doc. nr.	ISO/IEC JTC1/SGFS N 999
date 1993-08-18	total pages
item nr.	supersedes document

Secretariat: Nederlands Normalisatie-instituut (NNI)
Kalfjeslaan 2 P.O. box 5059
2600 GB Delft
Netherlands
telephone: + 31 15 690390
telefax: + 31 15 690190
telex: 38144 nni nl
telegrams: Normalisatie Delft

Title: ISO/IEC JTC1/SGFS

ISO/IEC JTC1 Special Group on Functional Standardization

Secretariat: NNI (Netherlands)

Title : Report of the SC21 Study Group on APIs

Source : JTC1/SC21 Study Group on APIs, during its meeting in Yokohama, June 15-25 1993

Status : For information

Note :

Title: Report of the Study Group on APIs

Source: JTC1/SC21 Study Group on APIs, during its meeting in Yokohama, June 15-25 1993.

This revision of the report includes the following amendments:

Reason for change	Changes made
To respond to points made at the HOD/C meeting on 28th June 1993	Clarification of the text in the synopsis and Section 6
To correct minor errors in the text	Small changes made to section 6

Synopsis

This report considers the background and requirements for the standardization of programming interfaces.

Its main conclusions are that:

1 In order to increase the benefits to users resulting from the wider use of existing standards, there is strong requirement for the standardization of programming interfaces to those functions.

2 There is a need for a framework in which this work can be carried out, and it is proposed that a Joint Working Group be established to prepare a standard which describes:

- The Architectural Framework for the standardization of programming interfaces
- Technical Guidelines for the work
- A Conformance Methodology

This document is to be based on existing work, in particular, but not exclusively, from SC21, SC22, and SGFS, and work should begin as soon as possible.

3 In the short term, within SC21, there remains the urgent need to meet the requirements for Standardized Programming Interfaces in relation to the standards for which it is responsible, and steps are proposed for the operation of the JTC 1 Fast-track procedure whilst the longer-term work is in progress.

CONTENTS

1. INTRODUCTION

1.1. Background

In response to increasing international interest in the standardization of programming interfaces, SC21, at its Ottawa meeting in June 1992, invoked the experimental JTC1 procedures for the study of large new work areas. The study period was announced in document SC21 N7209. An interim report was scheduled to be produced at SC21's 1993 meeting in Yokohama and a final report from the 1994 meeting in Southampton.

The JTC1 procedures for the study of a large new work area require an examination of the:

* requirement for the new work area,
* relationship to existing work,
* technical approach and feasibility,
* a work plan with timetable, resource requirements and resource availability.

The first meeting of the SC21 study group was held in Boulder, Colorado, USA, on 2nd-5th November 1992. The rapporteur's report is contained in SC21 N7424. In addition to the work of SC21, the study group took into account ongoing work in SC22/WG11 and WG15, as well as SGFS and ITU-TS SG VIII.

The Study Group met again from June 16th - 25th 1993, at SC21's meeting in Yokohama, and produced this report in response to comments from National Bodies and Liaison Organisations. The report was approved by the SC21 Plenary Meeting on 30th June 1993.

1.2. Scope of this Report

Where this document refers to short-term actions, it is assumed that its scope is limited to interfaces that are relevant to OSI communications, data management and ODP functions already defined or under development.

Where it refers to long-term activities, the scope may be extended to cover interfaces for distributed processing and other areas. The intention is that, in the long term, programming interface standardization will be able to use the ODP reference model, which will be the source of a wide range of future ODP standards.

1.3. The Importance of Programming Interface Standardization

The creation of effective and well co-ordinated standards for programming interfaces is central to the growth of open distributed systems. It will enable the users and providers of a wide range of components to benefit from the simplification and cost reduction that follows from the use of such standards.

The aim in creating such a framework is to incorporate not only published standards but also to draw upon the increasingly important work in consortia and industry groups. This report identifies the steps needed to create a framework and the actions in SC21 and JTC1 to make the process happen. These are described in section 6.

There should be overall benefits to the industry and its users as a result of the massive expansion in computing that should happen as Open Systems become a reality. But without the standardization of programming interfaces, Open Systems will take longer to develop, and the benefits will be delayed. In particular:

* Integration of Open Systems components on each vendor platform will be more difficult and expensive;
* Business application software will be developed using ad hoc proprietary interfaces, increasing the costs of integration into an enterprise's systems;
* The development of standard application packages will be hindered by the lack of widely available standard interfaces;
* Distributed applications will be less portable, making systems evolution more difficult;
* Because of the lack of standardization, the skills of applications programming staff will be limited to specific platforms. More staff will be needed to maintain these skills, and as a result staffing numbers and costs will remain high.

In short, programming interface standardization will help users and vendors alike dramatically to reduce the costs and timescales involved in building open systems and the business application systems that they support. This in turn will create demand, generating larger income.

It is vital to the industry that programming interface standardization is done within a co-ordinating framework, so that individual (and sometimes competing) initiatives contribute effectively to the overall industry needs.

This framework must provide methods (a process) for identifying what the important open systems "building blocks" are, and for prioritising their standardization.

The importance of programming interface standardization has already been recognised by other bodies concerned with industry standards and by several important consortia. In developing an ISO framework for programming interface standardization, and the associated processes, we must draw on the current work of other organizations so that immediate short term guidance can be given. If early results are not forthcoming, the ISO initiative will not receive the support of the industry or of users.

1.4. Terminology

A programming interface is an interface between one software component and its supporting infrastructure. No assumption is made about the nature of the interface defined, and it is recognised that there is no single 'layer' of programming interfaces in a system. Many different components may invoke each other's services, all through programming interfaces.

The term Application Programming Interface is in common use. However, because of the use of the word 'application', the expression has sometimes been interpreted simplisticly to mean an interface between a business application, such as an order processing system, and an operating system such as is described by POSIX. The distinction between an application and its supporting infrastructure, or platform, is relative, rather than absolute. As technology advances, infrastructure becomes more sophisticated, taking over more of the co-ordinating and integrating tasks. Thus today's application components may well become part of tomorrow's infrastructure, and the distinction between an API and an interface within the infrastructure is unduly restrictive and would hinder system evolution.

This report therefore prefers the simple term Programming Interface, allowing for consistent standardization of progressively more abstract interfaces as time goes on.

The ODP Reference Model has adopted the term 'programmatic', and this report has used it in that context.

Where the interface concerned has become a standard, the term Standardized Programming Interface (SPI) has been used.

2. REQUIREMENT FOR THE NEW WORK AREA

This section discusses the requirements for the standardization of programming interfaces, and the benefits that will come from the completion of the work.

There are significant indirect cost savings from standardization. It makes skills more transferable, reduces retraining costs and simplifies user understanding of the resulting system, reducing the scope for errors in operation.

For the information technology user, the key benefits arise through the portability and reusability of software. The proper implementation of programming interface standards can allow the business to purchase many information technology components, including both hardware and software, as a commodity. Standardization of interfaces also allows the business to implement new technology in a modular fashion.

* Portability is a related concept. It allows an application program to be moved from one computer to another, without worrying about differences in operating systems, or other vendor-specific parameters.
* Reusability allows modules that are written for a particular function to be used again and again to provide that function. In concert with portability, the module can be reused anywhere in the enterprise.

When these goals are achieved, the enterprise can use information technology much more effectively and economically. Processor power and communications bandwidth can be purchased as commodities where there is no dependence on a specific vendor's architecture or implementation. New technology can be introduced quickly, when required.

2.1. Benefits of Consistency

There are a number of different aspects of system design and specification where consistency is important:

a) consistency of architecture results in system building blocks which provide a well differentiated, but coherent, set of functions.

b) consistency of architecture results in a single set of assumptions about the way these building blocks interact; it leads to a coherent treatment of the effect of operations and a single interpretation of any errors which may occur.

c) consistency of notation results in a simplification of the programming environment and reduces the variety of support tools that need to be maintained and understood.

d) consistency of both architecture and notation encourages re- use, makes it easier to identify common functions and select suitable components from the set already available.

e) consistent representation of information resources, such as filing systems or document structures, unifies the whole family of interfaces making interpretation and exchange of information simpler. The representation of such resources involves the creation of an information model to capture the properties of the resource, not just establishment of a catalogue.

The savings in total system cost outlined above can be substantial. This implies that quite small amounts of inconsistency can dominate the system integration costs. If, for example, the cost of interfacing an incompatible new component were ten times the cost of integration and testing a component chosen from a consistent family, then inconsistency in even 10% of the total number of system components would double the system cost.

In the longer term it is important that there be only one underlying architecture and only one family of notations used in JTC1 standards. Fragmentation of the set of JTC1 standards into sets which use different underlying assumptions would reduce the benefits available from JTC1 standardization. This convergence is unlikely to come about by peer liaison between the groups working in different technical areas; it requires pro-active management, which can only come from JTC1. JTC1 should therefore promote a single architectural framework for use throughout its program of work, and encourage notational consistency by encouraging the production of a set of style guides to encapsulate preferred solutions. It should require that all its technical activities be based on this framework and these guides. Only thus will consistency be achieved and the potential benefits realised.

In the short term however, until the architectural framework and guidelines are available, work on SPIs must nevertheless progress.

2.2. Benefits to specific classes of user

There are several classes of IT user which have important specific requirements for the standardization of programming interfaces.

Organisations which are the ultimate purchasers of IT equipment will be able to benefit from the standardization of programming interfaces because there will be more applications available and they will be more portable across different systems. The costs of applications, and of their integration and maintenance, will be significantly reduced, and other financial benefits will result from new business opportunities.

Software suppliers will benefit from the increased ease with which they will be able to port applications between different systems and different communications protocols, and therefore the wider markets to which they will be able to promote their products.

The suppliers of computer systems and the associated systems software will benefit from the market growth that will result from the increased availability of applications and the increased confidence of users in the value of their investment.

For systems integrators the costs of integration and operation depends, in large part, on the degree of consistency between system components. Variation in style, divergence of architectural assumptions, variety in information models used, or mismatch of functional packaging, all require special action at integration time and so increase cost. It is much easier to integrate components that are designed within a single architecture, reference a single set of information resources and are specified using a single set of descriptive tools.

2.3. The need for balance

It is not easy to reconcile the need for work to be carried out within a proper framework with the imperative for standards to be made available in a timely fashion. This report has attempted to create a balance between the two by proposing steps for short-term activities whilst the framework is being agreed.

3. RELATIONSHIP WITH EXISTING WORK

3.1. Activities outside JTC1

In ITU-TS, recommendation T.611 defines a high-level Programming Interface for applications (word processing, spreadsheets), wishing to use communication services such as fax, telex, or teletex.

In addition, current work throughout the industry on Programming Interface specification has made available considerable experience and feedback on what may be considered good practice in the area. The existing (non-standardized) Programming Interfaces are providing building blocks widely used in the industry. In certain areas, they provide stable development environments and portability of applications between different vendors hardware and software. Yet areas for improvement have been identified.

Many organizations are actively developing interface specifications, but the work should be complemented with an overall plan. The problem for users is to select Programming Interfaces that will complement one another to accomplish the user's objectives. For example, a user who needs a transaction oriented RPC mechanism needs some guidance on the selection of the appropriate Programming Interface.

3.2. JTC 1 activities

In order to prepare a plan of work it is essential to establish programming interface standardization within the context of the overall programme of JTC 1. This is necessary both to establish the requirements for activity and to ensure consistency between programming interface standardization and other work. There are three areas to consider: base standardization, activities relating to Open System Environment (OSE) profiles in SGFS, and work on conformance.

3.2.1. Existing Programming Interface Standardization

In SC 24 there has for some time been work completed on the standardization of programming interfaces for graphics systems (GKS and PHIGS) and of language bindings to these interfaces.

In SC22 the POSIX standard ISO 9945-1 describes Programming Interfaces to Operating System functions.

There are a number of areas of base standardization where the definition of Programming Interfaces for access to the defined functionality is already an integral part of the activity. This applies:

* for work on graphics in SC24;
* for work on database in SC21, except for work on RDA (Remote Database Access).

At present, in some cases these interfaces are defined in abstract terms with specific language bindings defined from the abstract definition. In others the definitions are in terms of one or more specific language bindings only.

3.2.2. Potential Programming Interface standardization

The main area in which there is significant base standardization but, as yet, no formal standardization of Programming Interfaces, is that of OSI. OSI standards normally involve a service definition but this is intended as a means of identifying service users and providers. In general the specification of Programming Interfaces in this area requires agreement on the functionality to be supported by the protocol (for the FTAM standard, for example, there are a range of options from simple file transfer between two applications to fully functional remote file access).

The Reference Model of Open Distributed Processing (ODP) is now at the CD ballot stage (CD 10746 Parts 2 and 3) and the resultant programme of specific ODP standards is now being planned. These standards will define components of functionality to support system distribution. Where it is agreed that interfaces to these components should (potentially) be visible it will be necessary to define appropriate Programming Interfaces. It is currently assumed that the base standards will themselves provide abstract interface specifications and that, where visibility is required, it will only be necessary to develop appropriate language bindings. It is clearly essential for SC21 that there is alignment between guidelines for Programming Interface standardization and the requirements of ODP.

3.2.3. Relationship with IDNs and FDTs

There are currently coordinated activities on Interface Definition Notations (IDNs):

In SC 22, on
* Common Language Independent Data types (SC22 N970, CD 11404)
* Common Language Independent Procedure Calling Mechanisms (SC22 N1082, CD)

In SC 21
* as part of the work on the RPC (Remote Procedure Call) Model (CD11578-1)
* on an RPC Interface Definition Notation (SC21 N7348, CD11578-2).

Work on the standardization of programming interfaces will benefit from these activities: when mappings from the IDN to programming languages become available, all interfaces defined using the IDN will also be expressed in the form of a derived SPI specification with language bindings.

Having formal specifications of interfaces is, in general, very important to achieve a high degree of precision and absence of ambiguity in interface descriptions. SC21 and SC18 have defined several Formal Description Techniques (FDTs). In general, FDTs may vary widely, some requiring process description and others only data description. While work on Programming Interface standards will benefit from the underlying use of formal description techniques, standardization of FDTs is outside the scope of Programming Interface standardization.

3.2.4. OSE Profiles

Following the report of TSG-1, Standards necessary to define Interfaces for Application Portability, SGFS was charged with responsibility for extending the work on International Standardized Profiles (ISPs) for OSI to address the more general requirement identified in the report for profiling sets of base standards. Such profiles are termed OSE profiles.

Work is in progress on the extension of TR 10000, Framework and taxonomy for International Standardized Profiles, to cover OSE profiles. It is clear that Programming Interface standards will be an essential element of OSE profile specification both in terms of the functionality offered by the profile and in terms of how specifications of component functionality can be combined. The OSE profile work will also provide a process for identifying the important Open Systems building blocks. Thus, as with ODP, it is essential that there is alignment between any long term guidelines for Programming Interface standardization and the requirements of OSE profiling. It is also important that requirements for specific Standardized Programming Interfaces arising from OSE profiling are taken into account in any programme of work.

3.2.5. Conformance requirements

Conformance issues for ODP standards are discussed in Part 2 of the ODP Reference Model. There is a need to consider statements of conformance for programming interfaces, and the relationship between multiple programming interfaces and between programming interfaces and other interfaces for an ODP component. The New Work Item proposal on Open Systems Assessment Methodology (SC21 N8010) promises interesting results.

Work on Profile Conformance is also underway in the Regional Workshops. This work will be fed into SGFS.

In SC22 development is beginning of a standard for Test Methods for POSIX.

All this work should be taken into account in formulating conformance requirements for Programming Interface standards.

4. TECHNICAL APPROACH AND FEASIBILITY

4.1. The nature of Interface Specifications

There is a need for a framework which allows the interfaces in an Information System to be described in a way that is independent of whether the interfaces are local or distributed.

Interfaces specifications involve a number of levels of abstraction. The process of high level design produces a description of the desired system in terms of a set of functions and the interactions between them. The interactions take place at interfaces. The division of the system into functional components interacting at interfaces gives modularity and hides heterogeneity, making component reuse simpler. At this high level, the distinction between interfaces linking machines in a distributed system and interfaces linking software components within a single physical machine is not important. However, these distinctions do become relevant as the system design is refined, placing functions at certain locations, and then labelling each of the interactions with the type of communication involved.

The labelling takes the form of assigning each interaction to a specific logical location, known as a reference point, where the operation of the system could, in principle, be checked. A reference point may be programmatic (between software components) or interworking (involving physical communication), or may involve human interfacing or exchangeable media.

However, one single high level specification of an interface can capture the common features of both local and remote interfaces. This interface specification is not trivial; it must capture the operations that can be performed at the interface, the state of the communicating objects affected by the operations and the circumstances in which the operations are appropriate. The complete specification of an interface may, with advantage, be structured to reflect a number of different design concerns relating to specific behaviour, consistency or enterprise policy. The concept of viewpoint simplifies this structuring (see Annex A). The high level description may then be refined into distinct, more specific descriptions of the detail of programmatic, interworking or other forms of interactions.

In general, an interface specification will consist of two parts: the specification of the behaviour observable at the interface and the specification of the context within which the function being accessed operates, so that the overall policy or management constraints need not be expressed explicitly in each programmatic interface.

In order to meet the requirements identified in Section 2 the following approach is recommended:

4.2. Architectural Framework

An overall framework is needed to assist in the co-ordination of standardization activities and in the use of those standards, once produced. The framework should assist in the architectural placement of programming interfaces, and should guide the selection of candidates for standardization.

In the construction of a standardized open system environment, it is crucial that reference points and associated interface specifications be organised and structured to avoid complexity, unnecessary duplication of functionality or potential conflicts between independently specified interfaces. The aim must be to provide users with a clear, consistent and integrated view of the environment. This calls for a consistent architectural framework in which interfaces can be identified and positioned relative to one another. This architectural framework will evolve as technology matures, allowing definition of progressively higher-level interfaces. Such an architectural framework ought also to describe what composition or combination of interfaces are possible and meaningful.

The basis of the framework should be a set of technical concepts necessary to ensure semantic consistency across interface definitions. In particular, technical consensus is required on the concepts of interface, interaction, state, and behaviour.

Several kinds of reference points may coexist in a computer system: programmatic, interworking, perceptual, and interchange reference points. Standardized interfaces exist and can be developed for these different kinds of reference points. In specifying a system, it is very important to make explicit, and to understand, the relationships that exist between these different kinds of interfaces, since an interface, defined in abstract terms, could be realised in different ways. A framework is thus needed which distinguishes between these various kinds of interfaces and describes the relationships that may exist between them. This may contribute to the future definition of an ISP taxonomy for programming interfaces.

The framework must be described in ways that are not only meaningful to those involved in the processes of standards development, but also to users, so that the guide can be used in procurement.

4.3. Technical Guidelines

Within the context of the Architectural Framework described above there is the need for guidelines for the definition of programming interface standards. Programming Interfaces should in general be defined in an abstract (language-independent) form, complemented by language specific binding rules which allow the generation of language specific specifications from the language independent form. Such rules will in future permit the automation of the process.

This abstraction is essential to reach the goals of portability and interoperability in an heterogeneous environment, where sources of heterogeneity can be at hardware, operating system, communication, or programming language levels.

The producers of such specifications need clear guidance on what is necessary for standardization. This guidance should be consistent across all SubCommittees of JTC 1, and if possible even more widely accepted. Without such consistency there will be confusion for both the developers of standards and the users of the resultant documents.

The Framework should specify what are the requirements for:

* The form, and timing for the production, of a language independent specification.
* The structure, content, and vocabulary to be used in specifications.

4.4. Conformance Methodology

A conformance methodology is needed which makes clear, for a programming interface standard, what can be tested of a product.

It should specify that the product has to execute certain actions on the interface in response to actions executed on the same interface by the partner on the other side of the interface. These actions must be executed in a syntactically correct and timely way. The actions executed by the partner must be accepted if they are themselves syntactically correct and timely. Errors specified by the standard must be detected and reported as specified.

This methodology should remain simple, be provided quickly and take advantage of the existing practice in the area of OSI and language standards.

The compliance of an SPI standard to:

* the framework for SPIs,
* language independent specifications,
* languages (for the language bindings).

is to be studied. This should remain a longer term task with minimal impact on the short term activities.

4.5. The SC21 Contribution

The Reference Model for ODP (RM-ODP) provides the means to position programming interfaces with respect to other interfaces and functions, and to create a taxonomy for them. It also offers the basis for a clear interpretation of conformance to such interfaces.

The RM-ODP viewpoint languages provide a rich tool kit for interface specification. They independent of any programming languages. A language independent specification helps ensure that representations of the interface in different languages have consistent semantics.

More specifically, the ODP Reference Model provides the following answers to some of the technical requirements identified in section 3.2:

Part 2 of the RM-ODP provides a definition of the interface concept, as well as a definition of associated concepts such as interaction, interaction point, behaviour and state. The framework of abstractions in Part 3, which identifies five viewpoints on open distributed systems, provides a means to identify and separate general concerns that may intervene in the specification of an interface. The set of Viewpoint Languages defined in Part 3 of the RM-ODP provides a single interface specification technique. The ODP Computational Language, in particular, provides the means to define interface types and behaviour in a manner independent of actual localization of interfaces and of programming languages (which can be used if interfaces appear at programmatic reference points).

Part 2 of the RM-ODP provides a general conformance framework that identifies the different types of reference points that may coexist in an open distributed system, and discusses the conditions under which testing a system that claims conformance to several different interface specification standards (possibly at different types of reference points) can be done.

Part 3 of the RM-ODP, and in particular the Engineering Language, provides a general structure where several interfaces are identified and defined. services identified in the ODP engineering model cover functions such as:

* configuration and identification of interfaces;
* management of processing and communication resources;
* object and interface management;
* provision of multiple distribution transparencies, including access, location, migration, concurrency (transaction), replication, resource and failure transparencies.

Part 4 of the RM-ODP (Architectural Semantics) provides an interpretation of the basic concepts associated with the notion of interface using existing FDTs standardized by SC21.

The Interface Specification technique defined in the RM-ODP masks heterogeneity and allows expression of parallelism and exception handling while hiding details of associated control structures. A common interface specification style guide using this technique will also be defined. In addition, guidelines are also needed for the mapping of this interface specification technique to programming languages. The applicability of SC22's DTR 10182 will be investigated.

The interface specification technique and the attendant specification style guide and language mapping guide will allow the systems builder to integrate components of a distributed application from different vendors into a coherent system.

The RM-ODP has defined the four types of interface reference points, identified previously, through which conformance tests could be performed to ensure interoperability between distributed applications and to support system integration. Liaison between ODP experts and the conformance testing rapporteur group has been on-going. This liaison has resulted in work on the requirements for an ODP conformance testing methodology. It is anticipated that SC21 will define the methodological framework for ODP conformance testing, which will ensure a consistent approach to conformance testing of all interface reference points for an open distributed system.

In summary, RM-ODP provides a consistent framework for specifying programming interfaces of components and their placement in a distributed computing system, using a single standardized specification technique and a common style guide. This framework will allow implementors freedom to build the components using their preferred programming languages. As the software components are designed and built in accordance with the RM-ODP, they will be assured of interoperability, portability and reusability and that they can be easily put together to work as a coherent whole. the assurance will be qualified in accordance with the framework for ODP conformance testing methodology.

5. ORGANIZATIONAL ISSUES

5.1. Responsibilities

Where a base standard will require SPIs, JTC 1 should encourage the development of Language Independent Programming Interfaces in parallel with the development of the base standard, so that they can be closely coordinated. It is necessary to recognise, however, that significant programming interface development has already taken place outside JTC 1, and that this is likely to continue in the future - in particular for interfaces that address higher levels of abstraction than the base functionality.

There is a need for guidelines on the location within JTC 1 of work on the actual standardization of programming interfaces. There may be a need to involve several SCs in a cooperative development. However, on balance the work will be most effectively carried out if it is led by the group with the strongest interest in its success, which would normally be the SC that has responsibility for the underlying base standards. It is acknowledged however that a programming interface is not necessarily specific to a particular protocol stack, that there can be selection of the architectural levels at which standardization can take place, and that therefore the identification of the responsible group may not therefore always be clear.

Responsibility for activity on language bindings for specific programming languages may lie with either the specific programming language group, or the base standard group, or be shared by them in some way.

5.2. The use of the JTC 1 Fast Track Procedures

There are two areas were there would be value in modifying the Fast Track procedures to meet the needs of Programming Interface standardization.

First, the current Fast Track procedures permit only two extreme choices when balloting a document: affirmative or negative. If the vote is affirmative, comments may be given but there is no requirement for them to be accepted. If the vote is negative, the entire project is abandoned or must be restarted. This means that the process will generally fail where there exists technical consensus, but there is a need for alignment with JTC 1 style or a need for limited, identified, and agreed, technical change for alignment with other standards. It is proposed that a third, middle of the road alternative could greatly expedite progression of a number of documents to JTC 1 standards status. This third choice is "basic technical agreement, but alignment is required". This would permit national bodies to indicate their general concurrence, but submit comments that must be addressed prior to final acceptance.

Second, the process of submission of the document could be improved on the basis of lessons from the ISP process. Thus, it would be desirable:

* for the submitter to be encouraged to make JTC 1 aware at an early stage of their intent - this would allow early agreement on SC responsibility for subsequent action in JTC 1;
* for the submitter to be encouraged, at an appropriate stage, to make documentation available for informal review by experts from the relevant SC(s);
* for the submitter to be required to provide, with the document submitted for processing, an explanatory report to be circulated with the document covering:
  - the extent of and nature of international awareness of, and support for the document,
  - the degree of alignment with related JTC 1 standards and standards architectures,
  - the degree of alignment with any JTC 1 guidelines for the specification of APIs, and
  - proposals for maintenance and future development.

It is recognised that JTC1 is already addressing some of these procedural requirements, both by adoption of the JTC1 Berlin Resolutions 35 and 36 and through the referral to the JTC1 SWG-P of enhancements to the fast track process. SC21 should request JTC1 to direct the SWG-P to take the complete set of requirements, as presented here, into account when modifying the JTC1 Directives.

5.3. References to documents other than ISO, IEC and ITU-TS standards

In order to facilitate standardization of SPIs, source material from organisations other than ISO, IEC and ITU-TS needs to be considered. The principle guiding the definition of the program of work is that these references should identify potential input documents to the standardization process, rather than additional sources of information for the users of standards and ISPs.

Although there are many projects with potential, this report has not attempted to catalogue them without input from those involved. However, it is recommended that in preparation for longer-term work, an investigation be carried out into the requirements and plans of interested groups.

6. PROGRAM OF WORK

There is a body of work that must be accomplished, which this section describes. The longer term work should begin as soon as possible, but should not delay the implementation of the short term activities. As work proceeds on the long term study it may be possible for some of the intermediate results to be used to guide the ongoing conduct of the short term work, in particular the selection and presentation of documents that may be appropriate for Fast-Track.

6.1. Proposals for short term work within SC21

In the short-term, National Bodies should be encouraged to identify suitable candidates for Fast-Track standardization related to the standards for which SC21 is responsible, and should initiate appropriate discussions with industry groups.

Appropriate specifications could be standardized using the JTC 1 Fast-Track procedure. This may be applied either by a National Body or Liaison Organisation initiating a Fast-Track, or by SC21 following Resolution 35 of the JTC 1 meeting in Berlin, March 1993, and suspending the 5-stage process in favour of a Fast-Track. It is not clear whether the former case requires that the specification concerned be a national standard (in the case of a National Body submission).

Organisations intending to make use of the Fast-Track procedure are encouraged to make their plans known well in advance in order to gain consensus.

6.1.1. Proforma Information to accompany an NP or Fast-Track

If a document proposed for an NP or Fast-track originates in an industry group, the group concerned shall be asked to complete a proforma which would provide the following information about the specification:

1 The user requirement for the specification, and any accompanying evidence from user organisations.
2 A description of the process by which the specification was developed and agreed, and an indication of the level of consensus that has been achieved.
3 The degree of alignment of the specification with standards, including relevant base standards, architectures and models.
4 An assurance that the specification is complete, and would enable a third party to implement a supporting product without the necessity to use infrastructure functions which are not covered by standards.
5 Any commercial interest which the submitter may have in the specification.
6 The level of stability of the specification, and any expectations of change.
7 Assurance that the copyright requirements of ISO/IEC have been met.
8 The willingness of the submitter to provide support for maintenance activities during the process of standardization, and once the specification has become a standard.

In addition, it is proposed that any National Body or Liaison Organisation proposing an NP or Fast-track shall be requested to supply a statement that addresses the following questions:

a) Which SC(s) are responsible for the underlying function(s)?
b) Which SC(s) are responsible for the programming language(s)?
c) Will the Programming Interfaces specification require extensions to an existing programming language or service?
d) What is the kind of expertise required in the development of the Programming Interface specification?
e) What is the relationship of the Programming Interface specification work to other work of the SC(s) involved?

6.2. Long Term Activities, suggested for initiation by JTC 1

Action is needed to provide guidelines which cover the requirements described in section 4. As the guidelines would be touching on disciplines from many areas it is suggested that they be developed in a Joint Working Group administered by either SC21 or SC22. The Joint Working Group should have responsibility for producing a new Standard covering the following points:

6.2.1. Architectural Framework

This should identify the architectural placement of programming interfaces and guide the selection of candidates for standardization. Input documents to be considered would include:

SGFS: TR10000-3: Principles and Taxonomy for Open Systems Environment Profiles
SC22/WG15: Guide to the POSIX OSE
SC21/WG7: Reference Model for Open Distributed Processing.
and any additional input from other SCs, such as SC6, SC18 or SC24.

The aim of the work should be to agree one document based on the above work, and to relate it to the other documents. While SC21 would recommend that RM-ODP be the primary architecture, it commits itself to strenuous liaison to ensure convergence, and will contribute positively to the work.

6.2.2. Technical Guidelines

This should provide guidance for the definition of Language Independent Specifications and Language Bindings within the Architectural Framework. Input documents to consider will be:

SC22: DTR 10182
CD 11404 Common Language Independent Data types (SC22 N970)

CD on Common Language Independent Procedure Calling Mechanisms (SC22 N1082)

SC 21: CD 11578-2 Interface Definition Notation (SC21 N7348)
Potential output from WG7 on a specification style guide

The guidelines should reflect the requirement, stated in JTC 1 N2319, that when a standardization project for an Standardized Programming Interface includes multiple language bindings with common interface characteristics. the use of language independent specifications should be strongly encouraged.

The guidelines should also consider the possibility of accepting as base documents specifications which are expressed as language bindings, with the work of transforming the standard to a language independent form being carried out in parallel with the production of a second language binding.

6.2.3. Conformance Methodology

This methodology should describe the requirements for the conformance or compliance to:

* the framework for SPIs,
* language independent specifications (if any),
* languages (for the language bindings).

working on the basis of the OSI conformance standards, primarily ISO 9646, a newly-created work item on ODP conformance, and the output of the activities on Open Systems Conformity Assessment. Other inputs will be the activities on POSIX Conformance Testing, and work on Profile Conformance Testing from SGFS. It is important that there be convergence of this work in order to ensure that there is integration of the testing of SPIs, protocols, and other reference points.

6.2.4. Information Gathering in preparation for the Joint WG.

If the proposal to establish a Joint Working Group is agreed, it is suggested that steps be taken by JTC 1 to provide the Joint Working Group with additional input. Another benefit could be to guide the selection of appropriate specifications for Fast-Track. This could take the form of a request to each SC to provide input on these questions:

1. What are, as perceived by the projects within your group, the requirements and priorities for standardization of Programming Interfaces?
2. Have you identified any Programming Interfaces existing in the industry that may meet some of the requirement?
3. Have you already taken any actions in this regard or is there any that you would recommend?
4. Do you have any plans of particular interest for the area of standardization of Programming Interfaces?

6.3. Suggested Timetable for Long Term actions initiated by JTC 1

The timetable below assumes the agreement of JTC1 to form a Joint Working Group, and begins from the point at which the Group is formed.

It is assumed that all these activities are the responsibility of the Joint Working Group.

Month	Architectural Framework	Technical Guidelines	Conformance Methodology
6	Gather and review inputs, agree the requirement for components of the Framework and discuss criteria for the selection of the components. Request comments and input from NBLOs.	No activity	No activity
12	Meeting to decide which parts of the input documents best match the criteria for the components of the Framework. Request input from NBLOs on the conclusions of this work. First Working Draft.	Start Work	Start Work
18	Meeting to resolve issues. Second Working Draft.	Meeting to review and assess input documents, identify the issues, discuss and agree wherever possible. Ask for input from NBLOs.	Meeting to review and assess input documents, identify the issues, discuss and agree wherever possible. Ask for input from NBLOs.
24	Committee Draft	Working Draft	Working Draft
30		Committee Draft	Committee Draft
36	Draft International Standard		
42		Draft International Standard	Draft International Standard
48	International Standard		
54		International Standard	International Standard

If appropriate the opportunity could be taken to publish some of the intermediate results as Technical Reports. This could have the benefit of making documents publicly available up to 2 years before the corresponding standard.

6.4. Resource requirements and availability

Resource requirements	Resource availability
Secretariat, Convenor of JWG alternative 1: JWG created as Rapporteur Group within existing WG of an SC (may be interim, start-up phase) alternative 2: JWG created as separate group	 to be found to be found
Rapporteur (3 projects)	to be found
Editor of Architecture Framework	to be found
Editor of Technical Guidelines	to be found
Editor of Conformance Methodology	to be found

ANNEX A: INTERFACES AND THE RM-ODP

A.1 Introduction

The purpose of this annex is to provide some background on the basic Reference Model of Open Distributed Processing (RM-ODP) and explain, at least in part, how it relates to the definition of programming interfaces. While this is a technical subject, an attempt has been made to avoid unnecessary jargon and to present the material in a tutorial fashion so it will be accessible to the widest possible audience. This material is not a substitute for the RM-ODP and simplifications have been made in the interest of readability.

This annex begins by examining the objective of ODP and showing how that objective is inextricably related to the problem of interface definition. We then explore the RM-ODP approach to interface definition, and to programming interfaces in particular.

A.2 The Objective of ODP

The objective of ODP is to enable distributed system components to interwork seamlessly, despite heterogeneity in equipment, operating systems, networks, languages, data base models, or management authorities. an ODP system must supply the mechanisms which mask underlying heterogeneity from users and applications. these mechanisms will address a set of fundamental transparency properties, including:

* access transparency, which masks differences in data representation and invocation mechanisms between heterogeneous computer architectures, programming models and networking protocols.
* location transparency, which masks changes in configuration of application components, and enables the transfer of configuration-independent interface references between components.
* migration transparency, which masks dynamic relocation of components from both the components themselves and their clients.
* concurrency transparency, which masks scheduling of invocations of operations that act on shared state.
* federation transparency, which masks interworking boundaries between separate administration domains and heterogeneous technology domains.
* liveness transparency, which masks the automated transfer of components from active to passive state or vice versa.
* resource transparency, which masks variations in the ability of the local ODP infrastructure to proved the resources for an application component to engage in interactions with other remote components.
* failure transparency, which masks recovery of failed components, thereby enhancing fault tolerance.
* replication transparency, which masks replication of components, thereby enhancing performance and availability.

in order to achieve these ambitious goals, the RM-ODP must accomplish two things:

* RM-ODP must prescribe an integrated set of functions that can provide the required transparencies. We refer to the realised set of software components that provide these transparencies as the ODP infrastructure. The recommendations/international standards that will be developed under the umbrella of the RM-ODP will standardize the ODP infrastructure components. Distributed application components will interoperate through the ODP infrastructure. The ODP infrastructure is the platform that will make network computing a reality.
* RM-ODP must provide a technique for the specification of interfaces. The ODP infrastructure will allow client application components to access a server no matter where the clients are located in the network, no matter what programming languages were used for the clients or the server, no matter what local operating systems are involved, etc. The components of a distributed system might be developed by different teams, at different times, using differing technology q yet the components must interwork. It is therefore essential for developers of a client component to have a precise specification of the server's interface; the specification must be unambiguous and implementation independent. At run-time, the interface specification is the vehicle for ensuring that the interface expected by the client is compatible with that offered by the server, so that the infrastructure can effect a type-checked binding. finally, the recommendations and international standards for components of the ODP infrastructure must be rigorously specified by defining their interfaces.

A.3 Reference Points and Conformance Points

As we have seen, the problem of how to define interfaces has been central to the development of the RM-ODP. A related problem is the broad categorisation of interfaces based on their architectural placement. One of the tasks of any reference model is to specify reference points in the architecture. The RM-ODP identifies four types of reference points, any or all of which may be specified as conformance points in a particular standard or specification. They are:

* Perceptual reference point, at which a human-computer interface can be established. for example, a perceptual reference point might be established in a graphics standard.
* Interworking reference point, at which a communication interface can be established between two systems. OSI standards are based on the interconnection of interworking reference points (the physical medium).
* Interchange reference point, at which an interface to an external physical storage medium can be established. an interchange conformance requirement is stated in terms of the behaviour (access methods and formats) of some physical medium so that information can be recorded and then physically transferred, directly or indirectly, to be used on another system.
* Programmatic reference point, at which a programming interface can be established to allow access to a function. an interface at a programmatic reference point corresponds to the common notion of an API.

We believe that this categorisation of interfaces by reference point is consistent with the approach taken by X/Open and POSIX OSE. Although these undertakings recognise corresponding reference points in the architecture, they place programming interfaces between the application component and the infrastructure. ODP's object-oriented approach recognises that programming interfaces can also exist between two application components; it is a question of what conformance statements are made in a particular specification.

A.4 The Complexity of Interface Specification

An interface is a complex thing. Its behaviour can be described in various ways. It can be considered on a purely programming level, on the basis of the data types carried across the interface. It can be considered on a semantic level, on the basis of the information conveyed by the interaction. It can be considered on the basis of the future behaviours that may be enabled or inhibited as a result of the exchange.

One of the hallmarks of the RM-ODP approach to interface specification is the recognition that none of these approaches is complete by itself, that several descriptions of an interface in different abstractions are necessary for a complete definition of what the interface does and how it works. The RM-ODP has formalised these different abstractions and called them viewpoints.

A.5 ODP Viewpoints

A viewpoint is an abstraction mechanism. The adoption of a particular viewpoint allows a perception of a system which emphasises a particular concern, while ignoring other characteristics that are temporarily irrelevant to the concern at hand. RM-ODP recognises five viewpoints. Many others are possible, but these are sufficient for a complete description of an ODP system:

* enterprise viewpoint is concerned with the social, managerial, financial, and legal policy issues which constrain the human and machine roles that comprise a distributed system and its environment.
* information viewpoint concentrates on information modelling, flow and structure, and information manipulation constraints.
* computational viewpoint focuses on the structure of application components and the exchange of data and control among them.
* engineering viewpoint concerns the mechanisms that provide the distribution transparencies to the application components.
* technology viewpoint focuses on the constraints imposed by technology, and the realised components from which the distributed system is constructed.

The viewpoints do not comprise a layered architecture. There is no inherent ordering among the viewpoints, and most especially there is no implied methodological sequencing. the viewpoints are qualitatively different from one another; within each viewpoint, many levels of abstraction (detail) are possible.

The computation, information and enterprise viewpoints are used for the specification of a programming interface.

A.6 An Example Programmatic Interface Description

To illustrate the importance of the viewpoints in the definition of an interface, consider a simple hotel reservation system. The job of the system is to allocate rooms in advance to potential customers. To do so, it must interact with other systems belonging to customers (or to customer's travel agents). That is, it offers a programming interface to its clients:

* From the computation viewpoint, a computational object charged with making room reservations offers a server interface to client computational objects. This simple interface supports a single operation. A client that invokes the operation must supply certain parameters, and will receive certain parameters as a response. The types of the parameters passed form a signature for the interface. For example, the request might include check-in date, number of nights desired, customer name and credit card number. The server responds with a room price and confirmation number, or a reservation rejection. The computational specification tells us the data types that can legitimately be passed across the interface.
* From the information viewpoint, a successful exchange results in an increment in the cardinality of the class of reserved rooms for specific nights, a corresponding decrement in the class of available rooms for the same nights, additions to the classes of customer names and credit card numbers on file, the creation of a relation between the customer, the room night and a unique confirmation number, and the removal of the confirmation number from the of the class of available confirmation numbers (depending on the complexity of the information model, there may be a great deal more that happens as well). The information viewpoint specification models the universe of discourse that the customer and the hotel must share if the exchange is to be meaningful; it tells us what it means to have a reservation and to have a confirmation number.
* From the enterprise viewpoint, a successful reservation means that the hotel has incurred an obligation to have a room available should the customer choose to check-in on the specified day up to 4:00 p.m. The interaction has altered the deontic state of the hotel with respect to this customer. Deontic state refers to the future behaviours that are required, or permitted or forbidden at any given time. the hotel is now required to have a room available. The enterprise specification tells us what the interaction means in terms of real world behaviour and how our expectations may have changed.

Now let us alter the example slightly. The programming interface offered by the reservation application now supports 'guaranteed- late-arrivals'. Suppose the customer requests a guaranteed room:

* From the computation viewpoint the interaction is the same, except that an extra boolean value passed to the room reservation object is set to true. (gla=true).

* From the information viewpoint, the hotel is justified in booking the room revenue (i.e., increasing the cardinality of the class of rooms actually sold).
* From the enterprise viewpoint, not only has the hotel incurred a responsibility to have a room available no matter what time the customer checks-in, but the customer has incurred an obligation to pay for it whether he or she shows up or not. in other words, a change of one boolean value in the computation viewpoint has profound implications for the deontic states of the objects in the enterprise viewpoint.

Complete definitions of programming interfaces must include computation, information and enterprise viewpoint specifications.