Requirements engineering with viewpoints

by Gerald Kotonya and Ian Sommerville

The requirements engineering process involves a clear understanding of the requirements of the intended system. This includes the services required of the system, the system users, its environment and associated constraints. This process involves the capture, analysis and resolution of many ideas, perspectives and relationships at varying levels of detail. Requirements methods based on global reasoning appear to lack the expressive framework to adequately articulate this distributed requirements *knowledge structure*. The paper describes the problems in trying to establish an adequate and stable set of requirements and proposes a viewpoint-oriented requirements definition (VORD) method as a means of tackling some of these problems. This method structures the requirements engineering process using viewpoints associated with sources of requirements. The paper describes VORD in the light of current viewpoint-oriented requirements approaches and shows how it improves on them. A simple example of a bank auto-teller system is used to demonstrate the method.

1 Introduction

Requirements constitute the earliest phase of the software development life-cycle. They are statements of need intended to convey understanding about a desired result, independent of its actual realisation. The main objective of the requirements engineering process is to provide a model of what is needed in a clear, consistent, precise and unambiguous statement of the problem to be solved. The model is incomplete unless the environment with which the component interacts is also modelled. If the environment is not well understood, it is unlikely that the requirements as specified will reflect the actual needs the component must fulfil. Moreover, as the environment affects the complexity of the component design, constraining the environment can reduce the component complexity.

Studies by Boehm [1, 2] and others have shown that the potential impact of poorly formulated requirements is substantial. Boehm suggested that requirements, specification and design errors are the most numerous in a system, averaging 64% compared to 36% for coding errors. Most of these errors are not found during the development stage but at the testing and delivery stages. The resulting cost to correct these bugs increases with the time lag in finding them. A requirements error found at the requirements stage costs only about one-fifth of what it would if found at the testing stage, and one-fifteenth of what it would cost after the system is in use.

It has been observed that many of the problems of software engineering are difficulties with the requirements specification. It is natural for the developer to interpret an ambiguous requirement so that its realisation is as cheap as possible. Often, however, this is not what the client wants and it usually results in the system being reworked.

Discrepancies between a delivered system and the needs it must fulfil are common and incur very high costs [4]. In some extreme cases, these discrepancies may make the entire system useless. An example of these extreme cases is illustrated by the findings of a survey conducted by the US Government Accounting Office [5]. This survey reviewed nine software development projects that had recently been completed. Although the size of the projects was quite small (the total sum of the nine contracts was $7 million), the findings showed that 47% of the money was spent on software that was never used. 29% of the money was spent on software that was never delivered and 19% resulted in software that was either reworked extensively after delivery or abandoned after delivery but before the GAO study was conducted. The GAO report indicated that of the $317 000 spent on 'successful' projects, some additional modifications were required to be about $198 000 of it, and only $119 000 worth of software could be used as delivered. This implies that less than 2% of the amount spent resulted in software that completely met its requirements.

Requirements fall into two main categories; functional and non-functional [6]. Functional requirements capture the nature of interaction between the component and its environment. Non-functional requirements constrain the

solutions that might be considered. An ideal notation for requirements engineering should cover all aspects of functionality, performance, interface design constraints and the broader context in which the system will be placed [4, 7, 8].

The failure of software to satisfy the real needs of customers is the most visible manifestation of the problems of establishing an adequate set of requirements for a software system. Some of these problems are listed below.

- In most cases, the requirements engineer is not an expert in the application domain being addressed. Many problems in formulating requirements can be traced to misunderstandings on the parts of the requirements engineers and software engineers, and implicit assumptions by potential users.
- There is often inadequate communication between the requirements engineer and the system's potential users due to the differences in their experience and education. Specifically this means that the analyst and the users do not have a common understanding of the terms used [9].
- The notion of 'completeness' in requirements definition is problematic. There is no simple analytical procedure for determining when the users have told the developers everything that they need to know in order to produce the system required.
- Requirements are never stable. Changes in the environment in which the system has to work may change even before the system is installed, due to change in its operational environment.
- Natural languages are often used to describe system requirements. Although they aid users in understanding the system, they have inherent ambiguities that can lead to misinterpretations.
- No one requirements approach or technique can adequately articulate all the needs of a system. More than one specification language may be needed to represent the requirements adequately.
- There is a general lack of *appropriate* tools for supporting the requirements engineering process. There is a need for tools that can help the requirements engineer to collect, structure and formulate requirements in an efficient and consistent manner.

We believe that any requirements engineering method intended to solve these problems must have certain necessary properties. These properties are discussed below.

2 Properties of a requirements engineering method

Requirements reflect the needs of customers and users of a system. They should include a justification for the system, what the system is intended to accomplish, and what design constraints are to be observed.

A *software requirements specification* (SRS) is a document containing a complete description of what the software will do, independent of implementation details. The process of producing the requirements specification, including analysis, is denoted *requirements definition* [10].

The process of eliciting, structuring and formulating requirements may be guided by a requirements engineering method. Notations are associated with the method and provide a means of expressing the requirements. We believe that the following attributes are a necessary part of an effective requirements engineering method.

1. The precision of definition of its notation: this indicates the extent to which requirements may be checked for consistency and correctness using the notation. Imprecise notations may lead to errors and misunderstanding. It should be possible to check the requirements both internally and against a description of the real world.
2. Suitability for agreement with the end-user: this indicates the extent to which the notation is understandable (as opposed to 'writeable') by someone without formal training. A problem with formally expressed specifications and their notations is that they cannot be easily understood without special training. One solution to this problem may be to integrate both informal and formal descriptions of the system requirements.
3. Assistance with formulating requirements: this can be viewed in terms of two aspects:

   - how the notation organises the requirements *knowledge structure* for the system; understanding a system, the services required of it and its environment involves the capture, analysis and resolution of many ideas, perspectives and relationships at varying levels of detail; the requirements definition process should be guided by a problem analysis techniques that takes all these viewpoints and their requirements into account.
   - how the notation affords the separation of concerns; ideally, this means that readers of the requirements specification should need to find only those parts of the requirements specification that are relevant to their own area of interest.

4. Definition of the system's environment: the requirements model is incomplete unless the environment is modelled with which the component interacts. If the environment is not well understood, it is unlikely that the requirements as specified will reflect the actual needs the component must fulfil.
5. The scope for evolution: it must be recognised that requirements are built gradually over long periods of time and continue to evolve throughout the component's lifecycle. The specification must be tolerant of temporary incompleteness and adapt to changes in the nature of the needs being satisfied by the component. In essence, whatever the method or approach used to formulate the requirements, it must be able to accommodate changes without the need to rework the entire set of requirements.
6. Scope for integration: this can be viewed in terms of requirements approaches and types of requirements:

   - There is no single requirements approach that can adequately articulate all the requirements of a system both from the developers' and the users' viewpoints; for example, a data-flow model does not adequately reflect control requirements of a system and a formal language may not be able to express non-functional requirements properly.
   - non-functional requirements tend to be related to one or more functional requirements; expressing functional and non-functional requirements separately obscures the correspondence between them, whereas

stating them together may make it difficult to separate the functional and non-functional considerations.

7. Scope for communication: the requirements process is a human endeavour, and so the requirements method or tool must be able to support the need for people to communicate their ideas and obtain feedback.
8. Tool support: although notations and methods can provide much conceptual help with the process of defining requirements, it is their incorporation into, or support by, tools which makes the biggest contribution to improving our ability to manage complexity on large projects. Tools impose consistency and efficiency on the requirements process. It lets the requirements engineer collect, structure and formulate requirements in an efficient and consistent manner.

It is probably impossible for a single requirements engineering method to completely satisfy all of the above requirements. Method designers, however, should be aware of these desirable properties and should make explicit decisions about which are most important to them.

3 Viewpoints for requirements definition

The notion of viewpoints as a means of organising and structuring the requirements engineering activity is now well known. Viewpoints are implicitly present in SADT [11] and were first made explicit in the CORE method [12]. Since then there have been various other viewpoint-based approaches and proposals [13–16]. We have summarised these methods and described our own work on viewpoints for interactive system design elsewhere [17].

In our initial work, the model adopted for viewpoints was a service-oriented model, where viewpoints are analogous to clients in a client-server system. The system delivers services to viewpoints, and the viewpoints pass control information and associated parameters to the system. Viewpoints map to classes of end-users of a system or with other systems interfaced to it.

This approach can be used to support a user-centred design process [18]. Like user-centred design, it tends to focus the RE process on the user issues rather than organisational concerns. This leads to incomplete system requirements. To allow organisational requirements and concerns to be taken into account, we have extended the concept of viewpoints to consider other inputs apart from direct clients of the system. Viewpoints fall into two classes:

1. *Direct viewpoints:* these correspond directly to clients, in that they receive services from the system and send control information and data to the system. Direct viewpoints are either system operators/users or other subsystems which are interfaced to the system being analysed.
2. *Indirect viewpoints:* indirect viewpoints have an 'interest' in some or all of the services which are delivered by the system but do not interact directly with it. Indirect viewpoints may generate requirements which constrain the services delivered to direct viewpoints.

Although the concept of a direct viewpoint is fairly clear, the notion of indirect viewpoints is necessarily diffuse. Indirect viewpoints vary radically, from engineering viewpoints (i.e. those concerned with the system design and implementation) through organisational viewpoints (those concerned with the system's influence on the organisation) to external viewpoints (those concerned with the system's influence on the outside environment). Therefore, if we take a simple example of a bank teller system, some indirect viewpoints might be

- ☐ a security viewpoint concerned with general issues of transaction security.
- ☐ a systems planning viewpoint concerned with future delivery of banking services.
- ☐ a trade-union viewpoint concerned with the effects of the system introduction on staffing levels and bank staff duties.

Indirect viewpoints are very important as they often have significant influence within an organisation. If their requirements go unrecognised, they often decide that the system should be abandoned or significantly changed after delivery. This is particularly true for some classes of safety-related systems which must be certified by an external regulator. If certification requirements are not met, the system will not be allowed to go into service.

Note that the notion of viewpoint which we have adopted is distinct from the ideas used in other methods of requirements engineering, although it has something in common with Greenspan's SOS approach [19] and the requirements elicitation approach proposed by Leite [14]. In methods such as SADT and in the practical application of CORE, viewpoints are seen as sources or sinks of data flows. In the VOSE method [15], viewpoints are akin to what we would call engineering viewpoints; they recognise that there are many system models used by different engineers involved in system specification and design. These models often conflict, and the method proposed is geared to exposing and reconciling these conflicts.

Fig. 1 summarises the notion of viewpoints advanced by current approaches. Several features are summarised. Fig. 1 looks at whether some form of classifying mechanism is adopted in structuring viewpoints; we believe this is important as viewpoints may have similar characteristics but differing requirements. It also summarises viewpoint orientation adopted by these methods. Most approaches have an intuitive notion of a viewpoint and do not extend the viewpoint analysis beyond the data sink/source orientation.

Functional requirements do not exist in isolation. They are related to other requirements of the system, for example, non-functional requirements and control requirements. There is a need in a requirements method to provide a basis for integrating these requirements to expose this correspondence [17]. Fig. 1 shows that this kind of broad integration is lacking in most of the viewpoint-oriented methods. This is particularly true of the integration of functional and non-functional requirements. It is also useful if specifications can be expressed in several different representations. This aids the understanding of the requirements and promotes communication between the user and the software developers.

More than one specification may be needed to represent the requirements adequately. Fig. 1 shows that only VOSE and Leite's approach support multiple representations. We

feature	approach				
	SRD	SADT [11]	CORE [12]	VOSE [15]	Leite [14]
notation of viewpoint	intuitive	intuitive	weak	defined	defined
viewpoint classification	no	no	no	no	yes
viewpoint orientation	data sink/source	data sink/source	process	role/responsibility	role
integration of functional and non-functional requirements	no	no	no	no	no
provision for multiple representations	no	no	no	yes	no
support for event scenarios/control requirements	no	yes	yes	not explicit	no
support for object-oriented development	no	no	no	not explicit	no
support for indirect viewpoints	no	no	limited	no	no
tool support	no	yes	limited	yes	yes

Fig. 1 Summary of current viewpoint approaches

have already discussed the notion of an *indirect* viewpoint; we believe the requirements engineering process is incomplete unless these viewpoints are considered. With the exception of CORE, this notion is largely lacking in the methods discussed. The CORE notion of an indirect viewpoint is synonymous with an external entity that provides inputs to processes and receives outputs from processes. However, CORE focuses its main analysis on defining viewpoints, which are processes that transform the inputs to outputs. Each defining viewpoint forms the basis for further decomposition.

3.1 VORD viewpoints

Many viewpoint-oriented approaches consider viewpoints as data sinks or sources, sub-system processes or internal perspectives. Our proposed notion of viewpoint is based on the entities whose requirements are responsible for, or may constrain, the development of the intended system. These *requirements sources* comprise the end-users, stake-holders, systems interfacing with the proposed system and other entities in the environment of the intended system that may be affected by its operation. Each requirements source (*viewpoint*) has a relationship with the proposed system based on its needs and interactions with the system. It is therefore important that the techniques used should adequately capture and organise not only global, but also the specific requirements of the different viewpoints into a cohesive knowledge structure that is both complete and visible. Fig. 2 shows our proposed viewpoint structure. The notion is discussed below.

4 Viewpoints-oriented requirements definition (VORD)

Based on the foregoing notion of viewpoints, we have developed a method for requirements engineering called VORD (Viewpoint-Oriented Requirements Definition) which covers the RE (requirements engineering) process from initial requirements discovery through to detailed system modelling. For the purposes of this paper, the latter modelling stages of the method are not important. This dis-

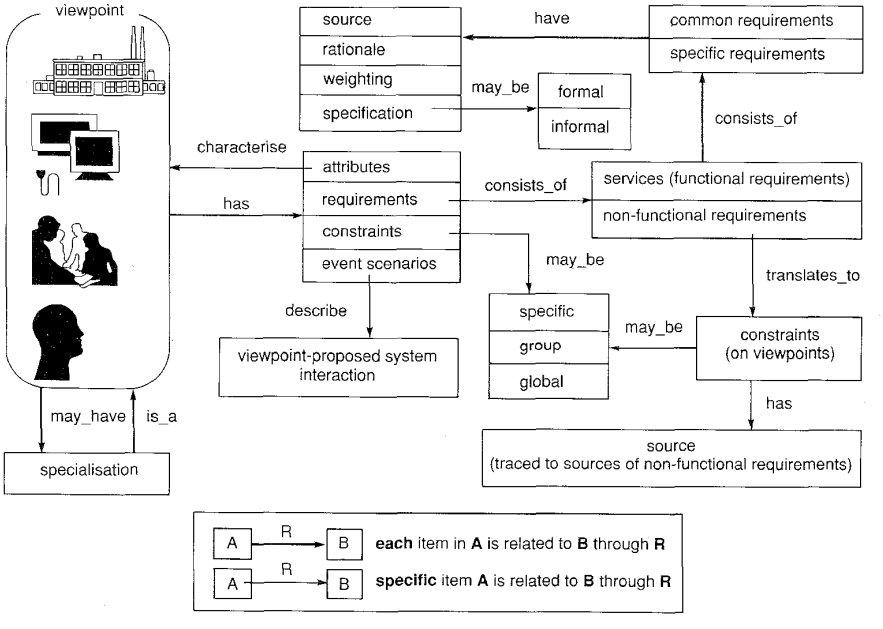

Fig. 2 VORD viewpoint and information structure

cussion therefore concentrates on the first three iterative steps in VORD:

- viewpoint identification and structuring.
- viewpoint documentation.
- viewpoint requirements analysis and specification.

Fig. 3 shows the VORD process model. The first step, viewpoint identification and structuring, is concerned with identifying relevant viewpoints in the problem domain and structuring them. The starting point for viewpoint identification is with abstract statements organisational needs and abstract viewpoint classes. This step is described in Section 4.2.

The second step is concerned with documenting the viewpoints identifed in the first step. Viewpoint documentation consists of documenting the viewpoint name, requirements, constraints on its requirements and its requirements source. Viewpoint requirements comprise a set of required services, control requirements and set of non-functional requirements. This step is described in Section 4.3.

The last step is concerned with specifying the functional and non-functional viewpoint requirements in an appropriate form. The notation used depends on the viewpoint, the requirements and requirements source associated with the viewpoint. Appropriate notations range from natural language (if the requirements source is concerned with non-technical requirements), through equations (e.g. if the requirements source is a physicist), to system models expressed in formal or structured notations.

Viewpoints and their requirements are collected into a central repository that serves as input to the requirements analysis process. The objective of the analysis process is to establish the correctness of the documentation and to expose conflicting requirements across all viewpoints.

4.1 ATM example

We use the example of an automated teller machine (ATM) to illustrate the VORD process model. The ATM contains an embedded software system to drive the machine hardware and to communicate with the bank's customer database. The system accepts customer requests and produces cash, account information, provides for limited message passing and funds transfer. The ATM is also required to make provisions for major classes of customers, the home customer and foreign customer. Home customers are defined as people with accounts in any of the branches of the bank to which the ATM

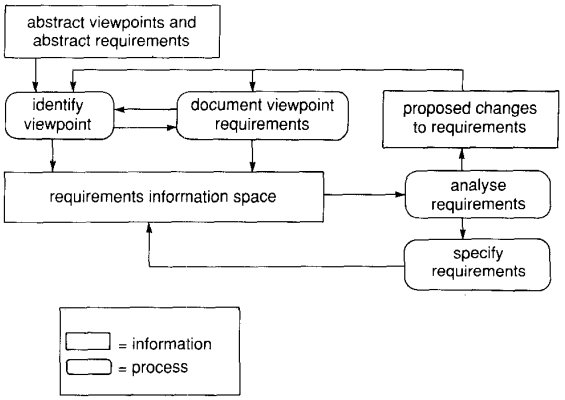

Fig. 3 VORD process model

belongs. These customers receive all the services provided by the ATM. Foreign customers are people with accounts in other banks affiliated to the bank concerned. Apart from providing services to its users, the ATM is also required to update the customer account database each time there is a cash withdrawal or funds transfer.

All the services provided by the ATM are subject to certain conditions, which can be considered at different levels. The top level sets out conditions necessary for accessing the services. These include a valid ATM cash-card and correct personal identification number (PIN). The level is concerned with service requests and is subject to the availability of particular services. Beyond this level, all services provided by the ATM are subject to specific conditions set out for their provision.

4.2 Viewpoint identification

All structured methods must address the basic difficulty of identifying the relevant problem domain entities for the system being specified or designed. The majority of methods provide little or no active guidance for this, and rely on the method user's judgement and experience in identifying these entities. We cannot claim that we have solved the problem of identifying relevant problem domain entities. However, our method provides some help to the analyst in the critical step of viewpoint identification.

The process of understanding the system under analysis, its environment, requirements and constraints places a lot of reliance on the 'system authorities'. These are people or documents with an interest in or specialist knowledge of the application domain. They include system

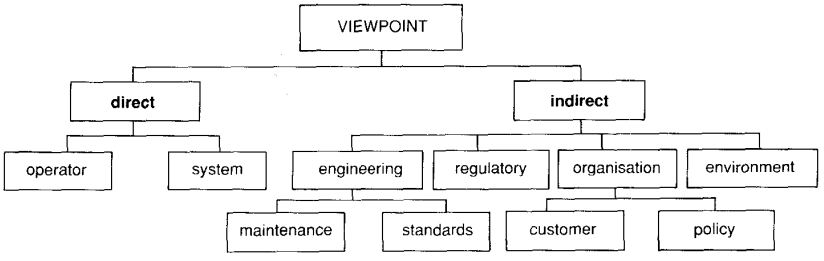

Fig. 4 Abstract viewpoint classes

end-users, system procurers, system engineers and documentation of existing system(s).

We have generalised these 'system authorities' into a set of viewpoint classes, which can be used as a starting point for finding viewpoints specific to the problem domain. Fig. 4 shows part of the tree diagram of the abstract viewpoint classes. Normally, the indirect viewpoints would be decomposed to greater depth than shown here. The organisation viewpoint, for example, would have policy, customer and training viewpoints as sub-classes, and the environment viewpoint may have people and others systems in the environment.

The root of the tree represents the generation notion of a viewpoint. Information can be inherited by viewpoint sub-classes, and so global requirements are represented in the more abstract classes and inherited by sub-classes. In the direct viewpoint class, the sub-system viewpoint represents the abstract class of systems within an organisation that may interact directly with the proposed system. These include shared databases and other sub-systems. The operator class represents the abstract class of people who will interact with the system directly.

Under the indirect viewpoint class, the customer viewpoint represents the requirements and policy of the organisation which is purchasing the system, the regulatory viewpoint represents legal and regulatory requirements associated with the system, the engineering viewpoint represents the engineering requirements for the system, and the environment viewpoint represents environment issues affecting the system development.

Of course, this class hierarchy is not generic. Each organisation must establish its own hierarchy of viewpoint classes based on its needs and the application domain of the systems which it develops. The information encapsulated in this class hierarchy is an important organisational resource.

The method of viewpoint identification that we propose involves a number of stages.

☐ Prune the viewpoint class hierarchy to eliminate viewpoint classes that are not relevant for the specific system being specified. In the ATM example, let us assume that there is no external certification authority and no environmental effect. We therefore do not need to look for viewpoints under these headings.

☐ Consider the system stake-holders, i.e. those people who will be affected by the introduction of the system. If these stake-holders fall into classes which are not part of the organisational class hierarchy, add these classes to it.

☐ Using a model of the system architecture, identify sub-system viewpoints. This model may either be derived from the existing models or may have to be developed as part of the RE process. In the ATM example, we can identify one main sub-system, the *customer database*. We note that architectural models of systems almost always exist because new systems must be integrated with existing organisational systems.

☐ Identify system operators who use the system on a regular basis, who use the system on an occasional basis and who request others to use the system for them. All of these are potential viewpoints. We can identify three instances of direct viewpoint in this example; the *bank customer* (regular), *ATM operator* (occasional), the *bank manager* (occasional).

☐ For each indirect viewpoint class that has been identified, consider the roles of the principal individual who might be associated with that class. For example, under the viewpoint class 'customer', we might be interested in the roles of 'regulations officer', 'maintenance manager', 'operations manager' etc. There are often viewpoints associated with these roles. In the ATM example, there are many possible indirect viewpoints but we confine our analysis to a security officer, a system developer and a bank policy viewpoint.

Based on this approach, the viewpoints that might be considered when developing an ATM specification are shown in Fig. 5. Home customer and foreign customer viewpoints are specialisations of the customer viewpoint and as such inherit its requirements and attributes. Likewise the bank manager, bank teller and ATM operator viewpoints are specialisations of bank employee.

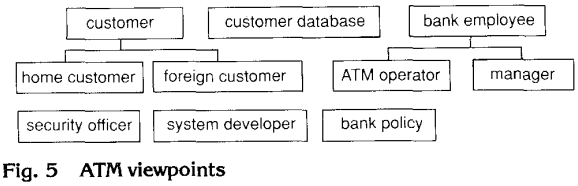

Fig. 5 ATM viewpoints

4.3 Documenting viewpoint requirements

Viewpoints have an associated set of requirements, sources and constraints. Viewpoint requirements are made up of a set of services (functional requirements), a set of non-functional requirements and control requirements. Control requirements describe the sequence of events involved in the interchange of information between a direct viewpoint and the intended system. Constraints describe how a viewpoint's requirements are affected by non-functional requirements defined by other viewpoints.

We do not have space to look at the detailed requirements of each viewpoint here. However, Fig. 6 shows examples of initial requirements which might apply to an auto-teller system. The ATM operator and customer database viewpoints are concerned with providing control information to the proposed system. The ATM operator is concerned with stocking the ATM with cash and starting and stopping its operation. The operator needs to be alerted whenever the cash dispenser is empty. The customer database stores the customer account information which is used by the system to process transactions.

Each viewpoint has an associated template, which is a collection of structured forms for documenting detailed viewpoint requirements. This template includes

- the requirements associated with the viewpoint; these may be either functional or non-functional requirements.
- associated sources for viewpoint requirements.
- a rationale for the proposed requirement
- constraints on viewpoint requirements and their sources
- viewpoint events; viewpoint events describe the interaction between the viewpoint and the intended system in terms of viewpoint events, system responses and exceptions.

viewpoint	service	non-functional requirements
bank manager	transaction reports	1. reports must be provided on a daily basis 2. reports should comprise the account name, transaction, date and time 3. failure rate of this service should not exceed 1 in 1000 requests 4. system must be operational within 6 months
ATM operator	operator paging	1. failure rate of this service should not exceed 1 in 10 000 attempts
home customer	1. cash withdrawal 2. balance enquiry 3. funds transfer 4. message passing 5. last five transactions	1. cash withdrawal service should be available 999/1000 requests 2. cash withdrawal service should have a response time of no more than 1 minute 3. cash withdrawal service should permit withdrawal in a choice of denominations 4. balance enquiry should not fail more than 1 in 1000 requests 5. funds transfer service should be reliable with a maximum failure rate of no more than 1 in 100 000 attempts 6. message passing should include request for cheque books and complaints on erroneous cash withdrawals
customer database		
foreign customer	1. cash withdrawal 2. balance enquiry	
security officer		1. all system security risks must be explicitly identified, analysed and minimised according to the ALARP principle 2. bank standard encryption algorithms must be used 3. system must print paper record of all transactions
system developer		system must be developed using standards defined in 'System Quality Plan xxx'
bank policy		1. cash withdrawal service should be available for 9 out of 10 requests 2. cash withdrawal service should have a response time of no more than 2 minutes 3. balance enquiry service should not have a failure rate of more than 1 in 50 requests

Fig. 6 Initial distilled list of viewpoint requirements

Certain items of the template are optional and need not have entries for all viewpoints. For example, an indirect viewpoint such as a government regulating body may not require services from the intended system, but may have certain non-functional requirements which place constraints on the system.

4.3.1 Viewpoint templates: as we do not have space to develop the complete requirements analysis for ATM here, we confine our analysis to two viewpoints; the *home customer* and *foreign customer*. For the most part our example is the *cash withdrawal* service. We believe this particular service has sufficient diversity associated with it in terms of usage and constraints to adequately demonstrate the usefulness of our approach. Fig. 7 shows the general viewpoint template for the customer viewpoint. The customer viewpoint represents the most abstract description of the home and foreign customer classes. Attributes and services described at the customer level are inherited by its two specialisations. The service template illustrates the provision of the *cash withdrawal* service to the home customer viewpoint.

Fig. 8 shows the detailed viewpoint template for the home customer and foreign customer viewpoints in relation to the cash withdrawal service. Event scenarios and service specifications are described in Sections 4.3.2. It is important to note that VORD provides the user with a framework for formulating very detailed requirements specification, yet maintains a clear separation of concerns. For example, the cash withdrawal service (Fig. 8) is intended for both the home and foreign customer viewpoints, but the source of, rationale for and constraints on the service differ in each instance. In the case of the home customer, the source of the requirement is the viewpoint itself, whereas in the case of the foreign customer, the source of the requirement is the bank policy viewpoint. Similar constraints (e.g. availability) on the service are less stringent for the foreign customer than for the home customer viewpoint. The template also shows that certain constraints can be specific, whereas others can be group or global con-

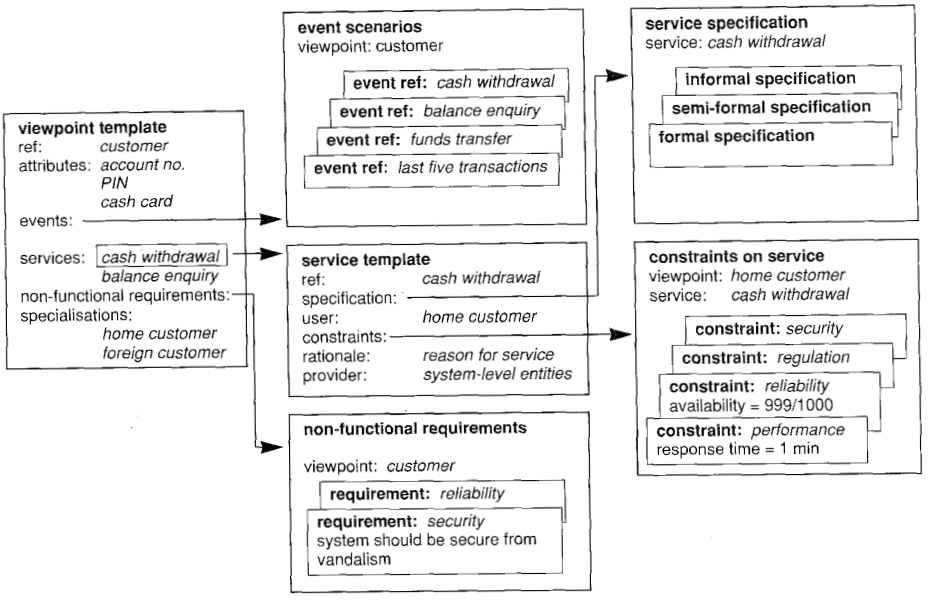

Fig. 7 Structure of ATM customer and service distribution

straints. Group constraints are associated with constraints affecting similar requirements across several viewpoints. Global constraints affect all system requirements.

4.3.2 Event scenarios and control: control requirements define how a system controls its environment and how the environment controls the system. Control influences occur not only between the environment and the system, but also between the elements of the environment themselves. Control relationships between the proposed system and its environment are largely due to the need to conform to, enforce or assist control relationships between elements of the environment. In essence, control can be viewed as a distributed layered process through levels of the environment culminating in the system as the service provider at the lowest level.

Several models have been proposed for extending current requirements definition approaches to incorporate control requirements. These models are typified, in structured analysis, by the Ward–Mellor [20] and Hartley–Pirbhai [21] extensions to the basic structured analysis notation, and in object-oriented analysis by the Rumbaugh dynamic model [22], the Shlaer–Mellor object state model [23], and Harel statecharts [24]. These models offer insights into the representation of control requirements.

The provision of a viewpoint service is the culmination of a series of events arising from the viewpoint layer and filtering through levels of control to entities that are ultimately responsible for its provision. Fig. 9 illustrates a simple event trace diagram involving a single viewpoint. Normally the provision of a service involves the participation of several viewpoints; each bringing its control influences to bear on the service. It is important in documenting a viewpoint service to identify other viewpoints affecting or participating in the provision of a service. This provides a means of tracing the impact of later modification in the requirements of one viewpoint on others.

Viewpoint events are a reflection of the control requirements as perceived by the user. System-level events reflect the realisation of control at the system level. Distinguishing between the two levels of control provides us with a mechanism for

- addressing control requirements from the user perspective.
- tracing system-level control to viewpoints.
- exposing conflicting control requirements.
- capturing the distributed and layered nature of control.

We have devised a simple mechanism to do the above, based on event scenarios. An event scenario is defined as a sequence of events together with exceptions that may arise during the exchange of information between the viewpoint and the system. A normal sequence of events may have exceptions at various points in the event sequence. At the system level, exceptions cause a transfer of control to exception-handlers. As exception-handlers describe alternative courses of action, they are treated as separate event scenarios. The top layer of an event scenario is referred to as the *normal scenario*; it represents the 'normal' sequence of events. Event scenarios can therefore be thought of as layered, with each subsequent layer comprising events that describe exceptions in the previous layer.

There are three steps in describing viewpoint events:

- describing isolated viewpoint event scenarios.
- tracing events and predicates to viewpoints.
- identifying viewpoints participating in a service provision.

Fig. 10 shows the event scenario associated with the service *cash withdrawal* of the customer viewpoint. The normal event scenario is shown in bold transitions. We use an extended state transition model, based on a model proposed by Rumbaugh [22], to represents the event scenarios.

viewpoint: *home customer*
service: *cash withdrawal*
source: *home customer*
weighting: *essential*

rationale

- to provide customers with the convenience of 24 hour cash withdrawal from any branch of the bank.
- to cut down on the paper work associated with withdrawals from inside the bank.

constraints

1.	**reference:**	reliability
	type:	availability
	definition:	availability $\geqslant$ 999/1000
	assignment:	specific
	source:	home customer
	weighting:	essential
2.	**reference:**	performance
	type:	response time
	definition:	response time $\leqslant$ 1 minute
	assignment:	specific
	source:	home customer
	weighting:	significant
3.	**reference:**	currency
	type:	currency selection
	definition:	ability to select several denominations
	assignment:	specific
	source:	home customer
	weighting:	moderate
4.	**reference:**	security
	type:	security risks
	definition:	security risks must be minimised
	assignment:	global
	source:	security officer
	weighting:	essential
5.	**reference:**	deadline
	type:	delivery time
	definition:	delivery time $\leqslant$ 6 months
	assignment:	group
	source:	bank manager
	weighting:	significant

event scenarios
described in another section

specification
described in another section

viewpoint: *foreign customer*
service: *cash withdrawal*
source: *bank policy*
weighting: *essential*

rationale

- to provide customers of banks affiliated to the home bank with the convenience of obtaining cash from a wide range of ATMs.

constraints

1.	**reference:**	reliability
	type:	availability
	definition:	availability = 900/1000
	assignment:	specific
	source:	bank policy
	weighting:	significant
2.	**reference:**	performance
	type:	response time
	definition:	response time $\leqslant$ 2 minute
	assignment:	specific
	source:	bank policy
	weighting:	significant
3.	**reference:**	security
	type:	security risks
	definition:	security risks must be minimised
	assignment:	global
	source:	security officer
	weighting:	essential
4.	**reference:**	deadline
	type:	delivery time
	definition:	delivery time $\leqslant$ 6 months
	assignment:	global
	source:	bank manager
	weighting:	significant

event scenarios
described in another section

specification
described in another section

Fig. 8 Viewpoint template for the home and foreign customers

Each transition has a triggering event, preconditions which must be satisfied before that transition can take place and actions which are associated with the transition.

Tracing events to viewpoints is usually straightforward. In most cases, the events can be traced to the viewpoint requesting the service. For example, the insert (card) event in an ATM is associated with initiating all customer services, and so is traced to the customer viewpoint. The preconditions can be traced to viewpoints by analysing the various states of the predicate variable (left-hand side of Fig. 10) to determine whether any external events are associated with causing the transitions. If no external events are involved, the variable is probably an internally generated value and may be traced to a database or data dictionary.

4.3.3 Service specification: the orientation of a service makes it easy to specify it using a variety of notations. We consider this important for two reasons.

□ One of the major problems associated with software development is a lack of adequate communication

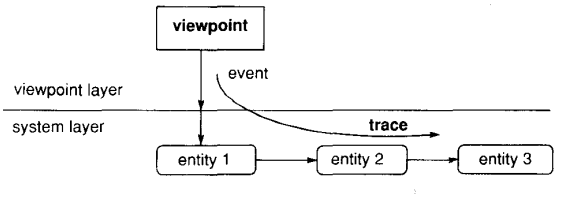

Fig. 9 Simple event trace diagrams

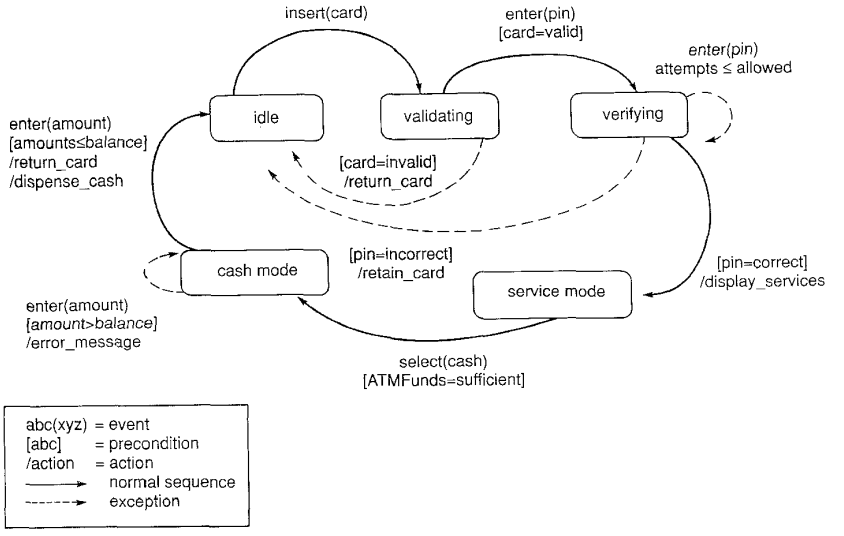

Fig. 10 Event scenario for cash withdrawal

between the requirements engineer and the system's potential users due to the differences in their experience and education. The ability to represent the same requirement in different notations that are familiar to different people enhances communication and aids understanding.

□ No single requirements notation can adequately articulate all the needs of a system. More than one specification language may be needed to represent the requirement adequately.

This aspect of a service provides us with a basis for replicating approaches such as VOSE [15], whose notion of a viewpoint is associated with different representation schemes.

We illustrate these aspects of a service by specifying a simplified version of the ATM's cash withdrawal service using a formal and informal notation. We use a simplified form of the formal notation Z and an informal notation to specify the service. In both cases, we assume that the customer has a valid cash card and has entered the correct personal identification number (PIN).

Fig. 11 shows an informal specification of the cash withdrawal service.

There are clearly a number of ambiguities in this description, but it is expressed at a level which could easily be understood by non-technical staff. A more precise specification can be developed and linked to this informal description (as shown in Fig. 7). Of course, we recognise that the problem with multiple representations of a service is the demonstration that these representations are equivalent. We have tried to address this problem. Finkelstein *et al.* [15] have identified a comparable problem, the VOSE approach, and discuss methods of equivalence demonstration.

```
Customer requests cash withdrawal
if any of the following conditions is true refuse withdrawal:
   condition1: The requested amount exceeds customer balance.
   condition2: The funds in ATM are less than request amount
else do the following:
   dispense cash
   update customer account
endif
```

Fig. 11 Informal specification of simplified cash withdrawal service

More precisely, the cash withdrawal service can be specified as a disjunction (OR) of two Z schemas; *PermitWithdrawal* and *RefuseWithdrawal* (Fig. 12). This is based on the following free types:

$$FundStatus ::= adequate \mid inAdequate$$
$$AccountStatus ::= overdrawn \mid goodStanding$$
$$criticalLevel = 1000$$
$$accountNumber: 0..10^6$$

FundStatus represents the stock of the ATM funds. An *inAdequate status* indicates that the ATM funds have fallen below 1000, represented by *criticalLevel*. AccountStatus represents the status of the customer account.

For a cash withdrawal to be permitted (Fig. 13), two conditions must be fulfilled.

- The customer account must be in good standing (i.e. not overdrawn).
- The ATM must contain adequate funds.

After a cash withdrawal, the customer account is updated. This is illustrated in the separate specification of PermitWithdrawal and RefuseWithdrawal.

5 Viewpoint analysis

The purpose of viewpoint analysis is to establish that viewpoint requirements are correct and 'complete'. There are two stages to this analysis.

CashWithdrawal

$$PermitWithdrawal \vee RefuseWithdrawal$$

Fig. 12 Specification for cash withdrawal

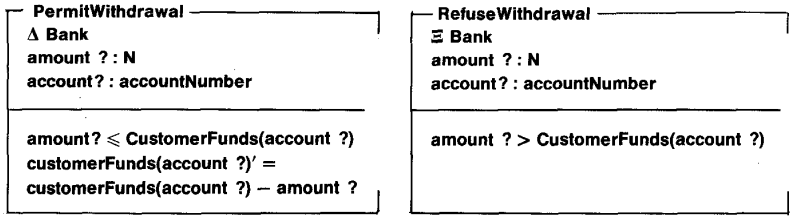

Fig. 13 Specification of PermitWithdrawal and RefuseWithdrawal

☐ correctness of viewpoint documentation; the viewpoint documentation must be checked to ensure that it is consistent and that there are no omitted sections.
☐ conflict analysis; conflicting requirements from different viewpoints must be exposed.

Analysis is a complex subject, which we cannot discuss in detail here. Frankly, we are sceptical about the usefulness of automated semantic analysis. Conflict analysis in VORD is performed by the requirements engineer with the help of the toolset. The VORD toolset has provisions for detecting a number of conflicting requirements and generating reports. Our checking facilities are therefore based on ensuring that information can be presented to the requirements engineer in such a way that manual analysis is simplified. We briefly describe the support for analysis below.

5.1 Viewpoint documentation checking

Checking the correctness of a viewpoint documentation involves verifying that it has been correctly entered and it is complete. We have defined a viewpoint as an entity consisting of a set of attributes, requirements, constraints and even scenarios. Although all viewpoints have attributes that characterise them, other viewpoint information may be omitted, depending on whether the viewpoint is a direct or indirect viewpoint. For example, an indirect viewpoint does not receive services or provide control information to the system, but may have non-functional requirements. Direct viewpoints may receive services, have non-functional requirements or provide control information to the system. Both classes of viewpoints may have constraints.

A detailed description of the viewpoint template, its contents and their inter-relationships is provided in Section 3.1. We have built into the VORD toolset a mechanism to analyse all these aspects of the viewpoint template for completeness and correctness.

5.2 Conflict analysis

Viewpoints have differing stakes in and interactions with the intended system and have requirements that are closely aligned with these interests. Conflicts may arise from contradictions among individual viewpoint requirements. Some related work in this area includes the work on domain-independent conflict resolution by EasterBrook [25] and the work on rule-based software quality engineering by Hausen [26].

In Section 2, we discussed how non-functional requirements tend to conflict and interact with the other system requirements. This kind of conflict may be quite specific, as in the following two cases:

- where the provision of a service across viewpoints is associated with different constraints of the same general type; for example, a conflict is reported in the case where the reliability of a service is specified in terms of its availability in one viewpoint, and in terms of its probability of failure on demand (POFOD) in another viewpoint.
- where the provision of a service across viewpoints is associated with similar constraints, but differing constraint values; for example, a conflict is reported in the case where the reliability of a service, specified in terms of its availability, has a value of 999/1000 in one viewpoint and a value of 900/1000 in another viewpoint.

It may also be the case that a requirement in one viewpoint contradicts a requirement in another viewpoint; for example, the security officer viewpoint requirement that the system must be maintained regularly conflicts with the availability requirements for the home customer and bank manager viewpoints. The home customer requires that the cash withdrawal service is available for 999/1000 requests, and the bank manager requires that transaction reports are provided daily with a failure rate of less than 1 in 1000.

These type of conflicts can be exposed by analysing the constraints associated with a particular service, for consistency, and by analysing one viewpoint's requirements against other viewpoint requirements for contradictions.

In addition to these specific viewpoint requirements are high-level organisational and other global requirements against which all requirements must be analysed. At this level, we are interested in establishing whether specific viewpoint requirements augment or contradict general organisational and global requirements.

Viewpoints place varying levels of importance on their requirements. It is important to characterise these varying levels of importance in order to sift the essential requirements from the non-essential and to resolve conflicts. One way of characterising requirements is to weigh them in order of importance. The weighing of non-functional requirements is especially important as they translate to constraints on services, which are reusable across viewpoints and may therefore have differing constraints placed on them that may conflict. Another important reason for characterising constraints is that it provides the designer with a basis on which to trade-off the less important constraints.

VORD incorporates a mechanism for weighting requirements that takes into account the viewpoint-requirement relationship, thereby accommodating differing perceptions and stakes. This mechanism can be used in conjunction with the stated rationales to resolve conflicting requirements or to suggest improvements. The VORD process

diagram in Fig. 3 shows that the result from analysis feeds back into the main requirements process through the proposed changes.

6 VORD toolset

Tools make a significant contribution to the requirements formulation process [10]. Their incorporation or support in methods improves the engineer's ability to manage the complexity associated with information collection, structuring, verification, consistency checking and integrity preservation. VORD is based on an extensible toolset whose framework lends itself to tailoring and component reuse. We would like to emphasise that the use of tools in VORD is an integral part of the method and is intended to provide support from the initial requirements formulation through to detailed specification.

The underlying philosophy of the VORD toolset is to afford users scope for creativity and experimentation in arriving at an expression of requirements, while enforcing the method. We believe the ability of a tool to accommodate potentially conflicting information without unduly restricting the user is very important. To this end, the VORD toolset incorporates interactive conflict report generation at all stages of requirements formulation.

Fig. 14 shows the general architecture of the toolset. The toolset has eight main components: the viewpoint editor, requirements space, constraint library, specification editor, proposed changes log, analysis process, entity identification process and mapping process. Straddling these six components are a report generation and method guidance tools.

The viewpoint editor facilitates the creation and structuring of viewpoint information collection. The requirements space is a central requirements repository; it maintains an updated record of all requirements, their sources, rationale, constraints, events scenarios, specifications and users. It serves as a source for reusable services as well as a reference point for other components of the toolset. The entity identification process (Fig. 14), for example, uses the requirements space to derive entities responsible for the provision of services and the viewpoint editor sees it as repository for reusable services.

The constraint library is a collection of user-defined non-functional requirements that can be associated with services. It comprises a tool for defining non-functional requirement templates, a constraint library browser and a facility for previewing and testing defined constraints. Both formal and informal constraints can be described using the tool.

The specification editor facilitates the definition of notation templates and the specification of services in various notations. The proposed changes log maintains a list of proposed changes to the requirements, and the analysis process tool provides a means of managing the analysis process.

The mapping process is concerned with mapping viewpoint-level information to system-level information and verifying that system-level information is consistent with the viewpoint-level requirements.

7 Limitations of VORD

A possible criticism of the method is that it does not explicitly support the analysis of the interaction across and within the viewpoints. This criticism is based on the fact that viewpoint interactions are addressed only in the context of services, i.e. a viewpoint is analysed for its role in the provision of a service. This is a reasonable criticism. We believe this type of analysis may provide system developers with additional information that may need to be taken into account in formulating the system requirements. Consider the example of direct interaction between the bank customer and bank manager resulting in the manager authorising cash withdrawal, even though the customer balance is less than the minimum prescribed level. It may be that the system requires this kind of flexibility built into it.

Currently, the model of control requirements adopted by VORD does not explicitly address control issues associated with concurrency. However, the method supports a framework that allows the engineer to reason about concurrency in relation to service provision. As services are explicitly identified with entities at the system level, it is possible to argue about possibilities of providing services concurrently, i.e. if they do not share similar entities.

One aspect of control that is usually ignored in many requirements methods is the flow of time. In structured analysis, the belief is that so long as data flows and data constraints are fully defined, time flow is not necessary because it follows as a property of the data flow/constraint information. This could be true only if you could guarantee a complete definition of the data flows and corresponding constraints. In practice, this is very difficult. In VORD we have not attempted to address the time issue beyond defining it as a constraint on system services.

VORD has been deliberately restricted to a service-oriented view of systems. A criticism of the method therefore is that it is difficult to apply to those systems which do not fit neatly into the service-oriented systems (SOS) paradigm. Service-oriented systems can be viewed as service-

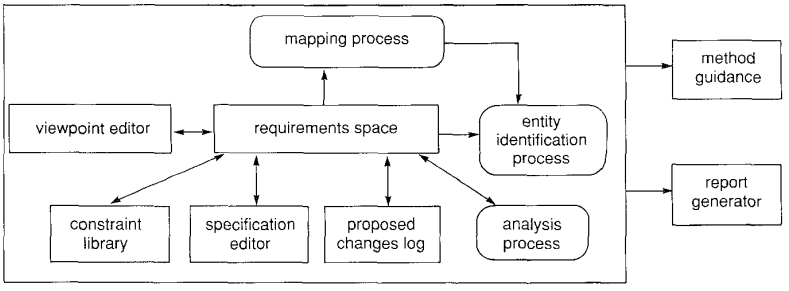

Fig. 14 VORD toolset architecture

providing enterprises; they employ systems composed of people, computer hardware and software, and other mechanisms to perform service actions in the customer environment [19].

We do not, however, consider this to be a serious limitation as we believe that most systems can be regarded as providing services of some kind to their environment. The intuitive end-user orientation of a service provides us with the ability to clearly distinguish between user needs on the one hand what is required (at system level) to meet those needs on the other. Secondly, the notion of a service also finds parallels in real life. Thus, for example, we can talk of the reliability of a service, the efficiency of a service and the cost of providing a service, all of which correspond to non-functional requirements. Thirdly, a service is a reusable commodity that is provided to many users, all (potentially) imposing differing constraints on it.

Issues relating to change control and the interface with existing software tools have not been explicitly addressed in VORD. The issue of change control is important as it may take several years to analyse requirements and to develop a large system and it must be expected that requirements changes will be identified during that time. It is therefore important that the inevitability of this is recognised and anticipated when producing a requirements document. A commercial version of VORD would need to incorporate a mechanism to support change control. It is important that VORD is able to interface with existing software tools, as this would allow inter-operability which would enhance the process of formulating requirements.

8 Conclusions

The notion of viewpoints proposed in VORD offers several added advantages over other viewpoint-oriented approaches to requirements engineering.

☐ Most existing viewpoint approaches lack any obvious framework for distinguishing between various user classes, types of user-system interaction and specific user requirements. Our proposed solution to this problem has been to address requirements from the user perspective (viewpoint), thereby creating a framework for distinguishing between user classes and specific user requirements. The intuitive end-user orientation of a service provides us with the ability to clearly distinguish between user needs on the one hand and what is required (at the system level) to meet those needs on the other hand.

☐ Unlike most viewpoint approaches whose concept of a viewpoint is largely intuitive, VORD is based on a clearly defined concept of viewpoints. Existing approaches have also failed to analyse viewpoints beyond considering them as data sources, data sinks or sub-system processes. These approaches focus most on their analysis on what are essentially internal perspectives of the proposed system. A VORD viewpoint is clearly defined by its attributes, services, events and specialisations.

☐ We have demonstrated the importance of incorporating indirect viewpoints into the requirements engineering process. Indirect viewpoints are very important because people associated with them are often very influential in an organisation and can make decisions on whether the system goes into service. The notion of indirect viewpoints is largely lacking in current approaches.

☐ The explicit identification of viewpoints with services in VORD has made it possible to create a framework where several related aspects can be encapsulated. It is possible, for example, to encapsulate within a viewpoint its rationale for a service and various constraints that it imposes on the service. This is a very useful attribute of VORD, as it improves the potential reusability of services and promotes incremental development.

☐ A lack of understanding of the terms used in requirements formulation, and hence a lack of communication between requirements engineers and systems users, has been cited as a major stumbling block to developing successful software systems (Section 2.1). A partial solution to this problem is to construct a framework that supports the integration of various formal and informal notations. Developing such a framework within existing requirement methods is difficult because they are usually associated with specific notations and are not based on extensible frameworks.

In VORD, there is no predefined notation for expressing service specifications. VORD allows an organisation to define a library of templates, based on different specification notations, and to use these in the specification of services. A template is intended to act as a guideline to the user by partitioning the specification into logical subsections. The tools allow the user to construct both whole and modular notation templates.

- Many methods do not address non-functional requirements explicitly, and those that address them have tended to address them as secondary to the 'central' issue of functional requirements. Existing methods lack support for the broad integration of functional and non-functional requirements. There is also a general lack of notations and tools that are flexible enough to accommodate the great diversity of non-functional requirements. VORD has addressed the issue of both global and specific non-functional requirements in relation to the system. Defined services may be associated with non-functional requirements that are derived from different viewpoints. Indirect viewpoints serve as a vehicle for collecting system-level non-functional requirements, and the viewpoint hierarchy allows these to be propagated to all services.
- VORD provides a framework that is amenable to requirements traceability.

In summary, we believe that VORD is a useful contribution to the field of requirements engineering. We have demonstrated that a method can be developed which takes into account both end-user and organisational considerations. The service orientation of the method ensures that system requirements, rather than high-level system specifications or designs, are derived by applying the method. We have developed a comprehensive toolset for VORD. Of course, the method is still being developed to address some of the problems identified earlier, but we would like to mention that an earlier version of the method was used to specify a fairly complex transactions-related system with some notable success. Suggestions arising from this early trial have been invaluable in the development of the current

model. Other user trials are underway, including the development of detailed requirements for an autonomous excavator.

9 References

[1] BOEHM, B.: 'Model and metrics for software management and engineering' (IEEE Computer Society Press, 1984), pp. 4–9

[2] BOEHM, B.: 'Industrial software metrics top 10 list', *IEEE Softw.*, 1987, **4,** (5), pp. 84–85

[3] SOMMERVILLE, I.: 'Software engineering' (Addison Wesley, 1992)

[4] ROMAN, G.C.R.: 'A taxonomy of current issues in requirements engineering', *IEEE Computer*, 1985, **18,** (4), pp. 14–21

[5] U.S. Government Accounting Office. Contracting for Computer Software Development: 'Serious problems require management attention to avoid wasting additional millions'. Report FGMSD-80-4, 1979

[6] RZEPKA, W.: 'Requirements engineering environment: software tools for modelling user needs', *IEEE Computer*, 1985, **18,** (4), pp. 9–12

[7] BORGIDA, A., GREENSPAN, S., and MYLOPOULOS, J.: 'Knowledge representation as a basis for requirements specifications', *IEEE Comput.*, 1985, **18,** (4), pp. 71–85

[8] DAVIS, A.M.: 'Software requirements analysis and specification' (Prentice-Hall International, 1990)

[9] BROWN, A.W., EARL, N.A., and MCDERMID, J.: 'Software engineering environments: automated support for software engineering' (McGraw-Hill, 1992)

[10] DORFMAN, M., and THAYER, R.H.: 'Requirements definition guidelines, and examples on system and software requirements engineering' (IEEE Computer Society Press, 1991)

[11] ROSS, D., and SCHOMAN, K.E.: 'Structured analysis for requirements definition', *IEEE Trans.*, 1977, **3,** (1), pp. 6–15

[12] MULLERY, G.P.: 'A method for controlled requirements specifications'. 4th IEEE Computer Society Int. Conf. on Software Engineering, pp. 126–135, Munich, Germany, 1979

[13] FICKAS, S., VAN LAMSWEERDE, A., and DARDENNE,: 'Goal-directed concept acquisition in requirements elicitation. 6th Int. IEEE Computer Society Press Workshop on Software Specification and Design, Como, Italy, 1991, pp. 14–21

[14] LEITE, J.C.P.: 'Viewpoints analysis: a case study', *ACM Softw. Eng. Notes*, 1989, **14,** (3), pp. 111–119

[15] FINKELSTEIN, A., KRAMER, J., NUSEIBEH, B., and GOEDICKE, M.: 'Viewpoints: a framework for integrating multiple perspectives in system development', *Int. J. Softw. Eng. Knowl. Eng.*, 1992, **2,** (10), pp. 31–58

[16] KOTONYA, G.: 'A viewpoint-oriented method for requirements definition'. PhD Thesis, Lancaster University, UK, 1994

[17] KOTONYA, G., and SOMMERVILLE, I.: 'Framework for integrating functional and non-functional requirements'. IEE Int. Workshop on Systems Engineering for Real-Time Applications, Cirencester, UK, 1993, pp. 148–153

[18] NORMAN, and DRAPER,: 1986

[19] GREENSPAN, S., and FEBLOWITZ, M.: 'Requirements engineering using the SOS paradigm'. RE '93 IEEE Int. IEEE Society Press Requirements Symp. on Requirements Engineering, San Diego, California, 1993, pp. 260–263

[20] WARD, P., and MELLOR, S.: 'Structured development for real-time systems' (Prentice-Hall, Englewood Cliffs, New Jersey, 1985)

[21] HARTLEY, D., and PIRBHAI, I.: 'Strategies for real-time systems specifications' (Dorset House, New York, 1987)

[22] RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F., and LORENSEN, W.: 'Object-oriented modelling and design' (Prentice-Hall International, 1991)

[23] SHLAER, S., and MELLOR, S.J.: 'Object-oriented systems analysis: modelling the world in data' (Yourdon Press, Prentice Hall, Englewood Cliffs, New Jersey, 1988)

[24] HAREL, D., LACHOVER, D., NAAMAD, A., PNUELI, A., POLITI, M., SHERMAN, R., and SHTULTRAURING, A.: 'STATEMATE: a working environment for development of complex reactive systems'. Tenth IEEE Int. IEEE Computer Society Press Conf. on Software Engineering, Washington, DC, 1988, pp. 396–406

[25] EASTERBROOK, S.M.: 'Resolving conflicts between domain descriptions with computer-supported negotiation', *Knowl. Acquisition*, 1991, **43,** (3), pp. 255–289

[26] HAUSEN, H.L.: 'A notion of rule-based software quality engineering'. Symp. on Applied Computing, Kansas City, USA, April 1991

FINKELSTEIN, A., KRAMER, J., and GOEDICKE, M.: 'Viewpoints oriented software specification'. 3rd Int. IEEE Computer Society Workshop on Software Engineering and its Applications, Toulouse, France, 1990, pp. 337–351

The paper was first received on 28 February and in revised form on 10 October 1995.

The authors are with the Department of Computing, Lancaster University, School of Engineering, Computing and Mathematical Sciences, Lancaster LA1 4YR, UK.