
Information technology — Programming languages, their environments and system software interfaces — Guidelines for the preparation of Language-Independent Service Specifications (LISS)

Technologies de l'information — Langages de programmation, leurs environnements et interfaces du logiciel d'exploitation — Lignes directrices pour l'élaboration de spécifications de service indépendantes du langage (LISS)

Reference number
ISO/IEC TR 14369:1999(E)



PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

© ISO/IEC 1999

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 734 10 79
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

This Technical Report is dedicated to Brian L. Meek in grateful recognition of his leadership and vision in the development of the concepts on programming language independent specifications, and his efforts in producing a set of standards documents in this area. Without his commitment this Technical Report never would have been published.

Contents

Page

INTRODUCTION	X
Background	x
Principles	x
1. SCOPE	1
2. REFERENCES	1
3. DEFINITIONS AND ABBREVIATIONS	2
3.1 Definitions	2
3.2 Abbreviations	4
4. OVERVIEW	5
4.1 Services, interfaces, service providers and service users	5
4.2 Information technology services	5
4.3 Services and language independence	6
4.4 Language-independent specifications	7
4.5 Problems of language dependence and inbuilt assumptions	7
4.5.1 Representational assumptions	8
4.5.2 Implementation assumptions	8
5. GUIDELINES ON STRATEGY	9
5.1 General guidelines	9
5.1.1 Guideline: Dependence of the interface on the service	9
5.1.2 ... Guideline: What to do when there are interoperability, concurrency, or time constraint issues.	9
5.1.3 Guideline: Use of marshalling/unmarshalling	9
5.1.4 Guideline: Recruiting expertise from a variety of backgrounds	10
5.2 What to do if starting from scratch	10
5.2.1 General guidelines	10
5.2.2 Specifying the service in language-independent form	10
5.2.3 Specifying the interface to the service in language-independent form	11

5.3 What to do if starting from an existing language-dependent specification.....	11
5.3.1 General guidelines	12
5.3.2 Converting an existing language-dependent specification of the service into language-independent form.....	13
5.3.3 Converting an existing implicit interface into an explicit language-independent interface.....	14
5.3.4 Specifying a language-independent interface to a service whose specification is language-dependent	15
6. GUIDELINES ON DOCUMENT ORGANISATION.....	17
6.1 Guideline: The general framework.....	17
6.1.1 Checklist of parts for inclusion	17
6.2 Guideline: Production and publication	18
6.3 Guideline: Document organisation when starting from a language-specific specification	19
7. GUIDELINES ON TERMINOLOGY	20
7.1 Guideline: The need for rigour	20
7.2 Guideline: The need for consistency.....	20
7.3 Guideline: Use of undefined terms.....	20
7.4 Guideline: Use of ISO 2382	20
7.5 Guideline: Use of definition by reference	21
7.6 Guideline: Terminology used in bindings	21
8. GUIDELINES ON USE OF FORMAL SPECIFICATION LANGUAGES.....	22
8.1 Guideline: Use of a formal specification language.....	22
8.2 Checklist of formal specification languages	22
8.2.1 Estelle.....	22
8.2.2 Lotos.....	22
8.2.3 VDM-SL.....	23
8.2.4 Z.....	23
8.2.5 Extended BNF.....	23
8.3 Guideline: Using formal specifications from the outset.....	24
8.4 Guideline: Use of operational semantics	24

9.	GUIDELINES ON INTEROPERABILITY	25
9.1	Introduction	25
9.1.1	Interoperability with what?	25
9.1.2	The nature of the interoperation.....	26
9.1.3	How interoperation is invoked.....	26
9.2	Guidelines on interoperability with other instantiations of the same service	26
9.2.1	Guideline: Identifying features affecting interoperability	26
9.2.2	Guideline: Precise definition and rigorous conformity requirements	26
9.2.3	Guideline: Importance of exchange values.....	27
9.3	Guidelines on interoperability with other services.....	27
9.3.1	Guideline: Interoperability with other services being defined at the same time.....	27
9.3.2	Guideline: Interoperability with a pre-defined service.....	27
10.	GUIDELINES ON CONCURRENCY ISSUES	29
10.1	Guidelines on concurrency within the service specification.....	29
10.1.1	Guideline: Avoidance of unnecessary concurrency requirements.....	29
10.2	Guidelines on concurrency of interaction with service users.....	29
10.2.1	Guideline: Handling of concurrent service requests	30
10.2.2	Guideline: Number of concurrent service requests handled.....	30
10.2.3	Guideline: Order of processing of service requests.....	30
10.2.4	Guideline: Criteria for prioritizing service requests	30
10.3	Guidelines on concurrency requirements on bindings.....	30
10.3.1	Guideline: Avoidance of concurrency requirements.....	30
10.3.2	Guideline: Specification of unavoidable concurrency requirements.....	31
11.	GUIDELINES ON THE SELECTION AND SPECIFICATION OF DATATYPES	32
11.1	Guideline: Use of ISO/IEC 11404:1996 Language-independent datatypes	32
11.2	Guideline: Specification of datatype parameter values	32
11.3	Guideline: Treatment of values outside the set defined for the datatype	32
11.4	Guideline: Specification of operations on data values.....	33
11.5	Guideline: Recommended basic set of datatypes	33
11.6	Guideline: Specification of arithmetic datatypes	33
11.7	Guideline: Approach to language bindings of datatypes	34
11.8	Guideline: Avoidance of representational definitions.....	34

12. GUIDELINES ON SPECIFICATION OF PROCEDURE CALLS	35
12.1 Guideline: Avoidance of unnecessary operational assumptions or detail	35
12.2 Guideline: Use of ISO/IEC 13886:1996 (LIPC) procedure calling model	35
12.3 Guidelines on the use of ISO/IEC 13886:1996 (LIPC)	36
12.3.1 Guideline: Selection of datatypes of parameters	36
12.3.2 Guideline: Selection of parameter passing modes	37
12.3.3 Guideline: Use of bindings to LIPC	37
12.4 Interfacing via remote procedure calling (RPC)	37
12.4.1 Guideline: Avoid limiting the service specification because of constraints on the interface specification	38
12.4.2 Guideline: Specification of RPC interface	38
12.4.3 Guideline: Use of subsets	38
12.4.4 Guideline: Use of ISO/IEC 11578:1996 (RPC)	38
12.5 Guideline: Guidance concerning procedure calling to those defining language bindings to the language-independent service specification	39
13. GUIDELINES ON SPECIFICATION OF FAULT HANDLING	40
13.1 Guideline: Fault detection requirements	40
13.2 Checklist of potential faults	40
13.2.1 Invocation faults	41
13.2.2 Execution faults	41
13.3 Guideline: Recovery from non-fatal faults	41
14. GUIDELINES ON OPTIONS AND IMPLEMENTATION DEPENDENCE	42
14.1 Guidelines on service options	42
14.1.1 Guideline: Optional service features	42
14.1.2 Guideline: Avoidance of assumptions about the use of the service	42
14.1.3 Guideline: Management of optional service features	43
14.1.4 Guideline: Definition of optional features	43
14.2 Guidelines on interface options	43
14.2.1 Guideline: Completeness of interface	43
14.2.2 Guideline: Interface to service with options	43
14.3 Guidelines on binding options	43
14.3.1 Guideline: Completeness of binding	43
14.3.2 Guideline: Binding to a service with options	44
14.3.3 Guideline: Binding to a language with optional features	44
14.4 Guidelines on implementation dependence	44
14.4.1 Guideline: Completeness of definition	44
14.4.2 Guideline: Provision of implementation options	45
14.4.3 Guideline: Implementation-defined limits	45

15. GUIDELINES ON CONFORMITY REQUIREMENTS.....	47
15.1 Guidelines for specifying conformity of implementations of the service	48
15.1.1 Guideline: Avoidance of assumptions about the implementation language	48
15.1.2 Guideline: Avoidance of representational assumptions.....	48
15.1.3 Guideline: Avoidance of implementation model.....	48
15.1.4 Guideline: Requiring end results rather than methods	48
15.2 Guidelines for specifying conformity of implementations of the interface.....	48
15.2.1 Guideline: Requirements on implementation-defined aspects	48
15.3 Guidelines for specifying conformity of bindings.....	48
15.3.1 Guideline: Propagating requirements to conforming bindings.....	48
15.3.2 Guideline: Adherence to defined semantics	49
16. GUIDELINES ON SPECIFYING A LANGUAGE BINDING TO A LANGUAGE- INDEPENDENT INTERFACE SPECIFICATION	50
16.1 Guideline: Use of bindings to LID and LIPC	50
16.2 Guideline: Adherence to defined semantics	50
16.3 Guideline: Binding document organisation	50
16.4 Guideline: "Reference card" binding documents	51
17. GUIDELINES ON REVISIONS	52
17.1 Kinds of change that a revision can introduce	52
17.1.1 Addition of a new feature	52
17.1.2 Change to the specification of a well-defined feature	52
17.1.3 Deletion of a well-defined feature	52
17.1.4 Deletion of ill-defined feature	53
17.1.5 Clarification of ill-defined feature.....	53
17.1.6 Change or deletion of obsolescent feature	53
17.1.7 Change of level definition.....	53
17.1.8 Change of specified limit to implementation-defined value.	53
17.1.9 Change of other implementation requirement	53
17.1.10 Change of conformity clause	53
17.2 General guidelines applicable to revisions	53
17.2.1 Guideline: Revision compatibility	53
17.3 Guidelines on revision of the service specification	54
17.3.1 Guideline: Determining impact on interface and language bindings	54
17.3.2 Guideline: Minimising impact on interface and language bindings.....	54
17.3.3 Guideline: Use of incremental approach to revision	54
17.4 Guidelines on revision of the service interface	54
17.4.1 Guideline: Buffering unrevised bindings from changes	54
17.4.2 Guideline: Use of incremental amendments.....	54

17.5 Guidelines on revision of language bindings following revision of the service interface 55

17.5.1 Guideline: Buffering application programs from changes 55

17.5.2 Guideline: Use of incremental amendments 55

17.6 Guidelines on revision of a language binding following revision of the language 55

17.6.1 Guideline: Use of new language features 55

17.6.2 Guideline: Buffering "legacy" application programs from changes 55

17.6.3 Guideline: Buffering application programs by use of options.....55

ANNEX A BRIEF GUIDE TO LANGUAGE-INDEPENDENT STANDARDS..... 56

A.1 Language-independent arithmetic.....56

A.2 Language-independent datatypes 56

A.3 Language-independent procedure calling 57

ANNEX B GLOSSARY OF LANGUAGE-INDEPENDENT TERMS 58

B.1 Source indications 58

B.2 Index of terms 58

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example), it may decide by a simple majority vote of its participating members to publish a Technical Report. A Technical Report is entirely informative in nature and does not have to be reviewed until the data it provides are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this Technical Report may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 14369 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Introduction

Background

This Technical Report provides guidance to those writing specifications of services, and of interfaces to services, in a language-independent way, in particular as standards. It can be regarded as complementary to ISO/IEC TR 10182 Guidelines for language bindings, which provides guidance to those performing language bindings for such services and their interfaces.

Notes

1. Here and throughout, "language", on its own or in compounds like "language-independent", means "programming language", not "specification language" nor "natural (human) language", unless explicitly stated.
2. A "language-independent" service or interface specification may be expressed using either or both of a natural language like English or a formal specification language like VDM-SL or Z; in a sense, a specification might be regarded as "dependent" on (say) VDM-SL. The term "language-independent" does not imply otherwise, since it refers only to the situation where programming language(s) might otherwise be used in defining the service or interface.

The development of this Technical Report was prompted by the existence of an earlier draft IEEE Technical Report (IEEE TCOS-SCC Technical Report on Programming Language Independent Specification Methods, draft 4, May 1991). The TCOS draft was concerned with specifications of services in a POSIX systems environment, and as such contained much detailed POSIX-specific guidance; nevertheless it was clear that many of the principles, if not the detail, were applicable much more generally. This Technical Report was conceived as a means of providing such more general guidance. Because of the very different formats, and the POSIX-related detail in the TCOS draft, there is almost no direct correspondence between the two documents, except in the discussion of the benefits of a language-independent principles below. However, the spirit and principles of the TCOS draft were of great value in developing this Technical Report, and reappear herein, albeit in much altered and more general form.

Note - The TCOS draft has not in fact been published, as the result of an IEEE decision to concentrate activities in other POSIX areas.

Principles

Service or interface specifications that are independent of any particular language, particularly when embodied in recognized standards, are increasingly seen as an important factor in promoting interoperability and substitution of system components, and reducing dependence on and consequent limitations due to particular language platforms.

Note - It is of course possible for a specification to be "independent" of a particular language in a formal sense, but still be dependent on it through inbuilt assumptions derived from that language which do not necessarily hold for other languages. The term "language-independent" here is meant in a much stronger sense than that, though complete independence from all inbuilt assumptions may be difficult if not impossible to achieve.

Potential benefits from language-independent service or interface specifications include:

- A language-independent interface specification specifies those requirements that are common to all language bindings to that interface, and hence provides a specification to which language bindings may conform.
- A language-independent interface specification is a re-usable component for constructing language bindings.
- A language-independent interface specification aids the construction of language bindings by providing a common reference to which all bindings can relate. Through this common reference it is possible to make use

of pre-existing language bindings to language-independent standards for common features such as datatypes and procedure calls, and to other language-independent specifications with related concepts.

- A language-independent service or interface specification provides an abstract specification of a service in isolation from language-dependent extensions or restrictions, and hence facilitates more rigorous modelling of services and interfaces.
- Language-independent service specifications facilitate the specification of relationships between one service and another, by making it easier to relate common concepts than is generally possible when the specifications are dependent on different languages.
- A language-independent interface specification facilitates the definition of relationships between different language bindings to a common service (such as requirements for interoperability between applications based on different languages that are sharing a common service implementation), by providing a common reference specification to which all the languages can relate.
- A language-independent interface specification facilitates the definition of relations between bindings to multiple services, including the requirements on management of multiple name spaces.
- A language-independent service or interface specification brings economic benefits by reducing the effort and resources needed to ensure compatibility and consistency of behaviour between implementations of the same service in different languages or between applications based on different languages using the same interface.

Information technology — Programming languages, their environments and system software interfaces — Guidelines for the preparation of Language-Independent Service Specifications (LISS)

1 Scope

This Technical Report provides guidelines to those concerned with developing specifications of information technology services and their interfaces intended for use by clients of the services, in particular by external applications that do not necessarily all share the environment and assumptions of one particular programming language. The guidelines do not directly or fully cover all aspects of service or interface specifications, but they do cover those aspects required to achieve language independence, i.e. required to make a specification neutral with respect to the language environment from which the service is invoked. The guidelines are primarily concerned with the interface between the service and the external applications making use of the service, including the special case where the service itself is already specified in a language-dependent way but needs to be invoked from environments of other languages. Language bindings, already addressed by another Technical Report, ISO/IEC TR 10182 Guidelines for language bindings, are dealt with by providing advice on how to use the two Technical Reports together.

This Technical Report provides technical guidelines, rather than organizational or administrative guidelines for the management of the development process, though in some cases the technical guidelines may have organizational or administrative implications.

2 References

ISO 8807:1989, *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*.

ISO/IEC 9074:1997, *Information technology – Open Systems Interconnection – Estelle: A formal description technique based on an extended state transition model*.

ISO/IEC TR 10034:1990, *Guidelines for the preparation of conformity clauses in programming language standards*.

ISO/IEC TR 10176:1998, *Information technology – Guidelines for the preparation of programming language standards*.

ISO/IEC TR 10182:1993, *Information technology – Programming languages, their environments and system software interfaces – Guidelines for language bindings*.

ISO/IEC 10967-1:1994, *Information technology – Language independent arithmetic – Part 1: Integer and floating point arithmetic*.

ISO/IEC 11404:1996, *Information technology – Programming languages, their environments and system software interfaces – Language-independent datatypes*.

ISO/IEC 11578:1996, *Information technology – Open Systems Interconnection – Remote Procedure Call (RPC)*.

ISO/IEC 13719:1995, *Information technology – Portable Common Tools Environment (PCTE)*.

ISO/IEC 13817-1:1996, *Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language*.

ISO/IEC 13886:1996, *Information technology – Language-Independent Procedure Calling (LIPC)*.

ISO/IEC 14977:1996, *Information technology – Syntactic metalanguage – Extended BNF*.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this Technical Report, the following definitions and abbreviations apply.

3.1.1 client

See service user.

3.1.2 datatype

A set of values, usually accompanied by a set of operations on those values.

3.1.3 formal language, formal specification language

See specification language.

3.1.4 interface

The mechanism by which a service user invokes and makes use of a service.

3.1.5 language

Unless otherwise qualified, "language" means "programming language", not "specification language" or "natural (human) language".

3.1.6 language binding

A specification of the standard interface to a service, or set of services, for applications written in a particular programming language.

3.1.7 language-dependent

Making use of the concepts, features or assumptions of a particular programming language.

3.1.8 language-independent

Not making use of the concepts, features or assumptions of any particular programming language or style of language.

3.1.9 language processor

The entire computing system which enables a programming language user to translate and execute programs written in the language, in general consisting both of hardware and of the relevant associated software.

Note - This definition comes from ISO/IEC TR 10176, Guidelines for the preparation of programming language standards.

3.1.10 mapping

(noun) A defined association between elements (such as concepts, features or facilities) of one entity (such as a programming language, or a specification, or a standard) with corresponding elements of another entity. Mappings are usually defined as being from one entity into another. A language binding of a language L into a standard S usually incorporates both a mapping from L into S and a mapping from S into L.

(verb) The process of determining or utilizing a mapping.

Note - Depending upon what is being mapped, a mapping is not necessarily one-to-one. This means that mapping an element E from system A into an element E' of system B, followed by mapping E' back into system A, may not necessarily get back to the original E. In such situations, if a two-way correspondence is to be preserved, execution of the mappings must include recording the place of origin and returning to it.

3.1.11 marshalling

The process of collecting the actual parameters used in a procedure call, converting them if necessary, and assembling them for transfer to the called procedure. This process is also carried out by the called procedure when preparing to return the results of the call to the caller.

Note - Marshalling can be regarded as being performed by a service user when preparing input values for a service provider, and by a service provider when preparing results for a service user, the service concerned being regarded as the procedure being called.

3.1.12 procedure

A subprogram which may or may not return a value (the former variant is sometimes called a subroutine, the latter is sometimes called a function).

Note - Some programming languages use different terminology.

3.1.13 server

See service provider

3.1.14 service

A facility or set of facilities made available to service users through an interface.

3.1.15 service provider

A computer system or set of computer systems that implements a service and makes it available to service users.

Notes

1. In this definition, "computer system" means a logical system, not a physical system; it may correspond to part of all of one or more physical computer systems.
2. The term "server" is often used in a similar sense, though sometimes implying a physical computer system that has no other function than to provide its service

3.1.16 service user

An application (typically a program in some language) which makes use of a service.

Note - The term "client" is often used in a similar sense, though sometimes implying the physical computer system on which the application is running, rather than just the application itself.

3.1.17 specification language

A formal language for defining the semantics of a service or an interface precisely and without ambiguity.

3.1.18 unmarshalling

The process of receiving and disassembling transferred parameters, and converting them if necessary, to prepare the values for further use. This process is carried out by the called procedure on receipt of the actual parameters for the call, and by the caller on receipt of the returned results of the call.

Note - Unmarshalling can be regarded as being performed by a service provider when receiving input values from a service user, and by a service user when receiving results from a service provider, the service concerned being regarded as the procedure being called.

3.1.19 Z

(1) (mathematics, e.g. ISO/IEC 10967-1:1993) the complex numbers

(2) (pronounced "zed") a formal specification language, see ISO/IEC WD 13568.

3.2 Abbreviations

3.2.1 LID

Language-independent datatypes, as defined in ISO/IEC 11404:1996.

3.2.2 LIPC

Language-Independent Procedure Calling, as defined in ISO/IEC 13886:1996.

3.2.3 RPC

Remote Procedure Call, as defined in ISO/IEC 11578:1996

4 Overview

4.1 Services, interfaces, service providers and service users

The concept of a "service" is a very general one. In some contexts it is customary to use it in a restricted sense, e.g. when talking about "service industries" as contrasted with "manufacturing industries". Despite such usages, almost any activity or behaviour can be regarded as a "service", if it serves some useful purpose to do so (for example, manufacturing spoons can be regarded as a service for those needing spoons).

With the concept of a service come the concepts of a "service provider" and a "service user". The provider performs the activity that constitutes the service; the user is the customer or the client for the service, for whom the service is performed. In the information technology field, the "client-server model" incorporates these concepts: the server provides, the client uses.

Between the service provider and the service user is an interface that allows them to communicate. The service user communicates through the interface the requirement for the service, and any relevant information (e.g. not only the need for spoons, but the number and size of spoons required), and the service provider communicates through the interface the response to the order for the service, and any additional information or queries (e.g. the spoons can be delivered in six days, do you want silver spoons or plastic spoons?). In the information technology field, such interfaces are usually explicit, realized in hardware or software or both. In the world in general, they are sometimes explicit, but sometimes subsumed in more general human or other interactions.

This distinction between provider and user (client and server) must not be assumed to correspond to identifiable distinct entities. The distinction, and the service interface, may be purely notional, and possibly not normally thought of in that way. The service itself may similarly not correspond to a distinct, separate activity, and again possibly not normally thought of as such; it may be subsumed in some other activity or group of activities, and may possibly be implicit.

Hence, for example, in a transaction between two parties, each one may be providing a service for the other: each is a client, and each a server. In another context, the provider is providing the service to itself; the provider is also the user. Though it may be possible to subdivide the provider/user into a provider part and a user part when considering provision of the service, this may be inconvenient in other respects.

In summary, "client" and "server", are roles that are carried out, rather than elements that necessarily must be implemented separately. Though the term "client-server" is sometimes used in the information technology field in ways that are more specific than it is used here, it is important not to carry over assumptions from particular client-server models when reading this Technical Report. It is still more important not to assume that implementation of any service, in the sense used here, has to be done using a client-server model.

4.2 Information technology services

The history of information technology has many instances of the technology, or a product, being used for very different purposes and in very different ways from those originally envisaged. The kinds of service that information technology and products provide have continually expanded and diversified, and this is still continuing.

It is as common in information technology as in the outside world for the term "service" in particular contexts to be used in a rather specific way. The history of the technology suggests that, for the purposes of formulating guidelines about services, the term should instead be used as generally as possible.

This Technical Report has adopted this very general approach to the concept of "service". It is therefore important that, when using this Technical Report and the guidelines it contains, no presuppositions should be made about what a service is, or about how and by what it is provided or how and by what it is used. The guidelines should be interpreted and applied in that light.

This Technical Report does, however, carefully distinguish between the service itself, and the interface used to communicate with it. In some usages the term "service" includes the interface, and the interface may be embedded in the service and its specification (as in the phrase "all parts of the service"). However, logically they are distinct, and this logical distinction is maintained throughout this Technical Report.

Services in the most general sense often simply evolve naturally, but information technology services are usually consciously designed. They are also often built from explicit specifications, though some are developed ad hoc. Whichever the case, it is useful to make a clear logical distinction between service providers, service users, and the interface between them, even if, when implemented, one or both of these distinctions will be purely notional, and will not be embodied in identifiable and separable artefacts like particular hardware components or particular blocks of code. Indeed, thinking about service provision in such a way, in an environment that is normally regarded as a more integrated whole, can help to improve a specification, or at least to test it and verify its validity.

This is especially so in the increasing number of cases where information technology environments and services, though originally conceived as self-contained, have to interact with external environments and services, many of which will need the distinction between providers and users to be made explicit. An instance of direct relevance to this Technical Report is where interacting entities are based upon different languages and hence different sets of underlying assumptions.

4.3 Services and language independence

The term "language-independent service (or interface) specification" means "language-independent specification of a service (or interface)", not "specification of a language-independent service (or interface)". Hence a language-independent specification of a service does not imply that the service itself is "language independent" in the sense intended here. The service specified may be relevant only to environments of particular languages.

Note - The implementation of a service which meets the specification will use some language or other, if only machine language, and so will in a sense be "dependent" on that language, but that is not the sense intended here.

Also, a language-independent specification of an interface does not imply that the service interfaced to is either itself "language independent", or specified in a language-independent way (though it may be).

A trivial instance is that of a language processor for a particular language providing a service by executing a program in that language. For one of the long-established languages (like Cobol or Fortran) the interface is the provision of input data and the output of results. The language was designed for particular forms of input and output media, presumed under the control of a human user. However, a language-independent interface specification could define the input and output in such a way that the data can come from, and the results be returned to, some other system, in general using a different language.

In a simple case like that, the user system and the interface are distinct and not closely coupled. The interface can be implemented as a "black box" which acts in the same way that a human interpreter would for two people with different languages conversing: it takes input from the client and translates it into the equivalent input for the service, and takes the output from the service and translates it into the equivalent output for the client.

In the more general case the interface might need to be embedded in the client system so that it appears to be integrated in that host environment. That environment may require invocations of the service to be expressed in more meaningful terms, not limited to the data transmitted to it and the results expected from it.

Note - One example involves the functional standards for graphics. In some languages the most suitable invocation method is a procedure call to an external library, while in others the most suitable method is the use of additional commands (keywords).

Both the simple and the general case are referred to as "binding" to the interface, though the binding is much tighter in the general case. A "language binding" to the interface binds a particular programming language (in general not the same programming language as the one used by the server), so that programs written in that language can have access to the service. A good language binding allows language users to use a style of accessing the service which is familiar to them, and will also accord with official standards for the language.

ISO/IEC TR 10182, Guidelines for language bindings, provides guidance to those performing language bindings and writing standards for them. This Technical Report provides complementary guidance to those specifying service interfaces in a language-independent way, and writing standards for them.

A useful way of looking at language independence is by considering levels of abstraction. The various elements of programming languages can be regarded as existing at three levels of abstraction: abstract, computational, or representational, where the middle, computational level can be divided into two sublevels, linguistic and operational. The linguistic elements are regarded as instantiations at the computational level of the abstract concepts, while the operational level deals with manipulation of the elements, which inevitably looks "downwards" to the realization of the elements in actual, processible entities at the representational level.

Note - The representational level does not necessarily mean the physical hardware level, or the logical level of bits and bytes; see the discussion in 4.5.1 below.

4.4 Language-independent specifications

As the preceding discussion has shown, a language-independent specification may be a service specification, specifying the service itself, or be an interface specification, specifying how the service is accessed by clients. It may of course cover both.

This Technical Report is concerned primarily with specification of the interface to the service, rather than of the service itself. The service may be predefined in a language-dependent way. How a service is specified is likely to depend to some extent on the nature of the service and its application area, so guidelines on specification of the service are definitely outside the scope of this Technical Report. However, where it is wished to produce a language-independent specification of a service, so that it can be implemented in a variety of different languages, then the guidelines presented may be useful, directly or indirectly. For example, they may draw attention to factors that should be borne in mind, and it may then be possible to adapt them to the particular circumstances.

This Technical Report therefore provides guidelines applicable in the following cases:

- specification of a service interface;
- specification both of a service interface and of the corresponding service itself, together;
- specifying from scratch (i.e. without anything pre-existing to base it on);
- specifying on the basis of an existing (probably language-dependent) service;
- specifying on the basis of an existing (language-dependent) binding.

Guidelines are grouped under various headings, dealing with different aspects. As far as possible each group is independent, in the sense that they can be referred to without necessarily working through preceding groups. Any necessary cross-references are provided.

4.5 Problems of language dependence and inbuilt assumptions

Producing a language-independent specification can present many problems, especially if starting from an existing service which was not originally designed to be language independent - typically, a service designed in and for a particular language environment. If a service is specified in the "wrong" way - it may of course not have been "wrong" in its original context - it can make producing a language-independent interface very difficult. In particular, it may depend on explicit or (more likely) implicit assumptions about the language that applications using the service will be written in. Languages that are similar in character to the original one may not have many problems, but a language-independent interface specification needs to be able to accommodate different language styles. This is one of the greatest challenges in developing language-independent specifications, whether for services or for interfaces.

Note - Examples of styles of language are: procedural, declarative, functional, interpretive, object-oriented, ..., and these are not necessarily mutually exclusive.

Such problems can still occur even if the service concerned is a new one yet to be developed. Since most service developers tend to come from a particular language environment, it is all too easy, even when consciously at-

tempting to produce a language-independent specification, to carry over implicit assumptions from that environment, simply because they are implicit and hence rarely questioned or even noticed.

4.5.1 Representational assumptions

An important class of language-dependent assumptions is that of representational assumptions. Some languages have explicit or implicit models of how language elements are represented at the hardware level, either physically or logically. Simple instances are storage of numerical values or of aggregate datatypes such as indexed arrays or character strings, or numbers of datatype Complex (assumed to be represented by two numbers of datatype Real, for the cartesian real and imaginary parts).

Such models tend to become implicit for those used to that language environment, even when the language definition makes the model explicit. Users of the language get so used to that model that they take it for granted. It is all too easy for such assumptions to get carried over into what is intended to be a language-independent specification.

Representational assumptions are not confined to the hardware level, but can occur at more abstract levels too; for example, a supposedly "language-independent" specification may use an integer datatype for a value which logically is not, or need not be, an integer. The fact that virtually all languages have an integer datatype or an equivalent is not relevant; the original language may have used the integer datatype because it was the best or only choice, but other languages may have alternatives which the original language did not. A language-independent specification should avoid requirements that constrain how things should be represented, and concentrate upon what should be represented.

Note - It is of course possible for a language-independent specification to be developed which is explicitly concerned with the representation of language elements. For such a specification the principles outlined above may not all apply - though some may still be relevant.

4.5.2 Implementation assumptions

Representational assumptions are a specific form of implementation assumption, though not all implementation assumptions are language-dependent. Service designers make implementation assumptions when they take it for granted that a particular implementation approach will be adopted. A simple example is the assumption that the service will be invoked by a procedure call or, even more specifically, will use procedure calls using a parameter passing mechanism of a particular kind.

Implementors of language-independent service specifications should not be required to adopt a particular implementation approach. Instead, the specification should require only what is needed for the service, or is needed to ensure that different implementations will be mutually consistent or (if interoperability is required) interact with one another correctly.

5 Guidelines on strategy

The discussion in clause 4 above shows that a large number of factors need to be taken into account when producing a language-independent service specification. This clause provides guidance on how to go about the task.

The guidelines that follow are divided into general guidelines (see 5.1) and more specific ones. Some of the more specific guidelines are in fact similar to one another, appearing in various modified and specific forms under various headings, and could have been made "general" guidelines. The apparent duplication increases the length of the document, but is intended to reduce the amount of interpretation and adaptation that will be needed in particular circumstances, and to emphasize the relevance in particular contexts. It also allows different Notes, specific to the context, to be appended.

5.1 General guidelines

5.1.1 Guideline: Dependence of the interface on the service

A service specification should be designed with the requirements for the language-independent interface in mind.

Note - If a service is specified in the wrong way, it can make the production of a language-independent interface very difficult, in particular when explicit or implicit assumptions are made about the languages that applications that will use the service will be written in.

An example is assuming a particular method for invoking the service, e.g. the use of object classes, or the use of low-level procedure calls (i.e. using only simple datatypes for parameters).

5.1.2 Guideline: What to do when there are interoperability, concurrency, or time constraint issues

Issues relating to interoperability with other services, or concurrency, or time constraints of other kinds may affect language-independent service and interface specifications. If this is the case, the nature of such issues makes it vital that they be addressed first, with the remainder of the service being designed later, around the aspects handling those issues.

Notes

1. Interoperability, concurrency, and time constraint issues can often cause difficulties, compared with which other issues are comparatively straightforward to deal with. They can also place requirements or constraints on other aspects of the service. It will therefore aid the design process to address those issues first. For example, if a service is to have multiple clients, this needs to be taken into account very early on.
2. Guidelines on interoperability appear in clause 9, and guidelines on concurrency appear in clause 10.

5.1.3 Guideline: Use of marshalling/unmarshalling

When specifying the way that values are communicated across the interface between the application using the language binding and the service, the marshalling/unmarshalling approach used in LIPC in relation to passing of parameters may prove useful.

Note - The marshalling/unmarshalling concept for communicating values is sufficiently general to be of use even when the service and its interface do not involve explicit procedure calling.

5.1.4 Guideline: Recruiting expertise from a variety of backgrounds

When developing a language-independent specification, every attempt should be made to recruit the involvement of, or to obtain input from, language experts from a variety of backgrounds, and also experts in language-independence issues. In any event, before the language-independent specification is finalized, arrangements should be made to get a complete draft reviewed by experts of that kind from outside the group designing the specification.

Note - Because of the particular nature of the problems involved in achieving language independence, it is preferable to choose language experts who have some experience of binding to language-independent specifications, and/or who are familiar with other languages than their own main language.

5.2 What to do if starting from scratch

It is rare for the designer of a language-independent service or interface specification to be able to "start from scratch", i.e. to be able to design without having to take into account an existing (and usually language-dependent) service or interface which is already in use (and with which compatibility is required, or expected even if not required). However, for completeness this Technical Report does need to cover the possibility. Furthermore, guidelines on what ideally might be done can serve as a benchmark against which to measure what has actually been possible, given the constraints that a pre-existing service or interface may have placed upon the design. In principle, they might even establish that it would be preferable to treat the pre-existing version simply as a prototype to be discarded.

5.2.1 General guidelines

5.2.1.1 Guideline: Avoidance of implementation assumptions

When designing a language-independent service specification, representational or other implementation assumptions should be avoided.

Notes

1. Languages differ greatly in character so a form of implementation suitable for one may be quite unsuitable for another. Furthermore some languages themselves make explicit or implicit representational or other implementation assumptions, not always consistent with those in other languages. Language-independence is therefore best assisted by avoiding all such assumptions, however attractive they may be in other respects.
2. This guideline reappears in various more specific forms throughout this Technical Report and the general question has already been introduced in 4.5. This has been done deliberately, both to stress its importance and to aid in interpreting the guideline in various contexts.

5.2.2 Specifying the service in language-independent form

5.2.2.1 Guideline: Allowing for different approaches

When specifying the service in language-independent form, it should not be assumed that implementations in every language will use the same approach, and implementations should not be required to adopt a particular approach. Instead, the specification should require only what is needed for the service, or is needed to ensure that different implementations will be mutually consistent or (if interoperability is required) interact with one another correctly.

Notes

1. It is not necessary to use an implementation model to specify requirements, whether these are needed to provide the service itself, to ensure mutual consistency, or to ensure interoperability. Such requirements can and should be expressed in an abstract, language-independent way.
2. Guidelines on interoperability appear in clause 9.

5.2.2.2 Guideline: Documenting external constraints and minimising their impact

If there are external constraints which the service is required to satisfy, these should be carefully examined to assess their impact, whether on implementation strategies for the service, or on the interface. The relevant aspects of the service should then be specified in a way which minimizes the impact of the constraints. The external constraints (including the rationale for their presence), and the steps taken in the specification to cope with them, should be documented.

Notes

1. Particular attention will be needed in the case of constraints which seem to require things to be done in accordance with some implementation model. In many cases it should be possible to avoid passing on these implementational requirements by absorbing them into the service, for example by internal conversions.
2. In general, it is preferable to leave as much as possible to implementations to handle as best they can, provided this can be done without compromising either the integrity of the service or of language independence.
3. Sometimes, the cost of an extra conversion interface will be justified by gains elsewhere, for example in resource terms or in safety or reliability terms.

5.2.2.3 Guideline: Allowing for different binding methods

When specifying the service in language-independent form, it should not be assumed that the interface will use or specify a particular binding method; rather, the specification should be neutral with respect to binding methods.

5.2.3 Specifying the interface to the service in language-independent form

When specifying the interface to the service in language-independent form, it should not be assumed that a particular binding method will be used by every language, and use of a particular binding method should not be required. The specification should require of bindings only what is to be passed across the interface, not how it should be passed.

Notes

1. Language bindings should be able to make maximum use of the facilities of the language. Assuming or requiring a particular binding method can lead to suboptimal bindings to the service and in extreme cases could make it impossible to specify an adequate binding.
2. Language bindings are also designed for many different purposes, and it can create many problems if a binding to one service is required to be markedly different from other bindings.

5.3 What to do if starting from an existing language-dependent specification

The task of producing a language-independent service or interface specification from an existing language-dependent specification is one of "reverse engineering". In general it can be expected that the original language-dependent specification will have treated the service, the interface, and the language binding as one, and will not, deliberately, have kept the different aspects separate. For a language-independent specification, whether for a service or for an interface, it is necessary to ensure that these different aspects are kept separate. Guidelines on identifying significant language-dependent aspects are provided in 5.3.1. The conversion of language-dependent features to language-independent form is addressed in 5.3.2 and 5.3.3 addresses the consequences for language

bindings. In 5.3.4 the situation is addressed where only the interface specification but not the service specification is to be made language independent.

Note - If more than one language-dependent specification exists, the following guidelines still apply, but the results for each binding should be checked against each other. Inconsistencies can be very helpful in reaching an appropriate language-independent formulation

5.3.1 General guidelines

5.3.1.1 Guideline: Identifying implementation assumptions

Any representational or other implementation assumptions in the original language-dependent specification should be carefully reviewed, and any which are derived from the particular language used, rather than dictated by the semantics of the service, should be identified.

5.3.1.2 Guideline: Identifying language-dependent terminology

The terminology used in the original language-dependent specification should be carefully reviewed from the language-independent point of view, to see if it is derived from the terminology of the particular language rather than from the service.

5.3.1.3 Guideline: Identifying aspects specified at the wrong level of abstraction

The language-dependent specification should be carefully reviewed for features which are specified at a level of abstraction that is inappropriate for the language-independent version. The review should in particular search for those at too low a level which do not involve overt representational or implementation assumptions as discussed in 5.3.1.1, but arise from the way the service has been conceived in the original language environment. Attention should, however, also be paid to any at too high a level, which may take the form of features being left unspecified because the missing aspects are taken for granted in that language environment, or because the language definition leaves such aspects implementation dependent.

Notes

1. The concept of levels of abstraction is discussed in 4.3.
2. An example of too low a level is specifying the service in terms of independent entities when in fact they naturally form fields of a Record datatype.
3. An example of too high a level is specifying a datatype without defining permitted or required ranges of values of the datatype.
4. When rectifying inappropriate levels of abstraction, care needs to be taken not to over-compensate.

5.3.1.4 Guideline: Identifying aspects derived from the language rather than inherent to the service

The language-dependent specification should be carefully reviewed for features which are not inherent to the service, but whose inclusion seems to have been prompted by the nature of the implementation language and its facilities. Particular attention should be paid to any such inessential features which could be difficult to provide in some other languages. Attempts should be made to discover how heavily users of the original specification use these features.

Notes

1. Some such features may in fact be included because they are useful elsewhere in the language, for purposes unrelated to the service itself.
2. It may be appropriate to include features of this kind in the specific language binding for the language concerned; though strictly inessential to the service, there may nevertheless be a continuing demand for them from that language community, which cannot readily be satisfied in another way (e.g. by the provision of separate services). If that is the case, the conformity rules should permit bindings to include these supplementary features, though they should not require them for all languages.