and model synchronization. The CRelation model can be viewed as a high-level repository that holds the data shared by the interconnected models. The case studies in Chapter 11 demonstrate how to use CRelation models to construct basic multi-view environments. An interesting future project will be using a well-established multi-view environment e.g. Rational Rose as a

benchmark to test to which extent that the MaramaCRelation approach can help to construct multi-view environments.

Q20: *Rate the potential for MaramaCRelation to provide structured, high level, and visual support for tool extension*

Evaluation feedback: Very promising

Evaluation feedback analysis: Tool extension is normally done at implementation level to achieve a set of functional goals. The CRelation model helps to extend tools by addressing the rationale and functional requirements (e.g. mapping sets, behavior descriptions, selection constraints) at high level. For experienced tool developers, the CRelation model can be used as structured functional requirements, so people can understand the rationale, the selection constraints, and the behavior information of the intended tool extension. For inexperienced tool developers, an existing CRelation model can be an effective guideline for the implementation of the intended tool extension, so they can follow the structured instructions to implement the functional requirements.

Q21: *Rate the potential for MaramaCRelation to provide structured behavior descriptions to feed third party event-handling technologies*

Evaluation feedback: Promising.

Evaluation feedback analysis: Visual event modeling environments have been researched extensively (Liu et al, 2007). Feeding the visual event modeling environments with the behavior information of the CRelation model makes it possible to visually construct full-blown behavior synchronization mechanisms for MI&T, model refinement, and multi-view support.

12.3 Summary

The initial feedback of the MaramaCRelation approach has been very positive. It takes some time for the users to understand and accept the motivations of the MaramaCRelation approach. They agree that the MaramaCRelation is an innovative technology to provide useful high level support for MI&T; and they

are interested in using the CRelation model to reorganize the issues that are currently treated as separate operational tasks, such as semantic consistency maintenance, traceability maintenance, and model and behavior synchronization. The interviewees give positive feedback to the features including: well-structured rationale; well categorized selection constrains; explicit representation of the associations involved in the intended MI&T; well-structured behavior descriptions; the generation of search conditions; and the easy traceability establishment. They have given valuable suggestions for the future work, including: improving the support for visual context in the CRelation model; using well-accepted transformation technologies to define selection constraints; and feeding third party event modeling environments to visually construct behavior synchronization mechanisms for MI&T. The feedback of the evaluation confirms the contributions of the MaramaCRelation research, and also prompts the future work for the research.

Chapter 13 - The Future Work of the MaramaCRelation Research

The MaramaCRelation approach provides high abstraction level solutions to the usually separated operational tasks involved in MI&T. Its strength and potential will be improved and better demonstrated when solving more concrete tasks such as visualizing and generating transformation scripts, supporting model refinement, and supporting multi-view environment. This chapter introduces the viable future projects that will explore the potential of the MaramaCRelation research.

13.1 Using the CRelation model to generate model transformation scripts

This will be the first natural move from the current status of the MaramaCRelation research. Section 10.2.2 has achieved positive results in using ATL to construct selection constraints for the CRelation model. The success shows the potential of the CRelation model visualizing and categorizing ATL scripts, as well as generating ATL transformation scripts. It is expected that the CRelation model's generating ATL scripts would be similar to the UML class diagram's generating Java classes. XSLT is also a very popular transformation language. Compared with ATL scripts, XSLT scripts are even harder to read and maintain. The initial success of using ATL in the CRelation model foresees the high chance of success of generating XSLT scripts from the CRelation model. VIATRA uses a graph-based transformation language. It will be interesting to explore how the diagrammatic CRelation model supports analysis and design of graph-based model transformation.

13.2 Formalizing the used algorithms and definitions

The MaramaCRelation tool uses empirical definitions and algorithms when supporting the CRelation modelling, including: the empirical definition of the paths between two construct vertices (refer to Section 9.4.5.2.1); the empirical algorithm to find out the model elements that are truly influential on the selection constraint results (refer to Section 10.2.3.1); the empirical definition of behaviour information

XML schema (refer to Section 9.4.1.4); and the empirical algorithm to translate *translatable semantic constraints* (refer to Section 10.3.2). For example, in Chapter 10, the MaramaCRelation tool rewrites the selection constraints by using an empirical algorithm. The algorithm has been successful in all of the sample cases, but there is no proof of how generic it is to retrieve the influential modelling information from any valid selection constraint. The empirical solutions need to be formalized to improve the accuracy of the CRelation modelling technology and provide more generic solutions for the problems encountered in the development of the CRelation modelling.

Another example is that the CRelation model uses an empirical definition to specify the paths between two construct vertices of a domain-specific meta-model (refer to Section 9.4.5.2.1). The empirical definition needs to be formalized by the knowledge of graph theory. The formalized definitions will improve the accuracy of the semantics of the SemanticAssociation and the effectiveness of the algorithm that calculates the available paths for the *associationMapping* property of the SemanticAssociation.

13.3 Providing visual context within the CRelation model

The CRelation model intensively communicates with the involved source and target meta-models. Currently, the CRelation model provides the limited visual context support across the models that are displayed in different Eclipse editors. As some of the CRelation model entities are not straightforward to understand, such as the *entityMapping* and *selectionConstraints* properties of the StructureMapping, and the *associationMapping* and *semanticTranslation* properties of the SemanticAssociation, it is very important to provide effective visual context within the CRelation model. The effective visual context within the CRelation model will allow users to access every piece of source and target meta-model information from within a CRelation model element. The context support within the Eclipse java programming environment (e.g. access to Java API, display method call hierarchy) will be the good benchmark for building up visual context support within the CRelation model.

13.4 Developing comprehensive running case studies

Model Integration and Transformation (MI&T) have been researched extensively. But it is hard to find the substantial running case studies of MI&T involving multiple models with various domains and abstraction-levels. Comprehensive case studies engage users instantly and can explain research concepts effectively. The Travel Planner case study in Chapter 11 of interconnecting four domain-specific models

via three CRelation models is a good beginning, but the models involved in the Travel Planner case study are too simplified. It is planned to develop large, real, complicated projects such as the Travel Planner to demonstrate the strength and explores the potential of the MaramaCRelation research.

13.5 Extending the CRelation model to support functional integration

The CRelation model is at its initial stage, and ready to extend to analyze and design more issues involved in MI&T. Supporting functional integration among domain-specific models will be a strong motivation for extending the CRelation model. A sample functional integration is to support the integration of the code generated from different domain-specific models. More specifically, in the MaramaMTE+ project, the Form Chart model generates client side program of web applications; and the architecture model generates server side program of web applications. The code is integrated manually in MaramaMTE+ in order to function conforming to users' mindset. It will be interesting to see if the CRelation model can be extended to provide structured, automatic support for this kind of functional integration.

13.6 Using the third party environments to construct behaviour synchronization mechanisms

In Chapter 11, the MaramaCRelation tool uses a very simple mechanism to maintain behaviour and view synchronization between the interconnected models. This mechanism is based on the behaviour descriptions of the CRelation StructureMappings, and simply passes the synchronization messages across the models. By leveraging the third party event-modelling technologies, the behaviour descriptions can be used to construct much advanced behaviour synchronization mechanisms. Katiaki (Liu et al, 2007), for example, is an advanced technology of event modelling. Katiaki provides visual notations for users to construct complicated events that may happen in a model. The CRelation behaviour descriptions provide the structured event information to feed Kaitiaki, and users can then design Katiaki models to process the events, such as passing around the information across the models; deleting the target model elements; or changing the property values of the target modelling elements. Combining the CRelation modelling with the Katiaki modelling will establish view and behaviour synchronization visually, structurally, and systematically.

13.7 Layered software architecture for multi-view environments

Many traditional multi-view environments are built on the low-level programming framework. The MaramaCRelation research motivates layered software architecture for building up flexible and adaptable multi-view environments. The layered software architecture would be consisted of: 1) MaramaCRelation users use a CRelation model to capture the requirements (through shared semantics) for a multi-view environment; 2) MaramaCRelation tool developers develop code generation scripts to generate MaramaCRelation search conditions; 3) MaramaCRelation users model behaviour synchronization mechanisms by using third party environments (e.g. Kaitiaki); and 4) the MaramaCRelation tool developers develop code generation scripts to generate behaviour synchronization program. When the multi-view environment needs to support a new domain-specific modelling technology, the tool users construct new CRelation models, and the tool developers develop new code generation scripts. The MaramaCRelation search conditions (java classes) are well-structured, and it is easy to adapt the existing code generation scripts to support new domain-specific knowledge.

The layered software architecture breaks down the monolithic task of multi-view environment development; and the workload between MaramaCRelation tool users and tool developers is balanced. The CRelation model is a high abstraction level presentation of the requirements and rationale of the multi-view environment. The layered architecture will be well modularized, loosely connected, and more flexible and adaptable than the traditional multi-view environments.

13.8 Supporting model refinement using the MaramaCRelation approach

One of the main goals of Model Driven Engineering (MDE) is to bridge the gap between informal, business-focused tasks (e.g. business analysis, requirement analysis) and low-level application design activities (e.g. design and implementation). A typical model-driven development process of the Travel Planner system may consist of: using the BPMN model to specify business processes (high level model, close to the business end); using the MaramaMTE+ models to specify software architecture and client behavior (high level model, in the software development domain); using the Web Service model to specify web services (high level model, in the software development domain); and using the UML OOD class diagrams to specify detailed Object Oriented classes (e.g. java classes). It is important to enable correct and consistent refinement of informal tasks into executable systems, and maintain the traceability of models across levels of refinement. At this stage, the support for refinement and traceability among

models is limited. It is difficult to refine a BPMN model to a MaramaMTE+ architecture model; to refine a Web service model to a MaramaMTE+ model; and to show the refining process from MaramaMTE+ to the java test bed.

The MaramaCRelation approach provides a high-level support to interconnect *any* models by capturing the shared semantics, recording the behavior synchronization information, and establishing traceability. The Interconnection Relationships among various models can be understood as the refinement relationships among the models. It is planned to draw a list of requirements from a benchmark refining mechanism, and then compare the MaramaCRelation-based refinement mechanism with the benchmark one. It will be interesting to see to which extent the MaramaCRelation approach can support a sound and flexible refinement mechanism.

13.9 Summary

All the future projects listed in the chapter are based on the initial success of the MaramaCRelation research. They are viable and will improve the strength of the MaramaCRelation research. It will be very interesting to see: how to use the MaramaCRelation approach to visualize and generate transformation scripts; to which extent the MaramaCRelation approach can support modelling refinement; to which extent the MaramaCRelation approach can support the construction of multi-view environments; and how to feed third party event-modelling environments with the behaviour information to visually construct behaviour synchronization mechanisms. These immediate future projects, in turn, will motivate more interesting further research.

Chapter 14 - Conclusions

Software architecture modelling and architecture performance evaluation have become very important in Software Development Life Cycle. The author's previous SoftArch/MTE research evaluates the performance of software architecture by generating test beds from software architecture models. The initial success of SoftArch/MTE technology prompted problems that need further research, including: how to support the test bed generation and performance evaluation using well-established CASE tools; how to analyze web user behaviour and use the analysis to improve the accuracy of software architecture performance evaluation; and how to provide high level support for model integration, transformation, and refinement. The three projects in this thesis are aimed for solving those problems. The ArgoMTE project improves the usability of the SoftArch/MTE technology by integrating it with the well-established ArgoUML tool. The MaramaMTE+ project automatically generates basic structure of the Form Chart model via web reverse engineering. The MaramaMTE+ generates third party load testing scripts from the Form Chart model and uses the Form Chart model to improve the accuracy of software architecture performance evaluation. The MaramaCRelation approach provides high level support to interconnect *any* domain-specific models, which can be used to support the model integration, transformation, and refinement. The three projects fall in the paradigm of Model Driven Engineering; and have made contributions to the research areas of: software architecture modelling and performance evaluation; realistic web user behaviour analysis and web load testing; and analysis and design of model integration and transformation. This chapter reports the conclusions drawn from the three projects.

14.1 The ArgoMTE project

The ArgoMTE project extends the well established ArgoUML tool to support software architecture modelling, test bed generation, and performance evaluation. ArgoMTE has significantly improved the SoftArch/MTE technology and shown the strength to deal with complicated, large cases. The conclusions drawn from the ArgoMTE research are as follows:

- *Usability of the SoftArch/MTE technology has been hugely improved*

The main incentive for the ArgoMTE research is to integrate the technology of test bed based performance evaluation with a well-established UML-based CASE tool. The ArgoMTE tool provides a much more appealing and effective environment than the previous stand-alone SoftArch/MTE tool. The ArgoMTE tool extends the UML meta-model, and uses UML class diagram look-like visual notations to model software architecture and generate test bed, which hugely reduces the learning load of the tool users. The ArgoMTE tool allows the software architecture modeling to leverage the existing UML OOA and OOD modeling via model and tool integration.

- *Using standard data format to achieve better data exchange*

The ArgoMTE tool extends XMI standard to save software architecture meta-models and models, which has increased the chance of data exchange between the ArgoMTE-styled software architecture models and UML OO models. The extended-XMI format can not use the standard XMI writer and reader to exchange data with other UML models, and still need its own writer, reader, and other support facilities. But the standard XMI paradigm provides the perfect guidelines for the extended-XMI to follow, which makes the standard XMI and the extended XMI very similar and easy to exchange data.

- *Leveraging third party tools to improve the maintainability and flexibility of complicated performance evaluation processes*

The ArgoMTE-styled performance evaluation process is tedious and error-prone. The process involves: test bed generation, test bed compilation and deployment, test bed execution, and result collecting and visualization. ArgoMTE uses the Ant build manager tool to automate the process, which greatly eased this complicated process. The use of third party tools to coordinate the test bed generation and execution process (Ant), deployment (SFTP), and web based client tests (Microsoft ACT) has proved much more scalable and flexible than using DOS batch files to perform these tasks in SoftArch/MTE.

- *Using a conceptual framework to improve the evolvability of test bed generation and performance evaluation*

The ArgoMTE approach uses a domain-specific meta-model to specify domain-specific knowledge. The meta-model manages essential information for test bed generation, including: rules for test bed

generation, test bed generation scripts, critics for validity of the architecture design and test bed, and test bed generation logic (refer to Section 5.3.1). An ArgoMTE domain-specific meta-model may evolve with new architectural concerns coming up. When a domain-specific meta-model evolves, its code generation scripts and logic need to evolve. The ArgoMTE approach provides a conceptual framework to support the evolvement of a domain-specific meta-model, and its code generation scripts and logic. This structured conceptual framework helps users to develop, modify, and reuse domain-specific meta-model in the ArgoMTE's meta-model specification tool.

The ArgoMTE research shows how commonly used Components Off The Shelf (COTS) or Open Source Off The Shelf (OSOTS) tools can improve the usability and maintainability of an in-house technology, and how the use of standard model representations can improve the tool integration. The ArgoMTE research motivated the MaramMTE+ to provide better model and tool integration.

14.2 The MaramaMTE+ project

The MaramaMTE tool, constructed using the Marama meta-tool, rebuilds the technology of test bed generation and performance evaluation of ArgoMTE. The Marama meta-tool supports efficient tool construction, and theoretically can build complicated tools like ArgoUML. The Marama meta-tool makes it easy for the MaramaMTE tool to integrate the software architecture modelling with broad range of other software modelling technologies (e.g. the UML modelling, the BPMN modelling, and the Form Chart modelling). The MaramaMTE approach leverages the strength of the Form Chart modelling to improve software architecture performance evaluation and support web load testing. The MaramaMTE+ improves the research of MaramaMTE by automatically generating the basic structure of Form Chart models, and generating well-analyzed test scripts for third party web load testing tools. The conclusions drawn from the MaramaMTE+ research are as follows:

- *The basic structure of the Form Chart model can be automatically generated via web crawling*

The Form Chart model formally analyzes the behavior of web application users. Manually constructing a Form Chart model can be tedious and error-prone, especially when the target website has complicated structure. The MaramaMTE+ approach retrieves target website information via web crawling, records the retrieved website structural data in the purpose-built database, and automatically generates the basic

structure of Form Chart models. The MaramaMTE+ has used automatic web reverse engineering to improve the efficiency of Form Chart model construction.

- *The Form Chart model can generate well-analyzed test plans for 3rd party stress testing tools*

Constructing web load testing plans and scripts has always been tedious and error-prone. The MaramaMTE+ approach has successfully used Form Chart models to generate JMeter-formatted and test-bed-formatted test plans. The Form Chart model has the potential to generate test plans for other third party testing tools. The test plans generated from the Form Chart model are not ordinary test plans, but the well analyzed ones that capture the realistic behavior of web users.

- *The MaramaMTE+ approach supports effective performance-oriented reverse-engineering*

The MaramaMTE+ approach has shown potential in performance-focused web application reverse engineering. It efficiently generates the basic Form Chart models from legacy web applications, which provides well-analyzed testing plans to evaluate the server-side software architecture designs. The MaramaMTE+ approach makes it efficient to compare the legacy server-side software architecture (may also be retrieved via reverse engineering) with intended, optimized server-side architecture designs.

MaramaMTE and MaramaMTE+ have integrated the traditional software architecture modelling with the web user behaviour modelling. They have shown how model integration can extend the applicable domain of a specific software engineering model (e.g. the MaramaMTE+ architecture model, the Form Chart model); and how the Marama meta-tool supports efficient tool extension. But the model integration and tool extension in both the MaramaMTE and MaramaMTE+ projects are done at implementation level and requires hard mental operations. The MaramaCRelation research was then proposed to provide high level support for model integration and tool extension.

14.3 The MaramaCRelation project

The MaramaCRelation project is designed to provide a structured approach to interconnect domain-specific models. The CRelation model maintains the rational of the interconnection of domain-specific models; records semantics maintained and lost across the interconnected models; tracks the evolvement of modeling elements through the traceability across the interconnected models; and maintains behavior

synchronization across the interconnected models. The conclusions drawn from the MaramaCRelation research are as follows:

- *It is useful and important to use an analysis and design abstraction level to guide the operational tasks of Model Integration and Transformation.*

The importance of UML models in software engineering is still arguable, but it does not stop people from using them to analyze and design Object Oriented (OO) Development. UML models specify isolated OO development tasks from a high abstraction level, which makes UML models the central places for software analysis and design, code generation, performance evaluation, and so on.

The current status of Model Integration and Transformation (MI&T) is like OO development without the support of UML models. Most of MI&T technologies are developed to achieve operational goals including: transforming models from one format to another, setting up traceability among models, combining and coordinating source code generated from various models, and synchronizing view and behaviour of the integrated and transformed models. An analysis and design level of MI&T would benefit MI&T research as UML models benefit OO development. The CRelation model of the MaramaCRelation approach has achieved positive results to be a central place for the important issues related to MI&T, including: capturing the semantics conceptually shared by two models, recording the rationale for an intended MI&T, detecting semantic inconsistencies during the intended MI&T, and establishing traceability and behaviour synchronization across models.

- *The maintainability of MI&T can be hugely improved by visually categorizing selection constraints*

The CRelation model uses StructureMappings, SelectionMappings, and StructureRefinements to visually categorize the selection constraints of an intended MI&T. The CRelation model has successfully used ATL scripts to construct selection constraints, which shows huge potential for the CRelation model to visualize and manage other third party transformation scripts.

- *The semantic inconsistency during MI&T can be detected and recorded.*

The CRelation model establishes associations between two StructureMappings, and explicitly represents the association information that is implied but ignored in the traditional model transformation technologies. The CRelation model uses a SemanticAssociation to explicitly represent the information

implied by two StructureMappings, which allows tool users to check if the intended StructureMappings are correctly designed, or what semantic inconsistencies have to be tolerated. The CRelation model *detects* semantic inconsistency, which is different from the traditional consistent transformation and consistency comparison

- *Traceability can be maintained from high level diagrammatic models.*

The traceability mechanism of the MaramaCRelation is flexible and adaptable. It consists of two layers: the CRelation model layer and the search condition generation layer. When interconnecting two domain-specific meta-models, tool users design the CRelation model and the MaramaCRelation tool developers develop the code generation scripts (Eclipse JET scripts). The layered architecture breaks down the monolithic programming-intensive traceability mechanism, and allows tool users to manage traceability of MI&T through a high level model (the CRelation model).

- *The events triggered by an MI&T can be automatically retrieved.*

The CRelation model automatically retrieves all the possible "removed" and "changed" events triggered by the interconnected models. The event information of the CRelation model represents well structured functional requirements for a behaviour synchronization mechanism.

- *The MaramaCRelation approach supports a flexible and adaptable multi-view environment structure*

One of the main goals of the MaramaCRelation research is to build up flexible and adaptable multi-view environment. The layered software architecture of the multi-view system would consist of : 1) MaramaCRelation users use a MaramaCRelation model to capture the requirements (through shared semantics) for the multi-view environment; 2) MaramaCRelation tool developers develop code generation scripts to generate MaramaCRelation search conditions; 3) MaramaCRelation users model behaviour synchronization mechanisms by using third party mechanisms (e.g. Kaitiaki); and 4) the MaramaCRelation tool developers develop code generation scripts to generate behaviour synchronization code for the used third party model. The high abstraction level of the MaramaCRelation approach improves the flexibility and adaptability of the MaramaCRelation-based layered multi-view system.

- *The MaramaCRelation approach supports a flexible and adaptable model refinement mechanism*

One of the main goals of Model Driven Engineering (MDE) is to bridge the gap between informal, business-focused tasks (e.g. business analysis, requirement analysis) and low-level application design activities (e.g. design and implementation). It is important to enable correct and consistent refinement of informal tasks into executable systems, and maintain the traceability of changes across levels of refinement. The MaramaCRelation approach provides a high-level support to interconnect *any* models by capturing the shared semantics, recording the behavior synchronization information, and establishing traceability. It is a well-structured place to record essential information involved in model refinement. The MaramaCRelation approach has the huge potential to support sound and flexible refinement mechanisms.

14.4 Summary

The thesis has made contributions in software architecture design, software architecture performance evaluation, web load testing, and model integration and transformation. The research results have been demonstrated and evaluated in the thesis. The main aims of the projects are to improve the automatic support, analysis and design support, and systematic and structured support for the interested software engineering modelling technologies.

The three projects presented in the thesis, bring innovative technologies, find new use for the existing technologies, and abstract problem domains at various levels. All three projects, especially the MaramaCRelation project, motivate interesting and promising future projects, which will lead more useful contributions towards Model Driven Engineering.

References

[Allen, 1997]
R. Allen, "*A Formal Approach to Software Architecture*.", Ph.D. Thesis, Carnegie Mellon University, CMU Technical Report CMUCS-97-144, May 1997.

[Amar et al, 2008]
Bastien Amar, Hervé Leblanc, Bernard Coulette, *A Traceability Engine Dedicated to Model Transformation for Software Engineering*, ECMDA'08 - Traceability Workshop, June, 2008

[Apache Ant, 2004]
Apache Ant, http://ant.apache.org/

[Apache JMeter, 1999]
Apache Jmeter, http://jakarta.apache.org/jmeter/

[Apache Xalan, 2004]
Apache Xalan, http://xml.apache.org/xalan-j/

[ArgoUML, 2003]
ArgoUML, http://argouml.tigris.org/

[ATLAS Transformation, 2006]
ATLAS, www.eclipse.org/m2m/atl

[Bakken, 2003]

David E. Bakken, *Encyclopedia of Distributed Computing*, Kluwer Academic Press, 2003.

[Balasubramanian et al, 2006]
Krishnakumar Balasubramanian, Jaiganesh Balasubramanian, Jeff Parsons, Aniruddha Gokhale, Douglas C. Schmidt, *A Platform-Independent Component Modeling Language for Distributed Real-Time and Embedded Systems*, Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium, Pages: 190 - 199, Year of Publication: 2005, ISBN ~ ISSN:1080-18120-7695-2302-1

[Balsamo et al, 2002]
Balsamo, S., Simeoni, M., Bernado, M. *Combining Stochastic Process Algebras and Queuing Networks for Software Architecture analysis*, Proc 3rd Intl Workshop Software & Performance, 2002, ACM Press

[Balzer, 1985]
Balzer, B. *A 15 year perspective on automatic programming*, IEEE Transactions on Software Engineering, vol. 11 no. 11, Nov 1985, pp.1257-1268.

[Barford et al, 1998]
Barford, P. and Crovella, M. *Generating representative Web workloads for network and server performance evaluation*, Proc 1998 ACM SIGMETRICS Joint Intnl Conference on Measurement and Modeling of Computer Systems, Madison, Wisconsin, 1998, pp. 151-160.

[Bachmann et al, 2000]
Bachmann, F., Bass, L., Carriere, J., Clements, P., Garlan, D., Ivers, J., Nord, R., Little, R., *Software Architecture Documentation in Practice: Documenting Architectural Layers,* available, http://www.sei.cmu.edu/publications/documents/00.reports/00sr004.html

[Beyer, 2005]
Beyer, D. *CCVisu - A Tool for General Force-Directed Graph Layout and Co-Change Visualization*, See http://directory.fsf.org/ccvisu.html

[Binns et al, 1996]
P. Binns, M. Engelhart, M. Jackson, and S. Vestal. "*Domain-Specific Software Architectures for Guidance, Navigation, and Control*.", International Journal of Software Engineering and Knowledge Engineering, vol. 6, no. 2, 1996

[Booch, 1994]
'*Object-Oriented Analysis and Design with Applications'*, Booch G. – 2nd ed., Addison Wesley Publishing Company, 1994

[BPMN, 2004]
Object Management Group, *BPMN and Business Process Management*, available, http://www.omg.org/Documents

[Briand, 1998]
Lionel C. Briand, Khaled El Emam Bernd G. Freimut *A Comparison and Integration of Capture-Recapture Models and the Detection Profile Method*, Proceedings of The Ninth International Symposium on Software Reliability Engineering, Page: 32, Year of Publication: 1998

[Cai and Grundy et al, 2004]
Cai, Y., Grundy, J.C., Hosking, J.G., Dai, X. *Software Architecture Modeling and Performance Analysis with Argo/MTE*, In Proceedings of the 2004 Conference on Software Engineering and Knowledge Engineering, Banff, Canada, June 20-24 2004

[Cai et al, 2004]
Cai, Y., Grundy, J.C. and Hosking, J.G. *Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool*, In Proc 2004 IEEE Intnl Conference on Automated Software Engineering, Linz, Austria, September 20-24, IEEE CS Press, pp. 36-45.

[Cai et al, 2007]

Cai, R., Grundy, J., Hosking, J., *Synthesizing client load models for performance engineering via web crawling*, Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA, SESSION: Model-based development 2, Pages 353-362

[Castell et al, 1998]
N. Castell and A. Hernández. *The Software Requirements Modelling in SAREL*, Fourth International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ'98) , pages 49-56, ISBN 2-87037-272-8, Pisa, Italy, June 1998. (in English)

[Chen et al, 2004]
Kai Chen, Janos Sztipanovits, Sandeep Neema, *Towards Formalizing Domain-specific Modeling Languages,* Model-Integrated Computing Workshop, Exploring the Synergy between MIC and MDA® Arlington, 2004

[Cheng et al, 1994]
Cheng, B. H. C., Wang, E. Y., and Bourdeau, R. H., "*A Graphical Environment for Formally Developing Object-Oriented Software*", Proceedings of International Conference on Tools with AI, Nov. 1994.

[Chikofsky et al, 1990]
EJ Chikofsky, EJ Chikofsky, JH Cross, *Software, IEEE*, Vol. 7, No. 1. (1990), pp. 13-17

[Clements, 1997]
P. C. Clements. "*Working Paper for the Constraints Sub-Group*.", EDCS Architecture and Generation Cluster (http://www.sei.cmu.edu/~edcs/CLUSTERS/ARCH/index.html), April 1997.

[Csertan et al, 2002]
Gyorgy Csertan, Gabor Huszerl, Istvan Majzik, Zsigmond Pap, Andras Pataricza, Daniel Varro, *VIATRA – Visual Automated Transformations for Formal Verification and Validation of UML Models*, in Proceedings of the 17th ASE, page267

[CSIRO, 2000]
CSIRO, http://www.csiro.au/files/mediaRelease/mr2000/Prmiddleware.htm

[Dai et al, 2002]
Dai, S. and Grundy, J.C. *Architecture of a Micro-Payment System for Thin-Client Web Applications*, In Proceedings of the 2002 International Conference on Internet Computing, Las Vegas, June 24-27 2002, CSREA Press.

[Dai et al, 2007]
Dai, X. and Grundy, J.C. *NetPay: An off-line, decentralized micro-payment system for thin-client applications, Commerce Research and Applications*, Elsevier, vol. 6, no. Electronic1, March 2007, Elsevier, pp. 91-101

[Denaro et al, 2004]
Denaro et al, Denaro, G., Polini, A., Emmerich, W. *Early performance testing of distributed software applications*, In Proceedings of the 4th Intnl Workshop on Software and Performance, Jan 14-18 2004, Redwood City, California, pp. 94-103

[Deng et al, 2003]
Gan Deng, Tao Lu, Emre Turkay, Aniruddha Gokhale, Andrey Nechypurenko, *Model Driven Development of Inventory Tracking System*, available, http://www.cs.wustl.edu/~schmidt/PDF/ITS_Modeling.pdf

[Sanches et al, 2008]
Sánchez, M., Villalobos, J., Deridder, D., "*Co-Evolution and Consistency in Workflow-based Applications*" in 1st International Workshop on Model Co-Evolution and Consistency Management, Toulouse, France, 2008

[Di Lucca et al, 2001]

Di Lucca, G.A., Di Penta, M., Antonniol, G. and Casazza, G. (2001), *An Approach for Reverse Engineering of Web-Based Applications*, Proc. 8th Working Conference on Reverse Engineering, WCRE'01, IEEE, pp231-240.

[Draheim et al, 2003]
Draheim, D. and Weber, G., *Modeling Submit/Response Style Systems with Form Charts and Dialogue Constraints*, LNCS Volume 2889/2003, Springer.

[Draheim et al, 2005]
Draheim, D., Lutteroth, C. and Weber, G. (2005), *A Source Code Independent Reverse Engineering Tool for Dynamic Web Sites*, Proc. 9th European Conference on Software Maintenance and Reengineering (CSMR'05), pp168-177.

[Draheim et al, 2006]
Draheim, D., Grundy, J.C., Hosking, J.G., Lutteroth, C. and Weber, G. *Realistic Load Testing of Web Applications*, In Proceedings of the 10th European Conference on Software Maintenance and Re-engineering, Berlin, 22-24 March 2006

[Eclipse 2001]
Eclipse, www.eclipse.org/

[ECPerf&JDBC Benchmark, 2002]
ECPerf&JDBC Benchmark, http://www.datadirect.com/products/jdbc/ecperfan dspecj/index.ssp

[ECPerf, 2002]
ECPerf Performance Benchmarks, August 2002, available, http://ecperf.theserverside.com/ecperf

[Egyed, 2001]
Egyed, A., "*Transformation, Ambiguity, and Trivialization*," Proceedings of the 2nd International Workshop on Living with Inconsistencies (IWLWI), co-located with ICSE 2001, Toronto, Canada, May 2001

[Egyed, 2001_2]
Egyed, A., *Scalable Consistency Checking between Diagrams – The VIEWINTEGRA Approach*, Automated Software Engineering archive. Proceedings of the 16th IEEE international conference on Automated software engineering, Page: 387, Year of Publication: 2001

[Elbaum et al, 2003]
Elbaum, S., Karre, S., Rothermel, G. *Improving Web Application Testing with User Session Data*, In Proceedings of the 2003 International Conference on Software Engineering, IEEE CS Press, 2003.

[Engels et al, 2002]
Gregor Engels, Reiko Heckel, Jochen M. Küster, Luuk Groenewegen, *Consistency-Preserving Model Evolution through Transformations (2002)*, Lecture Notes In Computer Science; Vol. 2460 archive Proceedings of the 5th International Conference on The Unified Modeling Language, Pages: 212 - 226, Year of Publication: 2002

[Falleri et al, 2006]
JR Falleri, M Huchard, and C Nebut (2006), *Towards a Traceability Framework for Model Transformations in Kermeta*. In: ECMDA-TW'06: ECMDA Traceability Workshop, Sintef ICT, Norway, pages31-40.

[Feast, 2002]
Alan Feast, *Using middleware for inter-enterprise integration across the UK electricity supply chain*, available, http://www.middlewarespectra.com/abstracts/5_99_02.htm

[Feldman et al, 2007]
Steve Feldman, Tim Moore, Ryan O'Neil, *Introduction to Software Performance Engineering,* available, *http://*www.edugarage.com/download/attachments/14551531/b2conference_**performance**_2004.pdf?version=1

[Fujaba, 2007]

Fujaba, http://wwwcs.uni-paderborn.de/cs/fujaba/

[Garcia, 2008]
M. Garcia. *Bidirectional synchronization of multiple views of software models*, In Proceedings of DSML-2008, volume 324 of CEUR-WS, pages 7–19, 2008.

[Garlan et al, 1997]
D. Garlan, R. Monroe, and D. Wile. "*ACME: An ArchitectureDescription Interchange Language.*" In *Proceedings of CASC ON'97*, November 1997

[Garlan, 1995]
Garlan, D., "*An Introduction to the Aesop System*." July 1995. http:// www.cs.cmu.edu/afs/cs/project/able/www/aesop/html/aesopoverview.ps

[Geoffrion, 1996]
Geoffrion, Arthur M., *STRUCTURED MODELING: SURVEY AND F UTURE RESEARCH DIRECTIONS*, 1996, available, http: //www.anderson.ucla.edu/faculty/art.geoffrion/home/csts/csts3.htm

[Gorton et al, 2000]
Gorton, I. and Liu, A. *Evaluating Enterprise Java Bean Technology*, In *Proc Software - Methods and Tools*, Wollongong, Australia, Nov 6-9 2000, IEEE

[Graaf et al, 2007]
Bas Graaf, Arie van Deursen, *Model-Driven Consistency Checking of Behavioral Specifications*, Proceedings of the Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, Pages 115-126, Year of Publication: 2007

[Green et al, 1996]
T. R. G. Green and M. Petre. *Usability analysis of visual programming environments: A `cognitive dimensions' framework*, Journal of Visual Languages and Computing, 7:131--174, 1996.

[Greenfield, 2001]
Greenfield, J., *UML profile for EJB,* 2001, available, http://www.jeckle.de/files/**UMLProfileForEJB**PublicDraft.pdf

[Grundy and Cai et al, 2001]
Grundy, J.C., Cai, Y. and Liu, A. *Generation of Distributed System Test-beds from High-level Software Architecture Descriptions*, Proc 2001 IEEE Intl Conf on Automated Software Engineering, San Diego, CA, Nov 26-29 2001

[Grundy and Bai et al, 2003]
Grundy, J.C., Bai, J., Blackham, J., Hosking, J.G. and Amor, R. *An Architecture for Efficient, Flexible Enterprise System Integration*, Proc 2003 Intl Conf on Internet Computing, Las Vegas, June 23-26 2003, CSREA Press, pp. 350-356

[Grundy and Cai et al, 2005]
Grundy, J.C., Cai, Y. and Liu, A. *SoftArch/MTE: Generating Distribute d System Test-beds from High-level Software Architecture Descriptions*, Automated Software Engineering, Kluwer Academic Publishers, vol. 12, no. 1, January 2005, pp. 5-39. PDF

[Grundy and Hosking et al, 2000]
Grundy, J.C. and Hosking, J.G., *High-level Static and Dynamic Visualization of Software Architectures*, In Proceedings of 2000 IEEE Symposium on Visual Languages, IEEE CS Press.

[Grundy and Hosking et al, 2006]
Gundy, J.C., Hosking, J.G., Zhu, N. and Liu, N., *Generating Domain-Specific Visual Language Editors from High-level Tool Specifications*, In Proceedings of the 2006 IEEE/ACM International Conference on Automated Software Engineering, Tokyo, 24-28 Sept 2006, IEEE

[Grundy et al, 1996]

Grundy, J.C. and Hosking, J.G. *Constructing Integrated Software Development Environments with MView s*, International Journal of Applied Software Technology, Vol. 2, No. 3/4, International Academic Publishing Company, pp. 133-160

[Grundy et al, 1998]
Grundy, J.C., Hosking, J.G., and Mugridge, W.B. (1998). *Inconsistency Management for Multiple-View Software Development Environments*, IEEE Transactions on Software Engineering, 24(11), 960-981

[Grundy, Hosking, and Li et al, 2006]
Grundy, J.C., Hosking, J.G., Li, L. And Liu, N. *Performance engineering of service compositions*, ICSE 2006 Workshop on Service-oriented Software Engineering, Shanghai, May 2006.

[Hann, 2008]
Johan den Haan, *DSL and MDE, necessary assets for Model-Driven approaches*, available, http://www.theenterprisearchitect.eu/archive/2008/08/11/dsl-and-m-model-driven-approaches de-necessary-assets-for

[Hu et al, 1997]
Hu, L., Gorton, I., *A performance prototyping approach to designing concurrent software architectures*, In Proc of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems, IEEE, pp. 270 – 276.

[IBM Rational, 2008]
Rational Architect, available, http://www-01.ibm.com/software/awdtools/architect/swarchitect/

[IBM, 2001]
Rational Rose, available, http://www-01.ibm.com/software/awdtools/developer/ rose/

[IBM, 2005]
Kim Letkeman, *Comparing and merging UML models in IBM Rational Software Architect: Part 3*, 2005, available, http://www.ibm.com/developerworks/rational/library/05/802_comp3/

[Jackson et al, 2005]
Robert Jackson, *Security and Cost Considerations of Using Middleware to Facilitate SAS Integration,* 2005, available, http://www2.sas.com/proceedings/sugi31/013-31.pdf

[Java EE, 2007]
J2EE, http://java.sun.com/javaee/

[Java RMI, 1999]
Java RMI, http://java.sun.com/javase/technologies/core/basic/rmi/ index.jsp

[Khare et al, 2001]
Rohit Khare, Michael Guntersdorfer, Nenad Medvidovic, Peyman Oreizy, and Richard N. Taylor, *xADL: Enabling Architecture-Centric Tool Integration With XML*, In Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34), 2001

[Kirwan et al, 2008]
Pat Kirwan, Jeannine Siviy, Lisa Marino, and John Morley, *Implementation Challenges in a Multimodal Environment*, 2008, available, http://www.sei.cmu.edu/prime/documents/multimodelSeries_wp5_implementation_052008_v1.pdf

[Kocsis et al, 2006]
VIATRA2 Framework, An Eclipse GMT Subproject, see, http://www.eclipse.org/gmt/

[Kruchten, 1995]
P. B. Kruchten. *The 4+1 view model of architecture*, IEEE software, pages 42-50, Nov., 1995

[Kuster, 2004]
J. M. Küster, *Systematic Validation of Model Workshop in Software Model Transformations*, Proceedings 3rd UML Engineering (WiSME2004), Lisbon, Portugal, October2004.

[Len Bass et al, 2003]
Len Bass, Paul Clements, Rick Kazman, *Software Architecture In Practice*, Addison-Wesley Professional; 2 edition (April 19, 2003)

[Li et al, 2007]
Li, L., Hosking, J.G., Grundy, J. C., *Visual Modeling of Complex Business Processes with Trees, Overlays and Distortion-Based Displays*, In Proceedings of the 2007 IEEE Symposium on Visual Languages and Human-Centric Computing, USA, IEEE CS Press

[Liu et al, 2005]
Yan Liu, Alan Fekete, Ian Gorton, "*Design-Level Performance Prediction of Component-Based Applications,*" IEEE Transactions on Software Engineering, vol. 31, no.11, pp. 928-941, November, 2005. http://doi.ieeecomputersociety.org/10.1109/TSE.2005.127

[Liu et al, 2007]
Liu, N., Hosking, J.G. and Grundy, J.C., *MaramaTatau: extending a domain specific visual language meta tool with a declarative constraint mechanism*, In Proceedings of the 2007 IEEE Symposium on Visual Languages and Human-Centric Computing, USA, Sept 23-27 2007, IEEE CS Press

[Liu et al, 2007]
Na Liu , John Hosking ,John Grundy , *A Visual Language and Environment for Specifying User Interface Event Handling in Design Tools*, Proceedings of the eight Australasian conference on User interface - Volume 64, Ballarat, Victoria, Australia, Pages: 87 - 94, Year of Publication: 2007

[Luckham et al, 1995]
D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. *"Specification and Analysis of System Architecture Using Rapide.",* IEEE Transactions on Software Engineering, vol. 21, no. 4, pages 336-355, April 1995.

[MacKenzie, 2002]

G. MacKenzie, G. Schulmeyer, and L. Yilmaz, *"Verification Technology Potential with Different Modeling and Simulation Development and Implementation Paradigms"*, V&V State of the Art: Proc. of Foundations '02, a Workshop on Modeling and Simulation Verification and Validation for the21st Century (CD), The Society for Modeling and Simulation International (SCS), 2002.

[Magee et al, 1995]
Magee, N. Dulay, S. Eisenbach, and J. Kramer. "*Specifying Distributed Software Architectures*", In Proceedings of the Fifth European Software Engineering Conference (ESEC'95), Barcelona, September1995.

[Marama meta-tool, 2007]
Marama meta-tool, https://wiki.auckland.ac.nz/display/csidst/Marama+Meta-tools

[MaramaMTE, 2007]
MaramaMTE, https://wiki.auckland.ac.nz/display/csidst/MaramaMTE

[MaramaTorua, 2007]
MaramaTorua, https://wiki.auckland.ac.nz/display/csidst/Marama+Torua

[Markovian, 1986]
Ibrahim H. Önyüksel, Keki B. Irani, *Markovian Queueing Network Models for Performance Analysis of a Single-Bus Multiprocessor System*, IEEE Transactions on Computers archive, Volume 39 , Issue 7 (July 1990), Pages: 975 - 980, Year of Publication: 1990

[MDT, 2008]
MDT, http://www.eclipse.org/modeling/mdt/?project=ocl

[Medvidovic et al, 1996]
N. Medvidovic, P. Oreizy, J.E. Robbins, and R. N. Taylor. *Using Object-Oriented Typing to Support Architectural Design in the C2 Style,* In Proceedings of FSE4, pages 24-32, San Francisco, CA, Octorber, 1996

[Medvidovic et al, 2000]
Nenad Medvidovic, Richard N. Taylor, *A Classification and Comparison Framework for Software Architecture Description Languages*, IEEE Transactions on Software Engineering archive, Volume 26 , Issue 1 (January 2000), Pages: 70- 93, Year of Publication: 2000

[Mensace et al, 2002]
Menasce, D.A., *Load testing of web sites*, IEEE Internet Computing, Jul/Aug 2002, vol. 6, no. 4, pp. 70-74.

[Microsoft, 2002]
Web Application Stress Tool, See: http://www.microsoft.com/downloads/details.aspx?familyid=e2c0585a-062a-439ea67d75a89aa36495&displaylang=en

[Miller et al, 1998]
Miller, R.C. and Bharat, K. SPHINX: *A Framework for Creating Personal, Site-Specific Web Crawlers*. In Proceedings of WW7, Brisbane Australia, April 1998, See: http://www.cs.cmu.edu/~rcm/websphinx/

[Model Driven Engineering]
http://en.wikipedia.org/wiki/Model_Driven_Engineering

[MOF, 2008]
MOF, http://www.omg.org/technology/documents/formal/mof.htm

[Moriconi et al, 1997]
M. Moriconi and R. A. Riemenschneider. "*Introduction toSADL 1.0: A L anguage for Specifying Software Architecture Hierarchies*." Technical Report SRI-CSL-97-01, SRI International, March1997

[MS .NET, 2007]

MS.NET, http://www.microsoft.com/NET/

[MS COM/DCOM, 2007]
MS COM/DCOM, http://www.microsoft.com/com/default.mspx

[MSDN, 2002]
Microsoft, *Using .NET to implement Sun Microsystem's Java Pet Store J2EE BluePrint application*, October 2002, http://msdn.microsoft.com/library/default.asp? url=/library/en-us/dnbda/html/psimp.asp.

[Nimmagadda et al, 1999]
Nimmagadda, S., Liyanaarachchi, C., Gopinath, A., Niehaus, D. and Kaushal, A. *Performance patterns: automated scenario based ORB performance evaluation*, Proc 5th USENIX Conf on OO Technologies & Systems, USENIX, 1999, 15-28.

[OMG MDA, 2001]
OMG MDA, http:// www.omg.org/mda/

[OMG QVT, 2001]
MOF QVT, Final Adopted Specification, 2001, available, www.omg.org/docs/ptc/05-11-01.pdf

[OMG, 1995]
CORBA, Common Object Request Broker Architecture, OMG, July, 1995, see www.corba.org/

[Oracle, 2006]
The state of middleware 2006, available, www.oracle.com/products/middleware/docs/state-of-middleware-part2.pdf

[Patel et al, 2007]

Reshma Patel, Frans Coenen, Russell Martin, Lawson Archer, *Reverse Engineering of Web Applications: A Technical Review*, 2007, available, http://www.csc.liv.ac.uk/research/techreports/tr2007/ulcs-07-017.pdf

[Petriu et al, 2000]
Petriu, D., Amer, H., Majumdar, S., Abdull-Fatah, I., *Using analytic models for predicting middleware performance*, In *Proc 2nd Intl Wkshop on Softwareand Performance*, A CM 2000, pp.189-94.

[PetStore, 2002]
Java Pet Store, http://java.sun.com/developer/releases/petstore/

[Pieter Van et al, 2005]
Pieter Van Gorp, Dirk Janssens: *CAViT: a Consistency Maintenance Framework based on Transformation Contracts*. Transformation Techniques in Software Engineering2005

[PrismTech, 2008]
Optimizing Development and Delivery of SCA-Compliant SDR Applications Using An Integrated Tool Chain, 2008, available, www.lynuxworks.com/products/webinar/sdr-prismtech-telelogic.pdf

[Proxy-Sniffer, 2008]
Proxy Sniffer, 2008, available, http:// www.proxy-sniffer.com/

[Ramos et al, 2007]
Ramos, R., Barais, O. and Jézéquel, J.M. *Matching Model-Snippets*, ACM/IEEE 10th International Conference on Model Driven, Engineering Languages and Systems, Nashville, USA, September30-October 3, 2007

[Robbings et al, 1998]
Robbins, J. Hilbert, D.M. and Redmiles, D.F. *Extending design environments to software architecture design*, Automated Software Engineering, vol. 5, No. 3, July 1998, 261-390.

[Robbings et al, 1999]
Robbins, J.E. and Redmiles, D.F. *Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML*, In *Proc CoSET'99*, Los Angeles, May 1999, pp. 61-70.

[Sabetzadeh et al, 2006]
M. Sabetzadeh and S. Nejati. TReMer: *A tool for relationship-driven model merging*. In FM, 2006. Demo.

[Sanchez et al, 2008]
Sanchez, P., Fuentes, L., Stein, D., Hanenberg, S. and Unland, R. *Aspect-Oriented Model Weaving Beyond Model Composition and Model Transformation*, ACM/IEEE 11th International Conference on Model Driven, Engineering Languages and Systems, Toulouse, France, 28 September - 3 October 2008.

[Schmidt, 2006]
Douglas C. Schmidt, *Model Driven Engineering*, 2006, available, http://www.cs.wustl.edu/~schmidt/GEI.pdf

[Sendall, 2004]
Shane Sendall, Rainer Hauser, Jana Koehler, Jochen Küster, Michael Wahler, *Understanding Model Transformation by Classification and Formalization*, 2004, available, www.zurich.ibm.com/pdf/csc/position-paper-sts04_sendall.pdf

[Shaw et al, 1995]
M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. "*Abstractions for Software Architecture and Tools to Support Them*." IEEE Transactions on Software Engineering, vol. 21, no. 4, pages 314-335, April 1995.

[Shaw et al, 1996]
M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996

[Smith et al, 2005]
Smith, C.U., Lladó, C.M., Cortellessa, V., Di Marco, A.,Williams, L.G. *From UML models to software performance results: an SPE process based on XML interchange formats*, In Proceedings of the Fifth International Workshop on Software and Performance, Palma, Spain, July 12-14, 2005, ACM Press, pp. 87-98

[Software Architecture, 2008]
Software Architecture, 2008, http://www.softwarearchitectures.com/go/Discipline/DesigningArchitecture/QualityAttributes/tabid/64/Default.aspx

[Soto, 2008]
Martín Soto, Jürgen Münch, *Using model comparison to maintain model-to-standard compliance*, May 2008 CVSM '08: Proceedings of the 2008 international workshop on Comparison and versioning of software models

[SPEC]
SPEC, available, http://www.spec.org/

[SPEC benchmarks, 2002]
SPEC Java Benchmarks, 2002, available, http://www.spec.org/benchmarks.html#java

[Sprenkle et al, 2005]
Sprenkle, S., Gibson, E., Sampath, S., Pollock, L. *Automated Replay and Failure Detection in Web Applications*, In Proceedings of the 2005 International Conference on Automated Software Engineering, Long Beach, California, November 7-11, IEEE CS Press, 2005.

[Subraya et al, 2000]
Subraya, B.M., Subrahmanya, S.V., *Object Driven Performance Testing in Web Applications*, In Proceedings of the 1st Asia-Pacific Conference on Quality Software (APAQS'00), IEEE CS Press, 2000.

[Taylor et al, 2008]
Nenad Medvidovic, Richard N. Taylor: *Exploiting architectural style to develop a family of applications*. IEE Proceedings - Software 144(5-6): 237-248 (1997)

[Tolvanen, 2008]
Juha-Pekka Tolvanen, *Domain-Specific Modeling in Practice*, 2008, available, http://tfs.cs.tu-berlin.de/gtvmt08/Program/DSMinPractice_Tolvanen_29March2008.pdf

[Tramontana et al, 2002]
Tramontana, P., *Reverse engineering Web applications, Software Maintenance*, 2005. ICSMapos; 05. Proceedings of the 21st IEEE International Conference on Volume, Issue , 26-29 Sept. 2005 Page(s): 705 – 708

[Ulrike Becker-Kornstaedt et al, 1999]
Ulrike Becker-Kornstaedt , Dirk Hamann, Ralf Kempkens, Peter Rösch, Martin Verlage, Richard Webby, Jörg Zettel
Support for the Process Engineer: The Spearmint Approach to Software Process Definition and Process Guidance, Lecture Notes In Computer Science; Vol. 1626 archive, Proceedings of the 11th International Conference on Advanced Information Systems Engineering, Pages: 119 - 133, Year of Publication: 1999

[UML, 1996]
UML Metamodel version 1.1 1 September 1997, available, http://www.omg.org/technology/documents/modeling_spec_catalog.htm

[Varr'o et al, 2003]
Varr´o, D. and A. Pataricza, *VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML*, Journal of Software and Systems Modeling 2 (2003), pp. 187–210.

[W3C, 2004]
OWL Web Ontology Language, www.w3.org/TR/owl-features/

[W3C, 2005]
Web Service Modeling Language (WSML), www.wsmo.org/wsml/

[WebLOAD, 2003]
webLOAD, http://www.radview.com/

[White et al, 1997]
Sharon A. White, Cuauhtémoc Lemus-Olalde, *THE SOFTWARE ARCHITECTURE PROCESS*, 1997, available, http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf

[Xiong et al, 2007]
Yingfei Xiong, Dongxi Liu,Zhenjiang Hu,Haiyan Zhao, Masato Takeichi, Hong Mei, *Towards Automatic Model Synchronization from Model Transformations, Automated Software Engineering archive*, Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA, Pages 164-173, Year of Publication: 2007

[XSLT Transformation, 2001]
XSLT, available, http://www.w3.org/TR/xslt.html

Appendix - Questionnaire

Category 1: Do you often come across the problems identified by the MaramaCRelation approach? Are those problems sensible and need to be solved?

Q: How do you normally capture the rationale of your Model Integration and Transformation (MI&T) designs?
A: Coding
More explanation: Rationale is usually captured in low-level code/formulae that specify target values in terms of source values. Is usually implicit in the formulae and structure/ordering of the transformation script or consistency management code.

Q: Have you ever experienced the need to maintain semantics and semantic constraints when doing MI&T?
A: Very often
More explanation: Yes – as this is the whole point in my mind of model integration and/or transformation. Semantically incorrect target models or related models make the integration/transformation of very limited use.

Q: How easy do you find maintaining model transformation scripts, especially after a long time since you wrote them?
A: Hard
More explanation: Because they are usually low level code, it's hard after a short time. Also the scripts/code don't lend themselves to easy maintenance due to the mix of constructs in the code i.e. mix of formulae doing transformation./linking vs. extraction of source./target elements esp. in XML based transformations.

Q: How do you currently solve issues such as traceability establishment and view synchronization during MI&T?

A: Both tool and coding

More explanation: Simple stuff: e.g. Marama OCL, Marama model/view view type mapping specifications, reusable event handlers. Complex stuff: code, transformations scripts

Q: Do you know any available solutions for the identified problems? Please list them.

A: Visual mapping languages help to abstract from the low-level representational structures.

Category 2: Is the CRelation model easy to understand?

Q: How hard is it to understand StructureMapping and its properties?

A: Reasonable

More explanation: This is the basic relationship between model elements/constructs. I am not sure the "construct" is so easy to understand as it is presented textually – a visual meta-model snippet might help to demonstrate these for the user if they ask for one of them.

Q: How hard is it to understand SelectionRefinement and its properties?

A: Easy

Q: How hard is it to understand StructureReifnement and its properties?

A: Reasonable

More explanation: This is somewhat more challenging – I do think a concrete example illustrating either meta-model elements/constructs in source/target would greatly help – or even better INSTANCE example (like in the ICSE paper – a small snippet showing example instances in source/target being linked by the structure refinement, structureMapping, etc.

Q: How hard is it to understand SemanticAssociation and its properties?

A: Hard

More explanation: I found this construct one of the harder ones to understand. However, again this perhaps could be helped in future version of MaramaCRelation tool by illustration

Q: How do you rate the understandability of the visual notations of the CRelation modelling elements?
A: Hard to understand
More explanation: The visual language is hard to understand at first. An interesting extension would be to show the source/target meta-model elements/constructs in the view – like Karen did with Kaitiaki – to give a visual indication of the context of the CRelation element.

Category 3: Can the CRelation model solve the identified problems? What is their potential?

Q: How do you rate the usefulness of CRelation model for capturing the rationale of MI&T?
A: Very useful
More explanation: This is the major contribution – a model-to-model mapping language with support for capturing not just element correspondences – which visual data mapping tools like MaramaTorua focus on - but complex constructs and constraints.

Q: How do you rate the potential of the MaramaCRelation approach for improving maintainability and readability of model transformation scripts?
A: Promising
More explanation: I am not 100% sure about the transformation scripts – as discussed, would be good to see how adding them to the CRelation elements goes, This would then give a quite different way to structure the scripts vs. current approaches in e.g. ATL, XSLT etc.

Q: Do you think the explicit representation of associations is helpful to correctly capture shared semantics during MI&T?
A: Very useful
More explanation: Yes I think so.

Q: Do you think the explicit representation of associations is helpful to highlight inconsistent semantics during MI&T?

A: Helpful
More explanation: I think so.

Q: How do you rate the usefulness of visually categorizing selection constraints?
A: Useful
More explanation: Current approach OK – but concrete illustration, even PBE, might work well in the future?

Q: How do you rate the usefulness of separating what from how when interconnecting models?
A: Very useful
More explanation: This is the thing most lacking in current script/code-based approaches – they get completely mixed up.

Q: What are the strength and weakness of the CRelation model?
A: Visual language is a bit obscure. Use text to represent source/target elements and constructs. Could use instance/examples vs. abstract representations in future to make easier for user to think about source/target model elements

Category 4: Is the MaramaCRelation tool easy to use?

Q: How do you rate the automatic support that the MaramaCRelation tool provides?
A: Good
More explanation: Good. As above, some usability enhancements around representing elements
Of source/target in the visual specifications I think would help.

Q: How do you rate the traceability mechanism of the MaramaCRelation approach?
A: Good
More explanation: This looks a good advance and uses the inter-model relationships well.

Q: How do you rate the usefulness of the behaviour descriptions of the StructureMappings?
A: Useful

More explanation: As discussed, these are good but augmenting with e.g. Kaitiaki would be very interesting.

Category 5: How to use the MaramaCRelation approach to improve software engineering?

Q: How do you rate the potential of using the MaramaCRelation approach to decompose monolithic, programming-intensive multi-view support?
A: Very promising

Q: Rate the potential for MaramaCRelation to provide structured, high level, and visual support for tool extension
A: Very promising

Q: Rate the potential for MaramaCRelation to provide structured behaviour descriptions to feed third party event-handling technologies
A: Promising
More explanation: There would seem to be good potential here. I also think very good potential in adding transformation scripting snippets/code to the inter-model constraints as discussed. This would make a nice follow-on project and be very interesting to compare to e.g. MaramaTorua approach.