CHAPTER 6

C.A.S.E. Technology

Introduction

The overall aim of CASE technology is to improve the productivity and quality of the resulting systems by assisting the developer throughout the different stages of the development process; from the acquisition of the functional and non-functional system requirements to the design and implementation of the system considering all the relevant technical and operational features.

CASE provides the software tools that support methodologies to employ in modelling all levels of an organisation. In this sense it is more appropriate to consider CASE in a wider context than just software production, that has normally been the case. Therefore, CASE may be described as software tools for enterprise support consisting of enterprise strategic planning, IS strategic planning, project planning, systems development, documentation and maintenance.

This chapter is organised as follows. Section 6.1 presents the possible advantages from the CASE technology and the need for CASE as part of system development and especially Requirements Engineering. Section 6.2 provides a classification of CASE technology. Distinctions are made between stand-alone CASE tools and integrated environments. Section 6.3 presents a generic architecture for CASE. Despite the plethora of CASE tools and supported methodologies in use today, the typical CASE architecture is built around the concept of a repository used for storing the various types of models which are created during software development, which are accessed by a number of assorted tools such as editors, diagrammers etc. The same section therefore specifies the requirements for repository functionality as part of CASE for Requirements Engineering. Issues to consider when selecting and integrating CASE tool for requirements are discussed in section 6.4. Finally, section 6.5 is concerned with properties and futures of research-oriented CASE tools for Requirements Engineering. Many of such tools are based on the principle that the utilisation of problem domain and methodical knowledge stored in the tool can automate (at least to an extent) many of the human labour intensive tasks of Requirements Engineering.

In summary, this Chapter presents an overview of the role of CASE technology within Requirements Engineering, keeping the discussion as much as possible free from references to specific tools and technologies. CASE is an indispensable feature of any modern software development approach and will be even more so in the future as its potential applicability within the Requirements Engineering process increases.

6.1 The Need for Computer-Aided Software Development

Requirements Engineering is one of the software development phases for which CASE is particularly suitable. This fact was realised in the early days of CASE and it was soon put into practice in the form of research prototypes first and commercial tools soon after. There are of course good reasons why CASE is particularly applicable to Requirements Engineering: Requirements Engineering produces vast amounts of information (in both textual and graphical forms). Obviously this information must be managed, captured, stored, retrieved, disseminated and changed. However the manual capture, storage, manipulation etc. of requirements increases the risk of errors creeping into the process and into the final outcome. Such errors cam manifest themselves in various forms i.e. as out-dated, inconsistent or incomplete or erroneous requirements models.

Another serious problem of manual practice of Requirements Engineering lies with the difficulty to enforce certain software standards. As it happens, users and software engineers have their individual ways of communication in textual or pictorial form. This however can cause serious communications problems in the context of software development. User reports written in a variety of formats and styles can be a nightmare for those responsible for their editing and translation into technical descriptions.

CASE tackles the problems related to software requirements in the following ways:

- First, by providing automated management of all the requirements related information.

- Second by enforcing standards. CASE tools provide standard formats for inputting, retrieving or changing information. Such standards can be for example document formats for textual requirements, diagrammatic notations etc. The use of a uniform set of standards across the software development team ensures that problems such inconsistency and misinterpretation are eliminated or greatly reduced.

CASE also affects the speed with which a requirements model is produced and updated. This is important, as Requirements Engineering has to deal with two contradicting demands, namely to involve the user as little as possible in the process (since users are probably too busy doing other things to participate in software development) and at the same time obtain all the information required from the user including feedback on the specified requirements. In resolving this conflict, CASE is invaluable since it offers two primary facilities, namely easy communication with the user through graphical models and prototyping. CASE allows the quick construction of quality graphical models which are easily understood by the user as well as their equally quick modification. Prototyping is also valuable in putting the user through life-like interaction with the software system in order to understand the user's true requirements (see also Chapter 5).

Most of the arguments in favour of using CASE in Requirements Engineering apply equally to the use of systematic methods for Requirements Engineering. As a matter of fact, CASE started as nothing more than an attempt to automate paper-and-pen software methods which in turn provided the much needed standardisation of documents and models, procedures for requirements change etc. to software development. It is a truism therefore that CASE and software methodologies have been dependent on its other for their success. Methods require automation in order to be practical; CASE on the other hand is of little help unless used in a systematic way within the development process, i.e. within a *method*. In a 'chicken and egg' fashion therefore the structured methodologies and CASE are responsible for each other's fast spreading of popularity in the 80's and 90's.

However, as the state-of-the-art moves beyond the structured approaches and into object-oriented and knowledge-based paradigms, the importance of CASE as the enabling technology for Requirements Engineering is moving into new areas. Sections 6.3 and 6.5 discuss the new role of CASE as a component of a Requirements Engineering approach, in both commercial and research-oriented settings.

6.2 Classification of CASE Technology

There exist several classifications of CASE in the literature. One of the first and most important ones classified CASE as *language-centred*, built around a programming language, *structure-centred* that were based on the idea of environment generation, *toolkit environments* that were primarily consisting of tools that supported the programming phase of the development and *method-based* which were centred around a specific methodology for developing software systems.

Another classification [Fuggetta 1993] is based on a framework consisting of three parts: *tools* that support only specific tasks in systems development, *workbenches* that support one or more activities and *environments* that support a large part of the software process. According to this classification, tools can be further classified into editing, programming, verification and validation, configuration management, metrics and measurement, and project management tools. Workbenches are classified depending on the activities that they support as: business planning and modelling, user interface development, programming, verification and validation, maintenance and reverse engineering, configuration management and project management workbenches. Finally, environments are classified as either toolkits which are integrated collections of products, language-centred, integrated, fourth generation environments or process-centred environments.

6.2.1 Upper and Lower-CASE

The most popular classification of CASE technology and tools is based on the distinction made between the early and late stages of systems development. Many of the current CASE tools deal with the management of the system specification only by supporting strategy, planning and the construction of the conceptual level of the enterprise model. These tools are often termed *upperCASE tools* because they assist the designer only at the early stages of system development and ignore the actual implementation of the system. The emphasis in upperCASE is to describe the mission, objectives, strategies, operational plans, resources, component parts etc. of the enterprise and provide automated support for defining the logical level of the business, its information needs and designing information systems to meet these needs.

UpperCASE tools support traditional diagrammatic languages like Entity Relationship Diagrams, Data Flow Diagrams, Structure Charts etc. providing mainly draw, store as well as documentation facilities. They support a limited degree of verification, validation and integration of the system specifications due to the inherent lack of formal foundations for the requirements modelling formalisms.

Other CASE tools deal with the application development itself with regard to the efficient generation of code. These are termed *lowerCASE tools* because they assist the developer at the stage of system generation and ignore the early stages of system requirements specification. The starting point of the system development with lowerCASE tools is the conceptual model of the information system. The conceptual modelling formalism is usually, based on formal foundations in order to allow for automatic mapping to executable specifications and efficient validation and verification of the system specification itself.

LowerCASE tools employ mapping algorithms to automatically transform formal specifications into an executable form. This includes, among others, transformation of specifications to relational database schemas, normalisation of database relations and SQL code generation. The majority of these tools facilitate rapid prototyping of specifications in terms of the functionality of the system. They do not support the development process itself but rather, they offer a powerful tool for making system design more effective and efficient.

The state of the art products of the CASE market nowadays claim that provide support for both the early stages as well as the implementation stages of information systems development. Clearly, from a users' perspective, this move towards *integrated CASE (ICASE)* is far more important [Gibson et al, 1989]. In this architecture, the repository plays a more active role in that all tools can interface and exchange data with it. A repository, holds data fields and definitions and ensures that data integrity is maintained throughout the development lifecycle. As a consequence, ICASE allow tools to work together relatively seamlessly and alleviates much of the stop-start nature of non-ICASE environments.

All the CASE environments mentioned above are often rigid and do not support the users' native methodology nor different methodologies. To avoid this, more flexible and customisable tools, called *CASE shells*, are emerging. They allow customisation of the CASE shell to a given methodology. Users are able to describe their methodology either through a set of meta-modelling editors or through a set of formal languages and tailor it to their specific requirements in order to create dedicated CASE tools. Many of the products and research prototypes of CASE shells move towards the support of different methodologies during the development of a single information system.

6.2.2 Integrated Software Development Environments

Central to the issue of CASE integration, is the concept of an Integrated Software Development Environment (*ISDE*). ISDE as the term implies, provides support for the coordination of all the different activities that take place during a software project. There are different types of support that an ISDE can provide. Typical examples of ISDE support include:

- automated coordination of development activities. For example, mechanisms may be provided which trigger the design activity at the end of the specification activity

- mechanisms for inter-activity communication. This means that data produced, for example by a specification tool can be filtered and subsequently transmitted to the design tool, to the project planning tool etc.

A major approach towards ISDEs is the European project *PCTE* (Portable Common Tool environment [Boudier et al, 1988]). The PCTE approach is similar to the idea of a modern operating system such as *UNIX* which offers a set of standard facilities such as text formatters, filters, process communications etc. which can provide the building blocks for creating sophisticated applications. In analogy, PCTE provides common building blocks to the developers of CASE tools which will run under PCTE. In turn, this can facilitate compatibility between the tools since they all use common facilities for storing and communicating their data.

6.3 A Generic CASE Architecture

6.3.1 Overview

Central to any CASE architecture is the concept of *data dictionary* or *repository* [Bruce et al, 1989; Burkhard, 1989; Martin, 1989b]. The role of the repository is to store all the logical and physical objects whose task is to provide control and integration in the development and maintenance of information systems. Essentially, the repository is the single point of definition in the software life cycle. In this role, the repository holds the metadata (data about data) which not only defines what and where the data is, but how it relates to other data and how the logical manipulation of data is mapped across physical structures like databases, file systems and, ultimately, physical objects such as networks and CPUs [McClure, 1988; Martin, 1989a]. In terms of the development life cycle, this means that any program or flow chart which is used in the building of an application and any tools which are employed in its construction, must receive and enter data to and from the repository.

The repository provides true integration of specifications from the different tools because it permits sharing specifications rather than converting and passing them between tools. CASE tools connect directly with the repository for specification storage and retrieval. The repository uses these specifications to drive an application generator and to generate operating systems commands, database calls, communication commands and user documentation.

A generic architecture for CASE tools is shown in Figure 6.1

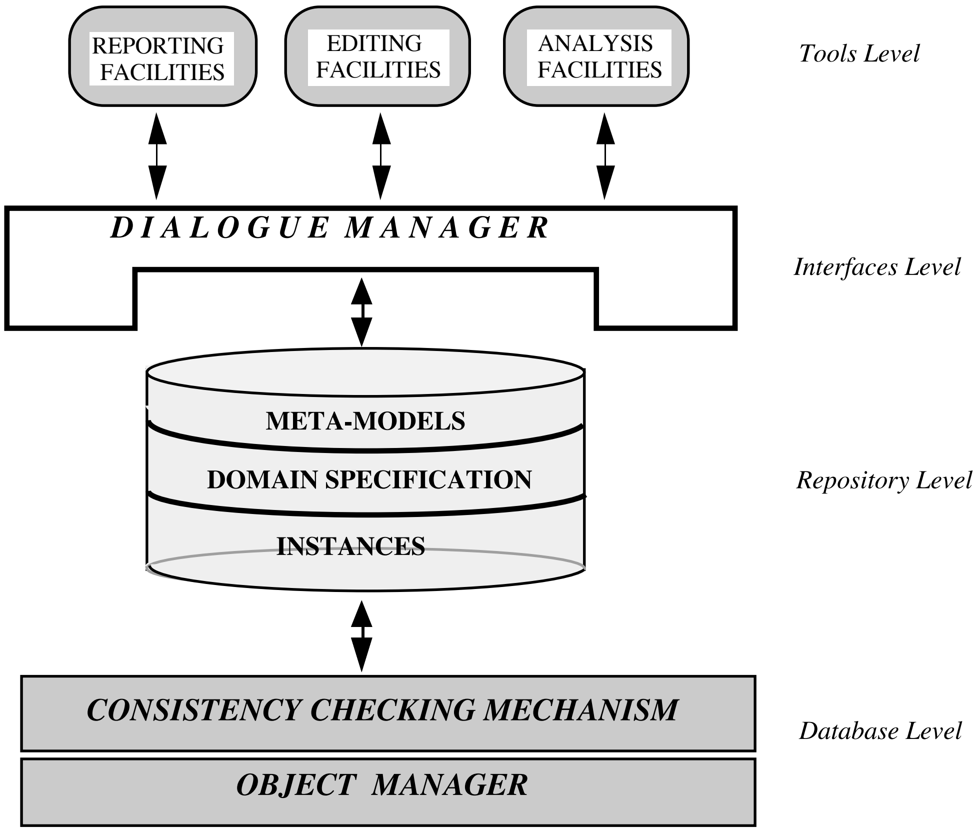

Figure 6.1: Architecture of a CASE tool

According to the architecture of Figure 6.1, a CASE tool consists of the following major components:

- *The repository*. This is usually a database or file system in which all the information about the current status of the development process is held. Depending on the particular phase of software development (i.e. analysis, design, testing etc.)

different types of data are recorded (e.g. diagrams, text, test data etc.) in a CASE repository. Note that a repository is frequently shared amongst different CASE tools. When all the tools are used within the same development phase (e.g. analysis) the repository must be capable of maintaining data about the individual projects that go on concurrently as well as about their inter-communication requirements. A repository that holds data from different development phases (e.g. analysis, design, etc.) must make them accessible from all tools which require them (i.e. analysis data must be accessible by the design tools, design data by the program generation tool etc.).

- *The assistant modules*. These can be considered as tools in their own right, responsible for performing some task within the particular development phase either in an entirely automatic fashion, or by assisting the user of the tool. In the requirements phase, assistant tools can, for example, automatically produce drawings (graphical models) from textual descriptions, check for consistency and completeness of the requirements models, propagate the changes in some of the models to the rest of the models (e.g. redrawing the corresponding graphical models when the user changes some textual descriptions etc.). In the following sections we will distinguish between assistant modules which are responsible for the most mundane/clerical tasks (such as screen drawing) and the ones which are capable of completely or in part undertaking intelligent tasks such as model validation, consistency maintenance etc.

- *The human-computer interface (HCI) component.* This component is responsible for handling the tool's communication with the user. The HCIs for CASE tools folow the general trends in user interface technology. There has been therefore a continuous evolution in the HCI technology employed by CASE tools from the early teletype style of interaction to the latest types of interaction using graphical user interfaces (GUIs) in multitasking environments.

- *The communications component.* The communications component is responsible for exchanging data with other CASE tools. This usually implies that the communications component receives data about the software project which was created in a previous development phase and transmits data for use by tools in subsequent development phases. Effective communication between CASE tools is a problem still to be solved in a satisfactory manner for a variety of reasons. The lack of widely acceptable standards between the CASE tool developers has resulted in a plethora of proprietary and often mutually incompatible formats used by the tools for the internal storage of development information.

6.3.2 Architecture and Functionality of a Repository for Requirements Engineering

The task of a repository is to help manage Requirements Engineering data by offering a variety of services that promote data sharing, data integrity and convenient access. The repository can:

- help users logically associate the various products of the process (documentation, formal specifications etc.)
- keep track of users' annotations which contain explanations and assumptions
- manage different versions of requirements, and the associated documentation
- control different views of the system under development
- help the managerial side of a team development (e.g. estimate required development effort)
- maintain historical data about the decisions taken through the process of eliciting, specifying requirements.

A CASE environment places specific demands on the repository technology used, especially in the case of large-scale team projects. It must support simultaneous access by team members, editing, and authorship in a computer network. Different versions of requirements must coexist, where team members work independently and then merge their specifications back into the main project. Analysts must be allowed to build specific configurations and version trees, and subsequently merge versions together. In summary, a CASE repository supporting Requirements Engineering must exhibit functionality in terms of:

Complex Information Modelling. A repository for Requirements Engineering must be able to store large variable-length objects, such as documents and graphical specification models.

Integrity Constraints and Triggers. Due to the size and complexity of the requirements models, the maintenance of data consistency should be performed by enforcing constraints as the data evolves over time.

Transaction Management. Advanced transaction management features for collaborative Requirements Engineering must be provided. The classical notion of serializability of a transaction is no more adequate, as it significantly reduces concurrency and is largely unsuitable for Requirements Engineering environments at large.

Specification Updates. The process of Requirements Engineering is incremental by nature. It is therefore compulsory to provide the means for changing and updating freely the structure of a preliminary requirements model as modelled by the schema of the repository.

Data Sharing. One of the key issues in collaborative Requirements Engineering is the sharing of data between various analysts. As in client-server architectures, requirements data may be partitioned based on various criteria. However, data must be accessible by all.

Distribution and Cooperative Work. Effective communication protocols between analysts is essential, since they are often unaware of each others developments. This sometimes results in lack of coordination, reduced parallelism, a considerable waste of time and resources, and faulty specifications due to misinterpretations of data. Mechanisms to support the distribution of tools such as distribution of the repository itself or at least distributed access to a repository residing on a database server are needed. In the same spirit, tools for handling and controlling the distribution must also be provided.

Concurrency. The repository should offer the same services as current database systems with respect to the handling of multiple users.

Recovery. In case of hardware or software failures, the system should recover, i.e., bring itself back to some coherent state of the data.

6.3.3 Features of CASE Technology

This section discusses the properties of contemporary CASE tools used in the Requirements Engineering phase. The common characteristic of all such tools is that they are built in order to automate aspects of software methodologies. Most of these methodologies existed before the introduction of CASE, albeit in a manual form. The origin of such methodologies is the paradigm of structured development which was initially applied to programming and subsequently to the design and analysis phases [Jackson, 1983] [Yourdon, 1989]. Structured methodologies are based on the principle of 'divide and conquer' i.e. on the stepwise decomposition of data and functional elements of a system to their parts. This approach ensures the efficient dealing of tasks of arbitrary size since the original task is progressively decomposed into smaller and easier to be dealt with sub-tasks. One serious drawbacks of the structured methodologies (especially when practised without tool support) is that they are usually cumbersome to apply, mainly because of the large amounts of information (documents, drawing etc.) they generate. The most important task of a CASE tool which supports a structured methodology therefore is to provide automated functions for the storage and manipulation of the information generated during software development.

The major application of CASE tools developed in the Seventies and early Eighties therefore, was to assist in the collection and manipulation of the voluminous information generated by the structured methodologies. The progress in the ability of CASE tools to store and manipulate project information, closely followed the progress in areas such as hardware (processor and graphics technologies), databases and computer communications.

Early CASE tools for Requirements Engineering support were handling requirements specifications, either as free text (in which case they were little more than word processors) or in a stylised form which allowed for some limited automatic processing of the requirements. Such automatic processing of requirements specifications usually included checking for *inconsistencies* (e.g. terms with multiple definitions) and *incompleteness* (i.e. terms which are used but not defined).

Advances in hardware technology such as high resolution graphics screens made possible the development of tools capable of manipulating graphical requirements models. The ability to manipulate graphical definitions has given an important boost to the acceptability of the CASE technology since the structured methodologies which the early CASE tools support, rely heavily on the use of graphical requirements models. Even in today's CASE the ability to draw and manipulate graphical specifications models on the screen remains the most heavily used and essential feature.

Technological progress, together with a more mature understanding of the Requirements Engineering (and more general of the software development) process, guided the incorporation of additional functionality in the CASE technology. Today's Requirements Engineering tools utilise the latest advances in processing, graphical user interfaces, database and communications facilities in order to provide effective support to the most fundamental activities of Requirements Engineering. Whilst proprietary tools differ on a number of issues such as the methodology they support, whether they are standalone or parts of an integrated environment etc., they nevertheless share a number of features such as:

- Facilities for prototyping. The importance of prototyping in activities such as requirements acquisition and validation is now generally accepted. The increased use of prototyping as an important technique within Requirements Engineering influenced the CASE developers into incorporating facilities for prototyping in their tools. Basic prototyping facilities usually encountered in CASE tools include report generation and screen painting. More advanced prototyping facilities include animation and symbolic execution of the requirements models. Also limited code generation (which allows for a crude program to be quickly generated from the specifications) is a facility offered by the more sophisticated tools.

- Facilities for data management. Whilst, the early CASE tools used simple file structures for storing the requirements models, contemporary ones utilise database technology in order to offer facilities such as concurrent access to the data by many developers, version control etc.

- Inter-tool Communications facilities. As the relationship between Requirements Engineering and other development phases becomes better understood, so does the ability of requirements CASE tool to communicate with other CASE responsible for tasks such as project planning, configuration control, design etc.

- Graphical user interface. It allows the representation of requirements models using the graphical notations used by the methodology which the tool supports . Usually the tool proves facility for dynamic redrawing of the model on the screen, each time the user changes some part of it. This facility significantly reduces the time it would take to manually redraw the model from scratch.

- Data administration. Managing the data generated during Requirements Engineering entails tasks such as keeping lists of the various types of data, enforcing standards (e.g. that the names of the various concepts follow certain naming conventions), checking for inconsistencies and omissions (e.g. duplicate names).

- Utilities for requirements animation and prototyping. Such utilities may include screen painters, program generators etc. which support the requirements validation process (Chapter 5)

- Data communication facilities. This includes facilities for importing and exporting data to and from other tools. A Requirements Engineering tool usually exports data to design tools and to project planning tools which use such data to inform about the current progress with the project and to plan ahead).

6.4 Selecting, Integrating and Using CASE tools for Requirements Engineering

6.4.1 Desired features of Requirements Engineering CASE

Modern CASE technology integrates the different categories of CASE that were mentioned before into the concept of an *Integrated Software Development environment* (ISDE for short). Very often therefore, these days, the prospective buyer/user of CASE has to choose between different ISDE options rather than the stand-alone CASE which is gradually becoming a rarity. Assuming that the dilemma of choosing between the ISDE and standalone one is circumvented, a number of technical and organisational issues that must be considered in the CASE selection process arises, i.e.:

- Support for specific formalisms and methodologies. The vast majority of commercial CASE support only one or two of the mainstream software methodologies, e.g. Structure Analysis [Yourdon, 1989] Jackson System Development [Jackson, 1983] Structured Systems Analysis and Design Method (SSADM) [NCC, 1990], Information Engineering [Macdonald, 1986] etc. Usually, there is little flexibility for customisation of the models used by the tool to suit the individual users' need. One exception, to this is the class of Generic or Meta-CASE [Alderson, 1993] tools which allow the customisation of existing models or even the creation of entirely new ones to support in-house developed methods.

- Support for specific types of applications. Although, most CASE tools can be used with varying degree of success for any type of application (e.g. data intensive, real time, process control, expert system etc.), some formalisms and methods better accommodate the needs of specific application types. It is essential therefore that, for example, CASE used for real time applications should provide support for formalisms such as StateCharts, Petri Nets etc., whilst data intensive CASE should be able to handle the entity-Relationship model or some of its variants.

- Repository capabilities. The majority of the modern CASE tools provide repository facilities similar to those discussed in Section 3. It is important however that a checklist is drawn by the prospective user which includes the following capabilities: Fully integration of data in the repository, database facilities, project management information, shareability of data amongst developers, automatic report generation and support for ad-hoc querying, import/export capabilities to other repositories, automatic analysis and control of changes in the data.

In the majority of real life software projects, even the most sophisticated CASE or ISDE tool on its own will fail to meet one hundred percent the demands of the requirements phase. One possible way to overcome this is by acquiring and using a set of different CASE with complimentary abilities, e.g. a diagramming tool combined with a prototyping tool and an executable formal specification language. An obvious problem in this approach is cost, since three times as many tools (compared with the one-tool option) will have to be purchased. Even if cost is not an obstacle, communication between the tools is also a potential source of difficulty. In many cases, special software (called *bridges*) will have to be written to allow the transferring of data between tools. Hopefully, this situation will gradually become a rarity as we move towards standard environments and repositories for CASE.

6.4.2 Integrating CASE tools

Integration and communication can be aimed at three levels.

The first level is that of *tools integration.* An example approach for tool integration is presented in Figure 6.2. The tools themselves are responsible for data structuring and control activities; in this way co-operation between tools is achieved through passing of streams of bytes. The advantage of this approach is that tools can be developed almost independently and generic file transfer utilities can be used for the communication between tools. There are however, many disadvantages with this approach such as difficulty in expansion, duplication of role and effort in tools, manual co-operation between tools rather than assistance in a team effort and loss of relationships between design data.

The second level is that of *data integration.* A set of data structures is agreed by all tools within an environment. A meta-schema is agreed upon by all tools before any development of these tolls commences. Any future expansion will need to conform to this meta-schema. The advantage of this approach is that most of the actions necessary for analysing, validating and converting data structures are no longer required within each tool. The disadvantage is that the way that the tools are used is not constrained or guided in any way. This can only be achieved by agreeing not only on the data structures but also on the *process* within which the tools will be used. The interest lies only in the management of information as a consistent whole and not how parts of the information are transformed or operated upon.

The third level is that of *method integration.* A set of data structures and the process model are agreed by all tools which then interact effectively in support of a defined process.

Data integration and method integration imply that the tools communicate by interacting at a level of abstraction higher than the operating system i.e. there is a project level interface which provides facilities and services for controlling the requirements specification process.

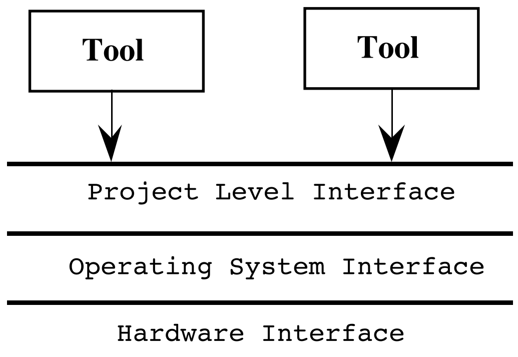

Figure 6.2: Tool Integration

6.5 Research CASE for Requirements Engineering

6.5.1 Introduction

In contrast to commercial CASE, research tools for Requirements Engineering do not fall easily into predefined categories. Generally, CASE research prototypes do not attempt to provide complete or integrated support to the tasks of Requirements Engineering, focusing instead on specific problems such as acquisition, specification, validation etc. Such tools have already been discussed in Chapters 3 and 6.

Research prototypes are usually concerned with proving the validity, feasibility or applicability of a certain paradigm for practising Requirements Engineering. A paradigm, in general, is a specific way of doing things (solving problems) and as such it cannot easily be proved true or false. Research paradigms for Requirements Engineering often materialise as 'tools' or 'environments' which are exclusively used in controlled experiments rather than real-life applications. Eventually, some of the research ideas find their way to real life practice by being incorporated in commercial tools and methodologies.

In recent years, the paradigm that has received the most attention in Requirements Engineering research has been *Knowledge-based Requirements Engineering*. The rationale behind viewing Requirements Engineering as a knowledge-based process has been discussed in many different occasions in previous chapters of this book. Viewing Requirements Engineering as a process which relies to a large extent to the availability of knowledge of various sorts, invokes a number of research issues:

- what types of knowledge is used in Requirements Engineering?
- how can such knowledge be formalised and represented within computers?
- how can the availability of this knowledge improve and sometimes automate the practising of Requirements Engineering?

Research addressing the above questions, has produced tools which can represent and store knowledge about the domain the software application belongs to (*Domain Knowledge*). This knowledge is used for purposes such as completing and validating the requirements model. Reusable requirements knowledge speeds up the overall process (since less interaction with the users is required) as well as improves the quality of the requirements model. Problems that are still to overcome in this approach are related with the difficulty of identifying suitable sources of reusable requirements knowledge and the cost of eliciting and formalising reusable requirements models.

Another source of knowledge which some research tools are based upon is method knowledge. It is fairly easy to automate the steps of a requirements method defined in an algorithmic way which has well defined inputs ad outputs. Structured methodologies fall into this category, i.e. some of their steps and deliverables can be applied mechanistically and are therefore suitable candidates for automation. Unfortunately, this cannot be said for the more unstructured processes such as elicitation and validation. Tools which attempt to (even partially) automate such processes therefore, are equipped with knowledge and methods of reasoning which mimics human knowledge and reasoning. Such tools are frequently called *intelligent requirements assistants* [Anderson and Fickas, 1989] [Reubenstein and Waters 1991]. Again, before such tools acquire commercially applicable a number of important issues related to their ability to stand up to real life software systems requirements must be resolved. Figure 6.3 shows a generic architecture for an 'intelligent' Requirements Engineering CASE tool.

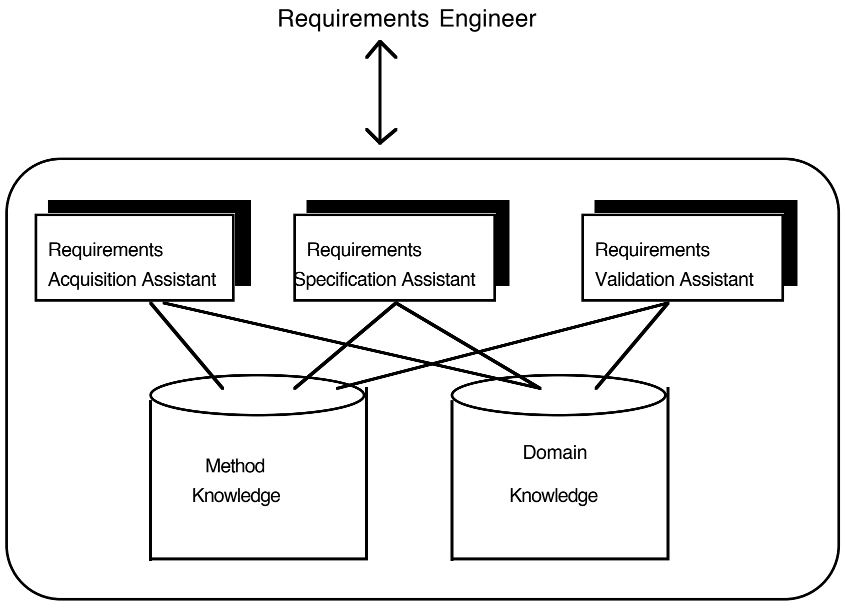
Requirements Engineer
Requirements Acquisition Assistant
Requirements Specification Assistant
Requirements Validation Assistant
Method Knowledge
Domain Knowledge

Figure 6.3: Architecture of an 'Intelligent' Requirements Engineering Tool

6.5.2 Intelligent CASE in Requirements Analysis and Specification

In general, these tools can be distinguished along two dimensions. The first dimension is concerned with tools supporting *building of the specification* whereas the second dimension is concerned with those supporting the *management of the specification and the building of the application out of a given system specification.*

The requirements for the next generation methods and CASE environments that are discussed in this section revolve around the activities of requirements capture and analysis, validation and management of the captured knowledge. These issues give rise to two lines of investigation regarding the requirements for the next generation development methods and CASE environments:

- Improved tools and techniques for assisting the *process of deriving a conceptual specification* of an enterprise.

- Improved tools and techniques for managing a conceptual specification and the building of the application out of a given system specification once such a specification has been developed.

These requirements give rise to a number of research issues discussed in the following sections that aim at providing intelligent support facilities during the process of conceptual specification design, conceptual specification management and the generation of the application itself.

6.5.3 CASE for Conceptual Specification Design

During the last decade, an intensive effort has been made by the industrial community as well as the research community to develop conceptual modelling formalisms that allow to describe Information System (IS) in high level terms, the so-called conceptual schema, and to reason upon this description. However, little effort has been paid to model the process by which one can reach the conceptual schema of an IS to be built. In other words, there exists a plethora of formalisms for the representation of the Requirements Engineering (RE) product whereas the number of techniques which deal with the RE process is very small (see also Chapter 2). As a consequence, CASE tools only concentrate on supporting the *population* of the repository in that they assist the requirements engineer to (1) enter the RE product in a diagrammatic form (2) store its contents in a repository and (3) document it. They do not help the requirements engineer in *constructing* the requirements product itself by supporting the transition process from an informal requirements description to a formal IS specification.

The upperCASE strand of research is strongly influenced by the application of Artificial Intelligence (AI) techniques. The purpose of applying AI techniques is to better understand and consequently, formalise the conceptualisation process. In contrast with the lowerCASE approach, emphasis is placed on the way that requirements are acquired and the way that these are transformed to populate the conceptual schema of the business and the information system.

The system development process is characterised as non-deterministic because of the difficulties in identifying the limits of the problem area, the scope and goals of the information system and the approach to conceptual schema definition. However, from a management point of view, developers wish to control the development process through the use of formal techniques and experimental knowledge.

Taking these two dimensions of the information system development process into account, it is clear that its automation cannot be based solely, on a pure algorithmic solution. This has been recognised by some researchers who developed CASE prototype toolsets based on an expert system approach (SECSI [Bouzeghoub, 1985], OICSI [Rolland, 1986; Cauvet, 1988]). In such an approach, both experimental and formal knowledge are represented in the knowledge base whereas the application domain knowledge is stored within the fact base. The development process is viewed as a knowledge based process which, progressively, through the application of the knowledge base rules on elements of the fact base, transforms the initial requirements into the final information system conceptual schema.

Some approaches to process modelling attempt to employ metamodelling formalisms and toolsets to explicitly represent the knowledge about the development method as well as the development product. The SOCRATES project [Wijers, 1991] follows this approach in that it aims at offering automated support of the information modelling processes by representing and manipulating the experienced practitioners' information modelling knowledge. This approach advocates a model independent architecture where the designer will be able to describe his method and the underlying modelling process using a number of formal languages.

Other approaches, attempt to automatically acquire the knowledge used in the analysis process. Whereas in most of the approaches the development process knowledge is provided by human experts, here it is deduced using automatic learning techniques. For instance, the INTRES tool [Pernici,1989] uses explanation based generalisation for specifying static properties of elements of well understood applications, based on examples of documents. In [Mannino, 1988], the approach is based on the strategy of learning from examples employed in a form definition system, they develop specific learning algorithms. The tool automatically induces the form properties and some functional dependencies. This work has been extended further in [Talldal, 1990] where the learning algorithms are optimised and the induced conceptual schema is expressed in an extended ER formalism.

Summary

This Chapter has been concerned with Computer Aided Software Engineering technology as applied to the Requirements Engineering phase. In the last two decades we witnessed a changing role of CASE in Requirements Engineering from simple clerical task handling to automating essential activities such as formal specification and validation.

Today's State-of-the-Art CASE tool acts as an editor and repository for the various models created during Requirements Engineering (discussed in Chapter 4) as well as an interface between Requirements Engineering and other software development phases. CASE technology particularly suits those Requirements Engineering methodologies which rely on graphical models, mainly because such models can be quickly drawn, manipulated and communicated (i.e. by using techniques such as prototyping, animation etc.) to the users.

The application of CASE technology to Requirements Engineering has been less effective when dealing with more 'hard' problems such as requirements elicitation and formal validation. Research however into intelligent CASE attempts to overcome the limitations of today's tools by utilising expert system technology as well as our improved understanding of the processes involved in Requirements Engineering.

An architecture for a Requirements Engineering CASE tool of the future is presented in Figure 6.4. Such a tool will be capable of performing tasks which are currently the responsibility of the human engineer. These will include:

- automatically completing the requirements model by utilising preexisting reusable requirements knowledge

- advising and supporting the human requirements engineer using domain and method-dependent knowledge and rules

- validating the requirements model using techniques such as natural language paraphrasing, symbolic execution etc.

- generating test cases directly from the requirements.

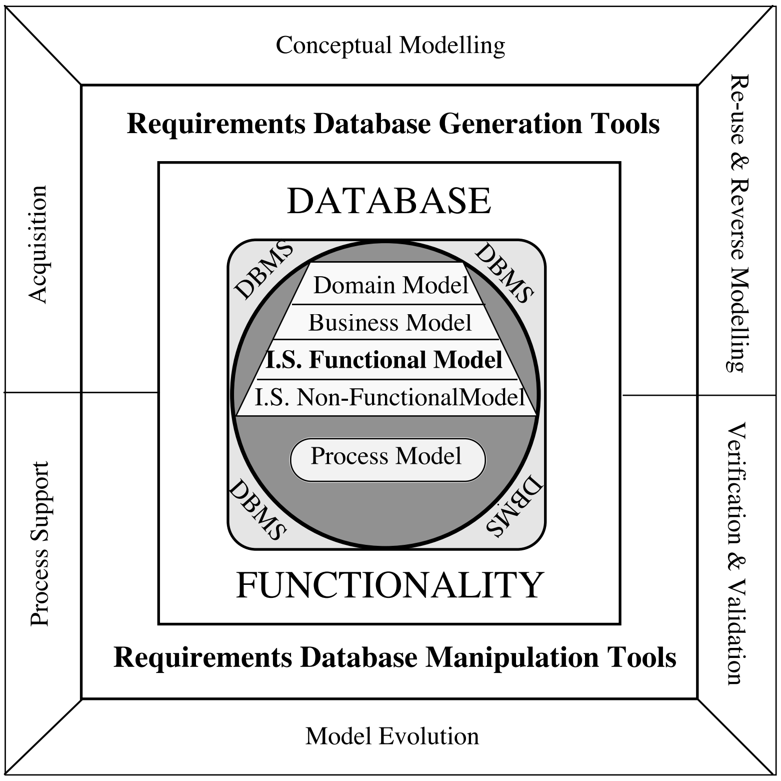

Figure 6.4: An Architecture for the Next Generation of CASE Tools

Such tools might be expected to lead to the CASE environments of the future that will support a very wide range (if not almost all) of stages in systems development with special attention paid to the specification of requirements. A CASE environment will then deal with the following functionality at the Requirements Engineering level:

- *Editing*. Inserting new information in the corporate knowledge base.
- *Browsing*. Viewing objects and sets of objects at various levels of detail.
- *Querying*. Editing and executing queries pertaining to the stored corporate knowledge base.
- *Reporting Facilities*. The reporting facilities range from simple access to data to more elaborated results used in decision support activities.
- *Analysis Facilities*. Analysis facilities will include scenario building and evaluation using different approaches including visualisation techniques.

References

Alderson, A. (1993) *Meta-CASE Technology*. IPSYS Software plc, Macclesfield, Cheshire, UK.

Anderson, J. & Fickas, S. A (1989) *Proposed Perspective Shift: Viewing Specification Design as a Planning Problem*. In Proc. 5th Int'l Workshop on Software Specification and Design, Pittsburgh, PA, IEEE.

Boudier, G., Gallo, F., Minot, R. & Thomas, M. J. (1988) *An Overview of PCTE and PCTE+*. In proc. 3rd ACM Symposium on Software Development Environments, October.

Bouzeghoub, M., Gardarin, G., Metais, E. (1985) *Database Design Tool: an Expert System Approach*, Proc. of the 11th VLDB Conference, Stockolm, August 1985.

Bruce, T. A., Fuller, J., Moriarty, T. (1989) *So You Want a Repository*, Database Programming and Design, May 1989.

Burkhard, D. L. (1989) *Implementing CASE Tools*, Journal of Systems Management, March 1989.

Cauvet, C., Rolland, C., Proix, C. (1988) *Information Systems Design: an Expert System Approach*, in Proc. of the Int. Conf. on Extending Database Technology, Venice, March 1988.

Fuggetta, A. (1993) *A Classification of CASE Technology*, IEEE Computer, Vol. 26, No. 12, 1993, pp. 25-38.

Gibson, M. L., Snyder, C. A. and Carr, H. H. (1989) *CASE: Claryfing Common Misconceptions,* Journal of Information Systems Management, 7(3), 1989.

Jackson, M.A. (1983) System Development, Prentice Hall.

Macdonald, I.G. (1986) *Information Engineering: An Improved, Automated Methodology for the Design of Data Sharing Systems*, in Information Systems Design

Methodologies: Improving the Practice, Olle,T.W et al (eds), IFIP TC8, North-Holland.

Mannino, M. V., Tseng, V. P. (1988) *Inferring Database Requirements from Examples in Forms*, 7th Int. Conf. on Entity-Relationship Approach, pp1-25, Rome, Italy, 1988.

Martin, J. (1989a) Information Engineering: Volume 1, Prentice Hall Inc., New Jersey, 1989.

Martin, J. (1989b) *I-CASE Encyclopedia Brings Consistency to IS*, PC Week, January 1989.

McClure, C. (1988) CASE is Software Automation, Prentice Hall Inc., New Jersey, 1988.

NCC (1990) National Computing Centre. SSADM Version 4 Manual, Manchester, UK.

Pernici, B., Vaccari, G., Villa, R. (1989) *INTRES : INTelligent REquirements Specification,* Proc IJCAI'89 Workshop on Automatic Software Design, Detroit, Michigan, USA, August 1989.

Reubenstein, H. B. & Waters, R. C. (1991) *The Requirements Apprentice:Automated Assistance for Requirements Acquisition.* IEEE Trans. on Software Engineering, Vol. 17, No. 3.

Rolland, C., Proix, C. (1986) *An Expert System Approach to Information System Design*, in IFIP World Congress 86, Dublin, 1986

Talldal B., Wangler B. (1990) *Extracting a Conceptual Model from Examples of Filled in Forms*, Proc. of Int. conf. COMAD, pp 327-350, N. Prakash (ed), New Delhi, India, Dec 1990.

Wijers, G. M., der Hofstede, A. H. M., van Oosterom, N. E. (1991) *Representation of Information Modelling Knowledge*, Proc of the 2nd Workshop on The Next Generation of CASE Tools, Trondheim, Norway, May 1991.

Yourdon, E. (1989) Modern Structured Analysis. Prentice-Hall.