SPEAR: Extending ER for Dynamic Behaviour and Refinement

Simon Wiseman and Bryony Pomeroy
Defence Research Agency
Malvern, Worcestershire WR14 3PS
{wiseman,pomeroy}@hydra.dra.hmg.gb

Abstract. SPEAR is an Entity–Relationship approach to modelling database systems which not only captures static requirements, but also dynamic behaviour. The notation can be used to give high level abstract requirements, or more detailed implementation level designs, and an additional refinement notation can be used to describe how specifications at different levels relate to one another. Tools to support the use of the notation, through a natural language interface, have been developed and have been used to construct the examples given.

1 Introduction

The Entity-Relationship approach to modelling database systems was originally proposed by Chen in 1976 [1]. Since then numerous extensions and variations have been put forward. One of these, SPEAR [2], was defined in order to assist with the specification of secure databases.

The original design of SPEAR was driven by three motivations. The first was to produce a single notation that was capable of capturing both high and low level specifications for database states. The second was to produce a notation which was itself specified mathematically. The third was to avoid the awkward terminology that surrounds the distinction between what Chen termed “entities” and “entity sets”.

The need to support high level specifications was seen to be important from a security point of view, because there is often a significant disparity between a security requirement and a security mechanism. For example, a requirement for purchases to be authorised by two people may need a mechanism that allows signatures to be collected on various forms. Thus a notation which can only describe low level specifications cannot help relay difficult security requirements.

The provision of a mathematical foundation to the model was seen as a way of ensuring that all details of the model were well understood by its designers. With hindsight, this was a very valuable strategy. Although design of the notation has seemed slower as a result, there have been many occasions where a construct whose meaning was thought to be intuitively understood turned out to have some awkward side effects when it came to define its semantics rigorously.

The importance of terminology cannot be over-emphasised. Existing E-R terminology can be very confusing, especially to the novice. Further, when it is used
to state security requirements, serious misunderstanding can result. Hence the
terminology used in SPEAR was carefully chosen, with some new terms, to allow
individual entities and collections of entities to be distinguished easily.

Once the initial model had been defined, a graphical notation was developed [3]. The
underlying model is defined using Z [4], but this is only readable by the
mathematically inclined. The graphical notation essentially provides a readable syntax
for the Z, making it a practical way of specifying databases.

To help produce the SPEAR diagrams, a simple tool was developed. To avoid the
expense of developing a diagram editor, this tool was written in Prolog on a
Macintosh and took simple sentence patterns as input and produced the diagram
described by the sentence, which could then be cut and pasted into a document.

Once use was made of this primitive tool it became apparent that there was merit in
the approach of using natural language to drive a CASE tool. As a result, a full
prototype tool, called SABRE, was produced to support the development of SPEAR
specifications. Microsoft Word is used to enter constrained-English descriptions of a
specification and then SABRE is invoked from the Word menus. The text is extracted
from Word, parsed by SABRE, the diagram is produced and automatically pasted into
the document. A knowledge base containing a description of the specification is
maintained and this is used to check for consistency and to generate schemas for
relational databases.

Initially, like other E-R models, SPEAR was designed to express the static structure
of a database. Though it did allow for constraints on how data was to be classified, it
was apparent that many security requirements are constraints on the dynamic
behaviour of a system.

While most methods for developing database systems include ways of describing
dynamic behaviour, such as Entity Life Histories and Data Flow Diagrams [5], these
are really specifications of orthogonal views of the system and not extensions to the
E-R model used for the static structure. Relating these different views, and reconciling
the differences in notation, is difficult, with the result that documentation often lacks
consistency and completeness.

For SPEAR the aim was to ensure that security requirements could be stated precisely
and unambiguously, hence a way of specifying dynamic behaviour which provides
close integration between the description of static structure and dynamic behaviour
was required. Fortunately it has proved possible to extend the existing SPEAR
notation with constructs for describing dynamic behaviour, and include these in the
SABRE tool [6], without any major reworking. The resulting notation is described in
[7] and [8].

In the development of large databases, implementation level specifications alone do
not provide adequate documentation for the maintenance of the database throughout its
lifetime. In order to manage changing requirements it is necessary to produce a
number of specifications, starting with one at a high level of abstraction which gives
a high level view of the requirements. Further specifications then refine previous ones
with additional detail, until an implementation level specification is produced.