Towards automatization of domain modeling

Iris Reinhartz-Berger

Department of Management Information Systems, University of Haifa, Carmel Mountain, Haifa 31905, Israel

Abstract

A domain model, which captures the common knowledge and the possible variability allowed among applications in a domain, may assist in the creation of other valid applications in that domain. However, to create such domain models is not a trivial task: it requires expertise in the domain, reaching a very high level of abstraction, and providing flexible, yet formal, artifacts. In this paper an approach, called Semi-automated Domain Modeling (SDM), to create draft domain models from applications in those domains, is presented. SDM takes a repository of application models in a domain and matches, merges, and generalizes them into sound draft domain models that include the commonality and variability allowed in these domains. The similarity of the different elements is measured, with consideration of syntactic, semantic, and structural aspects. Unlike ontology and schema integration, these models capture both structural and behavioral aspects of the domain. Running SDM on small repositories of project management applications and scheduling systems, we found that the approach may provide reasonable draft domain models, whose comprehensibility, correctness, completeness, and consistency levels are satisfactory.

1. Introduction

Domain engineering [34,36], also known as software product line engineering, aims at identifying, modeling, constructing, cataloging, and disseminating the commonality and variability allowed among applications in a specific domain. A domain in this context can be defined as a set of applications that use common concepts to describe requirements, problems, capabilities, and solutions. Domain artifacts are built and reused through domain analysis, which identifies the domains and captures their ontology to assist and guide system developers with particular applications in those domains. Despite the rapid growth of technologies and technical solutions, domain analysis models usually remain valid longer than domain designs and implementations, potentially justifying the cost and effort required for their development. Domain analysis artifacts may also serve as the basis for defining Domain-Specific Languages (DSL) [26].

Several domain analysis methods have been proposed over the years, (e.g., [13,15,20,27,45]). However, they all can be criticized as making the domain engineer alone responsible for developing correct, complete, and consistent domain analysis artifacts. Since domains may cover broad areas and are usually understood only during the development process, creating domain models can be a very demanding task. It requires expertise in the domain, reaching a very high level of abstraction, and providing flexible, yet formal, artifacts [20]. In this paper we suggest a partial solution to this challenge in the form of a Semi-automated Domain Modeling (SDM) approach. This approach aims to create draft models of emerging domains from families of relevant applications already developed in these domains. Since domain models in some areas may be wide and require close and detailed handling of domain experts, automatic generation of drafts or profiles to help with domain analysis in general and domain modeling in particular may be useful. Access to existing applications in a domain, which is an
essential requirement of the suggested approach, may not be easy. In real-world industrial settings, a specific organization may only have a few application models available. However, additional applications can be achieved from other organizations which, for example, participate in the same consortium; or open-source applications can be reverse-engineered and serve as inputs for the suggested approach.

The SDM approach matches similar elements according to syntactic, semantic, and structural aspects, and merges them into similarity groups which it generalizes into domain elements and models. The approach can be used in conjunction with various modeling languages such as UML, enabling its application to different modeling tasks.

The main contribution of this paper is the creation of domain models – an error prone and time-consuming task – semi-automatically. The aim is not to replace domain engineers but to facilitate their work by providing draft domain models that may later be enhanced. Furthermore, the choice of the application models on which the approach works is crucial for the quality of the resultant draft domain models: if the selected application models do not represent the entire domain, the created domain model will be partial and limited in its usefulness. Hence the approach is considered semi-automatic rather than fully automatic. From our experiments with the approach, we believe that the SDM-generated models supply reasonable draft domain models even for small repositories of application models.

Section 2 below reviews related work, including matching and integration approaches, domain engineering concepts, and ADOM, which is a general domain engineering approach. Section 3 supplies the required formalism of ADOM, while Section 4 elaborates on the SDM approach, exemplifying its capabilities. Section 5 reports two pilot experiments evaluating the quality of the SDM-generated domain models. Finally, the benefits and shortcomings of the proposed approach are discussed in Section 6.

2. Related work

2.1. Matching and integration

Matching and integration are two fundamental operations that have enjoyed great attention in the last two decades due to the rapid growth of the Web and the emerging needs to integrate data and models developed by different suppliers. Three important fields in which these operations are researched are database schema integration [37,43], ontology engineering [30,48], and model transformation [6,25].

Schema matching and integration are basic problems in many application domains of database engineering, such as heterogeneous database integration, e-commerce, data warehousing, and semantic query processing. Surveying existing approaches, Rahm and Bernstein [37] distinguish instance- from schema-level matchers, element from structure matchers, and language from constraint matchers. While instance-level matching exploits the data that underlie the schema to improve matching precision, schema-level matching uses only meta-data information. Orthogonally, matches can be performed for individual schema elements (e.g., attributes) or for combinations of elements (i.e., complex structures). Finally, a matcher can use linguistic approaches (based on names and textual descriptions of schema elements) or constraints (based on keys and relationships). Shvaiko and Euzenat [43] refine this classification, distinguishing approximate from exact techniques at the schema level, and among syntactic, semantic, and external techniques at the element and structure levels.

Doan and Halevy [10] divide matching approaches into rule-based and learning-based. Rule-based approaches employ rules for categorizing elements according to their names, data types, and ranges. Learning-based approaches consider a variety of learning techniques and exploit both schema and data information. Cupid [23], for example, is a schema level, rule-based algorithm that calculates the weighted similarity between schema elements according to their linguistic and structural similarity coefficients. Madhavan et al. [24] suggest a learning-based approach that uses a corpus of scheme from a specific domain to obtain statistics of matches, which can help merge particular schemes in that domain. This approach aims at improving the matchers by increasing the evidence about elements being matched. SemRol [28] is a corpus-based semantic role approach in information retrieval that uses two different statistical models, conditional Maximum Entropy Probability models and the TIMBL program, a Memory-based Learning method, for reducing the number of documents or passages retrieved for a given query. Generally, learning-based techniques require a large number of schemes (sometimes in the same domain) in order to learn the differences in representation of close terms. Moreover, data instances cannot always be used as a means to obtain better evidence about the schema elements.

Parsons and Wand [33] propose an attribute-based approach, based on an observation that data attributes that seem different in various sources may represent the same concept on a more generic level. Consequently, they define the notion of precedence among properties, and use it to create a common view and to integrate manifestations across independent data sources. Furthermore, they introduce the Principle of Minimality, which restricts the choice of the generic property to the lowest one in the precedence hierarchy.

In the area of ontology engineering, ontologies play the roles of content explication, query modeling, and verification [48]. Mapping is mainly used to relate the ontologies to the actual content of information sources, to connect local ontologies to an integrated, global ontology, and to associate terms from different (local) ontologies [5]. Although schema matching and ontology mapping are very similar in their tasks and the way data are analyzed, Noy and Klein [30] found the following differences: (1) ontologies more often use data (instances), (2) ontologies themselves incorporate semantics, (3) ontologies are more often reused, (4) ontologies are de-centralized by nature, and (5) ontology data models are richer. So ontologies are
mapped by a variety of methods, including text similarity measurement, keyword extraction, language-based techniques, structure resemblance, meta-annotation, graph mapping, and semantic correspondences [1,48].

The Model-Driven Architecture (MDA) initiative [31] bases software development on modeling and automated generation of implementations. The basic MDA pattern involves defining a platform-independent model (PIM) and automatically translating it into one or more platform-specific model (PSM). As opposed to database schemes and ontologies, models, which are the core of the MDA approach, are more complex structures. Czarnecki and Helsen [6] classify existing model-to-model transformation approaches into direct manipulation, relational, graph-transformation-based, structure-driven, and hybrid approaches.

They further suggest a taxonomy that makes the different design choices for model transformations explicit. Anaya et al. [2] suggest the Unified Enterprise Modeling Language (UEML) to support the integrated use of enterprise and information system models expressed in a variety of languages (including ARIS, BMM, BPMN, colored Petri nets, and IDEF3). UMLElement offers a structured approach to describe enterprise and information system modeling constructs; a common ontology to interrelate construct descriptions at the semantic level; a correspondence analysis approach to estimate semantic construct similarity; a quality framework to aid selection of languages; a meta-meta model to organize the UMLElement; and a set of tools to aid the different usages.

Grossmann et al. [16] observe that extending well-known data-integration techniques to support behavior-related elements remains a challenge. Some recent works tackle behavior integration by weaving statecharts, e.g., [14,12]. They mainly concentrate on integrating behavior of a single object represented in different statecharts and checking the consistency between static and dynamic components of object-oriented specifications. Whittle and Schumann [49] suggest merging multiple sequence diagrams by first converting them to statecharts and then performing the integration. Behavior integration has recently received special attention also in the area of business processes integration. Grossmann et al. [17] proposed a four-step approach to integrating business processes: (1) examining the relations of objects and activities in two models of discourse, (2) defining high-level integration operators, e.g., according to activity-execution order, (3) mapping each activity correspondence type to one or more potential integration operator, and (4) making an integration choice and applying the integration operator to perform the merge. However, this approach assumes designer-prepared identification of correspondences, hence requires user's involvement and time-consuming preliminary work.

Preuner and Shrefl [35] examine the integration of business processes in the framework of behavioral diagrams based on Petri nets. Their approach emphasizes the formal use of generalization to integrate two behavior diagrams. Generalization is defined as a concatenation of restriction and abstraction. The generalization techniques allow inclusion of alternatives, presented in only one local behavioral diagram, in order to preserve consistency and handle variability. Other approaches also introduce generalization in the context of model merging and integrating, e.g., [14,16,47], usually for readability improvement and consistency preservation.

As explained in the next section, domain engineering raises additional challenges to matching and integration operations which originate from dealing with two abstraction levels, namely applications and domains: the integrated (domain) model is required to show the commonality among the local (application) models, but also their allowed variability. Domain models should also specify typical behaviors in the domain and not just structural constraints.

2.2. Domain engineering

Domain engineering [34,36] builds reusable assets, such as specification sets, patterns, and components, in specific domains. It covers all activities for building software core assets, including identifying domains, capturing possible variation within a domain (domain analysis), specifying applications in the domain, constructing adaptable designs (domain design), and defining mechanisms for translating requirements into systems in specific domains (domain implementation). The products (or software assets) of these activities are domain models, design models, domain-specific languages, code generators, and code components. Most often the motivation for carrying out domain engineering activities is to achieve a significant increase in software productivity and reliability within the chosen domain. Hence, domain engineering involves reuse, knowledge representation, and validation. Domain artifacts can be reused in the context of particular applications (applying adaptation techniques if required); they represent the knowledge gained in the domain; and they can serve as validation templates for applications and their constraint fulfillment.

Several domain engineering approaches have been proposed over the years. Feature-oriented approaches, e.g., FODA [20] and PLUS [15], emphasize the common and different features of applications in a specific domain. A specific system uses the reusable architecture and instantiates a sub-set of features from the domain model. These methods enable modeling domains and applications at the same level of abstraction, where the application models are actually sub-sets of the domain models that satisfy all the required constraints. They handle variability by means of parameterization, generalization, and implicit variation point definition. They also guide the application designer in how to select the required features. However, they support only closed variation points, meaning that adding new application features, not modeled within the domain, are not explicitly supported, narrowing the variety of applications suitable for a specific domain. Application models in these approaches are validated by a check of whether the feature logics defined in the domain model holds in the specific application.

Metamodeling approaches to domain analysis, e.g., GME [22,45] and MetaEdit+ [27], enable definition of domain specific languages (DSL) and support their use to describe particular applications. These metamodels serve for capturing domain
knowledge and validating particular domain-specific applications. The validation rules induced by the metamodels prevent
syntactic and semantic mistakes in the initial stages of application development, reducing development time and improving
system quality. Most of them do not support any domain analysis methodology or formalism. Hence, the analysis is usually
ad hoc for particular domains. Although these methods support modeling at two different abstraction levels, the model and
the metamodel, they mainly enable specification of structural elements and fall short of supporting specification of behavior-
related constraints. Also, they suffer from limited accessibility as different jargons are used on the model and the metamodel
levels. This is also referred to as “impedance mismatch,” arising from the difficulty and ambiguity of translation between
these levels [36]. These methods mainly support definition and validation of closed variation points, i.e., a specific application
can select from a list of variants which is defined at the metamodel level.

To capture behavioral aspects of domains, semantic approaches have been suggested. Graph transformation (e.g., [8]) is
one way to specify the (behavioral) semantics of domain-specific visual languages. The most common formalization of graph
transformation is by category theory, which average software or domain engineers may find difficult to use. Maude [41],
which supports rewriting logic specification and programming of systems, has been proposed as a formal notation and envi-
ronment for specifying and effectively analyzing models and metamodels. Rivera et al. [40] further suggest a way to encode
models and their dynamic behaviors (given by graph-grammar rules) into their corresponding Maude specifications. Here
again, the models and metamodels are specified in two different languages and tools, raising transformation challenges
for the expected users (software and domain engineers).

Also relying on graph rewriting techniques and graph grammars, AToM³ [9] is a tool for multi-paradigm modeling. It has a
metamodeling layer in which different formalisms are modeled graphically using Entity-Relationship (ER) notation. AToM³
then generates a tool to process models described in the specified formalism. The models in AToM³ are internally repre-
sented using Abstract Syntax Graphs. Thus, transformations between formalisms are reduced to graph rewriting and the
transformations themselves can be expressed as graph-grammar models.

The UML-based language for specifying domain-specific patterns [13] defines a Role-based Metamodeling Language
(RBML) that modifies the UML metamodel to express domain variability in terms of element multiplicity. When specifying
a particular application, stereotypes serve to connect the application elements to the relevant domain (pattern) elements.
However, the approach supports variability to a limited extent. In particular, application-specific additions and their correct
usage are not explicitly supported, although allowed. In later work, Kim and Shen [21] suggest a conformance mechanism
which validates application models against the relevant domain models. But this mechanism lacks special treatments for
application-specific additions. For example, if a direct association in the structural domain model (termed Static Pattern
Specification, SPS) is replaced by indirect associations through an additional application-specific class in the class diagram,
the conformance mechanism will result in the conclusion that the class diagram does not conform to the SPS, limiting the
possible variants of a SPS.

Jarzabek et al. [18] suggest a modeling method and a Flexible Variant Configuration tool for modeling domains as domain
defaults and variants. Domain defaults contain requirements that characterize a typical system in a domain (i.e., they specify
the commonality in the domain). Variants are described as deltas in respect of domain defaults. The tool then interprets vari-
ants to produce customized domain model views for a system that meets specific requirements. In an earlier work, Zhang
and Jarzabek [53] describe a variant configuration language that allows equipping domain models with variation points and
record variant dependencies. An interpreter of this language produces customized views of a domain model, helping analysts understand and reuse software models. However, these works concentrate on representation issues and do not
guide the creation of such domain models.

The MetAmodel Recovery System (MARS), developed by Javed et al. [19], aims at inferring metamodels from collections of
instance models. A typical usage of this system is opening application models (termed instance models) in domain-specific
modeling environments, which often require the existence of the domain models (termed metamodels) with which the
application has been developed. When the metamodel is no longer available for an instance model, e.g., due to evolution
it has undergone, the instance model may not be loaded into the modeling tool. The system includes a program transforma-
tion tool and a grammar inference engine which transform a set of instance models (in XML) to an inferred metamodel (in
gME [22,45]). The authors claim that based on their case studies and experimental results, MARS successfully inferred metamodels for different (toy and real-world) domains. However, since it is impossible to “infer accurately the generalization hierarchy of a metamodel from the instance model, the total number of elements generated in the inferred metamodel is almost always higher than the elements used in the original metamodel”. Moreover, the system works with GME (and may be extended to other domain analysis metamodeling approaches), focusing on structural inference of domain artifacts.

In this study we provide an approach, supported by a tool, to generate draft domain models from existing applications in a
domain. To represent the application and domain models, we use a general domain engineering approach, called Application-
based DOmain Modeling (ADOM), which combines ideas from both feature-oriented and metamodeling approaches, in
an endeavor to furnish what they lack. A short overview of ADOM as well as reasons for choosing it in the current study
are provided next.

2.3. Application-based DOmain Modeling (ADOM)

The Application-based DOmain Modeling (ADOM) approach [38] handles applications and domains similarly, but on dif-
ferent abstraction layers, using regular application or software engineering tools and techniques. The general framework of
ADOM has three layers: (1) application, which is the most concrete layer, consisting of models of particular (information and software) systems, including their structure and behavior; (2) language, which is the most abstract layer, containing metamodels of modeling languages, such as UML; and (3) domain, which is the intermediate layer, consisting of specifications of various application families, including their common features and allowed variability. Separating the application and domain layers from the language layer, ADOM can be used in conjunction with different modeling languages, but when ADOM is adopted with a specific modeling language, this is used in the application layer and the domain layer, employing the same constructs and terminology in both, and easing inter-layer domain engineering activities such as guidance and validation of application models according to the domain constraints [39].

In the domain layer, the domain’s main concepts and their relations are specified. The domain artifacts are constructed from the knowledge gained from developing (previous) applications in the domains and studying the relevant literature. The domain artifacts also include information that may guide the creation of application models in a certain domain, e.g., whether an element is mandatory or optional, or whether different variants of an element may appear in a single application in that domain. This type of specification is modeled by means of multiplicity indicators supplied by the modeling language used in conjunction with ADOM. In UML [32], for example, the multiplicity indicators can be expressed as “multiplicity” stereotypes, which are associated with the top level Element metaclass to represent how many times a model element of this type can appear in a specific context. This stereotype has two associated tagged values, min and max, which respectively define the lowest and highest multiplicity boundaries. For clarity, four commonly used multiplicity groups are defined on top of this stereotype: (1) “optional many”, which indicates that any number (including 0) of application elements can be classified as this domain element, (2) “optional single”, which indicates that one application element at most can be classified as this domain element, (3) “mandatory many”, which indicates that at least one application element must be classified as this domain element, and (4) “mandatory single”, which indicates that exactly one application element must be classified as this domain element. Note that any range multiplicity constraint can be specified using the general “multiplicity min = n, max = m” construct.

In the application layer, application models are built using the domain terminology as expressed in the domain layer. The domain elements are used as domain classifiers in this layer, associating an application element to its relevant domain elements. Some generic elements may be omitted from the application model (these should be specified as optional elements in the domain model) and some new specific elements may be inserted into the specific application model (termed application-specific elements). Nevertheless, the domain knowledge embedded in the generic model must be maintained in the specific one.

As an example of the usage of ADOM, Appendices A–C provide models of a domain of process control systems (PCS) and two applications in it, a home climate control (HCC) system and a water level control (WLC) system. All models are expressed in terms of UML 2.0 class and sequence diagrams. PCS applications monitor and control the values of certain variables through a set of components that work together to achieve a common objective or purpose [11]. According to the domain model in Appendix A, all applications in the domain should define their sensors, controlled elements, controlled values, and controlled devices, as well as the relations among them. The applications should also specify scenarios of monitoring the controlled values and activating the controlled devices accordingly. In particular, each application in the PCS domain has to have exactly one controller, which exhibits at least one operation for monitoring and acting, and possibly several notification operations. A PCS application should also have at least one controlled element class and at least one controlled value class. Each controlled element class exhibits at least one attribute specifying its identity, zero or more enumerated attributes specifying its status, at least one Boolean operation checking certain conditions, and at least one operation for monitoring and activating the controlled element at hand. Each controlled value class exhibits two or more attributes specifying its range constraints, and at least one operation for getting these ranges. As the OCL constraint specifies, each controlled value class can be connected to either the controller class or to controlled element classes, so each corresponding association is optional. The cardinality of these associations indicates the cardinality that these associations have to obtain should the relations be included (instantiated) in a particular application in the domain.

In a typical monitoring and acting scenario, a controller object may or may not appear. If it appears, it may activate the monitoring and acting operation of its controlled elements, which in turn sample their controlled values and sensors and record them (if required).

The variety within the PCS domain is quite large. Applications in the domain differ in the number of the controlled elements, in the number and type of controlled values and sensors, in whether the history of measurements is recorded or not, in whether the range constraints are specified to the entire system or to each controlled element separately, in whether the system is configurable, etc. This variety is captured in the PCS domain model to guide the creation of new application models in the domain, as well as to validate the correctness and completeness of a particular process control system [39]. The HCC application, for example, whose model is set forth in Appendix B, ensures that the temperature and humidity in the rooms of a house satisfy certain constraints. Each person (user) has his or her preferred range constraints and the system has to adjust a room’s temperature and humidity according to its entrants’ requirements. The controlled elements in this application are the rooms of the house; its controlled values are temperature and humidity, which are independent of the entered rooms and reflect the wish of the persons entering; the sensors are human identifier components, thermometers, and humidity gauges; and, finally, the controlled devices are air-conditioners and water sprayers. The purpose of the WLC application, whose model is shown in Appendix C, is to monitor and control the water levels in the tanks, ensuring that the actual water level of tank $i$ is always in the closed range $[\text{Low}_i, \text{High}_i]$. If a problem occurs and some tanks do not satisfy their boundary
constraints, the system tries to resolve the problem internally, for example, by rebooting. If the problem cannot be resolved internally and at least one tank, tank \( i \), goes beyond its extended boundaries \([\text{Lowest}_i, \text{Highest}_i]\), an external-hazard handling system is activated to try to resolve the problem externally. In this application, the controlled elements are the water tanks; the controlled values are the tanks’ heights; the sensors are boundary sticks; and the controlled devices are filling and emptying faucets, which respectively inject and drain water when the water height in a certain tank reaches its low or high desired limits.

We chose ADOM for this research rather than other domain engineering approaches for these reasons: first, ADOM uses the same languages, techniques, and tools to specify applications and domains alike, whereas most reviewed works use different notations to specify domains (e.g., feature and class diagrams) and applications (e.g., use case, sequence, state, and other UML diagrams). Using the same languages, techniques, and tools in both layers may ease the task of creating domain models from application models as one terminology is used on the two levels. Second, ADOM is a general approach that can be used in conjunction with different modeling languages (e.g., UML, EPC, BPMN, and others). The only requirements from the modeling language used with ADOM are that it enables specification of multiplicity indicators and domain classifiers. The proposed approach can thus be easily extended to support different modeling languages and different development stages.

3. ADOM formalism

The formalism of ADOM is introduced in [39]. It is briefly provided here, as the basis of the SDM approach. In particular, the following definitions define application models, elements, and domain models in ADOM.

**Definition 1.** An application model \( A \) is a representation of certain aspects of the system of interest.

Each of Appendices B and C describes an application model.

The basic building blocks of an application model are elements, which are further divided into first-order, dependent, and relational elements.

**Definition 2.** A relational element in an application model \( A \) is a binary directional relationship between two other elements in \( A \). Notation: \( \text{re}_A = (s, t) \), where \( s \) and \( t \) are respectively the source and destination of \( \text{re}_A \).

UML associations and messages are examples of relational elements that respectively connect classes and objects (lifelines).

**Definition 3.** The \( V \)-type source elements group of an element \( e \) in an application model \( A \) is defined as

\[
\text{SRC}_V(e, A) = \{ s \in A \mid \exists \text{re}_A = (s, e) \land \text{type}(\text{re}) = V \}.
\]

Similarly, the \( V \)-type destination elements group of the element \( e \) in an application model \( A \) is defined as

\[
\text{DST}_V(e, A) = \{ t \in A \mid \exists \text{re}_A = (e, t) \land \text{type}(\text{re}) = V \}.
\]

In Appendix B all the associations are bi-directional, thus, for example, \( \text{SRC}_{\text{associations}}(\text{Room, Fig. 6}) = \text{DST}_{\text{associations}}(\text{Room, Fig. 6}) = \{\text{Climate Controller, Water Sprayer Item, Air Conditioner Item, Humidity Gauge Item, Thermometer Item, Human Identifier}\} \).

Since there are no other relational elements besides associations in Fig. 6, \( \text{SRC}_{\text{aggregations}}(\text{Room, Fig. 6}) = \text{DST}_{\text{aggregations}}(\text{Room, Fig. 6}) = \text{SRC}_{\text{dependencies}}(\text{Room, Fig. 6}) = \text{DST}_{\text{dependencies}}(\text{Room, Fig. 6}) = \emptyset \). Similarly, \( \text{SRC}_{\text{messages}}(\text{Room, Fig. 7}) = \{\text{Climate Controller, Room}\} \), \( \text{DST}_{\text{messages}}(\text{Room, Fig. 7}) = \{\text{Room, Thermometer Item, Human Identifier, Climate Controller, Air Conditioner Item}\} \).

**Definition 4.** A dependent element in an application model \( A \) is an element which has an implicit binary directional relationship with another element \( e \) in \( A \), such that the omission of \( e \) from the model implies the omission of \( d \). Notation: \( d \sim_A e \), where \( e \) is termed the dependee of \( d \) in \( A \).

Attributes and operations (methods) are examples of dependent elements in UML, as they depend on the existence of their owning classes; the omission of a class from a model implies the omission of all its attributes and operations. Objects (lifelines) in sequence diagrams depend on their corresponding classes in a class diagram, whereas messages depend on their owning combined fragments.

**Definition 5.** The \( V \)-type dependent elements group of an element \( e \) in an application model \( A \) is defined as

\[
\text{D}_V(e, A) = \{ d \in A \mid d \sim_A e \land \text{type}(d) = V \}.
\]

In Appendix B, for example, \( \text{D}_{\text{attributes}}(\text{Humidity Gauge Item, Fig. 6}) = \{\text{humidity gauge identifier, room humidity}\} \), \( \text{D}_{\text{operations}}(\text{Humidity Gauge Item, Fig. 6}) = \{\text{get room humidity}\} \), \( \text{D}_{\text{attributes}}(\text{Thermometer Item, Fig. 6}) = \{\text{thermometer identifier, room temperature}\} \), \( \text{D}_{\text{operations}}(\text{Thermometer Item, Fig. 6}) = \{\text{get room temperature}\} \), and so on.

**Definition 6.** A first-order element in an application model \( A \) is an element which is not relational or dependent in model \( A \).

Top-level classes are examples of first-order elements in UML.
Definition 7. A domain model DM in ADOM is a triple \((E_{DM}, MULT, m_i)\) where \(E_{DM}\) is a set of elements, \(MULT \subseteq N \times (N \cup \{\infty\})\) is a set of multiplicity pairs (where \(N\) is the set of the natural numbers), and \(m_i: E_{DM} \rightarrow MULT\) is a function. The elements in \(E_{DM}\) are termed domain model elements and the elements in MULT are termed multiplicity indicators.

In UML, MULT is specified utilizing the multiplicity stereotype and its two associated tagged values. Appendix A shows a possible domain model of process control systems in ADOM.

4. The SDM approach

The Semi-Automated Domain Modeling (SDM) approach gets application models (e.g., XMI files of UML models) and creates draft domain models that generalize the commonality and allowed variability of the given applications. We assume that a set of application models in the domain is given, and that these models’ correctness, completeness, and consistency are sufficient for the task in hand.\(^2\) We also assume that the element names consist of capitalized words in correct English, with no abbreviations, and with a space or underscore as a separator between the words.\(^3\) The domain models created by the SDM approach use the same terminology and notation as the application models and are represented in ADOM.

The process underlying the SDM approach takes three main steps (see Fig. 1): model matching, model merging, and model generalization. The rest of this section elaborates and demonstrates each step.

4.1. Model matching

As reviewed in Section 2.1, models can be matched by defining similarity among various elements. In the context of domain modeling, we distinguish four types of similarities: linguistic, meta-informational, dependent, and relational. Linguistic similarity uses the element names to measure the similarity of two elements. Meta-informational similarity uses meta-classes, types, and other kinds of meta-information to measure the similarity of two elements. For example, two elements modeled by classes are more alike than are two elements of which one is modeled by a class and the other by an attribute or an association. Dependent similarity measures the level of two elements’ similarity according to their internal structure, while relational similarity does so according to their context and their relations with other elements.

The general similarity of two elements is calculated in two phases. In the first, individual general similarity is calculated for each pair of elements without consideration of the relationships in which the elements participate. Thus, individual general similarity is a weighted average of the linguistic, meta-informational, and dependent similarities. In the second phase, general similarity is calculated as the weighted average of the individual general and relational similarities.

4.1.1. Linguistic similarity

Different similarity measurements have been proposed for calculating the distances between terms and sentences (see, e.g., [3] for a summary of WordNet-related measurements). Here we adopted Dao and Simpson’s [7] similarity measurement between two sentences, using WordNet [51]. We chose this measurement since it is simple and straightforward, and does not require a large corpus of statistics. We chose WordNet, which is an online lexical base for English language, because it is large, rich, and general-purpose, hence, can be used for creating different domains.

Definition 8. The linguistic similarity of two elements \(e_1\) and \(e_2\), \(L_{sim}(e_1, e_2)\), is a number between 0 and 1 that reflects the similarity between their names:

\[
L_{sim}(e_1, e_2) = \frac{\sum_{i=1}^{m} \max_{j=1}^{n} l_{i, u_j}}{m + n}
\]

where \(t_1 \ldots t_m\) is the name of element \(e_1\), \(u_1 \ldots u_n\) is the name of element \(e_2\), \(m\) and \(n\) are respectively the numbers of words in the \(e_1\) and \(e_2\) names, and \(l_{i, u_j} = \frac{2v_i u_j}{N_1 + N_2 + 2N_1 N_2}\) is Wu and Palmer’s formula [52] for comparing two words (see Fig. 2).

As an example of calculating the linguistic similarity of two elements, consider the elements Desired Temperature and Desired Water Level in Figs. 6 and 9, respectively.

Table 1 summarizes the pair-wise similarity values of the element names, while the formula below calculates their linguistic similarity.

\(^2\) The approach will work also on applications whose correctness, completeness, and consistency are not sufficient. However, the value of the resultant domain models in this case is unclear. The intention of this assumption is that only established application models, which have been already developed and tested, but not necessarily mathematically validated, will be provided as inputs to SDM.

\(^3\) If these assumptions are wrong, different spelling correction and abbreviation expansion techniques should be applied on the models as pre-processing.
$L_{\text{sim}}(\text{Desired Temperature}; \text{Desired Water Level}) = \frac{(1.0 + 0.77) + (1.0 + 0.27 + 0.77)}{3 + 2} = 0.76$

4.1.2. Meta-informational similarity

Meta-informational similarity is calculated by means of the metamodel of the modeling language and Wu and Palmer’s formula [52].

**Definition 9.** The *meta-informational similarity of two elements* $e_1$ and $e_2$, $M_{\text{sim}}(e_1; e_2)$, is a number between 0 and 1 that reflects Wu and Palmer’s measurement [52] between the meta-classes of the elements in the language metamodel. Formally expressed, $L_{\text{sim}}(\text{Desired Temperature}; \text{Desired Water Level}) = \frac{(1.0 + 0.77) + (1.0 + 0.27 + 0.77)}{3 + 2} = 0.76$

The meta-informational similarity of the two aforementioned elements, Desired Temperature and Desired Water Level, is 1, since both are represented by classes. Now, consider that Desired Temperature is modeled as an association (between the classes Room and Value) and not as a class. Fig. 3 depicts the relationships between these meta-classes according to the metamodel in [32]. Hence, the meta-informational similarity of these elements in this case is:

$$M_{\text{sim}}(\text{Desired Temperature}; \text{Desired Water Level}) = \frac{2 + 3}{2 + 1 + 2 + 3} = 0.67.$$

4.1.3. Dependent similarity

Dependent similarity takes into consideration the dependent elements of certain elements. The structure of compound elements, i.e., elements which exhibit dependent elements, can be used to measure their similarity. In class

---

4 To imitate a thesaurus, such as WordNet, only inheritance relations are taken into consideration for this calculation. Also, we assume that the language used in conjunction with ADOM has a metamodel, or that such a metamodel can be developed.
diagrams, for example, we can assume, with some confidence, that classes that exhibit similar attributes and operations are similar too.

**Definition 10.** The dependent similarity of two elements \(e_x\) and \(e_y\), \(D_{\text{sim}}(e_x, e_y)\), is a number between 0 and 1 calculated as a weighted average of the individual general similarity of its dependent elements. Formally expressed,\(^5\)

\[
D_{\text{sim}}(e_x, e_y) = \sum_{\text{type V}} W_{\text{DV}} \left[ \frac{\sum_{i=1}^{m} \max_{j=1-n} I G_{\text{sim}}(de_i, de_j) + \sum_{i=1}^{m} \max_{j=1-n} I G_{\text{sim}}(de_i, de_j)}{m + n} \right]
\]

where \(M_x\) and \(M_y\) are models (possibility the same one) such that \(e_x \in M_x\) and \(e_y \in M_y\), \(m = |D_V(e_x, M_x)|\) and \(n = |D_V(e_y, M_y)|\) are respectively the numbers of dependent elements whose dependees are \(e_x\) and \(e_y\), \(de_i \in D_V(e_x, M_x)\), \(de_j \in D_V(e_y, M_y)\), \(I G_{\text{sim}}(de_i, de_j)\) is the individual general similarity (after phase 1, without considering relationships) of the dependent elements \(de_i\) and \(de_j\), and \(W_{\text{DV}}\) is the weight assigned to the similarity of dependent elements of type \(V\),

\[
\sum_{\text{type V}} W_{\text{DV}} = 1.
\]

To calculate the dependent similarity of **Desired Temperature** and **Desired Water Level**, we need to calculate the individual general similarity of their attributes and operations. The individual general similarity of attributes takes into consideration linguistic and meta-informational aspects, since attributes have no dependent elements. The individual general similarity of operations takes into consideration linguistic (operation names), meta-informational (e.g., returned value types), and dependent (operation parameters) aspects. Assuming that the probability of modeling similar elements with different signatures in the process-control domain is relatively high, we can use 0.8 as the weight of linguistic similarity for both attributes and operations, 0.2 as the weight of meta-informational similarity for attributes, 0.1 as the weight of meta-informational similarity for operations, and 0.1 as the weight of dependent similarity for operations. Furthermore, since the importance of attributes and operations may be considered equal in a class specification, we can use similar weights for attributes’ and operations’ similarities. Hence, one can calculate the dependent similarity of the aforementioned classes as (see Table 2 for details): \(D_{\text{sim}}(\text{Desired Temperature}, \text{Desired Water Level}) = \frac{\text{Attrsim} + \text{Oppersim}}{2} = 0.91\).

**4.1.4. Relational similarity**

Relational similarity takes into consideration the relationships in which specific elements participate as sources or destinations.

\(^5\) Since an element does not depend on itself, there is no infinite recursion in this similarity formula.
The similarity matrix for the attributes and operations of the classes Desired Temperature and Desired Water Level.

### Table 2

<table>
<thead>
<tr>
<th>Type</th>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Bound</td>
<td>MI\text{sim} = 1 (both are double)</td>
<td>getDesired</td>
</tr>
<tr>
<td></td>
<td>L\text{sim} = 0.75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IG\text{sim} = 0.80</td>
<td></td>
</tr>
<tr>
<td>High Bound</td>
<td>MI\text{sim} = 1 (both are double)</td>
<td>getExtended</td>
</tr>
<tr>
<td></td>
<td>L\text{sim} = 0.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IG\text{sim} = 0.90</td>
<td></td>
</tr>
<tr>
<td>Minimum Bound</td>
<td>MI\text{sim} = 1 (both are double)</td>
<td>Water</td>
</tr>
<tr>
<td></td>
<td>L\text{sim} = 0.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IG\text{sim} = 0.85</td>
<td></td>
</tr>
<tr>
<td>Maximum Bound</td>
<td>MI\text{sim} = 1 (both are double)</td>
<td>Range</td>
</tr>
<tr>
<td></td>
<td>L\text{sim} = 0.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IG\text{sim} = 0.85</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Attr}_{\text{sim}} \approx 0.89 \]

\[ \text{Oper}_{\text{sim}} \approx 0.94 \]

**Definition 11.** The *relational similarity of two elements* \( e_x \) and \( e_y \), \( R_{\text{sim}}(e_x, e_y) \), is a number between 0 and 1 that is calculated as a weighted average of the individual general similarity of the relational elements in which \( e_x \) and \( e_y \) are sources or destinations. Formally expressed,

\[
R_{\text{sim}}(e_x, e_y) = \sum_{\text{type} \in V} W_{RV} \left[ \frac{1}{h} \left( \sum_{j=1}^{m} \text{max}\text{IG}_{\text{sim}}(s_j, s_j) + \sum_{j=1}^{m} \text{max}\text{IG}_{\text{sim}}(s_j, s_j) \right) + \sum_{q=1}^{p} \text{max}\text{IG}_{\text{sim}}(t_q, t_q) + \sum_{q=1}^{p} \text{max}\text{IG}_{\text{sim}}(t_q, t_q) \right] \right]^{1/N} + \frac{1}{k} \left( \sum_{j=1}^{m} \text{IG}_{\text{sim}}(s_j, s_j) + \sum_{j=1}^{m} \text{IG}_{\text{sim}}(s_j, s_j) \right) + \frac{1}{k} \left( \sum_{j=1}^{m} \text{IG}_{\text{sim}}(t_q, t_q) + \sum_{j=1}^{m} \text{IG}_{\text{sim}}(t_q, t_q) \right)
\]

where \( M_x \) and \( M_y \) are models (possibility the same one) such that \( e_x \in M_x \) and \( e_y \in M_y \), \( m_t = |\text{SRCV}\ (e_x, M_x)| \) and \( n_t = |\text{SRCV}\ (e_y, M_y)| \) are respectively the numbers of elements connected to \( e_x \) and \( e_y \) as sources of relational elements of type \( V \), \( s_j \in \text{SRCV}\ (e_x, M_x) \), \( s_j \in \text{SRCV}\ (e_y, M_y) \), \( k_y = |\text{DSTV}\ (e_x, M_x)| \) and \( p_y = |\text{DSTV}\ (e_y, M_y)| \) are respectively the numbers of elements connected to \( e_x \) and \( e_y \) as destinations of relational elements of type \( V \), \( t_q \in \text{DSTV}\ (e_x, M_x) \), \( t_z \in \text{DSTV}\ (e_y, M_y) \),

\[
h = \begin{cases} 
2, & (m_t + n_t) > 0 \land (k_y + p_y) > 0 \\
1, & (m_t + n_t) > 0 \lor (k_y + p_y) > 0 
\end{cases}
\]

\( \text{IG}_{\text{sim}}(s_i, s_i) \) and \( \text{IG}_{\text{sim}}(t_q, t_q) \) are respectively the individual general similarities of the source elements \( (s_i \text{ and } s_i) \) and the destination elements \( (t_q \text{ and } t_q) \) and \( W_{RV} \) is the weight assigned to the similarity of relational elements of type \( V \),

\[ \sum_{\text{type} \in V} W_{RV} = 1 \]

### 4.1.5. Calculating the similarity of behavioral elements

The SDM approach refers to both structural and behavioral aspects of the modeled applications. The different elements are divided into first-order, dependent, and relational elements, and then their linguistic, meta-informational, dependent, and relational similarities are calculated. As an example of a challenging usage of the SDM approach consider UML sequence diagrams. Automatic sequence matching is a very complex task, since (1) sequence diagrams express different procedural and behavioral aspects, some of them implicitly (e.g., message order), (2) sequence diagrams, which are similar in all aspects but the message order, may be considered similar, and (3) sequence diagrams depend on the structural aspects as expressed in the class diagrams (hence “weakly similar” structural elements may be not similar due to behavioral issues). Indeed, previous studies have tackled behavior integration of a single object represented in different state diagrams (e.g., [14,29]). Some approaches suggest merging multiple sequence diagrams by first converting them to state diagrams [49] or finite sequential processes [46].

The SDM approach refers to a sequence diagram as the dependee of its elements, that is, the combined fragments, messages, and objects in a sequence diagram depend on the sequence diagram. The elements in a sequence diagram are further divided into dependent and relational: objects, combined fragments, and messages are dependent elements, whereas messages are also relational. Besides depending on the sequence diagram, objects also depend on classes (from the class diagram), procedure calls on operations (from the class diagram), and combined fragments and messages on their owning combined fragments (if they exist). To consider message order while matching, relational elements \( re_S = (m_2, m_1) \) are implicitly added between any two messages \( m_1 \) and \( m_2 \) such that \( m_1 \) precedes \( m_2 \) in \( S \).
The linguistic similarity of two sequence diagrams refers to the names of these diagrams, while their dependent similarity refers to the similarities of the different combined fragments, messages, and objects that constitute the sequence diagrams. The meta-informational and relational similarities of sequence diagrams are calculated through their dependent elements (e.g., whether the sequence diagrams contain actor instances vs. class objects, procedure calls vs. asynchronous messages, and loop, alt, or opt combined fragments).

4.2. Model merging and generalization

To merge elements we refer only to elements whose general similarity, $G_{\text{sim}}(e_1, e_2)$, is greater than a certain threshold, named merge threshold (mTH). Procedure 1 returns $G_{\text{sim}}(e_1, e_2)$ only for “similar enough elements”.

Procedure 1: similar $(e_1, e_2)$

\[
\text{if } G_{\text{sim}}(e_1, e_2) > \text{mTH} \\
\text{then return } G_{\text{sim}}(e_1, e_2) \\
\text{else return 0}
\]

We then divide the different elements into groups of similar elements.

**Definition 12.** Let $A_1 \ldots A_k$ be $k$ application models in a domain. A Similarity Group, SG, is defined as a group whose elements are pair-wise similar. Formally expressed, $\forall e, e' \in \text{SG} \quad G_{\text{similar}}(e, e') > 0$.

Since all the elements in the same similarity group will be generalized into one domain element, the division of elements into similarity groups should be structured. The following definition specifies the relevant terminology.

**Definition 13.** Let $A_1 \ldots A_k$ be $k$ application models in a domain, and let $SG_1, \ldots, SG_m$ be $m$ similarity groups. $SG_1, \ldots, SG_m$ are well structured similarity groups of applications $A_1 \ldots A_k$ if:

1. The elements in the same similarity group are all of the same type (first-order, dependent, or relational).
2. Each element in the applications belongs to at least one similarity group.
3. The dependees of dependent elements that belong to the same similarity group belong to a single similarity group.
4. The sources of relational elements that belong to the same similarity group belong to a single similarity group.
5. The destinations of relational elements that belong to the same similarity group belong to a single similarity group.

Well structured similarity groups are created as follows. First, the pair of elements whose general similarity is the highest is retrieved (if more than one pair have the highest general similarity, one of them is chosen arbitrarily). The similarity group of this pair is calculated by inclusion of the two elements in the similarity group and addition of elements which are similar to all the elements that are already included in the similarity group but do not violate the structuredness constraints. This way we ensure that only elements similar enough to each other will be later merged into one

---

**Table 3**

<table>
<thead>
<tr>
<th>SG#</th>
<th>Included elements</th>
<th>Owning application</th>
<th>SDM generated element</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HumidityGaugeItem, ThermometerItem, WaterSprayerItem, AirConditionerItem</td>
<td>HCC</td>
<td>Item_Measuring_instrument</td>
</tr>
<tr>
<td>2</td>
<td>DesiredHumidity, DesiredTemperature, DesiredWaterLevel</td>
<td>HCC</td>
<td>Desired_State</td>
</tr>
<tr>
<td>3</td>
<td>WaterController, ClimateController</td>
<td>WLC</td>
<td>Controller_Entity</td>
</tr>
<tr>
<td>4</td>
<td>Room, Tank</td>
<td>HCC</td>
<td>Room</td>
</tr>
<tr>
<td>5</td>
<td>HumanIdentifier, AirConditionerItem, HumidityGaugeItem</td>
<td>HCC</td>
<td>Item_Entity</td>
</tr>
</tbody>
</table>

---

6 Well structured similarity groups exist for any set of applications: in the worst case, each similarity group will include only one element.
generalized element. However, the order of entry into a similarity group is important, and is determined according to descending order of the mean general similarity values of an element to the elements already included in the similarity group. When no more elements can be included in that similarity group, a new similarity group is defined, to include the two elements with the highest general similarity value where at least one of them is not included in the previously formed similarity groups. The process continues until each element appears at least in one similarity group (in the worst case this similarity group contains only one element). Note that the same element may appear in several similarity groups. Procedure 2 describes the similarity group creation procedure, which gets an array of k application models and returns a set of m similarity groups.

Table 3 summarizes the similarity groups that emerged for classes in the HCC and WLC applications. The elements Humidity Gauge Item and Thermometer Item are the most similar (their general similarity is 0.87). Hence, they are the core elements of the first similarity group. Next, both Water Sprayer Item and Airconditioner Item are potential candidates to be included in this similarity group. However, the mean similarity value between Water Sprayer Item and the elements already included in the similarity group is 0.76, while the mean similarity value between Airconditioner Item and these elements is 0.74. Thus, Water Sprayer Item is inserted into this similarity group. Following similar line of reasoning, the next element, the last to be inserted into this similarity group, is Airconditioner Item. The algorithm continues until all the elements are organized in similarity groups.

### Procedure 2: similarityGroupCreation (AppArray) 

```plaintext
SGS=null // initializing the expected set of m similarity groups
// ES = similar(ei, ej). ei, ej are elements in applications (possibly in the same application) in the array sent
// to the procedure as input, namely AppArray
ES=calculateElementsSimilarity(AppArray) // to the procedure as input, namely AppArray
// p is the next pair of elements whose general similarity is the highest
While (p=getNextCandidatePair(ES) != null) {
    SG = {p[0]} \cup {p[1]} // creating a similarity group that includes only the candidate pair
    // q is the next element whose mean general similarity value to the elements already included
    // in SG is the highest and above a certain threshold
    While (q=getNextCandidateElement(SG) != null) {
        // The wellStructured procedure checks all the conditions defined in Definition 13: if these
        // conditions are satisfied for SG \cup \{q\}, then SG is modified accordingly. Otherwise, q is not
        // included in SG.
        if (wellStructured (SG \cup \{q\}))
            then SG = SG \cup \{q\}
    }
    SGS = SGS \cup SG
}
Return (SGS)
```

#### Definition 14.

Let $A_1 \ldots A_k$ be k application models in a domain, and let $S_{G_1}, \ldots, S_{G_m}$ be m well structured similarity groups of these applications. The Appearance Frequency of a similarity group $SG$, $Fr(SG)$, is a number between 0 and 1 that represents the percentage of application models in which an element from SG appears. Formally expressed, $Fr(SG) = \frac{\sum_{i=1}^{k} f_i}{p}$, where:

- $f_i = \begin{cases} 1, & A_i \cap SG \neq \emptyset \\ 0, & \text{otherwise} \end{cases}$     \text{First order}

- $p = \begin{cases} k, & \text{first order dependent} \\ \frac{\{|A_i|d \in S_{G_1}, e \in S_{G_1} \text{ such that } e \in D_V(d, A_i) \land A_i \cap S_{G_n} \neq \emptyset\|}{\{|A_i|s \in S_{G_1}, t \in S_{G_2}, e \in S_{G_i} \text{ such that } e_{A_i} = (s, t) \land A_i \cap S_{G_1} \neq \emptyset \land A_j \cap S_{G_2} \neq \emptyset\|}, & \text{relational} \end{cases}$

Note that p represents the number of applications in which an element from SG could appear. Should the similarity group include only first order elements, p is the total number of applications (k). However, in the case of dependent elements, p is the number of applications in which dependees of elements from the similarity group appear. Similarly, in the case of relational elements, p is the number of applications in which both sources and destinations of elements from the similarity group appear.

Each (well structured) similarity group whose appearance frequency is greater than a certain frequency threshold is generalized into a single domain element, as described in Procedure 3. The other similarity groups, whose appearance frequency is equal to or less than the certain frequency threshold, contain elements considered application-specific, hence are not percolated to the domain model.
As an illustration to the name selection of the domain element, consider the second similarity group in Table 3: it contains three classes, named Desired Humidity, Desired Water Level, and Desired Temperature; the first two belong to the HCC application, and the last to the WLC application. This similarity group is generalized into a single element whose type is ‘first-order’. This element is mandatory and several variants of it may appear in the same application in the domain, since each of the two used applications has one or more elements from this similarity group. The name of the generalized element is Desired_Sate: ‘Desired’ repeats three times in this similarity group and State is the least common super-concept in WordNet of the remaining words (Humidity, Water, Level, and Temperature). The names generated for the other similarity groups in Table 3 are provided in its right column. Since ‘Room’ is an ancestor of ‘Tank’ in WordNet, the name generated to the fourth similarity group is ‘Room’, which does not explicitly capture the intention of the corresponding element in the PCS domain. However, this situation occurs due to the low number of applications used in this case and the low number of elements in the particular similarity group.

**Definition 15.** Let $A_1 \ldots A_k$ be $k$ application models in a domain, and let $SG_1, \ldots, SG_m$ be $m$ well structured similarity groups of these applications. A SDM-generated Domain Model, SDM, for the applications $A_1, \ldots, A_k$ is a domain model that satisfies for each $i = 1 \ldots m$ such that $Fr(SG_i) > frqTH$, there exists at most one element $g_i$ in SDM such that $g_i$ is the generalization of $SG_i$. Furthermore,

1. If $SG_i$ is a similarity group of dependent elements, and consequently $g_i$ is dependent, then $g_i$ appears in SDM only if the generalization of the dependees of the elements in $SG_i$, $h_i$, appears in SDM. In this case, $g_i_{SDM} \leftarrow h_i$.
2. If $SG_i$ is a similarity group of relational elements, and consequently $g_i$ is relational, then $g_i$ appears in SDM only if the generalization of the sources of the elements in $SG_i$, $g_{si}$, and the generalization of the destinations of the elements in $SG_i$, $g_{ti}$, appear in SDM. Furthermore, in this case, $g_i_{SDM} = (g_{si}, g_{ti})$.

The model generated by the SDM approach to the PCS domain using the HCC and WLC applications only is shown in Appendix D. As can be seen, the name selection in this model can be criticized as not representative. However, one should bear in mind the low number of applications and elements in this case and the low number of elements in the particular similarity group. Theorem. The SDM-generated domain model is well structured. In other words,

1. Each dependent element has its dependee in the model.
2. Each relational element has its own source and destination in the model.
Proof. Let g be a dependent element in SDM. Then, according to Procedure 3, it was generated from a similarity group SG of dependent elements. Since the set of similarity groups is well structured, a similarity group SG' exists such that \( \{ d | \exists e \in SG_i \cap D(d, A_i) \subseteq SG' \} \subseteq SG' \). According to Definition 15, g is included in SDM only if the generalization of SG' is included in SDM. Hence, the generalization of SG' is included in SDM and g depends on this generalization. Bullet (1) is proven.

Let g be a relational element in SDM. Then, according to Procedure 3, it was generated from a similarity group SG of relational elements. Since the set of similarity groups is well structured, similarity groups SG' and SG'' exist such that \( \{ s | \exists e = (s, t) \in SG_i \cap A_i \subseteq SG' \} \subseteq SG' \) and \( \{ t | \exists e = (s, t) \in SG_i \cap A_i \subseteq SG'' \} \subseteq SG'' \). According to Definition 15, g is included in SDM only if the generalization of SG' and the generalization of SG'' are included in SDM. Hence, the generalization of SG' and the generalization of SG'' are included in SDM and g connects these generalizations. Bullet (2) is proven.

Analyzing the complexity of the algorithm, one sees that procedure 2, which is the algorithm core, runs in \( O(|E|^2) \), where |E| is the total number of elements (first-order, dependent, and relational) in all applications that are generated into a single domain model. Creating the element similarity (ES) matrix (running procedure 1) requires \( |E|^2 \) runs, whereas creating the similarity groups requires an additional \( O(|E|^2) \) runs. Procedure 3 is performed for each similarity group. The maximal number of similarity groups is |E| (in this case each similarity group has only one element). Thus, the overall complexity of the algorithm is \( O(|E|^2) \).

5. Evaluation of the approach

To evaluate the approach, we implemented the SDM approach for UML 2.0 class and sequence diagrams. We ran the implemented SDM tool on different domains of various sizes (2–13 applications) and sources (e.g., reverse-engineered and forward developed). These runs were used for setting initial values for the different weights and thresholds, e.g., the linguistic similarity weight, \( w_L \), was set at 0.6, the dependent similarity weight, \( w_D \), was set at 0.25, and the relational similarity weight, \( w_R \), was set at 0.15. The tool, as well as some of the domains we used, can be found in [42]; a summary of this information is provided in Table 4. As the tool is expected to work with different domains, the initial values of the different weights and thresholds can be changed by the domain engineers.

We also carried out two controlled experiments whose purpose was to evaluate the quality of the SDM-generated models. Due to difficulties in conducting controlled experiments in industrial settings, because they require considerable time, effort, and resources [4], the two experiments reported in this paper were done in academia with students as subjects. Those who participated in both experiments were advanced information systems and software engineering students (final year undergraduate or graduate students). Most of them had experienced industrial work. In their courses they had gained deep knowledge in certain domains through different tasks so they were comparable to junior domain engineers. Nevertheless, these experiments are considered pilots; their replication in industrial settings is required and suggested as a future research direction.

In each experiment, the subjects were first requested to perform tasks that indicated their understanding of the domain and its models, and only then were they asked to evaluate the SDM-generated domain models. During the evaluation tasks the subjects were required to grade the quality of the SDM-generated domain models in terms of correctness, completeness, redundancy, consistency (if relevant), and comprehensibility. The precise questions classified for each criterion are listed in Table 5.

5.1. SDM-generated models vs. human-made domain models

The first experiment aimed at checking whether SDM-generated models are valuable, i.e., may help create qualitative draft domain models (to be improved by domain engineers). For this purpose, SDM-generated models were compared with human-made domain models. The experiment was conducted with an advanced class of 20 Management Information Systems students at the University of Haifa, Israel, who had taken a seminar on domain engineering in 2007. During the seminar

<table>
<thead>
<tr>
<th>#</th>
<th>Domain name</th>
<th>Number of applications</th>
<th>Size indicator</th>
<th>Way of development</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Process control systems</td>
<td>2</td>
<td>Maximum 10 classes</td>
<td>Expert development</td>
</tr>
<tr>
<td>2</td>
<td>Project management systems</td>
<td>4</td>
<td>Maximum 9 classes</td>
<td>Student development + corrections according to expert feedbacks</td>
</tr>
<tr>
<td>3</td>
<td>Scheduling systems 1</td>
<td>4</td>
<td>Maximum 12 classes</td>
<td>Student development + corrections according to expert feedbacks</td>
</tr>
<tr>
<td>4</td>
<td>Resource allocation and tracking</td>
<td>3</td>
<td>Maximum 15 classes</td>
<td>Student development + corrections according to expert feedbacks</td>
</tr>
<tr>
<td>5</td>
<td>Scheduling systems 2 (a wider scope that includes the applications in domains 2–4)</td>
<td>13</td>
<td>Maximum 29 classes</td>
<td>11 student development + corrections according to expert feedbacks, 2 reversed engineered from open source</td>
</tr>
</tbody>
</table>
students in groups of four were requested to develop domains. Each group was divided into two pairs: the first pair was responsible for creating a domain model according to the literature review, while the second pair reverse engineered and abstracted open-source applications in the domain. In addition, each member of the group was responsible for developing the model of an application in the domain. This way, five domains were developed during the seminar and each domain had four modeled applications. One of these domains included Project Management applications.

During the experiment, the subjects (i.e., the students who took the seminar that year) were asked to fill out a questionnaire, which can be found in [42]. It included (1) a paragraph describing the Project Management domain, (2) the domain model as generated by the implemented SDM approach on the four applications of the project management group, (3) a domain model created by reverse engineering and abstracting open-source applications in the domain, and (4) a domain model created by studying domain ontologies and literature. Each model contained UML 2.0 class and sequence diagrams. The participants were requested to map the core elements of the automatically created domain model to their counterparts (if any) in the two human-made domain models. They were also asked to evaluate the automatically created domain model according to different criteria, including correctness, completeness, redundancy, consistency, and comprehensibility (see Table 5). Grading these questions may be criticized as subjective; however, the subjects were requested to complete this part after the mapping task, as well as to provide justification for their grades. The grades they gave and their responses to the two tasks correlated well.

<table>
<thead>
<tr>
<th>Table 6</th>
<th>Performance in mapping SDM elements to the human-made model elements.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Literature review and ontologies</td>
</tr>
<tr>
<td></td>
<td>Literature review and ontologies</td>
</tr>
<tr>
<td></td>
<td>Literature review and ontologies</td>
</tr>
<tr>
<td></td>
<td>Literature review and ontologies</td>
</tr>
<tr>
<td>PM (%)</td>
<td>Others (%)</td>
</tr>
<tr>
<td>Classes</td>
<td>100.0</td>
</tr>
<tr>
<td>Objects</td>
<td>91.7</td>
</tr>
<tr>
<td>Messages</td>
<td>94.4</td>
</tr>
</tbody>
</table>

Tables 6 and 7 summarize the experiment results. In these tables we separated the performance of (and grades given by) subjects who belonged to the Project Management (PM) group from the others, as the former had prior knowledge of the domain, and indeed their performance was generally better than the others'. However, the grades that the PM group members gave the SDM model were very similar to those given by the rest of the class.

The results show that while the PM group performed almost equally well in both human-made models, the other subjects fared much better in mapping the SDM objects and messages to the model that was created by reverse engineering applications in the domain. A possible reason for this may be that the reversed engineered model was concrete and not very detailed – a circumstance that facilitated its mapping to the SDM objects and messages. The literature-based domain model, on the other hand, was rich and included numerous messages: some subjects just skipped the correct mappings to the SDM domain model, though these were quite explicit. Since the number of students was small and the errors were different, no classification of the errors could be done.

The grades given by the PM group members for diagram consistency and abstraction-level properness were higher than those given by the other participants to these categories (by 0.5 of a point and higher); the other categories were evaluated quite closely. A possible reason for this result may be that the prior knowledge possessed by the PM group about the domain at hand provided them a more general and wider viewpoint on the domain. We also observe that the grades given to the sequence diagrams in these categories were higher than those given to the class diagram. This is a complicated result to

7 Direct link:http://www.mis.hevra.haifa.ac.il/~iris/research/SDM/SDMexp1.pdf.
### Table 7
Grading the SDM model (the grades are out of 5) – first experiment.

<table>
<thead>
<tr>
<th>Model correctness</th>
<th>Model completeness</th>
<th>Redundancy</th>
<th>Diagrams consistency</th>
<th>Correct stereotype recognition</th>
<th>Abstraction level properness</th>
<th>General grade</th>
<th>Overall average</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>SD</td>
<td>Overall</td>
<td>CD</td>
<td>SD</td>
<td>Overall</td>
<td>CD</td>
<td>SD</td>
</tr>
<tr>
<td>PM AVG</td>
<td>3.5</td>
<td>3.9</td>
<td>3.3</td>
<td>3.7</td>
<td>3.5</td>
<td>3.1</td>
<td>3.8</td>
</tr>
<tr>
<td>Others AVG</td>
<td>3.7</td>
<td>4.2</td>
<td>3.9</td>
<td>3.6</td>
<td>3.9</td>
<td>3.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Overall AVG</td>
<td>3.6</td>
<td>4.1</td>
<td>3.9</td>
<td>3.5</td>
<td>3.8</td>
<td>3.4</td>
<td>3.7</td>
</tr>
</tbody>
</table>

Legend: CD – Class Diagram, SD – Sequence Diagram.

### Table 8
Evaluating the relevance and mapping of the SDM-generated model elements.

<table>
<thead>
<tr>
<th>Domain relevancy (%)</th>
<th>Correct mapping to ozone (%)</th>
<th>Correct mapping from ozone (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>73</td>
<td>65</td>
</tr>
</tbody>
</table>

### Table 9
Grading the SDM model (grades are out of 5) – experiment 2.

<table>
<thead>
<tr>
<th>Model correctness</th>
<th>Model completeness</th>
<th>Redundancy</th>
<th>Correct stereotype recognition</th>
<th>Abstraction level properness</th>
<th>General grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9</td>
<td>3.7</td>
<td>3.9</td>
<td>2.3</td>
<td>2.3</td>
<td>3.1</td>
</tr>
</tbody>
</table>

### Appendix A: The PCS Domain Model

![Diagram of the process control domain](image)

Fig. 4. An ADOM class diagram of the process control domain.
explain, but it may be due to the relative simplicity of the class diagram notation, which might have made the participants more circumspect about the results.

The participants’ main criticism of the draft domain model in this experiment was that the names of some relevant elements were too abstract or not compatible with their expected roles in the domain. However, since the repository used in this experiment was very small (four applications), yet the participants managed to correctly map the model’s concepts to the domain terminology, this shortcoming may be tolerable.

5.2. SDM models vs. ontologies

In the second experiment we set out to compare the domain comprehensibility as reflected in SDM-generated models with that of ontologies. The domain selected for this experiment was scheduling, which has many open-source applications and different ontologies that describe it. A common scheduling ontology is OZONE [44], which defines a reusable and expandable set of concepts for describing and representing scheduling problems, solutions, and constraints. The five basic concepts in OZONE are demand, activity, resources, product, and constraint. Accordingly, OZONE is a very abstract ontology and its usability in domain engineering may be questionable. However, to check the evaluation of the abstraction level of the SDM-generated domain model, which was greatly criticized as being too low (concrete) in the first experiment, we decided to use this abstract ontology in the second experiment. To suggest a competitor to OZONE, we automatically created a SDM model from 13 different scheduling applications. Some of these are reverse-engineered models of open-source systems. Others were developed by students and junior developers. Furthermore, since ontologies concentrate on the structural aspects of domains, we referred in this experiment only to the class diagram of the SDM model.

The subjects in this experiment were 15 fourth-year undergraduate Information Systems and Software Engineering students at the Ben Gurion University of the Negev, Israel. These students studied a (different) advanced course on domain engineering and ADOM. They were requested to fill out a questionnaire with three parts: (1) a brief explanation of the domain and the applications used for the SDM domain model, (2) the SDM model of the domain (in the form of a class diagram), and (3) a model of the OZONE ontology. The questionnaire also set three tasks. The participants were asked first,

Fig. 5. An ADOM sequence diagram describing a monitoring and acting operation in the process control domain.
Appendix B: The HCC Application Model

Fig. 6. An ADOM class diagram of the home climate control application.

to evaluate the relevance of the main SDM model elements to the Scheduling domain; secondly, to map core SDM domain model elements, such as classes, to their counterparts (if exist) in the OZONE ontology and vice versa, and thirdly, to evaluate the automatically created domain model according to different criteria, such as correctness, completeness, redundancy, correct stereotype recognition, and abstraction-level properness. The complete questionnaires can be found in [42].8

The results of this experiment are summarized in Tables 8 and 9. As can be seen, the subjects found that 75% of the SDM model elements were relevant to the domain. Some elements were evaluated as not relevant to the domain because of problems in comprehending their names. The SDM-generated model included, for example, a _WorkersClassificationDriver_Message_ class, which only third of the subjects thought relevant to the domain. This class had no attributes and operations from which its purpose could be drawn. Accordingly the students experienced difficulty understanding the role of this class in the scheduling domain. Another example is a class with the name _Ticket_Equipment_. The term 'Ticket' was added as a prefix to the generalized element name due to its high frequency of appearance in the similarity group. However, all elements which had the word Ticket in this similarity group came from the same application (_eTicket_), while the term Equipment was generalized from fifteen elements from seven different applications. Therefore, the term Ticket does not correctly reflect the role of the generalized element. The experiment results in fact showed that only 47% of the participants marked this concept as relevant to the domain, and only 33% succeeded in mapping it to the OZONE ontology.

Regarding the mapping task, 73% of the mappings from the SDM classes to their ontology counterparts were correct, while only 65% of those in the opposite direction were correct. We believe that the main reason for the low performance

8 Direct link: http://www.mis.hevra.haifa.ac.il/~iris/research/SDM/SDMexp2.pdf.
in this task was the far greater abstraction of the OZONE ontology than that of the SDM-generated model, so mapping especially from the ontology to the domain model was difficult since it required concretization of concepts and expertise in the domain. Another interesting finding was that all subjects found at most one OZONE element for each SDM-generated element, even though in some cases there were several possible mappings, a few already recognized by the subjects while mapping the SDM elements to their OZONE counterparts.

The participants graded the SDM model syntactic correctness as 3.9 (out of 5), its completeness (i.e., presence of the core elements) as 3.7, and its redundancy as 3.9 (where 1 indicates that the model contains too many redundant elements). However, the properness of the abstraction level of the SDM model and the correctness of modeling its multiplicity-related variability were graded relatively low: 2.3. We believe that one of the main reasons for this criticism is that the repository of the applications, underlying the SDM domain model, consisted of applications from different sub-domains (e.g., resource allocation, resource tracking and project management). This fact caused a higher variability in element features and affected their similarity measurements. Another possible reason may be the chosen ontology to which the domain model was compared; as noted, OZONE is a highly abstract ontology, which may cause our domain model to look too concrete.

5.3. Threats to validity

As usual in controlled experiments, the main threats to the validity of the results are divided to construct, internal, conclusion, and external validity [50].

**Construct validity** threats concern the relation of theory and observation. They are mainly due to the method used to assess the outcomes of tasks. The tasks conceived in these experiments were objective – such as mapping, and
subjective – such as grading. However, the subjects were requested to undertake the subjective tasks only after performing the objective ones, in order to obtain a deeper understanding of the domains. Furthermore, they were asked to provide justification for every subjective answer.

**Internal validity** threats concern external factors that may affect a dependent variable. In particular, the representativeness of the application models and domains can be questioned. We chose two known, established domains, namely project management and scheduling, which raise both structural and behavioral challenges. In each domain we selected different applications developed by different individuals (in the first experiment) or by different methods (e.g., open source vs. self development, in the second experiment). However, we do not claim that these applications and domains are representative in any manner. Further experiments have to check this point.

**Conclusion validity** concerns the relationship between the treatment and the outcome. Due to the small number of participants in both experiments, no statistical analysis was done. However, the raw results, as well as detailed explanations of the conclusions, are provided.

**External validity** concerns the generalization of the findings. The main threat in this area stems from the type of subjects. In our case they were advanced students with little experience. However, as already noted, we claim that their background and knowledge likened them to junior domain engineers. Furthermore, we strove that throughout the courses staff members would accompany the students and provide detailed guidance for their tasks. Nevertheless, additional experiments on SDM outputs are required.

### 6. Summary and future work

The SDM approach aims at assisting domain engineers in their time-consuming and error-prone task of domain-model creation. The created domain models may serve as the basis for creating DSLs and domain-specific applications. The main benefit of this approach is that the whole process of generating the draft domain models is automatic and can be conducted
on application models’ repositories of various size. The different parameters of the approach (namely weights and thresholds) can be set and tuned separately for each domain, such that in wide domains, for example, the weight of the linguistic similarity will be relatively low, whereas its weight in narrow domains whose terminology is well established can be higher. Furthermore, the SDM approach can be applied and implemented with different languages, and thus provide domain engineering solutions to different phases of system development. The elements of the used language are divided into three groups (first-order, dependent, and relational) and the SDM model is constructed in three main steps: matching (through calculating different types of similarities between pairs of elements), merging (through creation of similarity groups), and generalizing (through replacing similarity groups by domain elements). Note that the suggested approach depends on access to existing applications, which may not always be available (e.g., for proprietary reasons). Thus, different sources of application models, such as self-developed and open source, are acceptable.

Two pilot experiments were conducted to demonstrate the value of our approach. They both consistently showed that SDM generates reasonable models (in terms of correctness, completeness, consistency, redundancy, and comprehensibility) even on small repositories of application models. However, as the experiments were conducted on two (small to medium) domains and with students, their validity is threatened. In particular, additional experiments on SDM outputs with experienced users and domain engineers (in industrial settings) are required. We plan to conduct such experiments, in which the level of detail of the resultant SDM models will be checked in a wide range of domains, and improvements for their content will be suggested.

Further enhancements are also required to improve human satisfaction with the abstraction level of the resultant domain models. In particular, the usage of WordNet in the context of domain engineering should be queried and analyzed. If a domain-specific thesaurus, ontology, or lexicon exists, then using it for enrichment and creation of domain models may improve satisfaction with the abstraction level of the generated domain models. However, in the general case we cannot assume that such lexical bases exist to any domain. Another improvement should be representing additional variants, besides

**Appendix C: The WLC Application Model**

![Diagram of the Water Level Control Application Model]

Fig. 9. An ADOM class diagram of the water level control application.
multiplicity-related and meta-informational variability, in the SDM-generated domain models. Finally, a systematic way of setting weights and thresholds in different domains has to be developed.

Acknowledgment

The author would like to thank Zeev Tavor for his help in the implementation and evaluation of the SDM tool.

Appendix A. The PCS domain model

See Figs. 4 and 5.
Appendix B. The HCC application model

See Figs. 6–8.

Appendix C. The WLC application model

See Figs. 9 and 10.

Appendix D. The SDM-generated model for the PCS domain

See Figs. 11 and 12.

Fig. 11. The SDM-generated class diagram of the process control domain.

Fig. 12. The SDM-generated sequence diagram describing a monitor room operation in the process control domain.
Iris Reinhartz-Berger is a faculty member at the management information systems department, University of Haifa. She received her B.Sc. in computer science and mathematics in 1994, M.Sc. in information systems engineering in 1999, and Ph.D. in information systems engineering in 2003, all from the Technion, Israel Institute of Technology, Haifa, Israel. Her research interests include conceptual modeling, modeling languages and techniques for analysis and design, domain analysis, development processes, and methodologies.