Requirements Eng (1997) 2: 92–101
© 1997 Springer-Verlag London Limited

Requirements
Engineering

Telecommunications Service Requirements: Principles for Managing Complexity

Pamela Zave[a] and Michael Jackson[b]
AT&T Laboratories—Research, [a]Florham Park, USA and [b]London, UK

Telecommunications services are complex and rapidly becoming more so. If we wish to apply formal methods of requirements engineering in the telecommunications domain, then the primary obstacle we face is the sheer complexity of the behaviour to be described. This paper focuses on one kind of telecommunications system, the long-distance telephone network *(LDTN), and presents several ways of managing complexity in LDTN requirements. The alleviation of complexity comes from careful application of some general principles of requirements engineering. At the same time that these principles help us manage complexity, the productive focus on complexity enhances the motivation for these principles.*

Keywords: Feature interaction; Telecommunications; Telephony

1. Introduction

Telecommunications services are complex and rapidly becoming more so. If we wish to apply formal methods of requirements engineering in the telecommunications domain, then the primary obstacle we face is the sheer complexity of the behaviour to be described.

This paper focuses on one kind of telecommunications system: the *long-distance telephone network* (LDTN). An LDTN is part of the world-wide voice-transmission network, some components of which are illustrated in Fig. 1.

A *telephony device* is an input/output device for voice. A telephony device is usually connected to the network by a *line*, dedicated to the device and also capable of supporting a single two-way voice channel. The alternative to a line is a wireless channel, which is dedicated to many devices in succession. A line or wireless channel leads from the telephony device to a *switch*, which is a voice-handling node of the network. Switches are connected by many *trunks*, each of which is like a line in supporting a single two-way voice channel.

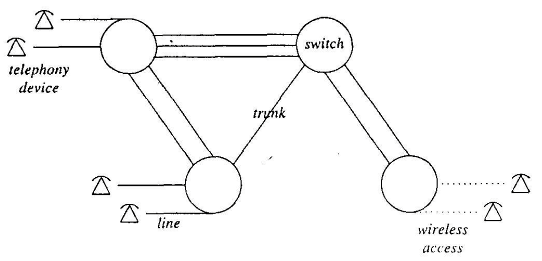

Fig. 1. Some components of the world-wide voice-transmission network.

The role of an LDTN in the world-wide voice-transmission network is illustrated by Fig. 2, where the boundaries of different *telephone systems*, owned and operated by different *service providers*, are also shown.

A *private branch exchange (PBX)* is a private switch, usually found on the premises of a business or institution. A *local network* or *local system* provides direct access by telephones to the rest of the world-wide network. An *LDTN* provides long-distance service. A *national network* provides telephone service for an entire country, combining the functions of local networks and LDTNs. A *cellular network* provides

Correspondence and offprint requests to: Pamela Zave, AT&T Laboratories - Research, 180 Park Avenue, room B211, Florham Park, NJ 07932, USA. Email: pamela@research.att.com mobile service; it may reach the rest of the world through any other type of system.[1]

LDTN requirements are complex in two orthogonal dimensions. They exhibit *feature complexity*, by which we mean complex services offered to their paying customers. This complexity has given rise to the well-known 'feature interaction' problem [1–3]. They also exhibit *system complexity*, by which we mean behavioural complexity arising from the properties of large distributed computing systems. Some sources of system complexity are race conditions, unsynchronised local clocks, resource failures, and the need for dynamic performance tuning.

System complexity concerns the service provider alone – even if customers are affected by it, they do not want to know about it and could do nothing about it if they did. Feature complexity, on the other hand, affects customer and service provider alike. If feature complexity is out of control, an LDTN will be difficult to build, use, maintain, extend, and market.

This paper presents several ways of managing feature complexity in LDTN requirements. The alleviation of complexity comes from careful application of some general principles of requirements engineering. At the same time that these principles help us manage complexity, the productive focus on complexity enhances the motivation for these principles.

[1]Other combinations are possible. For example, in the United States, there are now systems that act like national networks in the sense of combining the functions of local and long-distance networks, and that act like long-distance networks in the sense of having direct competition and needing the cooperation of other service providers for access to some local telephones.

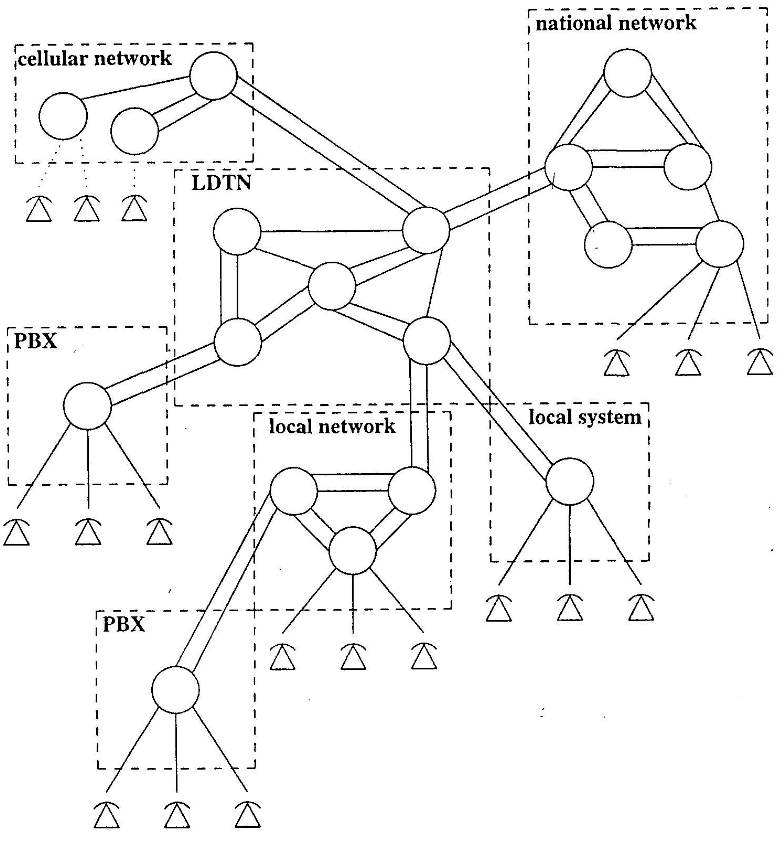

Fig. 2. Some systems and components of the world-wide voice-transmission network.

2. Principle: Requirements are About the Environment, not About the System

The requirements for a computer-based system concern the environment of the system as well as the system itself [4–6]. Recently we have shown that it is both possible and desirable to go much further in this direction, and to write requirements that do not mention the system at all [7–9]. The next few paragraphs give a very brief summary of how this is done.

The phenomena of the environment include those phenomena that are shared between the environment and the system. These shared phenomena constitute the system/environment interface. Thus, it is possible to describe interface behaviour without mentioning phenomena that belong exclusively to the system.

Conventional requirements focus on what the system to be built will do. If we cannot talk about the system, then we must have an alternative way to convey equivalent information.

The alternative is supplied by making a distinction between *indicative* and *optative* properties of the environment. An assertion in the indicative mood, commonly known as 'domain knowledge', describes the environment as it would be without, or in spite of, the system. A statement in the optative mood, commonly known as a requirement, describes the environment as we would like it to be because of the system. Thus a requirement constrains the system by describing its desired effect on the environment.

When the system to be built is as large and complex as an LDTN, then the absence of the system from the requirements is a big relief. Any unnecessary traces of the system architecture will cause unwanted complexity in the requirements.

The next three subsections present different aspects of LDTN requirements, showing for each one how a healthy focus on the environment can be maintained. Each aspect is interesting because it benefits greatly from this approach, because it is a challenge to this approach, or both.

2.1. Agreements

The service provider running an LDTN makes business agreements with its subscribers and with other service providers. These agreements control routing, feature application, billing rates, and other aspects of call processing.

The global state of an LDTN contains a representation of the current state of the agreements made by its service provider. In a requirements method based on description of the system to be built, formalisation of this part of the system state would be a major agenda item. How is it possible to specify LDTN requirements *without* mentioning this part of the system state?

The key fact is that agreements themselves are phenomena of the environment. Agreements relate entities such as subscribers, credit accounts, directory numbers, and features, all of which have a meaningful existence in the telecommunications environment. The relations among them, including 'subscriber sponsors credit account', 'directory number is assigned to subscriber', and 'subscriber purchases feature', are used by the LDTN but are not brought about by it. For this reason, LDTN requirements should describe them in the indicative mood. The indicative assertions will include such information as type constraints, arity constraints, and correlations among relations.

Provisioning is the usual term for getting data into the state of the LDTN. To keep the focus on the environment, we notice that provisioning events mark official changes to the agreements, and are therefore clearly phenomena of the environment. Provisioning events also happen to be shared with the system, which makes them interface phenomena as well. The relationships between provisioning events and the state of agreements, like relationships among the agreements themselves, are most accurately described in the indicative mood.

The requirements for call processing refer to the current state of agreements, for example to specify which features apply to which calls. The domain knowledge (indicative assertions) mentioned above will be used eventually to implement these requirements. The system must construct, from the shared provisioning events, an internal representation of the agreements in the environment. This internal representation is consulted during call processing.

Syntactically, domain knowledge relating agreements to provisioning events and to each other may not look much different from a specification of a system containing agreements. The benefits are in the interpretation. A statement of domain knowledge carries no implication that the information must appear in the system state, or if it does, that it appear in the same form.

2.2. Indirect Access

An LDTN is not connected directly to any telephones. The external trunks of an LDTN lead to other telephone systems, and through them, eventually, to telephones.

Because service providers wish to please their customers, requirements that describe what telephone users experience would seem to be the best telephony requirements. The only external behaviour that the implementors of an LDTN can guarantee, however, is the behaviour of its external trunks. So telephone-to-telephone requirements would leave a big gap between the behaviour that an LDTN can guarantee and the behaviour described in its requirements.

Ideally, this gap would be bridged by domain knowledge of the behaviour of other telephone systems [7–9]. But the reality is that very little can be said with certainty about the behaviour of another telephone system, and therefore very little can be proved about what the remote customers of an LDTN experience.

For example, suppose that law or policy requires that callers to a 900 number[2] hear an advisory announcement before the 900 charges begin. Even if the LDTN waits until its voice connection to the caller (which passes through a local network) is completely stable before playing the announcement, this does not guarantee that the caller will either hear it or disconnect before incurring 900 charges. The caller might use features of the local network to put a stable connection on hold and then return to talking and listening on it later.

The simplicity to be gained here is the simplicity of lowered expectations. It is no use writing many telephone-to-telephone requirements, because they cannot be satisfied. It makes more sense to concentrate on the external trunks, and to ensure that the system's behaviour at its interface is exactly right.

For those who are concerned with the interoperability of telephone systems, this discussion defines the problem. Requirements can constrain any part of the world, but a system can only control its own behaviour at its own interface. If a system is to satisfy far-reaching requirements, something must bridge the gap between the behaviour it can guarantee and the behaviour it is required to bring about. That something is domain knowledge, which allows us to reason about causes and effects in the environment. In the case of telephony, much of the environment of a telephone system consists of other telephone systems, so the satisfaction of requirements by any one telephone system depends on the properties of others.

2.3. System Complexity

In a requirements document, feature complexity plus system complexity is a deadly combination. If you do not believe this, you should examine Kay and Reed's specification of ordinary telephone service with races between the system and its telephones [10]. Fortunately, feature complexity and system complexity appear to be separable concerns.

The primary emphasis of requirements is customer service, so the immediate problem is to write sound feature requirements without straying into areas of system complexity. The solution is to write deliberately incomplete requirements, specifying the type or degree of incompleteness in the environment.

Consider, for example, the problem of stimulus–response race conditions. These are races between the system's response to a user stimulus and the user's subsequent stimulus. If admitted, these add greatly to the complexity of a specification or program. For this reason, many popular languages such as Esterel [11] declare them to be impossible – they assume explicitly that the system always responds before the next stimulus.

The trouble with this assumption is that it bears little or no relation to reality. A more realistic alternative is to declare explicitly that the requirements only constrain those behaviours in which the environment waits for responses before issuing additional stimuli. As with the agreements in Section 2.1, environment-oriented requirements may look much the same as system-oriented requirements, but they are interpreted differently.

The behaviours not covered by the formal requirements should be similar to those that are. It may be possible to formalise this using techniques such as those suggested by Jacob [12]. If not, telecommunications engineers are well accustomed to providing reasonable and customary implementations of incomplete specifications.

As another example of avoiding system complexity, consider resource failures that cannot be hidden from users because they make promised services completely unavailable. These exceptions can also add greatly to complexity, if their effects are fully specified.

In the case of resource failures there is no deterministic way to identify, in the environment, which behaviours will satisfy the ordinary requirements. Rather, the occasional non-conforming behaviour is 'chosen' by the system. So the incompleteness in this case must take a probabilistic form, such as '99% of all behaviours must satisfy the stated requirements.

It is a good idea to establish some general rules of etiquette for error handling. For example, in an LDTN a good policy is to play an explanatory announcement twice (ensuring that the customer hears and understands it) and then to disconnect the customer (ensuring that no further resources are wasted). Since these policies constrain all behaviours, they restore some of

[2]In the United States, callers to a 900 number pay a telephone company for the connection *and* pay the destination service a per-minute charge for connecting to it.

the certainty that was lost as a result of incomplete requirements.

3. Principle: The Real-world Problem Dictates the Formalisation, not the Other Way Around

New specification languages are designed for a variety of reasons. These reasons include exploring a new computational paradigm (e.g., logic programming) and exploiting a new verification technique (e.g., model checking). Even when the motivation was to make previous results more useful, one cannot assume that the language designer was thinking about the real world in its entirety rather than about computers themselves.

Sometimes the mismatch between formalism and the real world becomes ludicrous. Many formal specifications of a queue would be satisfied by a queue that surprises its user by spontaneously initiating its own put and get actions. Because the specification language abstracts away the control properties of events, it cannot express the requirement that only the environment can initiate gets and puts [13].

Problems of inappropriate formalisation are especially likely when requirements engineers cling to the use of a single formal language. Environments have widely differing characteristics, and large, complex systems usually have several distinctive aspects. No language is good for them all.

One of the worst effects of inappropriate formalisation is runaway complexity. It is from this perspective that we will look at the question of formalisation.

The ideal technique would be to write the requirements for each aspect of the system in the language best suited to it, and to compose the partial requirements specifications formally [14,15]. Since this is far more easily said than done, the subsections of this section present more specific techniques or principles, and show how they apply to managing the complexity of LDTN requirements.

3.1. Small is Beautiful

Small, clean, coherent languages are better for formal specification than large, inclusive ones. In addition to the obvious issues of aesthetics and semantic soundness, a small language is more likely to offer some major algorithmic capability – something that eliminates an entire task or subproject from the software development project.

An old, but still important, example is the case of the context-free grammar. A context-free parser is a valuable software component, and a context-free grammar is a specification of it. Not only is the specification small compared to the program, but it also has the wonderful property that the program can be generated automatically from it.

A more recent example is the broad and growing popularity of model checkers. If you specify higher-level properties in a version of temporal logic, and lower-level properties in an appropriate language based on communicating finite-state machines, then you can often prove algorithmically that the lower-level description satisfies the higher-level one.

It is certainly true that there are broad-spectrum specification languages, but they do not help reduce complexity as well as narrowly focused ones. There is a well known language trade-off between expressiveness and analysability, and broad-spectrum languages go too far in the direction of expressiveness to enjoy many analytic or generative powers.

It is already well known that finite-state machines are useful for telecommunications, so we shall give a different example. For formalising the agreements of an LDTN, we have found relational algebra to be both notationally convenient and analytically powerful.

For example, here are two useful partial functions (using the Z notation for relational algebra [16]):

$$\textit{dedicated-to}: \textit{trunk} \nrightarrow \textit{customer}$$

$$\textit{external-access-to}: \textit{trunk} \nrightarrow \textit{switch}$$

A pair (t,c) belongs to *dedicated-to* and if only if the trunk t is dedicated to the customer c, typically to provide direct communication between the LDTN and the customer's private switch. A pair (t,s) belongs to *external-access-to* if and only if the trunk t is an external trunk connecting LDTN switch s to some switch outside the LDTN.

It is an important constraint on the agreements that:

$$\mathbf{dom}\ \textit{dedicated-to} \subset \mathbf{dom}\ \textit{external-access-to}$$

A trunk cannot be assigned to a customer unless it is on the periphery of the LDTN. Also, the following declares and defines a useful function:

$$\textit{special-customers}: \textit{switch} \rightarrow \mathbf{P}\ \textit{customer}$$

$$\textit{special-customers}(s{:}\textit{switch}) \triangleq \mathbf{ran}((\textit{external-access-to}^{\sim} (\!| \{s\} |\!)) \lhd \textit{dedicated-to})$$

It says that the special customers of a switch as those customers to which the switch is connected by dedicated trunks. $(\textit{external-access-to}^{\sim} (\!|\{s\}|\!))$ is the relational image of the set $\{s\}$ under the inverse of the *external-*

access-to relation, i.e., it is the set of external trunks connected to *s*. This set is used to determine the subset of *dedicated-to* containing only those pairs in which the trunk is connected to *s*. The range of this restricted relation is the desired set of customers.

Like any proper algebra, relational algebra is unified and coherent; many laws provide a rich calculational potential. Relational algebra is easily type-checked. The analysability of relational algebra has recently been extended by the development of the Nitpick tool, which enumerates bounded state spaces automatically and checks them exhaustively for counterexamples to a specification [17]. Furthermore, a suite of tools for handling agreements in a large switching system relies on a formal agreements language close to relational algebra [18,19]. The suite includes support for writing update transactions and for specifying integrity constraints on the agreements. Other automated tools perform verification and code generation, guaranteeing database integrity and saving countless hours of work.

3.2. Use Domain-Specific Shorthands

Often a notation that is useful for many application domains can be made even more powerful in a particular domain by the addition of domain-specific shorthands. We shall illustrate this principle in the LDTN domain with the problem of unifying in-band and out-of-band signalling on trunks.

In LDTN requirements, call processing is driven by incoming signals associated with external trunks. It seems sensible to abstract the signals as events and to describe the events observable at each trunk in terms of a finite-state machine, but this plan is threatened by the existence of two radically different forms of signalling.

Out-of-band signals are messages travelling on a signalling channel that is completely separate from the voice channel. In-band signals are sounds travelling on the voice channel. For example, audible tones and announcements are in-bands signals from the LDTN to its environment. DTMF tones ('touch tones') are frequently used as in-band signals from the environment to the LDTN. Automatically recognised words, and changes between sound and silence, also sometimes serve as in-band signals to the LDTN.

Obviously the implementations of in-band and out-of-band signalling are vastly different, but the roles of these signals in requirements are often identical. For example, the directory number that a caller wishes to reach usually arrives as an out-of-band signal – the local network collects the dialled digits, determines that a long-distance call is being requested, and sends an initial message to the LDTN with the dialled digits in a data field. But the directory number can also arrive as an in-band signal. Suppose that a user sets up and makes a credit-card call. He then wishes to make another credit-card call without the bother of re-entering his credit information. Some LDTNs will accept the '#' DTMF tone as a signal to disconnect the current call, after which they will recognise in-band digits as the directory number of a request for a new call on the same credit account. Note that from the perspective of the local network and its interface with the LDTN, a sequence of credit-card calls made in this way is a single episode in which a voice connection is maintained continuously between the user and the LDTN. This voice connection carries the in-band signals by which the caller requests a sequence of long-distance connections.

The solution to the specification problem is a notation that expresses which changes of sound on the incoming voice channel are significant as signals. These significant changes can then be defined as events, and used on an equal footing with out-of-band events. This solution is illustrated by Fig. 3, which is a finite-state machine augmented by a special shorthand for in-band signalling.

Figure 3 is a partial specification of what can be observed at a callee's trunk. It is a finite-state machine with nested states as in Statecharts [20]. Event types spelled in lower-case letters represent events controlled by the environment (callee), while event types spelled in upper-case letters represent events controlled by the system (LDTN).

EXPEL, *withdraw*, and *answer* events all represent out-of-bands signals. A *withdraw* event ends the call on the initiative of the callee, while an *EXPEL* event ends the call on the initiative of the caller (as transmitted through the LDTN).

The remaining event types in Fig. 3 represent in-band signals. Recall that in-band signalling from the LDTN to the environment takes the form of audible tones or announcements. The tones or announcements to be played are specified as labels on appropriate states. These sounds are begun or ended by events that enter or leave the labelled states, respectively. If the callee is receiving a collect call, for instance, then an event of type *COLLECT-PERMISSION-NEEDED* begins the announcement that this is a collect call. An event of type *MESSAGE-ENDED* ends this announcement and begins the playing of a recorded speech from the caller. In the notation of Fig. 3, events can have attributes. The *s* attribute of a *COLLECT-PERMISSION-NEEDED* event is a recorded speech from the caller, which is why it appears as the label of the state in which this speech must be played to the callee. Another event of type

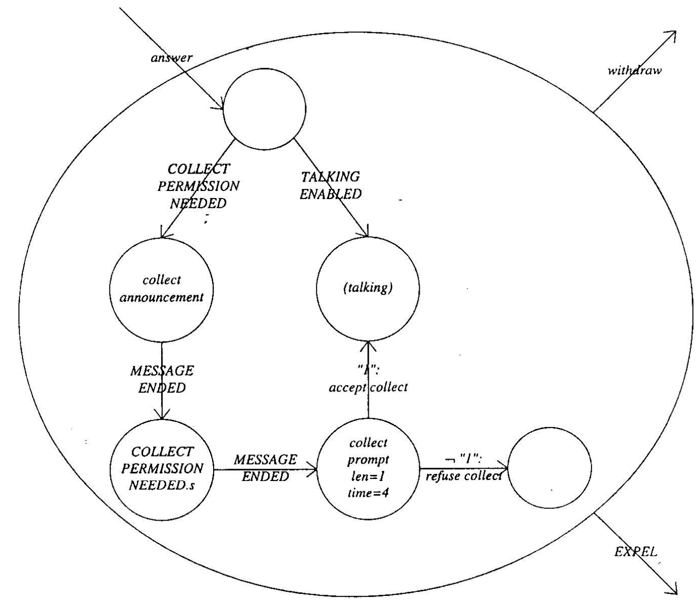

Fig. 3. A partial specification of a callee's trunk.

MESSAGE-ENDED ends this recording and begins the announcement that prompts for acceptance or refusal of the collect call.

To put in-band signalling from the environment to the LDTN into an event format, we need to do two things. The first is to indicate when incoming sounds might be significant. For example, the state labelled *collect-prompt* is also labelled *len = 1*, indicating that a sequence of DTMF tones of length up to 1 may have signalling significance in this state. Note that if a state has monitoring for DTMF tones *and* an outgoing tone or announcement, as this state does, then the outgoing sound ceases as soon as the first incoming digit is received.

Monitoring for DTMF tones can end in any of these ways: (1) The state is exited because of some event having nothing to do with monitoring, for example *EXPEL*. (2) One digit is collected. (3) Since the state also has *time = 4* in it, 4 second elapse without the arrival of a digit.

The second thing we need to do is to indicate which incoming sounds are interpreted as which symbolic events in the specification. In termination situations 2 and 3, the event that ends monitoring (a digit arrival or timeout) is an event represented in the specification. Its type in the specification depends on which of the string patterns found on transitions out of the monitoring state best matches the collected string. If '1' was collected then the final event is considered to be an *accept-collect* event. If nothing or any other digit was collected, then the final event is considered to be a *refuse-collect* event.

In summary, the special notations used in Fig. 3 accomplish two things: (1) they unify in-band and out-of-band signalling for purposes of convenient specification, and (2) they make it possible to specify requirements for in-band signalling easily and concisely. Another example of a telecommunications-specific shorthand – this time tables added to Z – can be found elsewhere [21].

3.3. Partial Aspects are Better than Partial Functionality

The current reality is that we cannot completely formalise the requirements for an LDTN; the magnitude of the problem is much greater than the power of any current solution technique. So we are faced with the prospect of doing partial formalisation or none at all, and faced with hard choices about where to invest our efforts.

All too often the 'choice' is made by the following scenario: (1) Begin with the ambition of specifying all requirements. (2) Start formalising the obvious, core

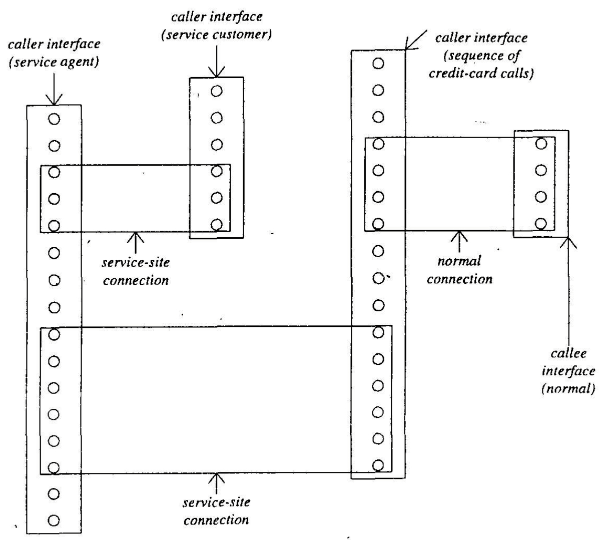
caller interface (service agent)
caller interface (service customer)
caller interface (sequence of credit-card calls)
service-site connection
normal connection
callee interface (normal)
service-site connection

Fig. 4. A Venn diagram of events involved in common LDTN features.

functions. In the case of an LDTN, modelling would start with POTS (Plain Old Telephone Service, the default or featureless service). (3) Add additional features, usually the easy ones, until everyone's ability to manage the complexity of the specification is exhausted. (4) Quit. The result of this scenario is that the formal requirements cover the functionality of the proposed system only partially. They are skewed toward the familiar and well understood; they avoid the troublesome and difficult.

It should be obvious that this approach expends most of the efforts where it will do the least good. It also incurs a big risk of false abstractions and generalisations, ones that cover the easy cases and fall apart on the hard cases. This has definitely occurred in telephony, where current thinking is dominated by call models.

A *call* is an attempt by one telephone (the *caller*) to establish one connection to one other telephone (the *callee*). The state of the call encodes or implies information about the states of all three entities. The term *call model* refers to any conceptual model for telephony that describes telephony exclusively in terms of calls.[3]

Many features subvert the one-to-one-to-one correspondence that is the essence of a call. For example, conferences connect sets of telephones, from three to a hundred. For a large conference, an appointment must be made ahead of time. Thus, at its starting time, a large conference can be thought of as a connection with no telephones. Furthermore, the conference can even initiate creation of the voice paths to all the participating telephones – very different from a call, which is always initiated by a caller.

Figure 4 shows another example, using only common LDTN features, full of exceptions to any call model. Figure 4 is a Venn diagram, where the set elements (small circles) are events and the sets (rectangles) are the scopes of applied specification modules. As an additional dimension of meaning, the vertical dimension of the diagram shows a rough time ordering on events.

Each tall box in Fig. 4 is concerned with an *interface episode* in which one trunk is being used to access one telephone; this is the closest that LDTN can come to talking directly about telephones. Each wide box is concerned with a two-way connection between such episodes; a wide box corresponds to an application of a specification module that coordinates events at the two ends of the connection.

The labels given some indication of the features being used. The telephone on the left is used by a service agent who is connected with a series of callers to a toll-free number. The callers to the virtual *service site* are queued and then connected to service agents in the order in which they called. The second telephone is simply one of the customers of the service site. The

[3]The most advanced call model in use today is the Intelligent Network Conceptual Model (INCM). It was developed under the auspices of the International Telecommunication Union (ITU) and the European Telecommunication Standard Institute (ETSI), and is being promulgated as a standard by those organisations [22,23].

third telephone is making a series of credit-card calls, using the feature described in Section 3.2. It first connects to a normal callee, and second to a service agent. The fourth telephone is the normal callee reached by the third telephone. Thus, even the commonest LDTN features create many-to-many correspondences among telephones and connections. Clearly many LDTN functions do not fit neatly into calls, and any call model will be partial at best.

A better alternative to partial functionality is modelling specific, well-defined aspects or views of the requirements. Each aspect may be narrowly focused, and the aspects together may not cover all the requirements, but an aspect is internally complete. For example, it is possible to specify everything observable at a telephone, quite separately from (and more simply than) the connection features that relate one telephone to another [24].

One advantage of the aspect approach is that it is clear where you have completeness and where you do not. Aspects can be chosen quite freely and the choice can be made based on what is most important.

Most advantageous of all, a well-formalised aspect – being tangible and complete – can really be useful. The tools for agreements mentioned in Section 3.1 [18,19] are an excellent example of this usefulness. They prove that incompleteness does not render a formal model ineffective, provided that the boundaries of its scope are drawn in rational places.

4. Conclusion

We have presented several ways of managing the complexity of LDTN requirements, along with the principles behind them. We hope that these observations will be interesting and suggestive to those who are primarily concerned with other software application domains.

We would like to be able to claim that these techniques are sufficient to bring the complexity of LDTN requirements down to a comfortable level, but that is far from true. All we can say for sure is that these techniques are helpful, and that we would not attempt to deal with LDTN requirements without them.

References

1. Cameron EJ, Griffeth ND, Lin Y-J et al. A feature interaction benchmark for IN and beyond. In Bouma LG, Velthuijsen H (eds), Feature Interactions in Telecommunications Systems. IOS Press, Amsterdam, 1994, pp. 1–23
2. Griffeth ND, Lin Y-J. Extending telecommunications systems: the feature-interaction problem. IEEE Comput 1993; 26(8): 14–18
3. Zave P. Feature interactions and formal specifications in telecommunications. IEEE Comput 1993; 26(8): 20–30
4. Balzer R, Goldman N. Principles of good software specification and their implications for specification language. In Proceedings of the Specifications of Reliable Software Conference, IEEE Computer Society, 1979, pp. 58–67
5. Jackson M. Information systems: modelling, sequencing, and transformations. In Proceedings of the Third International Conference on Software Engineering. IEEE Computer Society Press, Washington, DC, 1978, pp. 73–81
6. Lehman MM. Programs, life cycles, and laws of software evolution. Proc IEEE 1980; 68(9): 1060–1076
7. Jackson M. Software requirements and specifications: a lexicon of practice, principles, and prejudices. Addison-Wesley, Reading, MA, 1995
8. Jackson M, Zave P. Deriving specifications from requirements: an example. In Proceedings of the Seventeenth International Conference on Software Engineering. ACM, New York, 1995, pp. 15–24
9. Zave P, Jackson M. Four dark corners of requirements engineering. ACM Trans Software Eng Methodol 1997; 6(1): 1–30
10. Kay A, Reed JN. A rely and guarantee method for timed CSP: a specification and design of a telephone exchange. IEEE Trans Software Eng 1993; 19(6): 625–639
11. Berry G, Gonthier G. The Esterel synchronous programming language: design, semantics, implementation. Sci Comput Program 1992; 19: 87–152
12. Jacob JL. Refinement of shared systems. In John McDermid (ed). The theory and practice of refinement: approaches to the formal development of large-scale software systems. Butterworths, 1989, pp. 27–36
13. Lamport L. A simple approach to specifying concurrent systems. Commun ACM 1989; 32(1): 32–45
14. Nuseibeh B, Kramer J, Finkelstein A. A framework for expressing the relationships between multiple views in requirements specifications. IEEE Trans Software Eng 1994; 20(10): 760–773
15. Zave P, Jackson M. Conjunction as composition. ACM Trans Software Eng Methodol 1993; 2(4): 379–411
16. Spivey JM. The Z notation: a reference manual, 2nd edn. Prentice-Hall, Englewood Cliffs, NJ, 1992
17. Jackson D, Damon CA, Elements of style: analysing a software design feature with a counterexample detector. IEEE Trans Software Eng 1996; 22(7): 484–495
18. Corrico S, Ewbank B, Griffin T, Meale J, Trickey H. A tool for developing safe and efficient database transactions. In Proceedings of the Fifteenth International Switching Symposium of the World Telecommunications Congress, April 1995, pp. 173–177
19. Griffin T, Trickey H. Integrity maintenance in a telecommunications switch. IEEE Data Eng Bull 1994; 43–46
20. Harel D. Statecharts: a visual formalism for complex systems. Sci Comput Program 1987; 8: 231–274
21. Zave P. Secrets of call forwarding: a specification case study. In Formal description techniques VIII (Proceedings of the Eighth International IFIP Conference on

Formal Description Techniques for Distributed Systems and Communications Protocols). Chapman & Hall, London, 1996, pp. 153–168.
22. Duran JM, Visser J. International standards for intelligent networks. IEEE Commun. 1992; 30(2): 34–42
23. Garrahan JJ, Russo PA, Kitami K, Kung R. Intelligent Network overview. IEEE Commun 1993; 31(3): 30–36
24. Zave P, Jackson M. Where do operations come from? A multiparadigm specification technique. IEEE Trans Software Eng 1996; 22(7): 508–528