Design the Flexibility, Maintain the Stability of Conceptual Schemas

Lex Wedemeijer[1]

ABP Netherlands, Department of Information Management, P.O.Box 4476,
NL-6401 CZ, Heerlen, The Netherlands
L.Wedemeijer@ABP.NL

Abstract. A well-designed Conceptual Schema should be flexible enough to allow future change. But current methods for the design and maintenance of Conceptual Schemas are not based on insight into the actual evolution of the Conceptual Schema over time. The relationships between the demand of flexibility, the quality aspect stability, and evolution of the Conceptual Schema are not well understood or investigated. Advance in current design practices and maintenance of information systems depends on the understanding of the relation between actual properties of a Conceptual Schema and its evolution.

1 Introduction

Over time, every enterprise and its ways of doing business will change, technologic capabilities will continue to improve, and user demands will change. Hence, to safeguard business investments, all information systems are demanded to be flexible. In particular, the Conceptual Schema is required to be flexible, i.e. must prove to be a sound basis for longterm business support and for a long systems lifespan.
Figure 1 depicts the ANSI/X3/Sparc 3-schema architecture. According to this, a well-designed CS satisfies many quality requirements [6]. A common feature of current design theories and techniques is that the quality of a CS design is argumented from theoretical principles only, and concerns the design phase only. All quality aspects of a CS can be established at design time. The one exception is stability. For that, its operational lifecycle must be considered. No practical demonstrations of flexibility of a design in the operational phase are to be found in the literature. This lack is probably due to the difficulties that any research on operational CSs in a live business environment will encounter.

2 Flexibility of a Conceptuel Schema Design

Intuitively, flexibility means adaptability, responsiveness to changes in the environment [1]. And "more" flexibility will mean a smaller impact of change. A working definition is:

M. Jarke, A. Oberweis (Eds.): CAiSE'99, LNCS 1626, pp. 467–471, 1999.

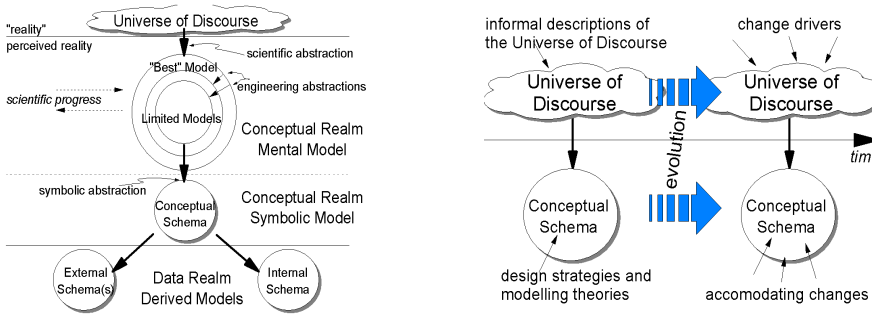

Fig. 1. CS models the perceived UoD (left) and should display joint evolution (right).

> *flexibility is the potential of the Conceptual Schema to accomodate changes in the structure of the Universe of Discourse, within an acceptable period of time*

This definition seems attractive as it is both intuitive and appropriate. It assumes a simple cause-and-effect relation between "structural change" in the UoD and changes in the CS. Also, it prevents inappropriate demands of flexibility on the CS by restricting the relevant environment from which changes stem to the UoD only. Three main design strategies exist that are widely accepted as "good design practices" and that do appear to work. Apart from these, many datamodelling theories exist that claim flexibility. The point is that there is no good understanding of their effectiveness in changing business environments.
Active flexibility or Adaptability is the strategy to improve the design by arranging the constructs of the CS in such a way that it is easy to modify (the *Engineering Abstractions* arrows in fig 1). Notice that no matter how easy the CS *description* can be modified, it is the operational environment that will resist the *implementation* of change. It assumes that certain types of change are easy to accomodate provided the CS is well-arranged, and that any future changes will be of exactly those types. Normalization, modularization, incremental design [3] and the use of component libraries (if applied to reduce the time for response) are based on this strategy. Some related fundamental problems are:

- what principles must be applied to select the right engineering abstractions
- how to detect and resolve conflicts in arranging the constructs
- how to match the perception of reality to a CS that arranges its constructs in special ways

Passive flexibility, or "stabilizing the CS", aims to decrease the need for future change in the CS. The experienced designer uses this strategy when he or she incorporates more requirements into the design than those originating from the current UoD (the *Perceived-Reality* and *Scientific Abstraction* arrows in fig 1). The assumption is that accomodating probable requirements will reduce the need for future change. (Re)use of "proven" designs, Business Data Modelling,

the use of component libraries (when taken to represent "good solutions") [2] and many rules-of-thumb [7] are based on passive flexibility. Related problems:

- how far ahead should future requirements be anticipated
- which requirements are relevant, and which are beyond consideration

The *Flexibility by abstraction* strategy puts less information into the CS design, thus making it more abstract. More features of the UoD are deemed "non-conceptual", increasing data independence (as contrasted to *relying* to data independence, which comes under active flexibility). Fundamental in the approach is that it is known which features must be abstracted from. Semantic datamodel theories, and Object-Oriented approaches, are based on this approach that comes under the *Scientific Progress* arrows in fig 1. Related problems to this strategy:

- deciding on the best level of abstraction
- how to transform the abstract CS into workable External and Internal Schemas

Problems with the Concept of Flexibility

There is no way for a designer to pick the best design strategy for a particular business case at hand. One must trust to experience and to state-of-the-art design practices. Further problems in assessing flexibility of a CS design are:

- flexibility is established on the fly. There is no way to verify that a CS has "enough" flexibility, or to discover beforehand that "more" flexibility is needed
- it leaves open what timespan is acceptable to accomodate a change
- it is not a persistent property. A CS that has adapted to many changes in the past may prove to be inflexible the next time.

Nor does the concept of flexibility help to understand the evolution of the CS. The main problem is in its dependence on future events. What is needed is a sound criterium that looks at the actual changes that occur over the operational lifespan of the CS.

3 Change

The CS is a well-defined, formal model that can be studied in isolation from its environment. In contrast, the UoD is informally known, a common understanding of what is "out there". But it can't be caught in formal definitions: any attempt at formalizing the UoD is equal to creating a CS. Over time, the common perception of "reality" will of course evolve as maintenance crews gain experience, employees shift to other jobs, new methods and tools are introduced etc. Although changes in (the perception of) the UoD can only be understood by referring to informal knowledge, there is hope yet. In many business cases,

structural change of the UoD can be deduced from informal communications as change requests, project definitions, budget justifications, etc.
A *taxonomy* lists the "elementary" changes that come with a datamodelling theory. But that is syntax only. The semantics of changes in the CS must be established by comparing the current CS description with the previous one, which will encounter serious complications:

- consecutive CS descriptions must be available. This is not usually the case!
- the datamodelling theory must not allow equivalent representations [5]
- the CS description must be faultless. Often, Reverse Engineering is called for
- constructs and constructions in the CS before and after the change must have a recognizable relation, or –even better– preserve their identity.

4 Stability of the Conceptual Schema

As the UoD is bound to change, as people will perceive the UoD in new ways, and as user requirements will change, so the CS must cope with change (Fig. 1). Flexibility is unsuitable to assess this quality aspect, as it depends on future events. Instead, the criterium of stability is proposed. A working definition:

> *a change in the CS is a stable change if it is absolutely necessary to accomodate change in the structure of the UoD*

A CS that changes for any other reason is *unstable*. Generally speaking, while flexibility is a demand that must be met by the designer, stability is proof that it has been delivered. The systems owner should decide on the necessity of the CS change and the spending of vital business resource, weighing it against other pressing business needs. Enterprises try to keep the impact of change as small as possible [4]. This restricts the freedom of choice for the maintenance people as the adapted CS is required to be a good model of the new UoD, and at the same time be "as close to the old CS as possible". This tendency towards stability is evident in the usual demand for compatibility, reducing the need for complex data conversions. Extensibility is a special form of compatibility. Other familiar ways to implement changes in a compatible way are relaxation of database constraints and misuse of data fields. These maintenance strategies are deemed less desirable as they are suspected –but rarely proved– to shorten the lifespan of the CS.
Even if CS changes are detected, it can be difficult to determine if the change is stable. It must be traced back to changes in the UoD, and the following complications might arise:

- lag time between change in the CS and change in the UoD
- update anomaly, if a UoD change causes multiple CS adaptations
- parts of the CS may be invalid; changes to those parts are irrelevant
- user perceptions (the *Mental Model* in fig 1) may change without there being a material change in the way of doing business

Stability or Flexibility

To demand that a new design CS will undergo only stable changes in the future, is equal to demanding its flexibility. But an existing CS can be checked for stability by looking at its changes in the past. The important point is that there is no convincing evidence that correct application of the three design strategies actually does improve the stability of the resulting CS's. To demonstrate effectiveness of the design strategies, changes occuring in operational CS's must be researched, their stability must be determined, and finally, the contribution of the design strategy to that stability must be established.

5 Summary and Conclusion

This paper discussed the problem of determining the flexibility of a CS. From a business point of view, it is important to know in advance whether a proper CS will prove to be flexible enough. Many authors demand that a CS design be flexible, but do not explain their concept of flexibility. They presume that CS changes are driven from the UoD only. State-of-the-art design strategies also claim to promote flexibility, but these claims are all founded on theory, not on proof. The relationship between changes in the UoD and changes in *operational* CS is not well understood. It can only be obtained by researching CS's in their natural environment, the operational business. The research must center on the concept of *stable changes*: changes in the CS that are driven by corresponding changes in the structure of the UoD. Such research will encounter serious difficulties as CS changes are very difficult to observe in real business environments. But it is only through an understanding of this relationship that we can hope to improve current practices.

References

1. Crowe T.J. *Integration is not synonymous with flexibility*, Int. J. of Operations & Production Management vol 12, no 10 (1992) 26–33
2. Di Battista G., Kangassalo H., Tamassia R. *Definition libraries for Conceptual Modelling*, Proceedings of the 7th Int. Conf. on the Entity-Relationship Approach, ed. Batini (1989) Elsevier Science Publishers 251–267
3. Filteau M.C., Kassicieh S.K., Tripp R.S. *Evolutionary Database Design and development in Very Large Scale MIS*, Information & Management vol 15 (1988) 203–212
4. Giacommazzi F., Panella C., Percini B., Sansoni M. *Information systems integration in mergers and acquisitions*, Information & Management vol 32 (1997) 289–302
5. Jajodia S., Ng P.A., Springsteel F.N. *The problem of equivalence for Entity-Relationship diagrams*, IEEE Trans. on Software Engineering vol 9, no 5 (1983) 617–630
6. Kesh S. *Evaluating the quality of Entity Relationship models*, Information and Software Technology vol 37 (1995) 681–689
7. Moody D. *Seven habits of Highly Effective Datamodellers*, Database Programming & Design (1996) 57–64