Domain Engineering: What is it?

Arne Sølvberg

Department of Computer and Information Science
NTNU – The Norwegian University of Science and Technology
7491 Trondheim, Norway
arne.solvberg@idi.ntnu.no

Abstract The term *domain engineering* has different meanings. In software engineering it is used for describing the software relevant features of the domain for which software is developed. In general the term denotes the act of creating the domain itself, including its artifacts. The paper argues that the use of the term in software engineering is not a happy choice and contributes to the terminological confusion often observed when disciplines meet in multi-disciplinary projects.

Keywords: information systems engineering, domain engineering

1 Introduction

When computers first came around, it made sense to distinguish between a domain and its computer applications. Around 40-50 years the so-called "Scandinavian School" of Information Systems made clear distinctions between the "total system", the "information system", and the "data system", the latter consisted of computer hardware and software.

These were pristine times, and it was simple to make such a distinction. An example of a "total system" could be a bank, with its vaults for the coins and bills, the "information system" would be the people and machines that exchange messages. A major task of the information system was to make sense of the numbers and texts that reflected the operation of the bank. The "data system" was the computers, the software and the data which could be stored and processed according to predetermined rules. It was a comparatively simple world.

As the years have gone by, and computers are everywhere, this simple distinction is no longer straightforward.

2 Domain layer and application layer, is this an appropriate distinction?

The call-for-papers explains the relationship between software and non-software as follows:

Domain engineering deals with two main layers: the domain layer, which deals with the representation of domain elements, and the application layer, which deals with software applications and information systems artifacts. In other words, programs, applications, or systems are included in the application layer, whereas their common and variable characteristics, as can be described, for example, by patterns, ontology, or emerging standards, are generalized and presented in the domain layer.

I find this distinction to be somewhat disturbing and potentially misleading. Man-made systems are always composed of parts. Most parts in our technical world increasingly consist of interrelated software and non-software. Take for example an automobile. The brakes consist of mechanical components and of process control software. So do other parts, like the fuel injection system, the power transmission system and many more. A car is an assembly of parts where many of them have very clear autonomous features. The various parts are made to interact through a combination of software and mechanical connections. How can one distinguish between a domain layer and an application layer for the car, when every component of the car is a system of both mechanical parts and software parts?

Approximately 50 % of the production cost of a modern automobile reflects the cost of electronics and software. So what is the domain and what is the (software) application? Does this distinction make sense?

The term domain engineering is used in contemporary software engineering. One example of a definition is "*Domain Engineering* is seen as a process for creating a family of programs so that programs in the family can be created efficiently" [3]. So domain engineering is seen as a technique for creating components with a wider applicability than the components would have when engineering makes one-of-a-kind solutions. The use of the term is seen more clearly in [4], where domain engineering is used in the meaning of reflecting software relevant features of a domain for the purpose of building software to be reusable for a wider market.

The use of the term *domain engineering* in this context is a typical example of picking a general term and applying it in a restricted setting. Such practices change the meaning of terms and lead to confusion and difficulties when communicating over discipline borders. A somewhat wider definition is given by [5], but the restriction to the software perspective is evident also here.

3 "Domain engineering" - old wine in new bottles?

Domain engineering has always been there, as long as people have built artifacts to satisfy human wishes and desires. The design of a road is domain engineering, as is the design of a business organization. The term "domain engineering" in the current context comes out of information technology and software engineering. The question is whether domain engineering in this restricted context means the same as domain engineering in the wider context. I propose that this is not the case and that this discrepancy leads to confusion.

If domain engineering means the engineering of a domain this is the same as the engineering of the total system of which the information system and the associated software are parts. This is very different from using the term domain engineering for describing the external properties of some part of the domain for the purpose of designing re-usable software solutions for this part of the domain.

An information system is usually seen as giving support to some other system, by keeping track of its state-of-affairs, by supporting the exchange of information between the other system and its environment, and by providing information needed for changing the behavior of the other system, either through direct intervention or through making information available for other change agents [1]. In general, “the other system” is known by many different names, e.g., the user system, the user domain, the Universe of Discourse (UoD), the real world, the business system.

The term *domain engineering* when used in this context implies that the software based information system and its domain should be seen as two different entities? But it is increasingly difficult to separate these two parts of “the total system” through all system layers, and collect the software parts in one bundle and the non-software parts in another. So, can domain engineering exist on its own, and be separated from information system engineering?

4 Traditional approaches to information systems engineering in organizations

Implicit to the traditional approaches to information systems engineering is that there is a primary system, the domain system, which is to be served by a secondary information system. Most of the practice of information systems engineering is done in the domain of administrative organizations, e.g., bank, insurance, public services. The information systems have been so central to the design of the administrative routines that it has been difficult to distinguish between the two.

The information services are to be determined by the needs of the primary system. So, the first step in designing an information system is to do a requirement analysis. The next step is to find out whether the requirements can be satisfied. This is done by developing concrete solution proposals, and evaluating those relative to the requirements. Traditional approaches recognize that solution proposals and requirements must be co-developed. The solution proposals will give rise to modifications in the requirements, and to modifications of the features of the primary system. So, domain engineering is implicit in the traditional approach to information systems engineering.

Traditional approaches to total systems realization are based on

- application platforms
- software- and process libraries
- application languages

These have been with us in different shapes and quality since we started to use computers.

The ERP's are among the most successful application platforms. Almost all sizable companies depend on an ERP. Process libraries are among the most important tools for consultancy companies who mostly make their profit by re-using solutions for every new customer. Software libraries have been with us from the start and provide for the reuse of software. Application specific languages have enjoyed less success.

Characteristic for all these approaches are that the tools that they apply reflect knowledge about the domain, that the tools limit the functional and structural properties of the "total system" through their own limitations to treat data, and therefore lead to a design process where the limited functional properties of the information processing software platforms, - libraries, and languages, provide a limited solution space for the design of "the total system".

5 Domain modeling in multidiscipline oriented approaches

Computer science is probably one of the few disciplines that cannot render useful results unless being related to another discipline. In many cases of traditional information systems engineering the computer part is large compared to the rest of the application system, and the modelling discipline of computer science dominates. For other situations there is more of a balance between the importance of domain disciplines and the IT discipline. Relevant domains are everywhere, in the material world, the biological world, and the worlds of organised and creative humans.

In practical system development cross-competence cooperative activities relate the IT core technologies to the application areas. Different modelling cultures meet, and sometimes they clash. We need to better clarify the relationship between the IT as a modelling discipline, and the modelling disciplines of the domains where IT is applied. We need a better framework for thinking about cross-competence systems design.

6 Conclusion: On the need and possibility of model integration

Computers are increasingly found everywhere, in almost every artifact, in the background of almost every organized human activity. This will have to result in a change in approach, from viewing the role of IT to mainly support "the other system", to become an integral part of "the other system". We should search for ways to avoid treating the domain discipline separate from the information system discipline, towards the integration of IT concepts, tools and theory into the modeling theories of the supported (domain) disciplines. If we manage this the two layers of domain engineering and application engineering may merge.

Fortunately the basic modeling concepts of the "domain sciences" and of information technology overlap. Information systems are concerned with data that represent facts in a Universe of Discourse. A fact is what is known -or assumed - to belong to reality. In science and technology one usually distinguishes the following kinds of facts: state, event, process, phenomenon, and concrete system e.g., a magnetic field [2]. We see that the basic modeling ontology is the same in the sciences as for IT. This gives some reason for optimism.

Because information technology provides component solutions to almost every other discipline we experience increasing fragmentation pressures on the discipline of IT itself. Every domain where IT is used seems to contain seeds for creating their own kind of discipline where IT is integrated with the domain specific knowledge. We often see labeling like, e.g., medical informatics, organizational informatics, and industrial informatics. And we sometimes see that common IT knowledge is reinvented in new application settings. Harmonization of domain modeling ontology with IT modeling ontology may be one approach to better sharing of IT knowledge among the various disciplines thus avoiding massive "wheel-re-invention".

References

1. Boman, M.; Bubenko, J.A. jr.; Johannesson, P.; Wangler, B.: Conceptual Modelling. Prentice Hall, 1997,269 p.
2. Bunge, M: The Philosophy of Science, Transaction Publishers, 1998, ISBNN 0-7658-0415-8
3. http://craigc.com/cs/de.html Accessed 23 April 2010
4. http://www.domain-specific.com/ Accessed 23 April 2010
5. http://burks.brighton.ac.uk/burks/foldoc/76/33.htm Accessed 23 April 2010