2013 22nd Australian Conference on Software Engineering

A Model-driven Approach for Ensuring Change Traceability and Multi-Model Consistency

Claudia Szabo and Yufei Chen
The University of Adelaide
Adelaide, Australia
Email: claudia.szabo, yufei.chen@adelaide.edu.au

***Abstract*—In model driven engineering, high-level models of an application are constructed to enable reasoning about functional and non-functional requirements independently of implementation issues and concerns. This allows for reduced maintenance, shortens development time, and permits automated model updates, system model executions, and impact assessment. Part of model driven engineering, multi-modeling integrates models that abstract various aspects of the system, such as I/O, behavioral, and functional among others, at different levels of granularity and using various domain specific modeling languages. An important challenge is to understand the relationship between these models towards preserving multi-model consistency as changes in one model affect other models in the multi-model. This paper presents a multi-modeling architecture that captures model relationships at syntactic and semantic levels. We define a taxonomy of change effects that relies on a relationship correspondence meta-model to highlight and trace the impact of changes across various modeling environments. Following the correspondence meta-model and associated change effects, our prototype implementation ensures that multi-model consistency is met and notifies stakeholders of significant changes. Our case study of a submarine tracking system checks multi-model consistency and highlights the impact of changes across system modeling tools that capture its functional and behavioral aspects among others. Our experiments show the feasibility of our approach while highlighting important challenges.**

Keywords: Model driven engineering, multi-modeling, change traceability, model consistency.

I. Introduction

Designing and developing complex software systems poses many challenges including the integration and proper specification of existing software artifacts and multi-vendor middleware [5] for execution on a multitude of hardware and software platforms [30]. The integration of software artifacts happens at different levels and includes data integration, functional integration, and presentation integration among others [19], and requires consistent management of artifacts that are often tightly coupled [30]. The specification of these artifacts is also challenging as today's commonly used programming languages fail to capture and express domain concepts effectively in a way that can be used in the design of the software [4], [5], [19], [30], [37]. Towards this, Model Driven Engineering (MDE) proposes the use of high-level modeling techniques to separate the intended purpose of the system from its software implementation [5], [19]. In MDE, the models, and not the implementation, are the fundamental concept in the software development life-cycle, decoupling designated architectural decisions from implementation issues and increasing understanding, accordingly reducing development time and maintenance costs. MDE relies on the concept of domain-specific modeling languages (DSML) [12], [34] to capture application structure, requirements, and behaviors within a particular domain. Not all domain aspects can be captured within a single DSML, and it is often the case that models of the same system are written in various DSMLs and are used in the development life-cycle.

The wide adoption of model-driven engineering in recent years has led to a variety of environments that model systems in different application domains from various perspectives [10], [16], [36]. However, these modeling environments do not usually model all system aspects, such as functional, non-functional, and behavioral among others, with the desired level of granularity. For example, it is often the case, especially in Defence, that functional requirements of a system are modeled in a SySML compatible environment such as Genesys [29], and behavioral aspects of the same system are modeled and analyzed in a system model execution (SEM) environment such as CUTS [16]. Other tools model components as mathematical functions with well-defined attributes, whose functionality is governed by equations [33], or represent behavior as state-machine and focus only on modeling component workloads [16]. This issue is addressed by *multi-modeling* or multi-view modeling [4], which aims to bring together different views of the same system in a consistent manner.

Multi-modeling approaches promise to facilitate system analysis on many aspects such as functionality, software evolution, change impact and traceability among others [10], [26]. This offers a more complete view of the system and allows stakeholders to make deployment decisions with reduced risk. For example, using MDE and SEM techniques, the development and deployment of submarine combat management systems can be analyzed at design time, thus providing a reduced-cost and risk analysis of spare power and weight constraints than if the system were deployed on real hardware [16]. In web application development, crucial concerns can be modeled using DSMLs that capture the functionality, security requirements, and user behavior respectively, permitting a well-rounded view of the system, with the desired level of granularity within each concern, while at the same time providing means of reducing complexity through modularization [8]. A multi-modeling approach enables the integration of fit-to-purpose DSMLs, while maintaining a holistic view of the system. An important advantage this provides is the ability to integrate DSMLs from the domain experts' own field [10], [19]. The use of knowledge expressed in a DSML fit for a particular domain reduces the number of iterations required to refine and improve models, consequently reducing development time [26]. Next, different DSMLs have different model-to-code transformation rules [27] that would be lost if a one-size-fits-all approach were employed, at the expense of implementation quality and software verification. This increased level of granularity and inherent variety across DSMLs in a specific domain also permits a diversified evolution of models within a multi-model, which is closer to a real software development life-cycle [26].

Despite the inherent appeal of multi-modeling, challenges include capturing multi-model interdependencies [10], maintaining multi-model consistency across various tools by propagating changes and analyzing their impact [10], [20], [32], and ensuring the semantic precision of data exchange between models in the multi-model [3], [10], [26], [37]. A multi-model is defined as being *consistent* when constraints on attributes and behavior within and across models are met [10]. Change traceability and ensuring multi-model consistency are crucial as they support software management, software evolution, and validation among others [10], [15], [26].

In this paper, we propose an extensible multi-modeling architecture that relies on meta-modeling and model mappings to achieve change traceability and consistency checking across various models in a multi-model. Our work focuses not only on improving change traceability but also on analyzing the impact of change across various models, and on ensuring correct semantic data exchange between models. Towards this, we introduce SeMMA (Semantic Multi-Modeling Architecture), an architecture that permits the integration of models defined in various languages that abstract different perspectives of the same system, e.g., functional and behavioral among others. SeMMA relies on a relationship correspondence meta-model to capture both direct and indirect relationships between entities in various DSMLs. Each relationship has attached a taxonomy of change effects and possible actions that define possible outcomes to ensure traceability and consistency. The correspondence meta-model is enhanced with semantic information using a two-level ontological domain, that aids in the evaluation of change effects and in ensuring semantic data exchange between models. The contributions of our work include:

- A relationship correspondence meta-model and a taxonomy of modeling aspects and their associated domain-specific languages
- A plug-in interface to facilitate the management of changes and ensure model consistencies across various domain-specific languages within the same application domain
- An integrated prototype that highlights the impact of changes across functional and behavioral models

The remainder of this paper is organized as follows. Section II discusses background information and presents a motivating example. We discuss related work in Section III. Section IV presents our proposed approach emphasizing our proposed correspondence meta-model and change effects taxonomy. A prototype implementation and example is discussed in Section V. Section VI concludes this paper and presents future work.

II. Background and Motivating Example

In model driven engineering (MDE), models are the fundamental drive in the software development life-cycle, allowing for reduced maintenance and development time. The use of staged transformations from models to executables incrementally transforms the model from a higher level to increasingly lower levels, until the final transformation to executable code [19], [27]. More importantly, changes to the software system are not performed ad-hoc on the code itself, but rather on the high-level model, and then incrementally transformed again into the final executable. This increases traceability of changes between various components of the system, but also provides early detection of errors in design, before the software is developed and deployed [19], [24]. MDE relies on domain specific modeling languages (DSML) to capture key semantics and constraints of the application domain [12].

Building on these domain specific modeling languages, a wide variety of tools for the modeling and generation of various aspects of a system in a multitude of application domains have been recently proposed, including CORE [35], Genesys [36], CUTS [16], MapleSim [38] and MathModelica [23], and several OMG standards support them, such as MetaObject Facility (MOF) [28] and Query-View-Transformation (QVT) [27] among others. The existing tools employ DSMLs that model the system from a variety of perspectives. For example, the CORE [35] and GENESYS [36] toolsuites model a system domain mainly focusing on its functional requirements towards offering support for verification, validation, and accreditation (VV&A), and for documenting every design step at a component and system level. The sequence of functional requirements is represented using Functional Flow Block Diagrams (FFBD) [11] that capture parallel and joining functional flows.

The Component Workload Emulator (CoWorkEr) Utilisation Test Suite (CUTS) system [16] proposes a suite of system execution modeling tools (SEM) in the domain of enterprise distributed real-time environments (DRE) that use model driven engineering, namely, i) emulation of application component behavior, in terms of computational load, resource utilization and requirements, and network communication; ii) configuration, deployment, and execution of the emulated components on top of real infrastructure components to determine their impact on actual runtime environments; and iii) feedback of results to enhance system architecture and components towards improving QoS. The CUTS toolsuite employs the Component Behavior modeling Language (CBML) to describe each CUTS component as a labeled transition system model for components in asynchronous concurrent systems. The actions of I/O automata are classified as input, output and internal actions, where input actions are required to be always enabled.

As large-scale software systems can now be developed using these new tools and standards, the importance of communicating changes to system stakeholders and ensuring change traceability and model consistency across various system models increases drastically as systems are developed and deployed on increasingly expensive hardware [10], [15], [26]. Consider for example a tracking component in a submarine combat management system. In a typical usage scenario, the functional requirements for this system will be modeled in Genesys as a Functional Flow Block Diagram (FFBD) as shown in Fig. 5. Using FFBD constructs, a submarine tracking system can track an existing threat *and* detect an incoming threat. If a threat is detected, then a threat message is initiated. Different characteristics of a threat are analyzed in the functional requirement called *Discriminate Threat*, and as a result a new database entry can be created, *or* previous database entries can be updated. Afterwards, the threat tracker is updated in the *Update Track* functional requirement, and the threat and operation result are displayed.

The components that implement these functional requirements are modelled in CUTS using the Generic Modeling Environment tool [21] as shown in Fig. 6. As it can be seen in Table I, the *Detect Threat* functional requirement is implemented by the component modeled in CUTS as the *Sensor Component*. Assume a simple example where the *Detect Threat* functional requirement is defined as "*Query sensor every 15 seconds*". This is in turn modeled in CUTS as an event that has an inter-arrival time of exactly 15 seconds. However, since existing tools and paradigms have no possibility of highlighting relationships between different modeling concepts, this relationship is not properly recorded in any of the modeling tools employed. This is important because a change in the functional requirement may impact the way the system is modeled in CUTS, and therefore affect the generated and deployed code, which in turn will affect the final system assessment. The relationship is also valid in the opposite direction, i.e. changes in CUTS models can affect functional and non-functional requirements in Genesys. This is a simple example of a one-to-one, variable-to-variable mapping, but more complex relationships can exist, as shown below. For simplicity, the architecture discussed in this paper is illustrated using this motivating example.

III. Related Work

Existing approaches to multi-modeling often aim to transform the various models abstracting a system into a common language or meta-model [7], [8], [39]. This provides a common syntactic representation to facilitate model integration, and is sometimes enhanced with semantic information to facilitate correct execution and analysis. However, this common representation contains a minimum modeling subset and may lose information that is important in ensuring model consistency and change traceability.

In a manner similar to our approach, the NAOMI project proposes to enable multiple models developed in different DSML to work together [10]. NAOMI defines models as Input/Output transformations where both inputs and outputs are represented as sets of input/output attributes respectively. Attributes may have attached constraints that raise warnings whenever the values of their attributes change beyond an accepted value. Multi-model consistency is ensured by verifying attribute constraints on all models in the multi-model. This offers a lightweight method for facilitating multi-model integration and ensuring change propagation. However, the Input/Output definition is not suitable for all DSMLs, such as those that capture functional and non-functional requirements [36] or contain analytic definitions [33]. Moreover, the notion of consistency and change propagation follows only attribute values and moves across models based on the model I/O connection. In contrast, our approach captures indirect relationships between models that go beyond I/O coupling, which allows for the analysis of more complex effects of changes across models in a multi-model.

Yie et al. [39] focus on the composition of views to facilitate model interaction, proposing the transformation to a low-level common subset language using a series of transformation chains. An important advantage of transformation chains is that concepts at higher levels can be related to counterparts in the application source code, thus allowing for the identification of low-level correspondences across corresponding model source code. However, the transformation to a common subset language preserves commonalities across DSMLs but may lose modeling artifacts[1] specific to each domain, thus impacting change effect analysis, traceability, and model consistency. Moreover, the analysis of the impact of change both within and across models is not addressed.

In a similar manner, Cicchetti and Ruscio [8] present a weaving model to define a web application using three independent models, namely, a data model, a composition model, and a navigation model. Common application concepts are represented in the weaving model, which becomes responsible for ensuring model interaction. As before, the focus is on the representation of concepts in a high level language that may lose important details, and change effects are not analyzed. Cibran and Hondt [7] propose a separation between business rules and application logic at the modeling level. Relationships between business rules and the application are modelled using a connection DSML. This allows different connection patterns between business rules and application code to emerge and be analyzed, but effects of changes are not considered.

Lastly, non-MDE approaches propose to analyze documentation artifacts [9], [13], source code [1], [25], or maintenance history records [14], [31] to facilitate change traceability. Approaches that analyze documentation [9], [13] propose to deduce inter-system dependencies from the system documentation, which can often be unreliable, or, in some cases such as legacy systems, may be missing entirely. Source code analysis approaches [1], [25] propose to extract dependencies from analyzing the system source code, but require a single-language implementation and a very good understanding of the source code. Other heuristic approaches [14], [31] propose to analyze maintenance records to deduce dependencies between components if components are frequently modified by the

[1]For example, the behavior of a water pump system can be modelled using discrete time, and the water flow using continuous time [2]. These two perspectives can be contradictory, and the reduction of one to another, e.g. continuous to discrete by sampling, can impact modeling.

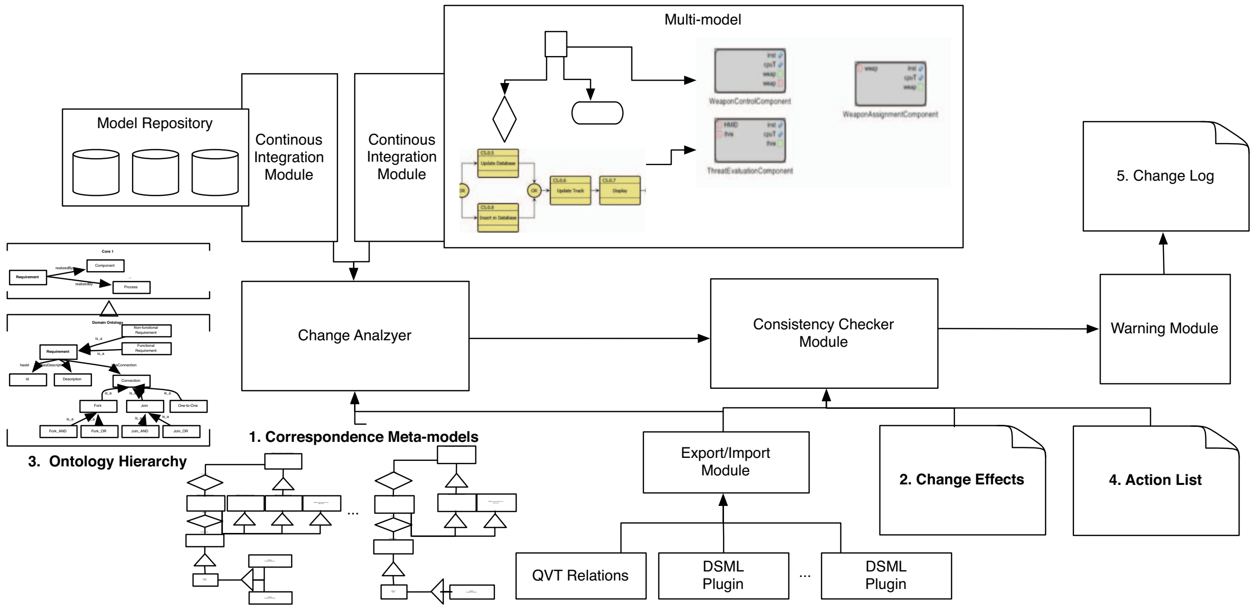


Fig. 1. The SeMMA Architecture

programmer at the same time, but require that maintenance records are available.

IV. The SeMMA Architecture

Towards ensuring model consistency and the traceability of change across various models in a multi-model, we propose SeMMA (Semantic Multi-modeling Architecture), an architecture that permits the semantic integration of models defined in various languages that abstract different perspectives of the same system, e.g., functional, behavioral, etc., as shown in Fig. 1. In SeMMA, a collection of models from different domains described using various DSMLs resides in a model repository.

Our architecture relies on three main modules, namely, the *Change Analyzer Module* (CAM), the *Consistency Checker Module* (CCM), and the *Warning Module* (WM). The CAM monitors the model repository for changes in the various models in the multi-model. When a change occurs, it is reported to the CCM. The CCM relies on a *relationship correspondence meta-model* that describes the relationships between the various entities and attributes in different models in the multi-model. The relationships defined in the correspondence meta-model can be classified as *direct*, one-to-one mappings, or *indirect*, in which some entities in both models are affected by a change in a single entity. Associated with the meta-model and the change effects are a list of actions that are used to analyze the impact of the change. Based on the evaluation functions defined in the actions, the CCM will highlight to the Warning Module if a significant effect has occurred. The Warning Module then logs this change in a change log and highlights it to the stakeholders.

When a new model or a new DSML is added to the repository, it is up to the system expert to define instances of the correspondence meta-model, as well as types of relationships and their associate actions. This process can become tedious for large models and DSMLs, and as such our prototype tool loads models, records mappings, and translates them into an XML format. As part of our future work, we propose a two-level taxonomy to describe concepts in various domain and semantic connectors between them respectively. This taxonomy, enhanced with reasoning capabilities, serves to identify possible correspondences across various models and DSMLs, and suggest relationships and possible actions to the system expert. The resulting ontology will then be used to ensure semantically consistent data exchange across various models.

In the following we discuss within SeMMA (i) our correspondence meta-model to capture relationships between DSMLs; (ii) a classification of change effects to capture relationships between DSMLs representing various aspects within the same domain; (iii) a classification of possible actions to take in the event of a change; and (iv) a set of taxonomies describing each application domain. In the following, we note that a concept C_A in model A can influence a concept C_B in model B, as $C_A \rightarrow C_B$. In the following, we will employ functional requirement *CS.0.1. Detect threat* as C_A, defined as *"Query sensor every 15 seconds"*, and C_B as the *SensorInterfaceComponent* as an example. Both concepts describe a submarine tracking system, in a functional requirement model and a system execution model in Genesys and CUTS respectively, as shown in more detail in Section V and Fig. 7. For simplicity, we write this relationship as *CS.0.1* $\rightarrow$ *SensorInterfaceComponent*. Changes to an entity in a DSML can affect other entities in the same DSML, even though the entities are not directly connected. Our correspondence meta-model detailed below captures relationships between entities within and across DSMLs.

A. Correspondence Meta-Model

Our relationship correspondence meta-model captures relationships between various entities across different DSMLs, as well as within a single DSML. For clarity, we employ the Entity-Relationship model to describe the correspondence meta-model, as it is well-studied and formalized [6]. We employ the symbols shown in Fig. 2, with their associated semantics. A *total* constraint ("t" in Fig. 2) is one in which all the possible entities have been enumerated, whereas a *partial* ("p" in Fig. 2) constraint has entities missing. A *disjoint* constraint ("d" in Fig. 2) is one in which one or the other entities are participating in the relationship. The opposite of a disjoint constraint is a *non-disjoint* constraint ("n" in Fig. 2).

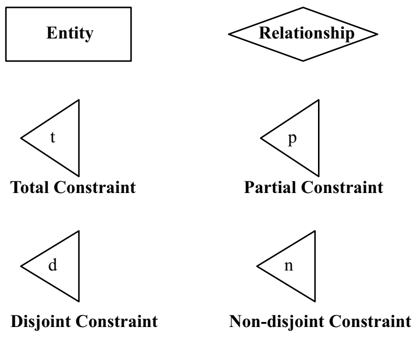


Fig. 2. Entity-Relationship Symbols

Fig. 3 presents an example of the correspondence meta-model for two DSMLs, $DSML_1$ and $DSML_2$, in which a *functional requirement* in $DSML_1$ influences various entities in $DSML_2$. In turn, various functional requirements in

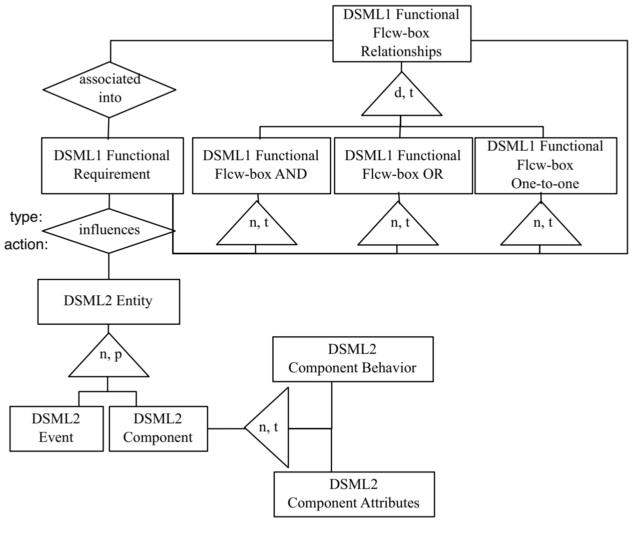

DSML1 Functional Flcw-box Relationships
associated into
d, t
DSML1 Functional Requirement
DSML1 Functional Flcw-box AND
DSML1 Functional Flcw-box OR
DSML1 Functional Flcw-box One-to-one
type:
influences
action:
n, t
n, t
n, t
DSML2 Entity
n, p
DSML2 Component Behavior
DSML2 Event
DSML2 Component
n, t
DSML2 Component Attributes

Fig. 3. Example of a Relationship Correspondence Meta-model for Two DSMLs

$DSML_1$ can be connected to form a functional flow block diagram shown in Fig. 3, the relationships between functional requirements in $DSML_1$ can be further classified as being relationships of type AND, OR, or *one-to-one* between functional requirements, and cascading combinations of all the types. An entity in $DSML_2$ is defined by components and events. Components in turn have behaviors and attributes. An instance of the correspondence meta-model is shown in Fig. 7, where correspondences between two models of a submarine tracking system, describing functional requirements and behavior respectively, are shown.

Each relationship has associated change effects and specific resulting actions. These are shown near the diamond representing the relationship name. Moreover, to recognize that relationships may be uni-directional, arrows are employed to show direction as shown in Fig. 7.

B. Classifying Change Effects

We propose to classify the effects that a change in one entity or concept in a model has across many concepts, into two main categories, namely, *direct* and *indirect*. A *direct* relationship is one in which there exists a one-to-one mapping between a concept defined in one DSML and a concept defined in another DSML, such that a change in the former entity directly affects the latter and is highlighted to the user. We classify *indirect* relationships into *internal* and *external*.

In an *indirect internal* relationship, if $C_A \rightarrow C_B$, a change in entity C_A affects the entities in $DSML_A$ that are related to C_A, and for each of these affected entities, a *direct* relationship affects more than one entity in C_B. For example, a change in the *CS.0.3 Track Threat* requirement affects the *Navigation Component* and all other components that have a (not necessarily direct) relationship with it according to the correspondence model.

In an *indirect external* relationship between entities C_A and C_B, a change in entity C_A may affect entity C_B, but the effect ripples to all the entities that are related to C_B. For example, a change in the *CS.0.1 Detect Threat* Genesys requirement affects all requirements connected to it in the Genesys functional flow block diagram. These requirements can have direct relationships to CUTS components and these in turn can be affected by a change in the former Genesys requirement.

C. Classifying Possible Actions

The actions the system should take when a change occurs can be classified as *analyze* and *execute*. The actions are introduced as labels in the correspondence meta-model in Fig. 3 and 7. An *analyze* action is defined as

$$analyze : [eval(C_B, f), threshold()]$$

where $eval(C_B, f)$ can be a quantitative or a qualitative evaluation, and can be performed by executing code or evaluating analytical representations among others. For quantitative evaluations, a *threshold* can be specified and the action determines whether the evaluation of C_B using function f is within the threshold. Qualitative threshold functions can also be defined and require the help of a system expert for evaluation.

From a system modeling execution perspective, an *execute* action transforms model B to code and considers the relationships between model concepts and code, ideally using the OMG Query/View/Transformation (QVT) relations, QVT-R. The code is then evaluated as in an *analyze* action. More formally, this can be written as

$$execute : [transform(B)];$$
$$[eval(QVT(C_B), f), threshold()]$$

For example, using the same *CS.0.1* $\rightarrow$ *SensorInterfaceComponent* as above, we have

$$execute : [transform(CUTS\_Model)];$$
$$[eval(QVT{-}R(SensorInterfaceComponent), f()), threshold()],$$

where $f()$, defined as

```
avg(SensDataInEvent),
```

is an aggregation function that calculates the mean inter-arrival times between any $SensDataInEvent$ events, and $threshold()$ is defined as `15` as per the functional requirement $CS.0.1$. We note that in some modeling environments the model-to-code transformation cannot be formally defined using QVT-R or other formalisms. In the case where no mode-to-code transformations are defined, the system expert specifies the mapping between model attributes and specific code variables in order to allow for execution and analysis.

D. Capturing Domain Knowledge

To facilitate the identification of relationships between concepts in various DSMLs, we propose to enhance the DSML definitions with domain ontologies in a two-level ontological domain, as shown in Fig. 4. Firstly, domain-specific ontologies contain classes that describe concepts in the various DSMLs for each domain. For example, a requirements ontological domain can have functional and non-functional requirements, which can be connected in a FFBD using various types of connections. Secondly, at a higher level, a core ontology defines the semantic connectors between concepts defined in the lower DSML ontologies. For example, a *Requirement* can be realized through a *Component* or *Process* among others, which are defined in other domain ontologies. When a new model is added to a multi-model or a new DSML is added to the repository, the semantic connector ontology is queried, and, using information supplied by previous correspondence meta-models, possible relationships are highlighted to the user. Our prototype implementation currently uses simple “is-a” relationships to select possible candidates. Reasoning support to enhance this process and to ensure that data exchange across various models in the multi-model is semantically correct is part of our future work.

V. Prototype Implementation

The SeMMA prototype implementation initially targets two system modeling tools that employ domain modeling specification languages to capture different aspects of the same system, namely, a functional aspect in Genesys [36], and behavioral and workload aspects in CUTS [16]. The modules shown in Fig. 1, namely the *Change Analyzer Module*, *Consistency Checker Module*, and *Warning Module* have been implemented and the *Export/Import Module* has been extended with DSML plug-ins to facilitate the querying of Genesys and CUTS models respectively. Genesys offers a C# API to query the Genesys model, whereas CUTS provides a C++ API that, together with the GME modeling tool [21] and the UNITE toolsuite, can be used to analyze CUTS models and the generated code as it is deployed on various platforms. As proof of concept, the *Change Analyzer Module* is implemented in Java with a series of Python scripts that monitor changes to Genesys and CUTS models. In future versions, these monitoring scripts will be replaced with a continuous integration toolsuite such as Jenkins [18], which monitors executions of repeated jobs, such as building a software project, and facilitates builds and execution of scripts once developers make changes to the projects. The CCM, WM and associated GUIs are implemented in Java. The interface between the CAM and CCM follows a push model, which means that the CCM is notified by the CAM once a change occurs. The CCM then traverses the correspondence meta-model for the relevant models and follows through the change effects, calling appropriate DSML plugins as defined in the evaluation function and configuration scripts.

The CUTS system [16] proposes a suite of system execution modeling tools that emulate application component behavior and facilitate the deployment and QoS analysis of the application on real runtime environments. An extension to CUTS, the UNITE toolsuite [17] additionally provides developers with techniques for understanding non-functional requirements by allowing them to identify key metrics of interest for data collection and extract and aggregate metrics towards useful data analysis. UNITE proposes to use log messages, which can be automatically parsed according to a pre-defined syntax, to capture metrics of interest, such as the

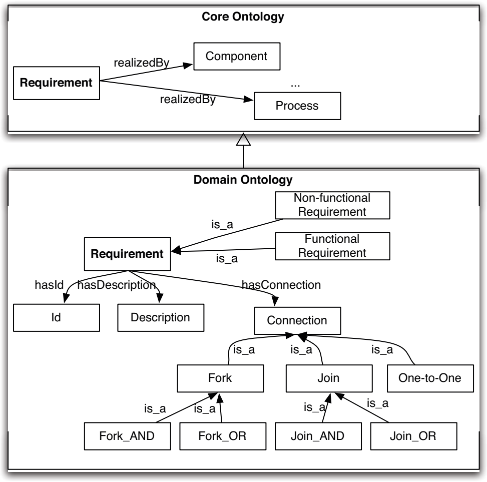


Fig. 4. A Hierarchy of Ontologies to Capture DSML Connections

time a message was sent or specific values of elements in an event. Moreover, UNITE permits the developers to define unit tests to analyze non-functional concerns, such as latency, utilization, etc. by formulating equations or aggregators using the specified metrics, that span more than one log message.

The GENESYS toolkit [36] proposes a model-centric approach to systems engineering and offers the possibility to build models that express the design intent of the system developer. Genesys offers a graphical user interface that allows the user to build a model by going through the various stages of construction, validation, and documentation. An important advantage of Genesys is that it provides an easy-to-use interface that is also SysML-capable [29]. Functional requirements of the system can be defined and connected to form a functional flow block diagram [11]. Moreover, a provided C# API allows for the querying of models programmatically.

A. Overview

The two target systems included in our initial prototype, namely CUTS and Genesys, define behavior and workload, and functional requirements, respectively. This section details these abstractions.

1) Behavioral Representation: Component behaviors are modeled in CUTS using the Component Behavior modeling Language (CBML) that abstracts component behaviors as I/O automata [22] that includes behavioral elements such as input and output actions, and states. The input action signifies the start of a behavioral specification. The state element represents a state in the I/O automata, which must occur between two consecutive actions. The output action signifies that an event was sent to trigger the start of another behavioral specification. Component attributes are not specifically represented in CUTS, and of interest is the workload of an individual component, which specifies workload behavior information such as profiles for processor, memory, database, and I/O behaviors. The workload information is specified in CUTS using the Workload Modeling Language (WML). The WML is used in the emulation of component execution to generate executable operations in the target platform.

2) Functional Flow Representation: The Functional Flow Block Diagram (FFBD) is a structured method for documenting process flow first exploited by NASA in the 1960s to visualize the time sequence of events in space systems and flight missions. It is widely used in systems engineering to show the order of execution of system functions [11]. The fundamental attributes in a FFBD are the function block, the flow connections, the flow directions, and the summing gates. In Genesys, the FFBD are represented pictorially using rectangular boxes connected through lines as shown in Fig. 5. The Export/Import module contains the implemented Genesys plug-in that uses the Genesys C# API to query the FFBD.

B. Tracking Module in a Submarine Combat System

As an example, consider a tracking module on a submarine combat system. The functional requirements of this module and their relationship can be represented in Genesys as shown in Fig. 5. As discussed above, a submarine tracking system can track an existing threat *and* detect an incoming threat. If a threat is detected, then a threat message is initiated. Different characteristics of a threat are analyzed (*Discriminate Threat*), and as a result a new database entry can be created (*Insert in Database*), *or* previous database entries can be updated (*Update Database*). Afterwards, the threat tracker is updated (*Update Track*), and the threat and operation result are displayed (*Display*).

The components that implement these functional requirements are modelled in the GME system modeling tool as shown in Fig. 6. The CUTS submarine combat system contains four packages, namely, *HMI*, *SensorInterfaces*, *Control*, and *EffectorInterfaces*. Each package contains one or more model components. For example, the Control package has *WeaponControlComponent*, *ThreatEvaluationComponent*, *SensorCtrlComponent*, and *Navigation Component*. Each component has at least one input event and one output event. For example, *SensorCtrlComponent* has a *SensCtrlIn* input event port that receives the *HMIControlOutEvent*, and a *SensControData* output event port that sends out the event call *sensCtrlDataOutEvent*.

An instance of the relationship correspondence meta-model for the CUTS CBML and requirement $CS.0.3$ is shown in Fig. 7. As it can be seen in Table I, the *Display* functional

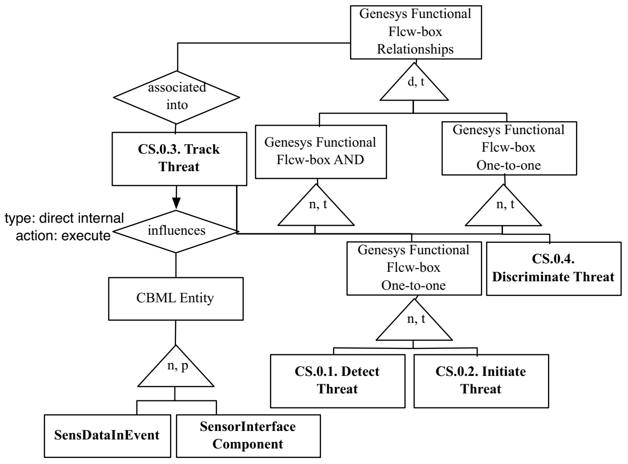

Genesys Functional Flcw-box Relationships
associated into
d, t
CS.0.3. Track Threat
Genesys Functional Flcw-box AND
Genesys Functional Flcw-box One-to-one
n, t
n, t
type: direct internal
action: execute
influences
Genesys Functional Flcw-box One-to-one
CS.0.4. Discriminate Threat
CBML Entity
n, t
n, p
CS.0.1. Detect Threat
CS.0.2. Initiate Threat
SensDataInEvent
SensorInterface Component

Fig. 7. Instance of the Correspondence Meta-Model

requirement is implemented by the component modeled in CUTS as the *HMI Component*, that contains various events related to environment sensing. All relationships are then classified as described in Section IV, in both directions, and specific actions are defined. Table I shows only the mapping between the functional requirements in Genesys to the CUTS components. For example, the *CS.0.1 Detect Threat* requirement directly impacts the interface definition in the *Sensor Interface Component*. The behavior of the *Sensor Control Component* affects all other CUTS components that make up the Tracking module of the submarine model, and as such the relationship between *CS.0.1 Detect Threat* and *Sensor Control Component* is classified as indirect external. Lastly, there is an indirect internal relationship between *CS.0.2 Initiate Track* and *HMI Component* because $CS.0.2.$ has an impact on the

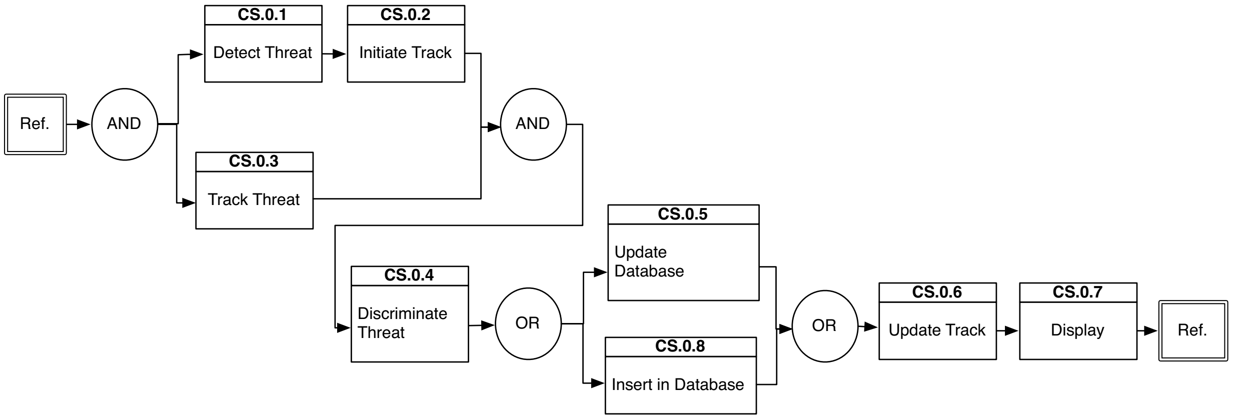


Fig. 5. Functional Flow Block Diagram of a Submarine Tracking System

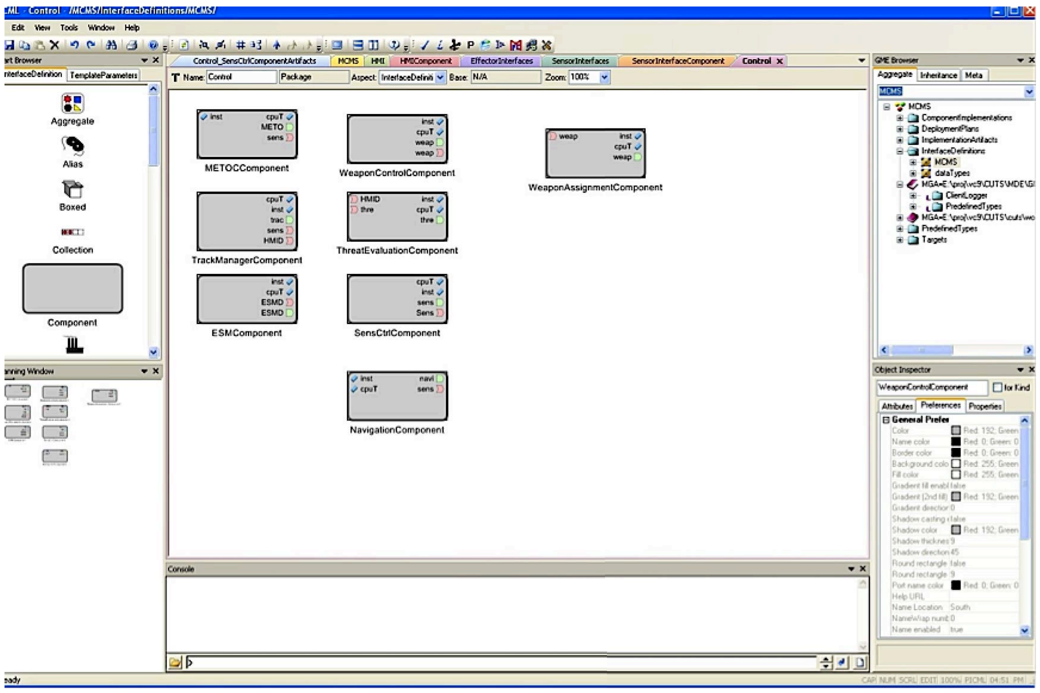


Fig. 6. Submarine Combat Management System in GME using CUTS

other functional requirements in the FFBD. In other cases, as the *HMI Component* is viewed as standalone within the other components, its relationships are classified as direct, e.g., between *CS.0.3. Track Threat* and *HMI Component*. In turn, an action is defined for each change effect, including those that are part of the indirect change effect definitions. It is important to highlight here that the definition of the relationship types and their associated actions is highly dependant on system expert knowledge and defining such relationships can become tedious without the help of specialized tools. Our prototype proposes a Graphical User Interface for the input of such relationships that is then transformed into XML. Our future work currently looks at employing our ontology to suggest possible relationships to the system expert based on existing models and correspondence meta-models in the repository. Based on the relationships described in the correspondence meta-model, our tool highlights to the user possible relationships between the CUTS model and the Genesys functional requirements as shown in Fig. 8.

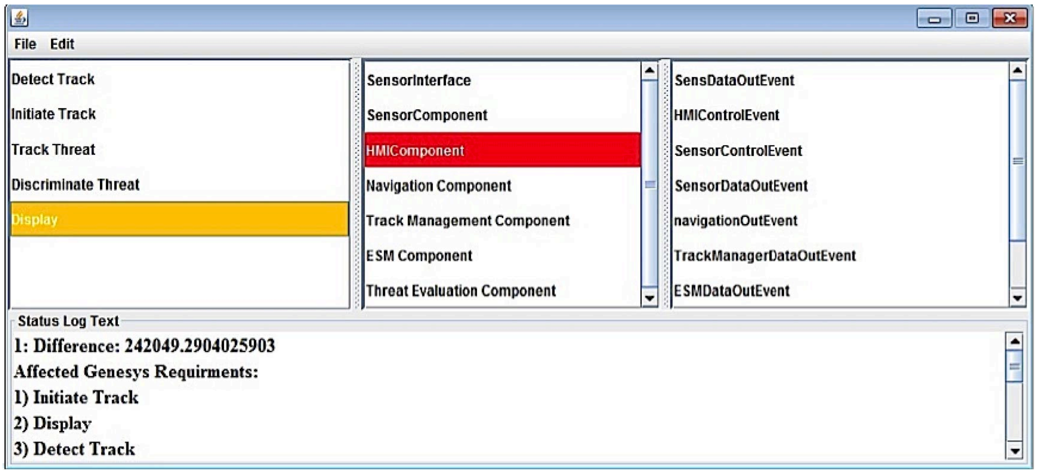
Following the motivating example presented in Section II, assume *CS.0.1 Detect Threat* functional requirement is defined as "*Query sensor every 15 seconds*". This is in turn modeled in CUTS as a *SensDataInEvent* with an inter-arrival time of exactly 15 seconds. This change relationship is classified as direct with an execute action, meaning that if a change in the functional requirement occurs, then the direct relationship with the event is analyzed. Next, all the functional requirements or other entities related to requirement $CS.0.1$ and their direct mappings to entities in CUTS are analyzed. Once a change occurs in the functional requirements, the Continuous Integra-

Functional Requirement	CUTS Components	Relationship Type
CS.0.1 Detect Threat	Sensor Interface Component Sensor Control Component	direct indirect external
CS.0.2. Initiate Track	HMI Component	indirect internal

CS.0.3. Track Threat	Navigation Component;
	Track Manager Component;
	ESM Component;
	HMI Component
		indirect internal
		indirect internal
		direct
		direct

CS.0.4. Discriminate Threat	Threat Evaluation Component;	indirect external
CS.0.5. Update Database	-	-
CS.0.6. Update Track	Track Manager Component;	direct
CS.0.7. Display	HMI Component	direct
CS.0.8. Insert in Database	Track Manager Component; ESM Component	direct direct

TABLE I
MAPPING OF GENESYS FUNCTIONAL REQUIREMENTS TO CUTS COMPONENTS



```
1: Difference: 242049.2904025903
Affected Genesys Requirments:
1) Initiate Track
2) Display
3) Detect Track
```

Fig. 8. Changes in CUTS Components Affect Genesys Models

tion module is woken up and will call the CAM to determine which functional requirement has changed, and to identify the change. This process is implemented in every DSML-specific plugin, as shown in Fig. 1. Based on the relationships specified in the correspondence meta-model, the CAM module will query the CUTS model looking at the relevant components and events, and following defined evaluation functions.

Since relationship $CS.0.1 \rightarrow SensorControlComponent$ is classified as indirect external, with an execute action, the CUTS Entities related to *SensorControlComponent* are analyzed their related functional requirements highlighted. Since this relation has an execute action attached to it, the CUTS model is further transformed into generated code, and following the model-to-code transformations, specific attributes in the code will be monitored using UNITE. The code is executed on the deployment platform and attribute values are logged. Using UNITE, the logs are analyzed using the evaluation function and values are compared with baseline values. If the differences are above a user-specified threshold, then the Warning Module notifies the user. The values obtained in this UNITE test will then be used as baseline values when new evaluation functions are executed by the CAM module.

Reversely, a change in the *HMI Component* refresh rate affects the Genesys requirements $CS.0.2$, $CS.0.3$, and $CS.0.7$, and is highlighted to the user, as shown in Fig. 8. The difference shown refers to the difference between the current CUTS UNITE tests of the HMI Component refresh rate and the initial baseline run. Our prototype is able to highlight the impact of changes across both DSML, within each DSML as well as from each DSML to the executed code.

VI. CONCLUSION

This paper proposes SeMMA, a multi-modeling architecture that focuses on the semantic traceability of changes across models and on ensuring multi-model consistency when changes across different models occur. Our architecture employs a relationship correspondence meta-model to describe change effects between DSML concepts across various DSMLs. The meta-model is accompanied by a taxonomy of change effects that guide the verification of model consistency, and a set of possible actions that can take place once a change has been identified. To aid the system expert in defining relationships and actions, as well as to facilitate semantic data exchange between models, we propose a two-level ontological domain, in which concepts in the various DSML within an application domain are represented in a lower ontology, and a core ontology defines the semantic connections between various DSML concepts.

Our experiments using our prototype implementation look at a case study of a submarine tracking multi-model system composed of models from two system modeling environments, namely, Genesys and CUTS, that model the functional requirements and the behavior of the tracking system respectively. Changes in concepts in one modeling tool are captured and their combined effects on the other models are highlighted. Budget models have also been added to the multi-model. Our prototype scenario shows the feasibility of our approach but also highlights important avenues for future work. Our immediate future work analyzes the integration of other DSMLs, such as WML, the workload modeling language in CUTS, with the existing DSMLs, and looking at representations of model-to-code transformations in the case where QVT-R cannot be used. To ensure the semantic correctness of data exchange across models in the multi-model, our proposed ontology will look at ensuring semantic consistency between data passed between models represented in various DSMLs.

Acknowledgement This work is partly funded by Defence Science and Technology Office Research Grant number A101692.

References

[1] M. A.Chaumun, H. Kabaili, and R. K. Keller. A Change Impact Model for Changeability Assessment in Object-Oriented Software Systems. In *Proceedings of the Third European Conference on Software Maintenance and Reengineering*, pages 155–174, 1999.

[2] J. Banks, J. Carson, B. Nelson, and D. Nicol. *Discrete-Event System Simulation*. Prentice Hall, USA, 2005.

[3] M. Brauer and H. Lochmann. Towards Sematic Integration of Multiple Domain-specific Languages Using Ontological Foundations. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, 2007.

[4] C. Brooks, C.-H. Cheng, and T. H. Feng. Model Engineering using Multimodeling. In *Proceedings of the International Workshop on Model Co-Evolution and Consistency Management*, 2008.

[5] A. W. Brown. Model Driven Architecture: Principles and Practice. *Software System Modelling*, 3:314–327, 2004.

[6] P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[7] M. A. Cibran and M. DHondt. A Slice of MDE with AOP: Transforming High-Level Business Rules to Aspects. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, pages 170–184, 2006.

[8] A. Cicchetti and D. D. Ruscio. Decoupling Web Application Concerns through Weaving Operations. *Science of Computer Programming*, 70:62–86, 2007.

[9] K. Dam, M. Winikoff, and L. Padgham. An Agent-oriented Approach to Change Propagation in Software Evolution. In *Proceedings of the Australia Software Engineering Conference*, pages 309–318, 2006.

[10] T. Denton, E. Jones, S. Srinivasan, K. Owens, and R. W. Buskens. NAOMI - An Experimental Platform for Multi-modeling. In *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, pages 143–157, 2008.

[11] T. Dufresne and J. Martin. Process Modeling for e-Business. *INFS 770 Methods for Information Systems Engineering: Knowledge Management and E-Business*, 2003.

[12] M. Fowler. *Domain-specific Languages*. Addison Wesley, 2011.

[13] J. Han. Supporting Impact Analysis and Change Propagation in Software Engineering Environments. In *Proceedings of the International Workshop on Software Technology and Engineering Practice*, pages 172–182, 1997.

[14] A. E. Hassan and R. C. Holt. Predicting Change Propagation in Software Systems. In *Proceedings of the International Conference on Software Maintenance*, pages 284–293, 2004.

[15] A. Hessellund, K. Czarnecki, and A. Wasowski. Guided development with multiple domain-specific languages. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, pages 46–60, 2007.

[16] J. Hill, J. Slaby, S. Baker, and D. Schmidt. Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 350–362, Sydney, Australia, 2006.

[17] J. Hill, H. A. Turner, J. R. Edmondson, and D. C. Schmidt. Unit Testing Non-Functional Concerns of Component-based Distributed Systems. In *Proceedings of the International Conference on Software Testing, Verification and Validation*, pages 406–413, 2009.

[18] Jenkins. Jenkins Continous Integration. http://jenkins-ci.org, 2012.

[19] G. K. J. S. K. Balasubramanian, A. Gokhale and S. Neema. Developing Applications Using Model-Driven Design Environments. *IEEE Computer*, 39:33–40, 2006.

[20] C. Kruegel, G. Vigna, and W. Robertson. A Multi-model Approach to the Detection of Web-based Attacks. *Computer Networks*, 48(5):717–738, Aug. 2005.

[21] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The generic modeling environment. In *Workshop on Intelligent Signal Processing*, 2001.

[22] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.

[23] MapleSoft. MapleSim 5. http://www.maplesoft.com/products/maplesim/, 2012.

[24] P. Mohagheghi and V. Dehlen. A Metamodel for Specifying Quality Models in Model-Driven Engineering. In *Nordic Workshop on Model Driven Engineering*, pages 20–22, 2008.

[25] L. Moonen. Lightweight Impact Analysis using Island Grammars. In *Proceedings of the International Workshop on Program Comprehension*, pages 219–228, 2002.

[26] B. Nuseibeh, J. Kramer, and A. Finkelstein. Expressing the Relationships Between Multiple Views in Requirements Specification. In *Proceedings of the International Conference on Software Engineering*, pages 181–196, 1993.

[27] OMG. Object Management Group: MOF QVT Final Adopted Specification. OMG document ptc/07-07-07, 2007.

[28] OMG. Meta-Object Facility. http://www.omg.org/spec/MOF/2.0/, 2012.

[29] OMG. Systems Modeling Language. http://www.omgsysml.org/, 2012.

[30] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39:25–31, 2006.

[31] J. Shirabad, T. C. Lethbridge, and S. Matwin. Supporting Software Maintenance by Mining Software Update Records. In *Proceedings of the International Conference on Software Maintenance*, pages 22–31, 2001.

[32] M. A. Simos. Organization Domain Modeling (ODM): Formalizing the Core Domain Modeling Life-cycle. In *Proceedings of the Symposium on Software Reusability*, pages 196–205, 1995.

[33] M. Tiller. *Introduction to Physical Modeling with Modelica*. Springer, 2001.

[34] J.-P. Tolvanen and S. Kelly. Integrating Models with Domain-specific Modeling Languages. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, 2010.

[35] Vitech. CORE. http://www.vitechcorp.com /products/core.shtml, 2010.

[36] Vitech. Genesys. http://www.vitechcorp .com/products/genesys.shtml, 2012.

[37] J. Warmer and A. Kleppe. Building a Flexible Software Factory Using Partial Domain Specific Models. In *Proceedings of the OOPSLA Workshop on Domain-Specific Modeling*, 2006.

[38] Wolfram. Wolfram Mathcore Products. http://www.mathcore.com/products/mathmodelica/, 2012.

[39] A. Yie, R. Casallas, D. Deridder, and D. Wagelaar. A Practical Approach to Multi-modeling Views Composition. In *Proceedings of the International Workshop on Multi-Paradigm Modeling*, pages 1–10, 2009.