Evolution and issues in metaCASE

Steven Kelly*, Kari Smolander

University of Jyväskylä, Department of Computer Science and Information Systems, P.O. Box 35, FIN-40351 Jyväskylä, Finland

Abstract

Customizable CASE environments (metaCASE) have begun to emerge in the marketplace. They offer tools and facilities for flexible method support and adaptation. This paper gives some views on the history of customizable CASE and metamodelling. The most difficult issues on the way to an integrated metaCASE environment are addressed, including representational problems, conceptual problems such as storing and processing recursive structures, method integration problems, and problems in creating repository support for metaCASE. These areas, and the divisions between them, are shown to need further work before a truly integrated metaCASE environment can be produced.

Keywords: MetaCASE; CASE

1. Introduction

Many scholars have claimed that Computer Aided Software Engineering (CASE) will substantially reduce the costs of developing software, standardize the systems specifications and thereby improve the quality of information systems [1,2]. However, recent evidence from using CASE tools1 shows that plain application of CASE technology will probably not change the situation substantially, as other dimensions in systems development must also be taken into account. For example, Siltanen [3] shows that the attitudes and orientation of IS management affect the impact of CASE. The study by Smolander et al. [4] revealed that the introduction of CASE is a major change which fails without the availability of sufficient resources and management attention. Other important factors included were the quality of the tools and their customization capabilities. LeQuene [5] (p. 260) argued that changes in working practices and changes in the use of systems development methods should be evolutionary when introducing CASE. This means that CASE tools should have customization capabilities for preserving successful systems development practices when introducing CASE. By customization capability we mean the ability of the tool to adapt to different systems analysis and design situations. A customizable tool’s languages, functions, and mechanisms can be somehow adapted (or “programmed”) to different situations and their particular needs.

Despite all these observations, until lately there has been only minor interest in the customization problem and in analysing the connections between systems development methods and practices and CASE. The development of CASE started from such text-based environments as SREM [6], PSL/PSA [7], and PDL [8]. These helped mainly in designing systems and software by offering understandable design languages and centralized data dictionaries, and were to some extent customizable. The design languages could be extended or adapted to the needs of the users and the schema of the data dictionary was largely definable.

The advent of powerful workstations and graphical user interfaces has brought, however, a certain inflexibility to the field of CASE. The customization of graphical CASE is not so straightforward a process as the customization of text-based environments, and the specification of graphical design languages is not so simple as the specification of linear design languages. Therefore, the method support of widely used commercial CASE tools is mainly fixed, and limited to the most well-known textbook methods.

To enable a flexible customization of the methods in CASE, new approaches and tools are needed. To cater for some of these concerns Bubenko [9] introduced the concept of a CASE shell, which would "... include mechanisms to define CASE tools for an arbitrary method or a chain of methods" (ibid. p. 11). In order to define a method in a CASE shell one must define a conceptual schema of the design objects constituting the method and specify how
these design objects are represented to the users of CASE tools. These conceptual and representational specifications form a metamodel and the specification process is called metamodelling. In the following, we will summarize the research that has been done to improve the state of the art in metamodelling and metaCASE environments, and what kind of methods and tools have been developed to carry it out. Finally, the most pressing issues in developing metaCASE environments are summarized.

2. Metamodelling

The field of metamodelling emerged in the 1980s. As noticed in Teichroew et al. [10], developers of CASE tools soon realized that underlying a system development methodology exists a model of what an information system is. By specifying the underlying model using, say entity-relationship diagrams, it is possible to compare methodologies and to construct methodology support in a CASE tool.

Brinkkemper [11] defines a metamodel as a conceptual model of a modelling technique. Brinkkemper's concept of modelling technique is similar to our concept of method, which we understand to embody a set of concepts that determine what is perceived, a set of linguistic conventions and rules which govern how the perception is represented and communicated, and a set of procedural guidelines which state in what order and how the representations are derived or transformed (cf. Smolander et al. [12]). Both Brinkkemper's concept of modelling technique and our concept of method consist of a linguistic or conceptual part which defines the concepts and languages to be used, and a procedural part which defines the procedures and activities embodied in a method or technique. Brinkkemper calls the linguistic or conceptual part a meta-data model and the procedural part a meta-activity model.

The first proposals for meta-data modelling were suggested in Teichroew et al. [10], which applied the entity-relationship model as a basis for methodology models. Teichroew et al.'s proposal was the starting point for the development of the OPRR model [13] (see also Sorenson et al. [14] for a similar model), in which the concept of role was used to extend the ER model. Other approaches suggested for meta-data modelling include, for example, the modelling of graphical notations with set-theoretical formalisms in RAMATIC [15], and the application of NIAM [16] to metamodelling [17].

Meta-activity modelling has produced fewer publications than meta-data modelling. Perhaps the solutions to the problems in meta-activity modelling are less commonly agreed, the activities in systems development vary more than its linguistic and conceptual structures, and the practical applications of meta-activity modelling are not so obvious. There are, however, some approaches for meta-activity modelling which include, for example, activity modelling with task structures [17,18]. In addition, the field of software engineering and programming has focused on software process modelling, the programming of the process of software production [19], which can be equated with meta-activity modelling. Also the software engineering concept software factory [20] implies the modelling of software processes as some kind of meta-activity model.

3. MetaCASE environments

Work in metamodelling theory has often been followed by some constructive work in the form of metamodelling environments. These metamodelling environments exploit the theories of meta-data modelling, and therefore they can be easily divided into two categories based on the languages they support. If the description languages that the environment can support are always linear in the sense that they consist of one dimensional sequences of predefined symbols, the environment is called a text-based environment. If the description languages can be n-dimensional (graphical languages, tables), the environment is called here a graphical one.

3.1. Text-based environments

Among the first text-based metamodelling environments was SEM [10,21,22] which was a textual environment based on a variation of the entity-relationship datamodel. SEM was composed of two parts: the information systems language definition system (ISLDS) and the systems encyclopaedia management system (SEMS). The definitions made by the language definition system specified the description language to be stored in the systems encyclopaedia management system, in which the information system descriptions were collected. This division between the meta level (ISLDS) and the instance level (SEMS) is characteristic of all metamodelling environments.

The architecture of SEM has had a strong influence on most successive metamodelling environments. Particularly, the text-based environments such as Plexsys [23], MetaPlex [24], and QuickSpec [25] have derived many of their features from SEM. Each of these environments has separated the language definition part (meta level) and the actual information system specification part (instance level) which uses the language defined on the meta level. The meta level itself is based on a still higher level model which can be called the meta-meta model. The meta-meta model in all these environments is some variant of the entity-relationship model.
The modelling of linear languages has well-established solutions, and therefore the development of text-based environments is easier than the development of graphical environments. Unfortunately, most developers of information systems do not want to use textual design languages. Powerful workstations and windowing systems have offered possibilities of graphical modelling and created a need for visual tools that the textual environments do not meet.

3.2. Graphical environments

The availability of cheap graphical interfaces has created new requirements for software engineering environments. The environments must support graphical modelling languages of the user’s choice. The environments should also conform to the conventions of the particular interface used, i.e. the windowing functions, mouse and cut and paste operations should be used in a standardized way.

The support of graphical modelling languages in commercial CASE tools is on the whole predefined. They support a few of the known text-book methods and the possibilities for method customization compared to the text-based environments are very limited. One reason for the inflexibility of graphical CASE tools is the nature of graphics. The field of conceptual modelling is well established and has therefore provided sound principles for the modelling of concepts, but the field of graphics has not provided similar principles for the modelling of graphical modelling languages.

Although graphical CASE shells are not so widely used yet, there are some environments that offer partial solutions to the modelling of graphical methods. The Swedish CASE shell RAMATIC [15] uses set-theoretical constructs to model graphical notations; the British Eclipse [26] uses directed graphs to represent design diagrams; and the Finnish MetaEdit [12] uses an entity-relationship model extended with the concept of role for the modelling of concepts and their graphical representations in the methods. Its meta-metamodel is based on the OPRR model [13]. Fig. 1 is a presentation of the level structure of MetaEdit: this structure can be generalized to apply to the majority of available metaCASE environments. A metaCASE environment may have several different metamodels, which then determine the methods that are used in modelling information systems.

4. MetaCASE issues

So far, we have explored the historical development of metamodelling and metaCASE, and the current state of the art. In this section we will examine some of the more pressing issues facing metaCASE. Dividing them into representational (the way a method looks), conceptual (the underlying data in a model or metamodel), methodological and implementation (repository; the data store) problems, we will identify areas where we feel research or tools are at present weak, and where they could be extended. These issues in particular have been chosen, because they are required by methods that are currently in use in organizations, either with paper and pencil or with method-specific CASE tools, and yet they are not supported well by existing metaCASE tools.

4.1. Representational problems

Our experience of building a metaCASE tool has shown that problems concerning the support of the representations used in methods are often more difficult than those concerning methods’ concepts. The meta-metamodel, in this case OPRR, required only four basic concepts, and needed little adaptation during the course of implementation; the need for a general system to support all representations and their associated semantics constantly demanded attention. In this section we will examine some further extensions to the support of different representations needed within the next generation of (meta-) CASE tools.

Graph-Matrix-Text. Several methods, many of which are concerned with developing business systems, use matrices rather than graphs (i.e. pictures of graphical symbols connected by lines or curves) as their main representation, such as IBM’s Business Systems Planning [27]. Andersen Consulting’s Method/1 [28,29], Business Information Analysis and Integration Technique [30], and relational modelling [31]. Support for these methods has largely been restricted to purpose-built, fixed method CASE tools, rather than metaCASE tools.

A more common extension to (meta-) CASE tools has been support for the production of textual representations, such as reports. In some specialized systems this has been widened to include hypertext, for instance as a medium for recording design decisions [32]. There exist still further possibilities of considering (hyper-) text on the same level as graphs and matrices, or of using it to provide an infrastructure for the whole CASE environment [33].

![Fig. 1. Level structure of MetaEdit.](image-url)
Conditional graphics. The graphical symbols used in certain models [34,35] can vary not only according to the type of the object, but also according to values contained in the object's properties, whether it has duplicates, whether it has been decomposed, or the number of types of relationships attached to that object. The need to present different views to support the activities of different developers or phases of development also motivates towards the capability to represent the same object in contextually different ways.

Considerations internal to the application itself, such as locking or activity modelling, require additional ways of marking or changing the appearance of objects, over and above the normal visual indication that an object has been selected. These internal characteristics must be represented as consistent a way as possible, whether the object is currently displayed in a graph, matrix or text form.

Specialized behaviour. Most graphical methods attach semantics to the vertical and horizontal direction of flow of relationships or position of objects, with varying degrees of formality. To support or enforce (depending on the level of formality) this feature of the method requires specialized behaviour within the graphical editor, for instance by fixing objects to a grid or constraining the direction of relationships. This behaviour must be made available by the metaCASE tool, and then instantiated with any necessary values by the metamodel.

4.2. Conceptual problems: recursive structures

Among the most powerful functions of the human brain are the abilities to group entities, to form hierarchies and abstractions. Consequently, any tool hoping to empower a human designer must support these ways in which people are the abilities to group entities, to form hierarchies and abstractions. Thus, in arriving at a repository and schema for metaCASE, it is essential to take stock of the data needs of different methods to achieve intended results. The concepts in these methods are somehow tied together so that the transitions from a method to another (and maybe back) can be successfully executed. In metaCASE environments enforcing these connections between methods leads to difficult problems, and no satisfactory solution is available. Moreover, we must maintain consistency of design information along at least two dimensions (cf. Marttiin et al. [36]):

(1) **Horizontal consistency**, the consistency between different descriptions on the same level of abstraction, for example between an ER-model and a data flow diagram.

(2) **Vertical consistency**, which is the consistency between semantically equivalent descriptions on different levels of abstraction, for example between an ER-model and an equivalent data base schema.

Despite their great importance, systematic studies of the principles of method integration are missing.

4.3. Methodological problem: method integration

Usually in systems development people use several different methods to achieve intended results. The concepts in these methods are somehow tied together so that the transitions from a method to another (and maybe back) can be successfully executed. In metaCASE environments enforcing these connections between methods leads to difficult problems, and no satisfactory solution is available. Moreover, we must maintain consistency of design information along at least two dimensions (cf. Marttiin et al. [36]):

(1) **Horizontal consistency**, the consistency between different descriptions on the same level of abstraction, for example between an ER-model and a data flow diagram.

(2) **Vertical consistency**, which is the consistency between semantically equivalent descriptions on different levels of abstraction, for example between an ER-model and an equivalent data base schema.

Despite their great importance, systematic studies of the principles of method integration are missing.

4.4. Implementation problem: repository

Requirements for a database differ according to its intended purpose, as noted by Brown [37] in comparing those for a commercial database with those for a software engineering database. For a multi-user metaCASE tool, the requirements are a witches' brew of those two, with added peculiarities. Thus, in arriving at a repository and schema for metaCASE, it is essential to take stock of the data needs of such an environment, recognizing where previous work on other kinds of databases can be used, and where new theories and practices must be developed. Table 1 shows the requirements from Brown (p. 39), with those which we feel apply to a metaCASE tool in bold: the precise nature of access patterns will only be possible to determine later, by examining a real multi-user metaCASE and methodology engineering environment in operation.

The fixed meta-meta model allows a fixed schema on the conceptual level: we need to store the meta-model for each method, then the conceptual data for each model can be mapped via this to the meta-meta model. By keeping the representational data free-form or unformatted, so no
Most information is static and can be described \textit{a priori}, so the schema is also static and compiled.

Table

<table>
<thead>
<tr>
<th>Commercial database</th>
<th>Software engineering database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous evolution of information—data about the environment itself (tools, methods, etc.), and the products being developed.</td>
<td>Change to the schema is expected and frequent. Many users will need to change the schema.</td>
</tr>
<tr>
<td>Data stored is atomic but also structured. Could be graphical data, design documents, programs, and so on. Also, data items may be large and complex, and of variable length.</td>
<td>Data may be conveniently divided into three groups: \textit{meta data}, containing the types, rules and symbols of the method, \textit{model data}, which is representational, conceptual, or a combination of the two—for instance the name of an object may be both conceptual and also a graphical label on instances of that object, and \textit{user data}, such as agent models, configurations, views, and permissions to access data.</td>
</tr>
<tr>
<td>Small number of entity types, with large numbers of instances of each type. Often only simple, fixed relationships between entity types exist.</td>
<td>Less data initially loaded. Rapid growth of database (both of structure and contents), which slows down after completion of design phase.</td>
</tr>
<tr>
<td>Initially loaded with large amount of data. Shows, constant rate of data growth.</td>
<td>Versions of data items are vital. Dependence on versions, and relationships between versions must be explicitly recorded.</td>
</tr>
<tr>
<td>Single-valued data items, which are updated in place.</td>
<td>Long-lived transactions (&gt; hours) which may leave the database inconsistent for long periods. Cannot be conveniently used as locking units.</td>
</tr>
<tr>
<td>Transactions are short, atomic, and can be used as the basis for concurrent data access.</td>
<td></td>
</tr>
</tbody>
</table>

5. Conclusions

The field of metaCASE environments, both in terms of products and research, has been around for some time now. The early text-based metaCASE environments, although simpler to make, were not found to meet the needs to which they were addressed. Initial attempts to expand into graphical representations have also proved insufficient for development needs, because of their over-simpler meta-meta models, single editor or tool, and single representational paradigm. Although initially growing slowly, the field has gathered pace. However, in the current surge of interest, it is important that basic issues are re-examined, to provide a common foundation for data interchange and future expansion. Without this, metaCASE stands in danger of missing the wave of free data exchange and communication, leading to either a diverse, dissipated marketplace, or one, possibly two, vendors or schools of thought, whose frameworks are mutually exclusive and opposing.

One of the main considerations throughout the history of metaCASE has been to recognize different types of data: conceptual/representational, meta/model, user-definable/fixed. These distinctions, foundational as they are to the whole principle of metaCASE, still require further refinement, in the light of the facts that there are no fixed boundaries between the types, and that parts of individual pieces may well lie on both sides of the divide. These items of data with a mixed type must be examined closely, both because of the difficulty of dealing with them within the current framework of divisions, and also to refine those divisions.
References