Iris Reinhartz-Berger · Arnon Sturm
Tony Clark · Sholom Cohen
Jorn Bettin *Editors*

Domain Engineering

Product Lines,
Languages, and
Conceptual Models

Springer

Iris Reinhartz-Berger · Arnon Sturm
Tony Clark · Sholom Cohen
Jorn Bettin   *Editors*

Domain Engineering

Product Lines,
Languages, and
Conceptual Models

Springer

Domain Engineering

Iris Reinhartz-Berger • Arnon Sturm •
Tony Clark • Sholom Cohen • Jorn Bettin
Editors

Domain Engineering

Product Lines, Languages,
and Conceptual Models

Springer

Editors

Iris Reinhartz-Berger
Dept. Information Systems
University of Haifa
Haifa
Israel

Arnon Sturm
Dept. Information Systems Engineering
Ben-Gurion University of the Negev
Beer-Sheva
Israel

Tony Clark
Business Information Systems
Middlesex University
London
United Kingdom

Sholom Cohen
Software Engineering Institute
Pittsburgh
Pennsylvania
USA

Jorn Bettin
S23M
Melbourne
Australia

Printed on acid-free paper

Preface: Introduction to Domain Engineering: Product Lines, Languages, and Conceptual Models

A *domain* is an area of knowledge that uses common concepts for describing phenomena, requirements, problems, capabilities, and solutions. A domain is usually associated with well-defined or partially defined terminology. This terminology refers to the basic concepts in that domain, their definitions (i.e., their semantic meanings), and their relationships. It may also refer to behaviors that are desired, forbidden, or perceived within the domain. *Domain engineering* is a set of activities that aim to develop, maintain, and manage the creation and evolution of domains.

Domain engineering has become of special interest to the information systems and software engineering communities for several reasons. These reasons include, in particular, the need to maintain and use existing knowledge, the need to manage increasing requirements for variability of information and software systems, and the need to obtain, formalize, and share expertise in different, evolving domains.

Domain engineering as a discipline has practical significance as it can provide methods and techniques that may help reduce time-to-market, development cost, and projects risks, on the one hand, and help improve product quality and performance on a consistent basis, on the other hand. It is used, researched, and studied in various fields, predominantly: software product line engineering (SPLE), domain-specific language engineering (DSLE), and conceptual modeling.

This book presents a collection of state-of-the-art research studies in the domain engineering field. About half of the chapters in this collection originated from a series of workshops, named domain engineering, which were associated with the Conference of Advanced Information Systems Engineering (CAiSE) during the years 2009–2011 and with the international conference on conceptual modeling (also known as the ER conference) in 2010. The authors of the other chapters were personally invited to contribute to this book. The chapters are organized in three parts. The first part includes research studies that deal with domain engineering in SPLE. The second part refers to domain engineering as a research topic within the field of DSLE. Finally, the third part presents research studies that deal with domain engineering within the field of conceptual modeling.

Part I: Software Product Line Engineering

A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1, 8]. While reuse has always made sense as a means to take advantage of the commonality across systems, most reuse strategies fail to have any real technical or economic impact. SPLE is a discipline that addresses technical and economic benefit and achieves strategic reuse of software across the product line through: (1) capturing common features and factoring the variations across the domain or domains of a product line; (2) developing core assets used in constructing the systems of the product line; (3) promulgating and enforcing a prescribed way for building software product line assets and systems; and (4) evolving both core assets and products in the product line to sustain their applicability.

Although SPLE has been a recognized discipline within the software engineering community for two decades, the practice of SPLE in industry still faces significant challenges in achieving strategic reuse. Challenges are seen in each of the four areas mentioned above. Specific examples include: (1) capturing commonality—modeling and representation approaches, tools, and analysis for variation and variation management; (2) developing assets—architecture design approaches including real-time embedded, design patterns, automatic generation of software, and design approaches including aspect-orientation; and (3) building software—implementation of software assets for reuse, composition techniques, and use of domain-specific languages (DSLs) for software construction. In addition, there is a need to deal with the evolution of the core assets and the product line systems and a need for specific tools to assist the coevolution of product line core assets and dependent systems.

In the field of SPLE, domain engineering deals with specifying, designing, implementing, and managing reusable assets, such as specification sets, patterns, and components, that may be suitable, after customization, adaptation, or even extension, to families of software products. The focus of domain engineering in this field is on conducting commonality and variability analysis and representing the results of this analysis in a comprehensible way. Commonly, feature-oriented methods and UML profiles are used for this purpose.

The chapters in this part of the book deal with application of domain engineering to address some of the aforementioned challenges. Two chapters deal with domain engineering to capture commonality and manage variation across a software product line:

- "Separating concerns in feature models: Retrospective and support for multi-views" by Mathieu Acher, Arnaud Hubaux, Patrick Heymans, Thein Than Tun, Philippe Lahire, and Philippe Collet looks at managing common features and their variants in software product lines with thousands of features. This chapter describes the separation of concerns, that can be applied to partition the feature space.

- "A survey of feature location techniques" by Julia Rubin and Marsha Chechik examines over 20 techniques that offer automated or semi-automated approaches to isolate features from existing software. This chapter provides a description of the overarching technology for isolating features for software code and analyzes the potential that each of the 20+ techniques offers. The chapter also provides guidance in selecting the appropriate technique.

One chapter deals with architecture and design for software product lines, specifically those software product lines that are real-time (RT) embedded:

- "Modeling real-time design patterns with the UML-RTDP profile" by Saoussen Rekhis, Nadia Bouassida, Rafik Bouaziz, Claude Duvallet, and Bruno Sadeg applies domain engineering to design RT patterns for capturing commonality and managing variation across the software product line. This chapter introduces UML-based models that represent the static and dynamic patterns of a software product line architecture for RT systems. It describes the application of these models in an example of an RT control system.

The two remaining chapters in this part apply domain engineering to develop techniques for building core assets and systems in the software product line:

- "When aspect-orientation meets software product line engineering" by Iris Reinhartz-Berger discusses an approach that melds aspect-oriented and SPLE methods through domain engineering. The approach uses the Application-based DOmain Modeling (ADOM) method to support families of aspects and weave them to families of software products. Reuse is enhanced through the three levels addressed by ADOM: language, domain, and application.
- "Utilizing application frameworks: a domain engineering approach" by Arnon Sturm and Oded Kramer also uses ADOM. In this chapter, the modeling approach supports specification and use of frameworks as a construct to support reuse across the software product line. ADOM also contributes to domain-specific languages to reduce the development effort and increase their reusability and code quality.

Part II: Domain-Specific Language Engineering

DSLs are specification or programming languages tailored to specific domains [6, 7]. These languages are developed in a domain engineering process and are later used to develop and maintain solutions (i.e., software systems) in the specific domains. The focus on a specific domain is achieved by abstracting from general programming language implementation details such as variable locations and control structures. The resulting DSL features correspond to domain elements and are often referred to as declarative (as opposed to imperative) because they focus on expressing desirable states of the problem domain rather than computations in the solution domain. The benefits of adopting DSLs include increased productivity,

improved quality, reuse of experts' knowledge, and, perhaps most importantly, better maintainability [6, 7].

Conceptually, a DSL is a formal language that is expected to be understood by domain experts and that can be interpreted by domain-specific software tools to produce lower level specifications. Practically, a DSL is a preferably small language that focuses on a particular aspect of software systems (e.g., [3]) and that is used by domain-specific software tools to generate code in a general purpose programming language or other lower level formal specification languages, e.g., [7]. Examples of domain-specific programming approaches are elaborated in [4, 5].

Domain-specific language engineering (DSLE) is concerned with methods and tools for specifying and utilizing such languages. This includes the identification of relevant language concepts and their relationships, the determination of the most appropriate level of abstraction for the envisaged users of the language, and the specification of all required transformations.

Although the practice of DSLE has evolved considerably in the last two decades, a number of challenges remain. There is a need to study the ways in which people use DSLs and to what extent. In line with the increasing popularity of DSLs, there is a need to evaluate the ways in which DSLs are composed, to examine maintainability and interoperability, and to devise mechanisms that enable end users to extend DSLs. The trade-offs between using general purpose languages and DSLs also merit further discussion. From an engineering perspective, there is a need to explore how DSL elements can be reused, which types of transformation are required, what best practices can be distilled from detailed case studies, how to define and evolve the semantics of a DSL, and how to evaluate the design and implementation of a DSL.

In this part of the book, we have gathered five chapters that address some of the challenges mentioned above.

The first chapter discusses the notion of domain-specific languages:

- "Domain-specific modeling languages—requirement analysis and design guidelines" by Ulrich Frank attempts to provide instructions for developing a domain-specific modeling language. In particular, this chapter introduces a set of guidelines, which consist of requirements for the meta-modeling language, as well as a detailed process description of the stages to devise a new DSL.

The following two chapters discuss the design process of domain-specific languages. Designing a DSL involves the creation of a new formal language, and therefore it is important to investigate the emergence of new languages as well as their engineering:

- "DSLs and standardization: Friends or foes?" by Øystein Haugen argues that creating a good language requires knowledge not only of the domain but also of the language design process. This chapter discusses the tension between DSLs and standardization efforts, demonstrates how DSLs can benefit from standardization, and provides a comprehensive example of language evolution and standardization.

- "Domain engineering for software tools" by Tony Clark and Balbir Barn proposes a language-driven approach that elaborates the notion of domain-specific tool chains and related tool interoperability challenges. The approach presented in the chapter views domains as languages and emphasizes the need for modularity, in particular the need for modular composition of domains and tool chains. The suggested approach for tool design involves a model of the semantic domain, a model of the abstract syntax, a model of the concrete syntax, as well as a model of the relationships between the semantic domain and the abstract syntax.

A key motivation for developing one or more DSLs for the same domain is the desire to capture all the meta-data that is needed to automate the production of detailed artifacts (such as code) from the abstract concepts supported by the DSLs. A common way of producing derived artifacts is through model transformation. Although a large number of model transformation languages have been developed, there are only few heuristics for engineering model transformation languages. The fourth chapter in this part tackles this issue:

- "Modeling a model transformation language" by Eugene Syriani, Jeff Gray, and Hans Vangheluwe introduces a technique for developing model transformation languages that refers to each language as a DSL and that includes a model of all domain concepts at the appropriate level of abstraction.

As developing a DSL is a complex task that involves stakeholders from different disciplines, a cooperative environment that supports cross-disciplinary collaboration is required. The fifth and last chapter in this part addresses this challenge:

- "A Reconciliation framework to support cooperative work with DSM" by Amanuel Alemayehu Koshima, Vincent Englebert, and Philippe Thiran proposes a communication framework that links the changes made by the language engineers and their effects on DSL users. This framework is concerned with the effects of language evolution and the propagation of changes in tool chains and across the stakeholders and the language user community.

Part III: Conceptual Modeling

Before any system can be collaboratively developed, used, and maintained, it is necessary to study and understand the domain of discourse. This is commonly done by developing a conceptual model. The main purposes of conceptual models are: (1) supporting communications between different types of stakeholders and especially between developers and users; (2) helping analysts understand the domain of interest, its terminology, and rules; (3) providing input for the next development phases, namely top level and detailed design; and (4) documenting the requirements that originate from the real world for maintenance purposes and future reference.

The process of building conceptual models, conceptual modeling, involves developing and maintaining representations of selected phenomena in the application domain [12]. These representations, the conceptual models, are usually developed during the requirements analysis phase of software or information systems development. Such models aim to capture the essential features of systems in terms of the different categories of entity, their properties, relationships, and their meaning. They are used for representing both structural and dynamic phenomena, usually in a graphic way. Once a conceptual model is constructed and agreed on, it forms a foundational basis for subsequent engineering activities.

Although research in conceptual modeling has existed for many years, its boundaries are quite vague. In particular, conceptual modeling has significant overlap with the field of knowledge engineering [9]. Many of the features of modern notations for conceptual modeling can be traced to examples in both early system design notations and knowledge representation notations, such as conceptual structures [10]. Conceptual modeling also has a strong relationship to ontologies [11], and the question whether conceptual models and ontologies are alternatives of each other is open. Clearly, Conceptual modeling is also related to model-driven architecture (MDA), which has become a significant research topic in recent years. MDA promotes the idea that systems should be modeled at a high level of abstraction and then systems are partially or completely generated from the models.

In light of technology improvements, many challenges that concern domain engineering in the context of conceptual modeling arise. In particular, how can the real world be modeled to better support the development, implementation, use, and maintenance of systems? [1] Conceptual modeling applies to many different application domains which raises the question of how to support the representational needs of each domain. Should there be a single universal language for conceptual modeling or several different languages? Do methods that apply to one type of application (finance for example) also apply in another (for example an embedded system)? The representational issue is often addressed using meta-techniques that allow the conceptual modeler to use a standard notation to design a bespoke notation that is used to express the conceptual model. While the best meta-technology is an open question, UML provides profiles that allow the UML standard to be tailored in a number of ways to support new concepts in terms of abstract modeling elements and the ways they are represented on diagrams. Also, following the evolvement of the MDA approach, it is interesting to examine how conceptual models fit into various MDA technologies and processes. Finally, the management of conceptual models is also a challenge, especially those that involve meta-technologies [2]. In addition to the usual problems related to distributed multi-person development, a conceptual model written using a notation that has been specifically defined for this purpose requires care when the meta-model is evolved, otherwise the conceptual model becomes meaningless.

The chapters in this part of the book address some of the challenges mentioned above. The first chapter in this part suggests using domain engineering for formalizing the knowledge of domain experts.

- "Model oriented domain analysis and engineering" by Jorn Bettin presents a model-oriented domain analysis and engineering methodology. This methodology, whose roots are in both SPLE and conceptual modeling, can be used to uncover and formalize the knowledge that is inherent in any software-intensive business or any scientific discipline.

The second chapter analyzes the relationships between different abstraction levels of modeling in order to support the definition of domain-specific modeling languages.

- "Multi-level meta-modeling to underpin the abstract and concrete syntax for domain-specific modelling languages" by Brian Henderson-Sellers and Cesar Gonzalez-Perez discusses the relationships between models, meta-models, modeling languages, and ontologies. They further provide a theoretical foundation for the construction of domain-specific modeling languages, exemplifying this foundation on two languages: ISO/IEC 24744 that can be used to define software-intensive development methods and FAML that can be used for the specification of agent-oriented software systems.

The third chapter discusses an ontology-based framework for evaluating and designing conceptual modeling languages.

- "Ontology-based evaluation and design of visual conceptual modeling languages" by Giancarlo Guizzardi addresses another methodological issue and focuses on the evaluation of the suitability of a language to model a set of real-world phenomena in a given domain. In the proposed approach, the suitability can be systematically evaluated by comparing the level of homomorphism between a concrete representation of the worldview underlying the language and an explicit and formal representation of a conceptualization of that domain (represented as a reference ontology).

The fourth chapter addresses the challenge of managing conceptual models in distributed multi-person development.

- "Automating the interoperability of conceptual models in specific development domains" by Oscar Pastor, Giovanni Giachetti, Beatriz Marín, and Francisco Valverde discusses the model management, interoperability, and reuse. In particular, it discusses the problems related to conceptual interoperability across applications in a domain. This chapter introduces a framework for describing levels of conceptual interoperability and the challenges that must be overcome to achieve the various levels and then outlines a process for achieving and automating interoperability through the integration of modeling languages.

As mentioned before, MDA promotes the idea that systems should be modeled at a high level of abstraction and then systems are partially or completely generated from the models. The benefits that are claimed for this approach are that it shields the developer from constantly changing technology platforms, increases quality, and

makes change easier to manage. The last chapter exemplifies this notion for the domain of geographic databases.

- "Domain and model-driven geographic database design" by Jugurta Lisboa-Filho, Filipe Ribeiro Nalon, Douglas Alves Peixoto, Gustavo Breder Sampaio, and Karla Albuquerque de Vasconcelos Borges describes the use of the MDA approach in the design of databases in the geographical domain. In particular, a UML Profile, called GeoProfile, is proposed and is aligned with international standards of the ISO 191xx series. This chapter also shows that with the automatic transformation of models it is possible to achieve the generation of scripts for spatial databases from a conceptual data schema in a high level of abstraction.

Concluding and Further Remarks

As elaborated above, domain engineering is closely related to several fields, primarily SPLE, DSLE, and conceptual modeling. This book provides a collection of research studies that are related to these three fields. The fields promote domain engineering differently; however, they do have significant overlap. In particular, some of the studies could pertain to more than one field. Therefore, we confirmed our classification with the authors in these cases. Moreover, as the studies are very diverse, they address a variety of important topics related to domain engineering, stressing the importance of this field, providing solutions, and further clarifying existing related challenges.

We believe that the chapters in this book are of interest to researchers, practitioners, and students of domain engineering in general and of the fields of SPLE, DSLE, and conceptual modeling in particular. Furthermore, given the exponential growth of data on the Web and the growth of the "Internet of Things," Domain Engineering research may be relevant to other disciplines as well. For example, the emergence of deep chains of Web services highlights that the service concept is relative. A set of Web services developed and operated by one organization can be utilized as part of a platform by another organization. This calls for appropriate conceptual models as well as for DSLs that together facilitate the design of service-oriented architectures. Furthermore, as services may be used in different contexts and hence require different configurations, inspiration to their design as families of services can be taken from the field of SPLE.

Another new opportunity for research is related to the Big Data domain. Big Data is characterized by the "three Vs": **v**olume, **v**ariety, and **v**elocity (rate of change). The ability to process such data depends on understanding and manipulating the information. Conceptual models can be of great help since they capture the semantics of the information which is important for making matches in the presence of incomplete and noisy input. Furthermore, meta-processing, such as dependency analysis, model transformations, model merge, and slicing, can be used to address

multiple data sources. DSLs can be used to develop languages that express domain-specific data patterns and SPLE can be utilized to help address the need to modify the patterns on a regular basis.

Lastly, we would like to thank the authors for their contribution to this book and to wish the readers enjoyable and fruitful reading.

References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns, 3rd edn. Addison-Wesley Professional, Boston (2001)
2. Di Ruscio, D., Iovino, L., Pierantonio, A.: What is needed for managing co-evolution in MDE? In: Proceedings of the 2nd International Workshop on Model Comparison in Practice (IWMCP '11), pp. 30–38 (2011)
3. Fowler, M.: Domain-Specific Languages. Addison-Wesley Professional, Boston (2010)
4. Freemanand, S., Pryce, N.: Evolving an embedded domain-specific language in Java. In: Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, pp. 855–865 (2006)
5. Kabanov, J., Raudjärv, R.: Embedded type safe domain specific languages for Java. In: Proceedings of the 6th International Symposium on Principles and Practice of Programming in Java, pp. 189–197 (2008)
6. Kelly, S., Tolvanen, J-P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley, Hoboken (2008)
7. Mernik, M., Heering, J., Sloane, A. M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005)
8. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Heidelberg (2005)
9. Schreiber, G. T., Akkermans, H.: Knowledge Engineering and Management: The Common KADS Methodology. MIT Press, Cambridge (2000)
10. Sowa, J.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley Longman Publishing, Boston (1984)
11. Sugumarana, V., Storeyb, V.C.: Ontologies for conceptual modeling: their creation, use, and management. Data Knowl. Eng. **42**(3), 251–271 (2002)
12. Wand, Y., Weber, R.: Research commentary: information systems and conceptual modeling—a research agenda. Inf. Syst. Res. **13**(4), 363–376 (2002)

Haifa, Israel	Iris Reinhartz-Berger
Beer-Sheva, Israel	Arnon Sturm
Hendon, London	Tony Clark
Pittsburgh, PA	Sholom Cohen
Mordialloc, Australia	Jorn Bettin