Generic Domain Models in Software Engineering

Neil Maiden

Department of Business Computing
City University
London EC1V OHB, UK.
Tel: +44-71-253-4399 x3422
E-mail: cc559@city.ac.uk

Abstract

This paper outlines three research directions related to domain-specific software development: (i) reuse of generic models for domain-specific software development; (ii) empirical evidence to determine these generic models, namely elicitation of mental knowledge schema possessed by expert software developers, and; (iii) exploitation of generic domain models to assist modelling of specific applications. It focuses on knowledge acquisition for domain-specific software development, with emphasis on tool support for the most important phases of software development.

Introduction

Domain-specific software design has aroused considerable interest over the last decade. Most of the research effort has focused on supporting the latter stages of software development, typified by program transformational techniques and systems (e.g. Feather 1987). However, it is now agreed that most costly problems occur during the early stages of system development, when systems' requirements are ill-defined and poorly understood. Therefore, domain-specific software development (as opposed to design) must provide effective guidance during requirements engineering and high-level software design as well as during system implementation. Unfortunately requirements engineering differs from system design in its focus on the identification and embedding of systems in their environment rather than prescribing systems' functionality. This broad view can often preclude the complete capture of all domain knowledge, implying only partial automation of domain-specific software development. This paper proposes, as a first research direction, that it is more beneficial to model generic domain models rather than specific application domains, and to exploit these generic models for guiding rather than automating requirements engineering and high-level software design.

Domain modelling is needed for domain-specific software development. However, case histories of successful domain modelling and effective methods for modelling complex applications have been lacking in the literature. Innovative work by Neighbors (1980) indicated that domain analysis was both difficult and time-consuming, even for experienced analysts. Recent findings have supported this view, for instance Prieto-Diaz (1991) reports difficulties in maintaining a domain model represented as a faceted classification scheme supporting reuse within a single application. Furthermore, models of specific applications can only support development within that application, while many organisations develop software for many applications, thus reducing the potential payoff from such application modelling. Generic domain models provide an alternative domain knowledge source which can provide greater payoff to software developers because of their applicability to many applications. Reuse of such models has been proposed elsewhere (e.g. Reubenstein & Waters 1991), although little is known about the nature, contents and applicability of generic domain models for effective requirements engineering. As a result, a second research direction proposed in this paper is to determine the knowledge structures of generic domain models which support effective requirements engineering.

Generic domain models have been proposed to support requirements engineering activities, however they may also provide effective guidance for longer-term domain modelling activities. The problem is akin to knowledge acquisition during knowledge-based system (KBS) development. Recent advances in knowledge acquisition techniques promote reuse of generic, partial domain models as templates supporting top-down knowledge acquisition and modelling (e.g. Wielinga et al. 1991,

Chandrasekaran 1986). A third research direction proposed in this paper is to exploit generic domain models to assist application modelling within a comprehensive domain modelling framework.

The remainder of the paper investigates these three research directions, namely reusing generic models for domain-specific software development, determining the nature of these generic models from empirical studies, and exploiting generic domain models to assist subsequent modelling of specific applications.

Evidence for Generic Domain Models

Evidence for the likelihood of generic domain models to assist requirements engineering comes for current software engineering research, recent advances in knowledge acquisition and empirical evidence of software engineering expertise. Each is examined in turn.

Generic Domain Models in Software Engineering

Generic domain modelling in software engineering research has arisen as an issue in both automated software development and domain analysis. Reusable generic domain models have been proposed in several research projects (e.g. Reubenstein & Waters 1991). The well-known Requirements Apprentice (Reubenstein & Waters 1991) exploits cliches representing general software engineering concepts, including domains, however few clues are provided about the nature and boundaries of these cliches. Furthermore object-oriented paradigms have been limited to design and implementation phases of software development while object-oriented analysis has focused on object definition rather than object structure within domains. This would suggest that abstraction in software engineering is poorly understood, and requires further investigation.

Iscoe (1991) reviewed evolving research in domain modelling, with emphasise on meta-models instantiated into application domains. His research issues include domain classification and analysis, implying the need for a theory of software engineering abstraction, however he gives few clues about the nature of this abstraction. Several domain meta-models have been reported in the literature (e.g. Lubars 1988, Dardenne et al. 1991, Chung et al. 1990), however this work has not been sufficiently developed as application examples and in practice to determine generic domains. Prieto-Diaz (1990) also reviewed domain analysis and emphasised the importance of abstraction in domain modelling. However, he could offer no guidance for this abstraction process beyond current structured analytic techniques such as SSA (De Marco 1978) and domain analyst expertise. Furthermore abstraction was limited to identification of important domain features rather than generification from application instances.

Generic Knowledge Structures in Knowledge Acquisition

Knowledge acquisition techniques and methods (reviewed in Neale 1988) have implications for domain analysis for at least three reasons. First the task of requirements analysis is similar to knowledge acquisition. Aspects of KBS development such as information analysis, application selection, project management, user requirement capture, modular design and reusability are similar to those encountered in software development. Indeed the KADS project (Wielinga et al. 1991) proposes a sequential development method based on modelling activity and an operational model that exhibits some desired behaviour in terms of real-world phenomena, similar to many existing software development methodologies including SSADM (Cutts 1987) and JSD (Jackson 1983). A second reason is that knowledge acquisition techniques like KADS are relevant to requirements engineering because they focus support on the earlier, analytic stages of KBS development while domain-specific software design paradigms support later stages such as program specification, transformation and maintenance (e.g. Feather 1987). Finally knowledge acquisition approaches introduce techniques not found in otherwise equivalent software development methodologies, so a review of knowledge acquisition techniques in respect to requirements engineering is warranted. The following knowledge acquisition projects were identified as having implications for generic domain models.

Generic Tasks: Chandrasekaran and his colleagues at Ohio State University propose generic tasks to provide an outline or framework for expert system design. This framework claims that complex knowledge-based reasoning tasks can often be decomposed into generic tasks, each with associated types of knowledge and family of control regimes (Chandrasekaran 1986). Six generic expert system tasks are identified in terms of knowledge types and control regimes: classification, state abstraction, knowledge-directed retrieval, object synthesis by plan selection and refinement, hypothesis matching, and assembly of compound hypotheses for abduction. These tasks encompass both declarative and procedural knowledge in reoccurring patterns. They emphasise the importance of domain knowledge and the reuse of large knowledge structures akin to complex objects.

The KADS Project: KADS is an ESPRIT project (ESPRIT-1 P1098), providing the knowledge engineer with reusable partial knowledge models as templates to support top-down knowledge acquisition and modelling, based on recognition that parts of the model are not specific to certain applications. The success of this approach has been documented in many domains, including diagnosis of

movement disorders, paint selection, commercial wine making and statistical consultancy (Wielinga et al. 1991), suggesting the potential effectiveness of the retrieval and exploitation of generic knowledge structures in complex, ill-structured modelling activity. Generic models are categorised by system structure, solution type and the discrepancy between observed and expected behaviour, based on a modified and extended version of Clancey's (1985) description of problem types.

KADS's domain meta-model is based on a tentative topology of primitive problem solving actions, or knowledge sources, consisting of concepts, their attributes, the values of these attributes, the structure of concepts, sets and set instances. It is derived from the type of operation that is carried out by the knowledge source, demonstrating the importance of contextuality linked to functionality of knowledge needs. KADSs' generic models demonstrate the importance of a topology of primitive problem solving actions based on a taxonomy of problem solving types. This approach has lead to considerable modelling success in a number of complex applications. Unfortunately the meta-model is weak due to the varied nature of domains tackled by the KADS approach.

Generic Mechanisms: Klinker et al.'s generic mechanisms (1991) result from comprehensive research to develop constructs which are both usable and reusable during knowledge acquisition and modelling. These mechanisms represent generic tasks reoccurring in many domains, for example *sizing* and *scheduling* tasks occur in both the computer and aerospace industries. Klinker's current knowledge acquisition tool is populated with at least 14 such mechanisms which are also aggregated into larger applications in which they often occur. A theory of mechanisms is currently being developed from experiences with the knowledge acquisition tool in new applications, leading to a more refined and complete mechanism library. The approach of Klinker and his colleagues differs from those of Chandrasekaran and KADS in terms of the research methods used, which employ empirical evidence to determine generic task mechanisms and their aggregation. This most comprehensive generic domain analysis demonstrates the importance of multi-level abstraction and granularity for generic domain models, with a need to aggregate domain models in several dimensions such as common application groupings.

Summary: Recent knowledge acquisition approaches demonstrate the feasibility of guidance based on generic domain and task models during complex modelling activities like requirements engineering. However, a model of generic tasks and domains, implying an underlying theory of abstraction, is not readily available for software engineering researchers. Such a theory must identify several determinants of generic domain models, such as their appropriate level of abstraction, granularity and effective knowledge structures, to decide how big or small these generic domain models should be. Intermediate findings point to potential research directions, namely the contextual nature of these models and the need to validate them through empirical evidence in software engineering, for instance software engineering domains are very different to those of commercial winemaking or diagnosis of movement disorders (Wielinga et al. 1991).

Software Engineers' Expertise

Software engineers' expertise offers one form of empirical evidence for validating generic domain models. Expert software developers possess preformed abstract mental schema of domains which allow them to classify, structure and scope each problem (Guindon 1990) and develop multiple mental domain models (Pennington 1987). Experts' mental schemata can be assumed to be effective generic representations due to successive refinement during requirements engineering experiences in many applications, which may suggest why experienced software engineers are much sought-after individuals. Intelligent software development mimicking experts' knowledge structures may be one direction for research to proceed. Again however, current empirical evidence of software engineers' mental schema is limited due to a lack of relevant and comprehensive studies, so more effective, empirical research is needed to determine generic domain models in software engineering.

An Initial Model of Software Engineering Abstraction

Studies of generic models in software engineering, knowledge acquisition and expert analytic behaviour suggest the validity of a generic domain modelling approach to domain-specific software development. However, the nature of these generic models is less clear, so a three-phase research strategy was adopted at City University to determine their contents and structure:

- investigation of analogical specification reuse as one means of determining generic domains underlying this reuse, to be followed by validation and extension of these generic domain models using:
- empirical studies of software engineers' mental knowledge structures via knowledge acquisition techniques, and
- domain analyses of large, real-world applications to verify generic domains in terms of recognisable instantiations and instantiation aggregations.

The first phase is partially complete while the second and third phases are the focus of an ESPRIT Basic Research Action. The first phase has led to a tentative model of software engineering domains which provide the basis for

a retrieval mechanisms supporting analogical specification reuse, and described in Maiden & Sutcliffe (1991).

Generic Domain Models Supporting Specification Reuse

Maiden (1991) identified an initial model of generic domain models through studies of analogical specification reuse, such that two specified domains are analogous if they are both instances of the same generic domain class, as demonstrated in Figure 1. As such the scope, granularity and level of abstraction of these generic knowledge structures is constrained to most effectively support reuse of functional specifications.

Maiden's model (1991) proposes that generic domain classes are differentiated by key state transitions, hence a generic resource hiring domain, of which library loans is an example, can be distinguished from a generic resource containment domain (e.g. stock control) by the key transition of return (see Figure 2). Similarly two classes of object allocation domain can be differentiated by the transitions *send to* and *remove* from waiting lists, for example the reservation system of a local cinema may not include waiting lists once all seats for a performance are sold. Additional determinants of distinct domain classes were identified in terms of these critical transitions between domain states. The following meta-schema for describing critical generic domains and their instantiations was developed, with each knowledge type describing one or more critical dimensions:

- actions leading to state transitions with respect to a knowledge structure. These actions represent system intervention in the domain to maintain or change the domain from a possible to a required state. Actions and state transitions are central to the model, for example the allocation action in the theatre reservation example causes the object (*theatregoer booking*) to change state from an in-requirement state to an occupying-resource state (from *required-booking* to *reserved-booking*);
- object structural knowledge describing both problem and required domain states in the form of conceptual relations between objects. For example, *theatre contains many seats*, each *containing one or no theatregoer booking*. Furthermore, required knowledge structures such as *maximise seat occupation*, can be imposed on these domain states;
- pre/post-conditions on state transitions identified from values describing the current state of objects, for example a state transition moving the theatre reservation to the seat only occurs if the reservation and the seat have similar constraints such as *non-smoking*, *price <£20*, *seat is unreserved*, etc.;
- object types describe object roles in the context of state transitions, for example customer bookings is a type of *requirement* while theatre seats are *resources* available to satisfy those *requirements*;
- functional transformations which may be causally-related to state transitions in the domain model, for example the functional transformation *allocate from waiting list* results in a state transition moving the theatre booking from the waiting list to theatre seats while functional transformations in library systems are typically *lend* and *return*;
- state transitions can also be distinguished by their triggering events. Domain events which cause state transitions are either initiated by the information system or by events external to it, for example the theatre reservation domain may in part be distinguished by the scope of triggering domain events because allocating customer bookings to the seats available is initiated by the information system while removing customers from allocated seats results from external events.

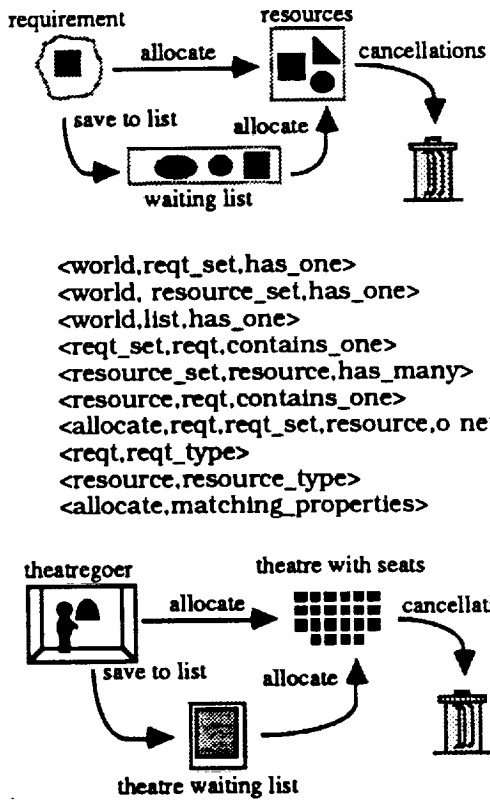

Figure 1: simple theatre reservati on domain and its generic domain class, incl uding partial definition of that class

To sum, this model of generic software engineering domains was developed from example-based studies of such domains in the context of reuse. Its development was driven by domain-based studies of important knowledge structures in software engineering, a constraint which

distinguishes it from existing meta-models of software engineering domains such as TELOS, (Chung et al. 1990). The extent and nature of this example-driven analysis is described briefly in the following section.

Example Generic Domain Models

Current research has identified 35 generic domain models through the relatively weak proof of trial by example, see Figure 2 and Maiden (1992). These models were hierarchically-structured to identify classification and specialisation of basic domain types, for instance library and stock control domains are both specialisations of a more generic object containment domain. Furthermore generic domains were aggregated to identify *standard* applications incorporating many domain classes in unique patterns, for example a comprehensive library system can involve lending, stock updating, allocating and reserving activities which are all instantiations of different domain classes. The validity of this current approach is suggested by a prototype specification reuse tool incorporating 10 such generic domain models in a specialisation hierarchy to support successful retrieval and explanation (Maiden 1992). However, further work is needed to extend and validate the current model.

Domain Modelling From Generic Domain Models

This paper reports studies which reveal domain analysis to be a problematic task akin to knowledge acquisition. Parallel experiences in knowledge acquisition suggest that generic domain models may assist in this task. A domain modelling framework incorporating reuse, similar to the KADS method, is needed to make effective use of generic domain models. In particular such models provide pieces of the generic skeleton to be instantiated and fleshed out with additional knowledge types until the domain model is complete.

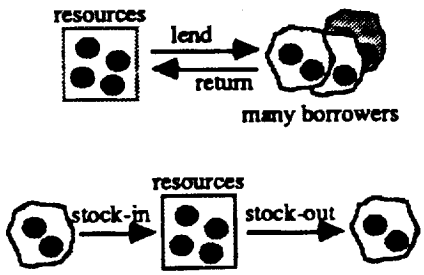

Figure 2: examples of generic doma in models:
(i) renewable resource, e.g. libra ry,
(ii) non-renewable resource, e.g. stock control.

Summary

This paper proposes that greater benefits can be achieved from modelling generic domains rather than specific applications, so overcoming domain modelling bottlenecks by mimicking expert software engineering practice. Intelligent tool support founded on generic domain knowledge can assist during requirements engineering in the following tasks:

- identification and validation of application models to assist effective requirements capture, providing intelligent feedback on system requirements and models;
- procedural guidance for requirements engineering tasks, using generic domain hierarchies to focus on critical domain features and incrementally specialise them;
- support for reuse through categorisation of problems based on generic domain classes (Maiden & Sutcliffe 1991).

We would also intuitively expect generic domain models to provide the basic building blocks for complex application modelling then domain-specific software design. Acquiring these knowledge structures therefore takes on considerable importance for intelligent support during requirements engineering and software design. To this end we suggest that much research effort should be focused on practical and empirical research to determining the most effective knowledge structures for supporting domain-specific software development.

References

Chandrasekaran B., 1986, Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, *IEEE Expert* 1(3), 23-30.

Chung L., Katalagarianos P., Marakakis M., Mertikas M., Mylopoulos J. & Vassilou Y., 1990, From Information System Requirements to Designs: A Mapping Framework, Technical Report CSRI-245, University of Toronto, September 1990.

Clancey W.J., 1985, Heuristic Classification, *Artificial Intelligence* 27, 289-350.

Cutts G., 1987, *SSADM - Structured Systems Analysis and Design Methodology*, Paradigm Publishing.

Dardenne A., Fickas S. & Lamsweerde A., 1991, Goal-directed Concept Acquisition in Requirements Elicitation, Proceedings of 6th Intl Workshop on Software Specification and Design, Como (It) 25-26th October 1991, IEEE Computer Society Press, 14-21.

De Marco T., 1978, *Structured Systems Analysis and Specification*, Prentice-Hall International.

Feather M.S., 1987, A Survey and Classification of some Program Transformation Approaches and Techniques, *Program Specification and Transformation*, ed. L.G.L.T. Meertens, Elsevier Science Publishers.

Guindon R., 1990, Designing the Design Process: Exploiting Opportunistic Thoughts, *Human-Computer*

Interaction 5, 305-344.

Iscoe N., 1991, Domain Modelling: Evolving Research, Proceedings of 6th Knowledge-Based Software Engineering Conference', Syracuse NY, 22-25th September 1991, 300-304.

Klinker G., Bhola C., Dallemagne G., Marques D. & McDermott J., 1991, 'Usable and Reusable Programming Constructs', *Knowledge Acquisition* 3, 117-135.

Lubars M.D., 1988, A Domain Modelling Representation, MCC Technical Report STP-366-88, Software Technology Program, MCC, Austin Texas, November 1988.

Maiden N.A.M., 1992, Analogical Specification Reuse during Requirements Analysis, PhD Thesis, Department of Business Computing, City University.

Maiden N.A.M., 1991, Analogy as a Paradigm for Specification Reuse, *Software Engineering Journal* 6(1), 3-15.

Maiden N.A.M & Sutcliffe A.G., 1991, Analogical Matching for Specification Retrieval, Proceedings of 6th Knowledge-Based Software Engineering Conference, Syracuse NY, 22-25th September 1991, 101-112.

Neale I., 1988, First Generation Expert Systems: A Review of Knowledge Acquisition Methodologies, *The Knowledge Engineering Review* 2, 105-145

Neighbors J.M., 1980, Software Construction using Components, Ph.D. Dissertation, Department of Information and Computer Science, University of California, Irvine.

Pennington N., 1987, Comprehension Strategies in Programming, *2nd Workshop of Empirical Studies of Programmers*, ed. G. Olson, S. Sheppard and E. Soloway, Ablex, 100 - 113.

Prieto-Diaz R., 1991, 'Implementing Faceted Classification for Software Reuse', *Communications of the ACM* 34(5), 88-97.

Prieto-Diaz R., 1990, Domain Analysis: An Introduction, *ACM SIGSOFT Software Engineering Notes* 15(2), April 1990, 47-54.

Reubenstein H.B. & Waters R.C., 1991, 'The Requirements Apprentice: Automated Assistance for Requirements Acquisition', *IEEE Transactions on Software Engineering* 17(3), 226-240.

Wielinga B.J., Schreiber A.Th. & Breuker J.A., 1991, 'KADS: A Modelling Approach to Knowledge Engineering', Technical Report ESPRIT Project P5248 KADS-II, May 1991.