Empirical Modelling in Support of Constructionist Learning: A Case Study from Relational Database Theory

Meurig Beynon, Antony Harfield
Department of Computer Science, University of Warwick, Coventry CV4 7AL
{wmb,ant}@dcs.warwick.ac.uk

Abstract

Conventional programming paradigms have limitations where support for constructionist learning is concerned. This paper illustrates the merits of an alternative approach to giving support for constructionist learning, based on the principles of Empirical Modelling (EM), with reference to an algorithm from database theory. Effective model-building for constructionist learning has to support activities relating to three roles: that of student, teacher and developer. This paper aims to show that EM brings far greater conceptual unity to interactions in these roles than is typically found in conventional approaches to educational software development.

1. Introduction

Constructionist computer-assisted learning can be seen as ideally unifying three roles: that of the student, the teacher and the developer. The learner first explores in ignorance and confusion in the role of a student, then identifies concepts and objectives for model-building to support and direct their exploration, then constructs appropriate models with which to repeat a similar cycle of interaction (see Figure 1a). As discussed in [1], with conventional programming techniques, each role corresponds to a radically different perspective on a program, corresponding to its use, design and implementation. This paper illustrates an alternative scenario, where the constructed model acts as a common construal to support concurrent interaction in the three roles (see Figure 1b).

2. An EM construal for the TLJ algorithm

The Testing_Lossless_Joins (TLJ) algorithm, as specified in Ullman [3] (see Algorithm 7.2 on p227), is a standard component of the relational database theory. The essential principles of the algorithm can be inferred from the following brief informal description. The first stage of the algorithm is to set up an array in which each entry is a symbolic element of the form a_i or b_{ij}, where i (respectively j) is the index of the row (respectively column) in which the element is located, and an a_j appears in location (i,j) if and only if the attribute associated with the j-th column appears in the i-th subscheme. The algorithm then proceeds step-by-step by taking account of the functional dependencies (FDs) in turn in cyclic order. At each step, when a particular FD of the form $X \rightarrow Y$ is being considered, the array is processed so that if any two rows have identical entries in the columns associated with all the attributes in X, they are modified so as to agree on all attributes in Y. In this process of modification, b_{ij} entries are replaced by a_j entries wherever possible, and agreement is otherwise established by assigning the same indices to all the relevant b_{ij} entries. The algorithm terminates when no further modification of the array results from the application of any of the given FDs, at which point the join is declared lossless if and only if there is a row comprised of a_j entries.

An EM *construal* is a computer-based model that embodies the patterns of observation, dependency and agency that are observed in its referent [1]. A detailed account of the principles and tools used in developing construals in EM is beyond the scope of this paper (cf. [2] for more details), but the essential ideas can be illustrated with reference to our chosen case-study.

For the TLJ algorithm, the primary observables are the contents and attributes of the table that is generated in executing the algorithm and the associated FDs. Both teacher and student come to understand the algorithm in terms of just these observables; building a

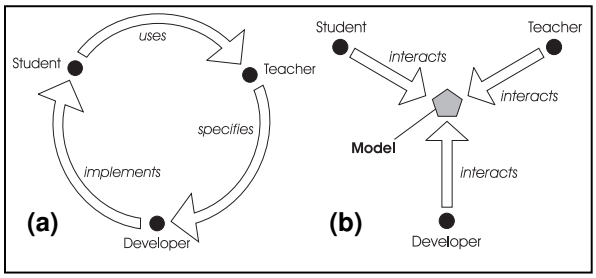

Figure 1: Roles in constructionist learning

construal to embody these observables, and the patterns of dependency and agency to which they are subject, is also a most appropriate way for the developer to provide support for the manual, semi-automated or fully automated interaction that must accompany the learning of the algorithm.

Learning the TLJ algorithm is linked to a pattern of observation that applies at each step. The learner consults the current state of the table with a specific FD X→S in mind, observes the pattern of tuples that arises in the columns associated with the left-hand side X of the FD to detect where there are duplicates, then observes how this pattern applies to the column associated with the right-hand side S of the FD. The core step of the algorithm is the substitution of the resulting transformation of the column associated with S for the original column.

For a particular table and FD, the above ingredients of the core pattern of observation can be displayed pictorially as in Figure 2. The arrows in this figure represent dependencies between observables, expressing the way that a given state of the TLJ table, and a given FD determines the set of columns LHS and a column RHS, and how the duplicate rows in the set of columns LHS then determine the updated entries in the column RHS. In the modelling environment used to develop the construal, these dependencies can be directly specified and are automatically maintained. This makes it possible to explore, in an experimental fashion, the way in which the current instance of this pattern of observation is affected by changing the current state of the TLJ table, or the current FD.

3. Developing and deploying the construal

The exploratory activity that surrounds the identification of observables and dependencies is a core activity that is central to the interests of the student, the teacher and the developer. As Figure 2 illustrates, the contexts for observation with which the student must become familiar in learning the TLJ algorithm are rich and subtle: they involve moving from global observation of the entire table to localised observation of the entries in specific rows and columns. It is also significant that the activities denoted by the arrows in Figure 2 are best conceived as mental operations on the part of the student, preparatory to the action of updating the table. From a teacher's perspective, each of the arrows can be interpreted as a link in a chain of observation involved in executing a step of the TLJ algorithm. As such, it can be the subject of an exercise: for instance, identifying the columns LHS and RHS, given a table and a FD. Decomposing the pattern of observation into a chain of simpler observations also has potential value as a diagnostic tool: for instance, helping the teacher to detect where a student understands the updating mechanism correctly, but is mistaken in their interpretation of a FD relation.

From the perspective of this paper, the relevance of Figure 2 for the developer has particular interest. There is a very direct correspondence between Figure 2 and the EM construal for the TLJ that was first constructed as an open interactive environment by the first author, and subsequently extended by the second to provide specific interfaces to the construal. This correspondence is best appreciated by interacting with the dynamic script development environment that is supported by the EM tool used in this development: the `tkeden` interpreter, but it is to some extent apparent from the relationship between Figure 2 and Listing 1. Just as the pattern of observation depicted in Figure 2 is the core of the TLJ algorithm, so the script of five definitions linking observables and dependencies in Listing 1 is the core of the TLJ construal. The names of the observables in Listing 1 have been made more expressive, and the code for operators (such as `index_duplicated`, and `makelistcol`) has been omitted, but the definitions are essentially as they

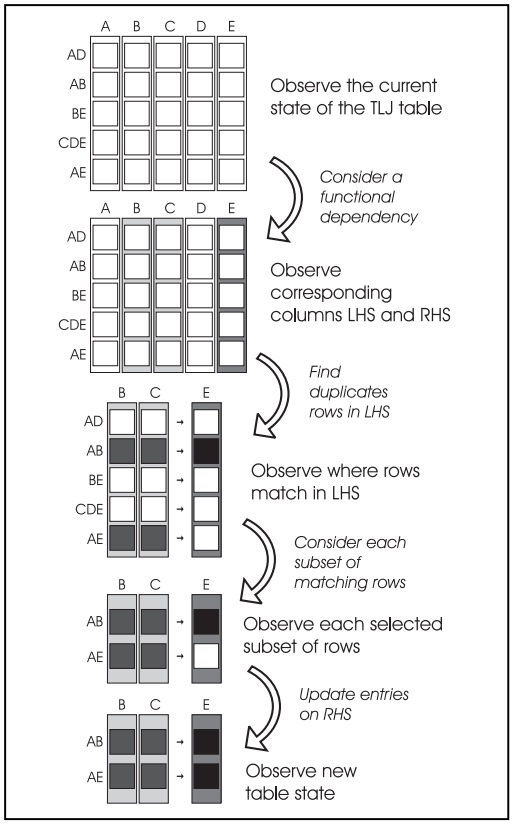

Figure 2: The TLJ pattern of observation

```
project_table_LHS_FD is project(current_table, makestrlist(FDs[current_FD][1]));
project_table_RHS_FD is project(current_table, [FDs[current_FD][2]]);
pattern_duplicate_rows is index_duplicated(tail(project_table_LHS_FD));
newcol is transformcol(makelistcol(project_table_RHS_FD), pattern_duplicate_rows);
newtable is apply_current_FD_current_table(current_table, newcol);
```

Listing 1: Observables and dependencies in the TLJ construal

appear in the `tkeden` source. Since our aim is to illustrate the convergence of viewpoints of student, teacher and developer suggested by Figure 1b, a brief explanation of how this script was developed, and relates to the pattern of observation in Figure 2, is appropriate.

As is evident by inspection, the values of all the observables in the script in Listing 1 are determined from the index of the FD that is currently of interest (`current_FD`) and the current contents of the TLJ table (`current_table`). The first two definitions determine the contents of the columns that correspond to the LHS and RHS of the current FD respectively. The third definition identifies the pattern of duplicate rows in the columns in the LHS of the FD; the fourth expresses the way in which the new contents of the RHS column is to be updated by consulting the pattern of duplicate rows. The final definition expresses the relationship between the original value of the table and the value that it takes after the FD has been processed. These definitions correspond closely to the links in the pattern of observation in Figure 2: in establishing the definitions using the `tkeden` interpreter, the operators introduced to specify the relationship associated with each link are tested in isolation by supplying different test values for the parameters in much the same way that the student might confirm that they have understood each observational link in mastering the algorithm. Though the development otherwise has more of the characteristic flavour of conventional programming, it remains anchored in this way to the learning domain. The missing elements of the `tkeden` source are the specifications of the operators themselves, which take the form of rather straightforward procedural code to compute an output from an input without side-effect. The script illustrates other features that are of interest from a computational perspective. These include:

- the re-use and adaptation of standard operators (such as the operator `project`, borrowed from the relational database extension of `tkeden`).
- the use of definitions to maintain dependencies between different modes of observation that are a common concern for traditional programmers, namely those that are associated with two or more data structures for a particular application (such as the conversion function `makelistcol`).

For the experienced developer using `tkeden`, the model-building task is greatly simplified by a combination of these three techniques: programming of relatively simple functions without side-effects; re-use of existing functions and scripts; and the use of definitions to maintain many different consistent concurrent representations of a given family of observables.

4. Conclusion

The difficulty of unifying the roles of student, teacher and developer is one of the obstacles to constructionist computer-assisted learning. In activities such as developing micro-worlds for children current development techniques do not enable the learners to build the models. Our case study is of interest because it proves that in principle there can be a high degree of synergy between interactions that are demanded of the learner in the roles of student, teacher and developer. For the target group of learners (viz. computer science students following an advanced module in database theory), there is no great conceptual or practically significant distinction between the kind of activity involved in learning about the lossless join algorithm and that involved in constructing the associated EM construal. It remains to be seen to what extent, subject to appropriate tool refinement and suitable training in the application of EM principles and tools, the same synergy between learning and model-building can be demonstrated in other learning contexts.

5. References

[1] W.M.Beynon, C.Roe, Computer Support for Constructionism in Context, In *Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies* (ICALT), 2004, .216-220.

[2] W.M.Beynon, A.Harfield, Empirical Modelling in support of constructionism: a case study, CS-RR-412, University of Warwick, April 2005

[3] J.D.Ullman, *Principles of Database Systems*, Computer Science Press, 1982.