9

Application-Oriented Methods for Systems Development - A Review

D. Millington and I. M. Tulloch

Department of Computer Science, University of Strathclyde, 26 Richmond Street, Glasgow G1 1XH, United Kingdom.

Abstract

The need for *application-oriented methods* has not yet been established, but a method which supports the incorporation of domain knowledge into the development life-cycle is known to be attractive. This paper discusses the development of computer applications using methods and tools which in some way reflect the nature of the underlying applications. It identifies the objectives of such an approach, and discusses work carried out in the health-care domain, including the HEAL language which is currently under study at Strathclyde University. In addition, the basis of a pre-fabrication method (PFM) is outlined.

Keyword Codes: D.2.1; D.2.2
Keywords: Software Engineering, Requirements/Specifications; Tools and Techniques

1. INTRODUCTION

An earlier paper by Millington, Gray and Tulloch [1] discussed the role of application-oriented *tools* for software development in the creation of new systems. Software engineering tools are built to support the application of particular *methods*. This paper reviews the area of application-oriented methods - a topic merely touched on in the earlier paper. The basic question to be considered is "How far may any method for systems analysis and design be application-oriented and what advantages would result ?". The answer to this question is explored by considering work in the health-care applications domain in particular.

2. OBJECTIVES OF APPLICATION-ORIENTED METHODS

The prime objective of methods and tools oriented towards a particular domain of applications is increased user participation in systems development. This is achieved through expressing application requirements and designs in terminology similar to the normal language of the application area, and by providing tools which convert such requirements and designs into operational computer systems. The consequences are:

- improved software quality, particularly in respect of 'fitness-for-use'.
- improved software productivity, through automating more of the translation and transfer processes required to move from a user-model of a required system to an implemented model.

Methods and tools should therefore recognise commonly-occurring entities, as well as commonly-occurring tasks and their sequencing. For example, in the health-care domain, they should 'know' what a patient record is (at least in a default version) and what "make an appointment" means.

M. Lee et al. (eds.), *Software Quality and Productivity*

3. APPLICATION-ORIENTED METHODS FOR HEALTH-CARE SYSTEMS

3.1 Background

The contents of this section reflect the authors' involvement with the development of health-care (particularly hospital) applications over many years. Three distinct areas of activity relating to this domain are now discussed :-

(i) **HEAL** (HEalth Applications Language), which is currently under study at Strathclyde University in Glasgow.

(ii) The **CBS** (Common Basic Specification), which was developed centrally for the National Health Service within England and Wales.

(iii) Projects under the European Community's AIM (Advanced Informatics in Medicine) Research and Technology Development Programme.

The first two relate to specific work which sought to provide an applications-oriented context for *methods* used in systems analysis and design. On the other hand, the AIM projects had a stronger emphasis towards the provision of *tools and environments* for developing health-care applications.

3.2 HEAL (HEalth Applications Language)

Work on HEAL developed from a past hospital systems analysis study [2], which originally set out to express the data processing functionality occurring in each of a hospital's departments or areas of activity (over 70 in all). That it was necessary to record similar requirements in different areas prompted the analysts involved to formulate a formal syntax for specifying the processing procedures required.

Each procedure commenced with a verb (from a set of 21) and had its own defined specification structure. One example was SCHEDULE, defined as

> "arrange a timetable of activities or an activity within a timetable, STORE the details in the appropriate file(s), and initiate any consequent actions necessary". **(1)**

where STORE was another procedure pitched at a more basic level. The formal syntax of SCHEDULE was defined as

> Schedule event for individual / location (into file). **(2)**

where (i) the parentheses indicates an optional clause within the procedure, (ii) the stroke ('/') indicates permissible alternative nouns which may be used and (iii) the underlined words are replaced by specific examples in practice. For example, an actual invocation of SCHEDULE recorded in the original study was

> Schedule outpatient clinic for patient. **(3)**

The "consequent actions necessary" in definition (1) above were expected to be different for different actual events. In the case of the event represented by (3), these might include initiating a booking for patient transport and the scheduling of pre-clinic X-ray and ECG examinations for some categories of patient.

This specification language was developed primarily as a *method of systems analysis*. It was also envisaged to be used in both design and implementation; however, for several reasons, this did not occur. Tulloch [3] used this work as a basis for considering the design of **HEAL**, an implementation language for health-care applications.

Current studies are concerned with how HEAL could be implemented. Possible delivery systems include fourth-generation languages and object-oriented systems; indeed the latter appear, at least initially, to offer an attractive implementation route.

3.3 Common Basic Specification

Within the National Health Service for England and Wales, the extent to which hospitals and health authorities worked with similar *data* was recognised early in the development of computer use. Its recognition was formalised first in the reports of the Steering Group on Health Services Information (the Korner Committee) and in the consequent "Korner Data Model" [4]. From these was developed a wider "NHS Data Model" [5], which sought to support data modelling work involved in developing health-care applications.

The NHS Information Management Centre later supplemented this model by incorporating *processing* information. This led to the formulation of the Common Basic Specification (CBS), a model which claims to provide precise definitions of all NHS activities and the data they require, making "available in one source all the basic material required by anyone specifying information requirements in the NHS" [6].

The nature and use of the CBS are outlined by Dallimore [7]. It sets out to provide a generic model of health-care systems requirements, which can be tailored to the needs of a specific hospital (or hospital department). The CBS itself does not include implemented components for specific software/hardware environments, but it clearly offers the prospect of modular construction of applications through reuse of components.

The initial version of the CBS model was developed using SSADM (Structured Systems Analysis and Design Method) - the established mandatory method for developing software applications for the UK government.

In practice the CBS appears to have received a mixed reception from its intended users i.e. health authorities and external systems suppliers. This may be due to

- a perceived inherent complexity of the model, and/or
- a reluctance to apply the necessary effort and resources (e.g. training) to use the model

Nonetheless, the CBS appears to be the most extensive (and the most promising) attempt so far to incorporate domain knowledge and the principle of reusability into both the analysis and design phases of health-care application development.

3.4 European Initiatives

Hospitals and other health-care facilities throughout Europe share similar functions and organisational structures. Consequently, the need for and the perceived benefits of domain-oriented methods and their supporting tools are of interest within this wider context. The Committee of the European Communities (CEC), through its Research and Technological Development programme entitled "Advanced Informatics in Medicine" (AIM), recognised and supported this area of research.

In the preliminary phase of AIM (1989 - 1990), two projects - HEALTHBENCH and HELIOS - concentrated on methods and tools for health-care application development *in general*, rather than on the development of specific applications.

HEALTHBENCH set out to identify an appropriate architecture for an integrated software engineering environment to support the development of health-care applications. It surveyed the use of general-purpose software tools in the development of such applications [8], to gauge *inter alia* their perceived limitations. The project subsequently determined a general architecture for a health-care software engineering environment [9]. This perspective was developed further by Millington, Gray and Tulloch [1].

HELIOS set out to specify a "Software Engineering Environment" and a "Ward Information System" which would include an "Image Management Subsystem". It appeared to employ elements of an application-oriented method in its work [10]. Its successor in the current phase of AIM (1992 - 1994) is HELIOS II, whose quoted objectives are "to develop a fully integrated medical software engineering environment supporting analysis, design and implementation of medical applications and to use this platform for building significant parts of a multimedia medical workstation" [11]. One component of the environment being developed is an object-oriented database management system, which will "provide the developers with a

core set of medical classes needed to construct basic medical applications" (ibid). In this case, a *general* method is employed (i.e. the object-oriented method), and the application-orientated perspective is achieved by providing a pre-defined set of reusable classes.

4. APPLICATION-ORIENTED METHODS AND TOOLS IN OTHER DOMAINS

Millington, Gray and Tulloch [1] reviewed the use of application-oriented tools in general, noting in particular the work of the ATMOSPHERE project [12] under the CEC's ESPRIT programme of research and development. The ATMOSPHERE project has designed system engineering environments for five different application domains (i.e. embedded aerospace systems, distributed information processing, digital computer networks, communications, and process control). Each environment is intended to support the methods appropriate for its application domain - but this does not necessarily imply methods which are used only for that application domain and not for other domains.

A particular advantage of *object-oriented* methods is seen to be their potential to create reusable components of applications systems. Such components inevitably relate to a particular application domain, and the topic of *domain analysis* is identified as a major theme in the context of reusability. Arango [13] provides a valuable comparison of published domain analysis methods, and maps these onto a "Common Process". He also identifies *evaluation* (or alternatively *validation*) of domain analysis as a topic barely touched on so far. That requires consideration of the subsequent use of the results of domain analysis. In respect of this paper's theme, even if it may be argued logically that domain analysis as a method should not itself be domain dependent, the use of its results may be - but this aspect has still to be explored.

5. THE FUTURE OF APPLICATION-ORIENTED METHODS AND TOOLS

Earlier sections have shown that domain-orientation has influenced the work of systems development at several points in the system life-cycle. However, a fundamental question remains - "Is there a need for application-oriented methods for analysis and design, or is the need solely for software engineering tools which incorporate domain knowledge within general methods?".

If the need is for application-oriented methods, it is necessary to identify the particular capabilities required of the method beyond those of general methods. Indeed structured methods of analysis and design have for real-time and control applications different expressive capabilities from those required for the bulk of business and administrative applications. However, it may be argued that "real-time" and "control" do not identify *application domains*; rather they are *types of processing* which may occur across a range of application domains.

Irrespective of the answer to the first part of the above question, the answer to the second part appears to be a definite "Yes" - there *is* a need for tools enabling domain knowledge to be used within general methods. The present popularity of object-oriented methods is largely built upon their promise of enhanced "re-usability" of system components within a range of applications. Such components may very definitely include application domain knowledge. However although re-usability has been exploited in the implementation phase in object oriented programming, its use in other phases of the life-cycle appears to have received only limited attention so far.

If re-usability of system components is to be taken seriously there is a need to develop a "pre-fabrication method" (PFM) which supports the use and management of system components within and across the analysis, design, implementation and maintenance phases of systems development. Such a method requires not only that a development life-cycle should be meaningful for an individual component, but also that there should be well-established techniques for the integration of components into the overail system. Such integration techniques need to be available at each phase of the development cycle, and might be expected

to entail more than just the provision of a library or catalogue facility. They also need to cover the documentation for the final system (e.g. user documentation), as well as that which is part of the development process. Figure 1 gives a simple diagrammatic representation of the basic requirements for integration.

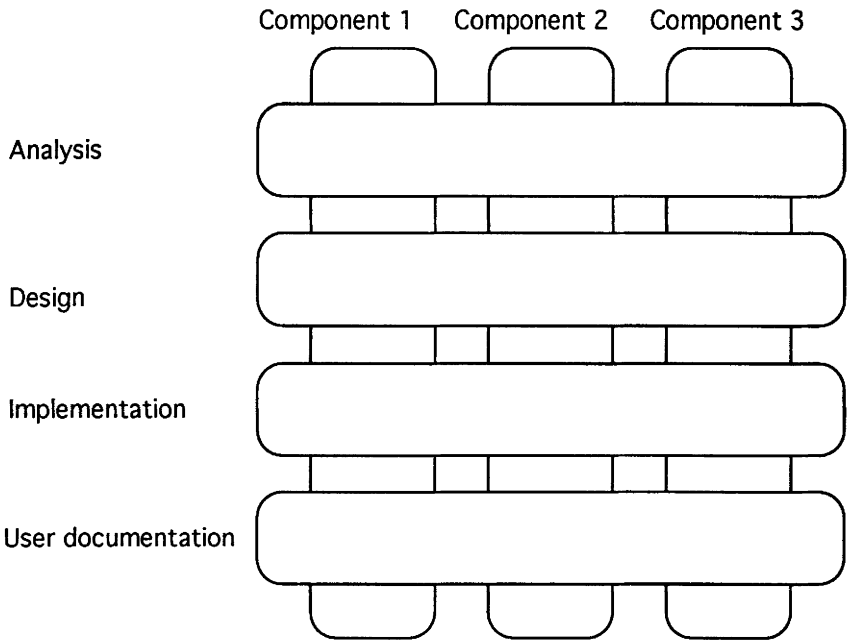

Figure 1 Integration needs

The set of techniques comprising PFM should additionally cover the requirements for ensuring quality through appropriate verification and validation checks in each phase. Again techniques have to be concerned not only with individual components, but also with their integration mechanisms.

This provides an agenda for further defining a PFM and appropriate tools for its operation. Significant inputs for this task will be experience to date with

- applications software libraries
- generic analysis and design specifications such as the CBS previously discussed.

The PFM is likely to be used by the end-user application developer; therefore it, rather than methods of domain analysis, is the prime contender to be application-oriented.

6. CONCLUSIONS

It is established that the incorporation of generic application domain knowledge into software development can improve quality and productivity, through its support of re-usability. How this knowledge is made available for incorporation into a specific application needs

further study. The approaches of HEAL and the CBS are two distinctive attempts to achieve this within the health-care domain, but a more extensive pre-fabrication method (PFM) is required - one which is applicable to *all* phases of application development.

Since application development will be undertaken increasingly by end-users, it is likely that a PFM will itself be application-oriented, reflecting particular characteristics of the user community as well as the underlying application domain.

REFERENCES

1. Millington, D., Gray, E. and Tulloch, I.M.: 'Application-oriented tools for software development' in: Stowell, F., West, D. and Howell, J. (eds), System Science: 'Addressing Global Issues', (Plenum Press, 1993), pp. 361-366.
2. Nuffield Medical Data Processing Unit, Notes to accompany a session entitled 'Methods of Systems Analysis in the Liverpool Project', given to the National Health Computer Services Group, April 7th, 1970.
3. Tulloch, I.M., 'Application Specific Languages (Including Introduction to HEAL)', M. Phil. dissertation, University of Strathclyde, Glasgow, 1992.
4. Steering Group on Health Services Information: Reports on the collection and use of information about activity in hospitals and the community in the National Health Service, H.M.S.O., London, 1984.
5. 'The National Health Service Data Model', Vol.1. Management Summary, NHS Corporate Data Administration, Birmingham, United Kingdom, 1986.
6. 'Introduction to the Common Basic Specification', NHS Information Management Centre, Birmingham, United Kingdom, 1990.
7. Dallimore, T.: 'Common Future', *British Journal of Healthcare Computing*, 1991, Vol.8, (3), pp. 26-28.
8. Millington, D., Tulloch, I.M. and Gray, E.M., 'Current Use of Software Tools in the Development of Medical and Health Services Computer Applications - a Survey', *Journal of Information Technology*, 1991, Vol.6, pp. 86 - 93.
9. Hooymans, M., 'HEALTHBENCH - Health Information Decision Support Workbench -General Architecture of Healthbench', in: Timmers, T. & Blum, B.I. (eds), *Proc. IMIA Working Conference on Software Engineering in Medical Informatics*, Elsevier, Amsterdam, The Netherlands, 8 - 10th October 1990.
10. C.E.C., Exploitation of results from the AIM exploratory action, C.E.C., DGX111/F/A11516, Brussels, 1990.
11. AIM 1993 - Annual Technical Report on RTD: Health Care, CEC, DGX111, Brussels, 1993.
12. Obbink, H., 'Systems Engineering Environments of Atomsphere', *Proc. European Symposium on Software Development Environments and CASE Technology*, Springer-Verlag, Berlin, 1991, pp. 1 - 17.
13. Arango, G., 'Domain Analysis Methods' in: Schafer, W., Prieto-Diaz, R., & Matsumoto, M.(eds), Software Reusability, Ellis Horwood, London, 1994.