Contexts, Domains, and Software

Alfs Berztiss

[1] University of Pittsburgh, Pittsburgh PA 15260, USA
[2] SYSLAB, University of Stockholm, Sweden

Abstract. We survey some issues that relate to context dependence and context sensitivity in the development of software, particularly in relation to information systems by defining a range of context-related concepts. Domain models are identified as the appropriate framework for dealing with context-related issues.

The importance of context is being increasingly recognized in many areas of computing. The aim of this note is to arrive at a classification of contexts in a rather informal way, by defining a number of terms. We hope that this will further the discussion of what effect context has on the design of software systems, and on the relationship between contexts and domain modeling.

The interpretation of the word context shows considerable variety — indeed, the definition is itself context-sensitive. Linguists may see it as a psychological construct, a subset of a hearer's assumptions about the world that is used in interpreting an utterance [1]. To an organizational theorist context is a social environment in which actions are taken [2]. To a sociologist, context is provided by macrosocial forms, such as gender, national ethos, and economic maturity of a society [3]. An axiomatic approach is taken in [4].

As a first step toward a classification of contexts we take a dictionary definition of context: the parts of a written or spoken statement that precede and follow a specific word or passage, or a set of circumstances or facts that surround a particular event, situation, etc [5]. This two-way split suggests a partitioning of contexts into internal and external. The term *internal context* relates to cases in which the actions of a software system depend on internal factors. For example, computer systems contain clocks, and an operation may be triggered by such an internal clock. When the software actions depend on external factors, such as a temperature reading or a stock market quote, we shall use the term *external context*. In all cases the effects of the context can be expressed as rules, which we call context rules. A set of context rules relating to a particular setting of interest will be called a *contextual domain model* (CDM).

For uniformity we assume that all context rules of CDMs have the form "if c then q." An example: "If x is in the United States, then temperatures at x are measured in Fahrenheit." The c will be called a *context*; the q will be called an *effect*. Both c and q form populations that can be partitioned into classes that can be related to domains.

Not all software systems are context-dependent. To identify those that are, we partition computations into *procedural* and *transactional*. A procedural compu-

P. Bouquet et al. (Eds.): CONTEXT'99, LNAI 1688, pp. 443–446, 1999.

tation transforms inputs into outputs by means of some algorithm. An example is the computation of the cosine of an angle, and a characteristic of the cosine of angle x is that it is always and everywhere the same. To generalize, procedural computations are independent of context. Transactional computations are not: the result of a transactional computation depends on the state of the environment in which it is carried out. The distinction between procedural and transactional systems is not clear-cut: a predominantly transactional system often contains procedural components, e.g., an interest-computing procedure in a system that manages bank accounts.

We equate the *domain* in which a software system operates with a context for this system. To start off, let us establish a rationale for this interpretation. The environment or domain in which the word pen is used, a children's nursery, a cattleyard, a writers' workshop, or a bird watchers' excursion, lead to the interpretation of the word as a play area, a cattle enclosure, a writing implement, or a female swan. In terms of the if-then approach, we can state "If pen is located in a children's nursery, then it is likely to refer to an enclosed play area." The pen as play area becomes a working hypothesis, and the hypothesis remains in force unless additional evidence negates it.

The expression "If inventory level for item x is below c, then reorder y of x from z" is also a context-effect expression, but there is a taxonomic difference between its consequent and that of the children's nursery expression. In one case the consequent suggests a definition for a term, in the other it initiates an operation. Moreover, and this is a crucial difference, in the case of the pen there is context sensitivity in that a choice of interpretation exists, while there is no choice in the reordering situation. There is no obvious difference in the antecedents. Now consider the rule "If inventory level for pens is below c, then reorder y pens from z." If this expression is used in an office, then it is unlikely that pens will be interpreted as female swans. This enables us to distinguish between a *linguistic context* and a *nonlinguistic context*.

The defining characteristic of context sensitivity is choice. Of course, there is choice in all instances of the ordering of pens: the pens are either to be ordered or not ordered. The choice we have in mind has to be more complex than a simple yes/no decision. To emphasize this difference we shall talk of context sensitivity only when a choice is more complex than a yes/no decision. In all cases there is a context dependence: every transactional system that aims at influencing its environment has to be aware of the current state of the environment.

Whether a domain model provides an internal or external context depends on how it is used. If it is closely tied to a particular application, e.g., if it serves to allow greater sophistication in the use of a particular data base, or if it tells when pens are to be reordered, then it is internal. When the model requires, for example, information on the possibility of a strike at the factory supplying pens, then it becomes external because the c of our "if c then q" is then supplied in part by the external environment of the software system. The internal-external distinction does not much matter, except that an external domain is more likely to undergo change. The meaning of words changes slowly,

and changes in a linguistic domain are therefore quite slow. On the other hand, business environments can change very rapidly.

What is an appropriate domain model depends on the purpose to which it is put. We have examined domain models for reuse in software development [6], conceptual modeling [7], and decision support systems [8]. In all these cases the domain models have to have a process orientation, and we call them *process domain models* (PDMs). Another type of domain model relates the concepts that pertain to a domain, and thus defines the static structure of the domain. This type of model will be called a *structural domain model* (SDM). What matters here is that the domain model is to represent contexts, and for a CDM the form "if c then q" appears the most suitable. Note that this form is used also by Brezillon and Pomerol in their work on context in decision making [9]; such forms have been used extensively in business models and knowledge-based systems.

We have considered two types of context change, *context evolution* and *context switch* [8]. An example of context evolution relates to a rental company that is taking note of changes in its environment and is now allowing rental reservations to be made not just by mail or telephone, but also electronically. The interactions between personnel, software system, and customers are different for the two existing reservation modes, and the addition of the third mode requires that a third interaction pattern be added to the domain model. Context switch relates to the migration of a software capability from one domain or subdomain to another. For example, although car rental and video rental are similar applications, there are differences.

We noted earlier that in a context-sensitive situation the rule "if c then q" expresses a choice. Actually we should limit the use of the simple form to cases in which there is just context dependence, i.e., where merely a yes/no decision is to be made. In the more complex cases we should put the rule in the form "if c_1 then q_1 else if c_2 then q_2 else if c_3 then q_3 ...". We call the c of the simple case a *trigger*; the c_i of the forms expressing choice are called *selectors*. Let us look at an example, the query "Was it cold on September 5?" The reference to a calendar and a determination of where the questioner is located allow the the system to sharpen the query to "What was the minimum temperature in Fahrenheit at Pittsburgh on September 5, 1999?" Here we have several rules with selectors: if reference made to cold use minimum temperature; if the calendar reference is incomplete, complete it to the closest relevant calendar reference before today; if the question originates in place x, relate the answer to this place; if x is in the United States, use Fahrenheit.

The effect of a context rule is a control action, the furnishing of information to a user, or a request for further inputs. Systems based on rules of these types are, respectively, control systems, information systems, and dialog systems. Of course, a system can be a hybrid. Control systems are in some respects the easiest to develop, in some respects the most difficult. They are easy because a context rule is defined by a domain expert, and the software developer merely implements the rule. But control systems are not just aggregates of independent rule-based actions. They implement processes, so that the appropriate model

is a PDM rather than a CDM, and the construction of a PDM from individual context rules is a complex, intellectually demanding cooperative effort of domain experts and software engineers. We are not considering dialog systems here.

Information systems operate in a much broader context than control systems. The model of the environment of a control system, i.e., the model of the host system into which the control system is embedded, does not change; only values of the parameters of the model change, and the purpose of the control system is to keep these values within acceptable ranges. As regards information systems, the CDM may itself be changing — the context evolves. At its mildest, context evolution or context switch manifests itself merely in a different effect q when the context c is refined. For example, although normally temperatures in the United States are measured in Fahrenheit, in a hospital they may be expressed in Centigrade. This is a context effect.

We have introduced a large number of terms. To begin with, we partitioned contexts into linguistic and nonlinguistic, and internal and external. We saw that an action can be independent of context, can depend on context, or can be context-sensitive. We identified context with a particular type of domain model, the CDM. Two other domain models, PDM and SDM, were introduced as well. We noted that a domain model can change slowly or rapidly, and that a context change can be context evolution or context switch. The CDM was seen as being composed of context rules of the type "if c then q," where the c could be either a selector or a trigger. Turning to computing, we identified procedural and and transactional software, and the latter was partitioned into control, information, and dialog systems.

References

1. Sperber, D., Wilson, D.: Relevance — Communication and Cognition. Harvard University Press (1986).
2. Weick, K.E.: Sensemaking in Organizations. Sage Publications (1995).
3. Layder, D.: New Strategies in Social Research. Polity Press (1993).
4. Buvac ,S., Buvac, V., Mason, I.A.: Metamathematics of contexts. Fundamenta Informaticae 23 (1995) 263-301.
5. Webster's Encyclopedic Unabridged Dictionary of the English Language. Portland House (1989).
6. Berztiss, A.T.: Domains, patterns, reuse, and the software process. In: Domain Knowledge for Interactive System Design. Chapman & Hall (1996) 79-89.
7. In: Information Modelling and Knowledge Bases VIII. ISO Press (1997) 213-223.
8. Berztiss, A.T.: Domain models for flexible decision support systems. In: Context Sensitive Decision Support Systems. Chapman & Hall (1998) 216-226.
9. Brezillon, P., Pomerol, J.-Ch.: Using contextual information in decision making. In: Context Sensitive Decision Support Systems. Chapman & Hall (1998) 158-173.