INTERNATIONAL STANDARD

ISO 16100-2

First edition
2003-11-01

Industrial automation systems and integration — Manufacturing software capability profiling for interoperability —

Part 2: Profiling methodology

Systèmes d'automatisation industrielle et intégration — Profil d'aptitude du logiciel de fabrication pour interopérabilité —

Partie 2: Méthodologie d'élaboration de profils

Reference number
ISO 16100-2:2003(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

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

ISO 16100-2 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 5, *Architecture, communications and integration frameworks*.

ISO 16100 consists of the following parts, under the general title *Industrial automation systems and integration — Manufacturing software capability profiling for interoperability*:

— *Part 1: Framework*

— *Part 2: Profiling methodology*

— *Part 3: Interface protocols and templates*

— *Part 4: Conformance test methods, criteria and reports*

Introduction

The motivation for this International Standard stems from the industrial and economic environment noted in the strategic plan of ISO/TC 184/SC 5, in particular:

a) a growing base of vendor-specific solutions;

b) user difficulties in applying standards;

c) a need to move to modular sets of system integration tools;

d) a recognition that application software and the expertise to apply that software are assets of the enterprise.

ISO 16100 (all parts) is an International Standard for the computer-interpretable and human readable representation of a software capability profile. Its goal is to provide a method to represent the capability of manufacturing software relative to its role throughout the life cycle of a manufacturing application, independent of a particular system architecture or implementation platform.

Industrial automation systems and integration — Manufacturing software capability profiling for interoperability—

Part 2:
Profiling methodology

1 Scope

This part of ISO 16100 specifies a methodology for constructing profiles of manufacturing software capabilities, and is applicable to software products used in the manufacturing domain.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 16100 (all parts), *Industrial automation systems and integration — Manufacturing software capability profiling for interoperability*

REC-xmlschema-1-20010502, XML Schema Part 1: Structures — W3C Recommendation 02 May 2001

REC-xmlschema-2-20010502, XML Schema Part 2: Datatypes — W3C Recommendation 02 May 2001

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 16100-1 and the following apply.

3.1
association
semantic relationship between two or more classifiers that specifies connections among their instances [ISO/IEC 19501-1]

3.2
base specification
base standard or widely accepted and available specification

3.3
capability class
element within the capability profiling method that represents software unit functionality and behaviour with regard to the software units role in a manufacturing activity

3.4
capability profile integration
process in which two or more software units interoperate using equivalent interfaces that are configured in a compatible manner as indicated by their capability profiles

3.5
classifier
mechanism that describes behavioural and structural features [ISO/IEC 19501-1]

NOTE Classifiers include interfaces, classes, data types, and components.

3.6
element
atomic constituent of a model [ISO/IEC 19501-1]

3.7
entity
any concrete or abstract thing of interest [ISO/IEC 10746-2]

3.8
interface
abstraction of the behaviour of an object that consists of a subset of the interactions of that object together with a set of constraints on when they may occur [ISO/IEC 10746-2]

3.9
object
model of an entity [ISO/IEC 10746-2]

NOTE An object is characterized by its behaviour and by its state. An object is distinct from any other object. An object is encapsulated, i.e. any change in its state can only occur as a result of an internal action or as a result of an interaction with its environment. An object interacts with its environment at its interaction points. Depending upon the viewpoint, the emphasis may be placed on behaviour or on state. When the emphasis is placed on behaviour, an object is informally said to perform functions and offer services (an object which makes a function available is said to offer a service). For modeling purposes, these functions and services are specified in terms of the behaviour of the object and of its interfaces. An object can perform more than one function. A function can be performed by the co-operation of several objects.

3.10
profile
set of one or more base specifications and/or sub-profiles, and, where applicable, the identification of chosen classes, conforming subsets, options and parameters of those base specifications, or sub-profiles necessary to accomplish a particular function, activity, or relationship

NOTE This definition is adapted from ISO/IEC TR 10000-1.

3.11
role
named specific behaviour of an entity participating in a particular context [ISO/IEC 19501-1]

NOTE A role may be static (e.g. an association end) or dynamic (e.g. a collaboration role).

3.12
taxonomy
classification scheme for referencing profiles or sets of profiles unambiguously [ISO/IEC TR 10000-1]

4 Abbreviated terms

CORBA	Common Object Request Broker Architecture
IDL	Interface Definition Language
OMG	Object Management Group
PSL	Process Specification Language
UML	Unified Modeling Language
XML	eXtensible Markup Language

5 Capability profiling method

5.1 Capability profiling concept

The main focus of ISO 16100 is manufacturing software interoperability. Figure 1 depicts the use of a capability profile concept to integrate interoperable software.

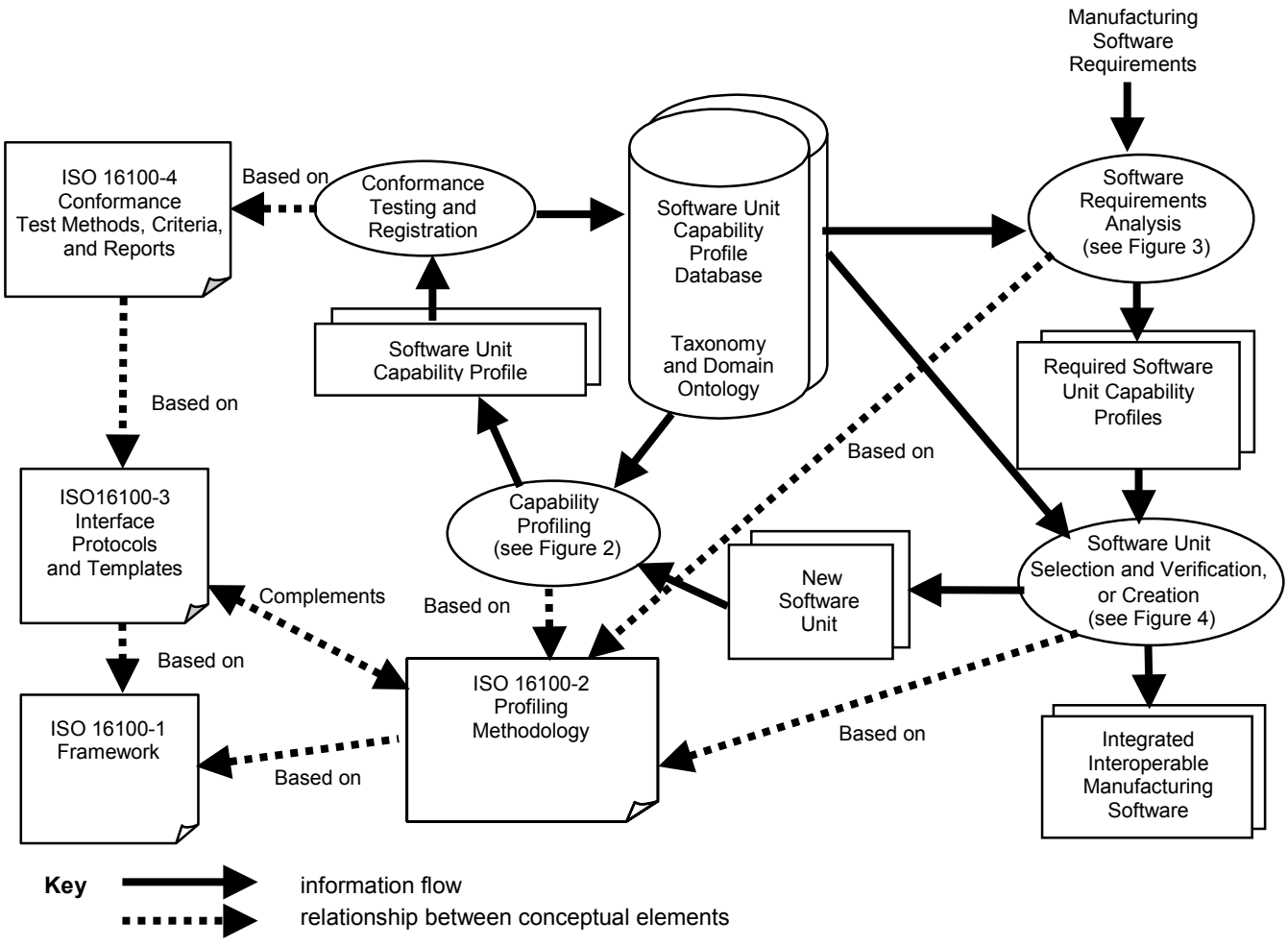


Figure 1 — Concept of capability profile for software interoperability

The interoperability of software units can be described in terms of their capabilities that are associated with the aspects of functionality, interface and structure. These aspects, based on the framework and domain specific application system model defined in ISO 16100-1, are defined in Clauses 5 and 6, and are further detailed in ISO 16100-3.

A manufacturing process has a structure that is both nested and hierarchical. At each level, the manufacturing software requirements can be modelled as a set of capability classes organized in a similar structure. Manufacturing software requirements are met by the integration of several manufacturing software units.

In this methodology, manufacturing software requirements shall be expressed in terms of software unit capability profiles. The profiling of a software unit involves the generation of a concise statement of manufacturing capabilities enabled by the software unit in terms of the functions performed, the interfaces provided, and the protocols supported as required by the target manufacturing capability.

The capability profiling methodology shall be defined in terms of the rules and elements provided in Clause 6. The methodology shall make use of the domain-specific attributes and methods associated with each specific software unit to describe capability profiles in terms of unit name, manufacturing functions, and other needed class properties.

The required profiles are compared to existing profiles in the database. When a match occurs, the software unit being profiled shall be considered to be ready for integration. When no match occurs, a new software unit with the required capabilities shall be developed, profiled, and registered in the capability profile database.

The software units capability profile definition shall be registered in an appropriate database after passing the conformance test which will be provided with the conformance test methodology and its abstract test suites to be defined in ISO 16100-4.

The profile database shall have a set of taxonomies for use in describing the capability profiles.

5.2 *Capability profiling* process

The part of the concept of capability profile for software interoperability shown in Figure 1 related to the *capability profiling* process is detailed in Figure 2.

A software unit to be profiled shall be analyzed in terms of the supported paths within the capability class structure, the concept for which is described in 6.2.1. The structure itself is defined in ISO 16100-3.

The supported paths shall then be used in the search for a matching template from the database. When a matching template is found, the fields of the template shall be filled to make a profile. When no matching template is found, a new template shall be formed using the set of capability classes.

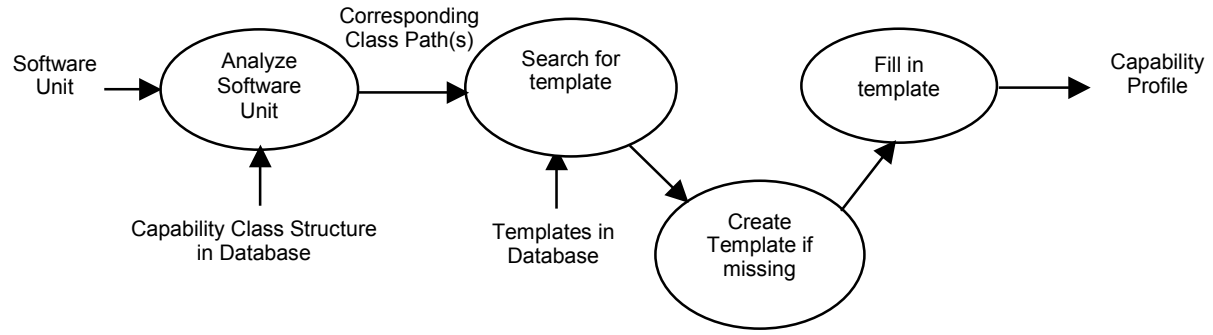


Figure 2 — *Capability profiling* process

5.3 *Software requirements analysis* process

The part of the concept of capability profile for software interoperability shown in Figure 1 related to the *software requirements analysis* process is detailed in Figure 3.

Capability profiles for each manufacturing software unit shall be derived from manufacturing software requirements in the software requirements analysis process. As a first step, manufacturing software requirements shall be decomposed into several primitive requirements which are fulfilled by capability classes that are selected from the database. When a template that corresponds to the class exists, the template shall be filled with specific requirements in order to generate a required capability profile. When such a template does not exist, a new template shall be created based on rules for template creation described in 6.3.

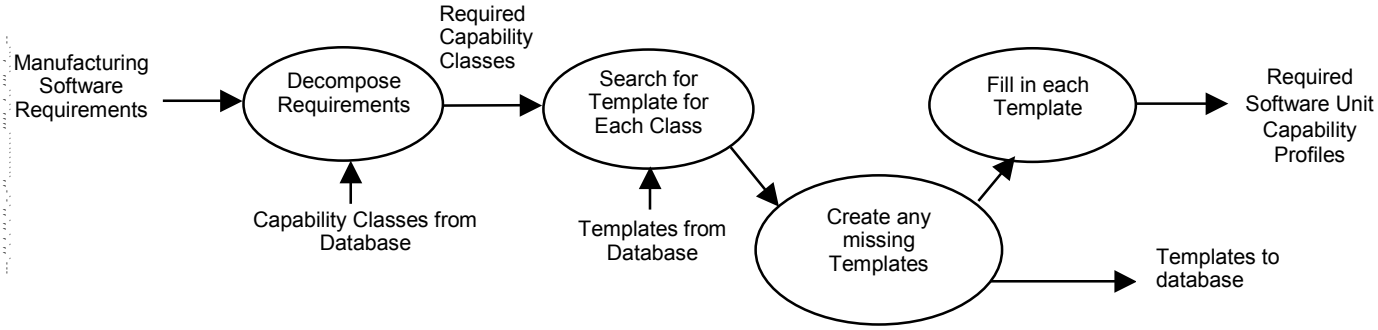
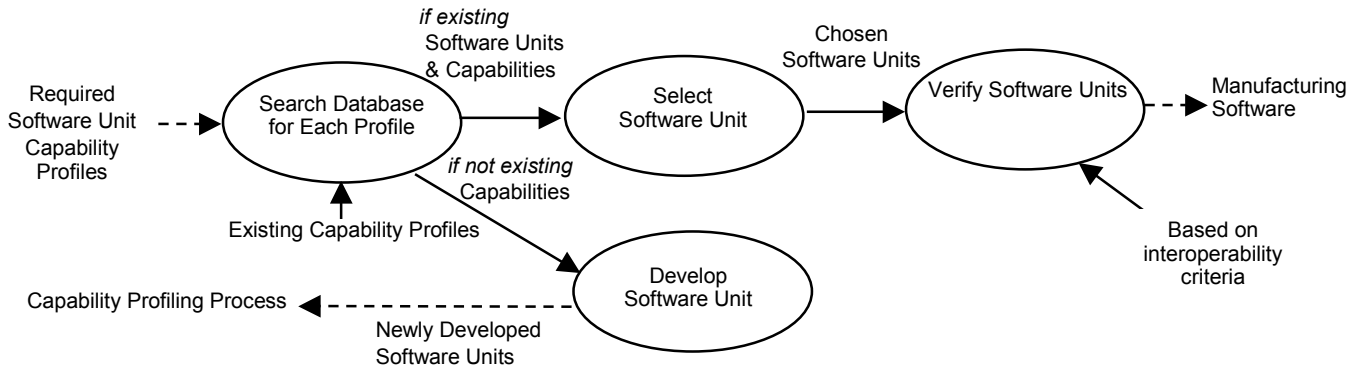


Figure 3 — *Software requirements analysis* process

5.4 *Software unit selection and verification, or creation* process

The part of the capability profile integration related to the *software unit selection and verification, or creation* process shown in Figure 1 is detailed in Figure 4.



Key

flow within the process

flow entering from, or leaving to, another process within the capability profiling concept of Figure 1

Figure 4 — *Software unit selection and verification, or creation* process

For each required capability profile, a search of matching capability profiles that represent available software units shall be performed. Matching shall be performed according to the rules given in 6.6. When a match exists, the software unit shall be added to a list of candidates. When a match does not exist, one of the following shall occur:

a) a new software unit is developed to meet the required profile;

b) the required profile is decomposed into a combination of several profiles;

c) requirements are reconsidered against existing profiles.

The profile for the new software unit shall be registered to the database according to the profiling process in 5.2. The selected software units shall be verified against the manufacturing software requirements according to interoperability criteria.

6 Elements and rules for capability profiling

6.1 Taxonomy

A key aspect of a taxonomy for capability profiles is its ability to identify the contents that constitute a capability class definition. A taxonomy shall be constructed that provides a means for the interchange of the capability information.

The taxonomy shall describe a partial set of activities undertaken within the lifecycle of a manufacturing enterprise. If there is a need to add a new activity to the taxonomy, then the capability class associated with the new activity shall be constructed according to the rules in 6.2.

6.2 Capability classes and their content

6.2.1 Software unit capability class content

The capability of manufacturing software unit shall be expressed in terms of capability classes. These classes shall be derived from the manufacturing activities noted in ISO 16100-1, Figure 4. These classes shall also denote the manufacturing function, resource, and information handled by the manufacturing software unit according to the requirements of the manufacturing process.

The contents of a software unit capability class shall include, but may not be limited to, the following:

a) type of manufacturing domain;

b) type of manufacturing activity as differentiated by the process it is part of, the resources involved in conducting the activity, and the information types exchanged during the activity;

c) type of computing system as differentiated by the operating environment, the software architecture, and the design pattern used;

d) type of services, protocol, and data types used in running the software unit;

e) supplier name, software version, and change history;

f) performance benchmarks;

g) reliability indices;

h) service and support policy;

i) pricing terms and conditions of use.

More capability class content rules and their details are described in ISO 16100-3.

A conceptual structure for a capability class is shown in Figure 5.

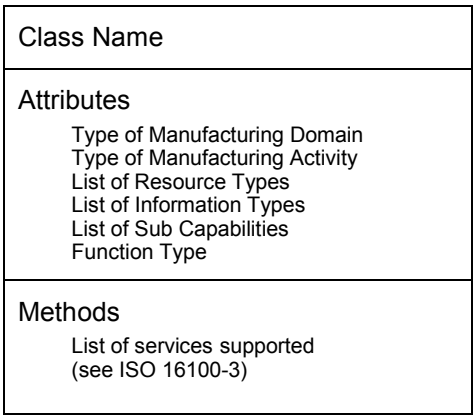


Figure 5 — Conceptual structure for a capability class

6.2.2 Manufacturing application domain

6.2.2.1 Manufacturing application activity model

The domain specific manufacturing application activity model and its three associated models for information, processes, and resources shown in ISO 16100-1, Figure 4 use the common requirements and the software interoperability framework for the entire set of software units offered as the solution for an application's requirements and framework.

As an offered software unit may cover only some portion of the entire application, the target part in the entire model shall be labeled appropriately using the taxonomy registered and the sequence and layer number of the activity models.

EXAMPLE (See ISO 16100-1, Figure 2)
A) Manufacturing Activity
AA) Develop Products
AA1) Design Product
AA11) Develop Conceptual Design
AA111) Define Product Functions and Constraints
AA112) Generate Product Behaviours
AA113) Decompose Functions Constraints and Behaviours
AA114) Specify Product Configuration
AA12) Develop Detailed Design
AA121) Design System/ Component
AA122) Analyze System/ Component
AA123) Evaluate System/ Component Design
AA124) Optimize Design
AA125) Finalize System/ Component Design
AA126) Produce Assembly Drawings
AA2) Engineer Process
AA21) Develop Conceptual Process Plan
AA211) Select Manufacturing Process
AA212) Select Manufacturing resources
AA2121) Select Machines
AA2122) Select Tools/ Fixtures
AA2123) Select labor Skills
AA213) Estimate manufacturing Cost/ Time
AA22) Develop Detailed Process Plan
AA221) Generate Process Sequence
AA222) Generate Operations
AA2221) Determine Intermediate Machining Features
AA2222) Specify Part Setups and machining Resources
AA2223) Calculate Intermediate Machining Tolerances
AA2224) Develop machining Instructions

AA223) Define Manufacturing Parameters
AA224) Generate Control Programs
AA2241) Generate Tools Paths
AA2242) specify Process Control Parameters
AA2243) Generate Machine Control Program
AA225) Generate Shop Floor Routing
AA2251) Determine Shop Floor Configuration
AA2252) Determine Means for Transportation
AA2253) Specify Timing
AA3) Plan Enterprise Resources
AA4) Acquire Resources
AA5) Execute Manufacturing Orders
AA51) Develop Operation Sequence & Detailed Schedule
AA52) Dispatch Production Units
AA53) Track Production Units& Resources
AA54) Manage Factory- Floor Data/ Document
AA6) Control Equipment & Process

6.2.2.2 Manufacturing process model and its profile

The manufacturing process model class is the (automated function) model derived from the implementation of the required specific application activities with its appropriate taxonomy and index number (see example in 6.2.2.1) in the system with specific resources and its information flow.

Using a modeling language such as IDEF0, the activity model describes the requirements and data flow of the application system.

The process model depicts the automated functions with the selected resources and information flow between them. The process model shall be named and labeled appropriately using the taxonomy registered and the sequence and layer number of the related (targeted) activity models.

Each process model shall be represented as an appropriate profile.

6.2.2.3 Manufacturing resource model and its profile

The manufacturing resource model is the model representing the selected resources such as devices, equipment, communication networks, humans, and materials used in the process model to fulfill the requirements of the information model which specifies the information flow among the resources.

The resource model shall be named and labeled appropriately using the taxonomy registered and the sequence and layer number of the related (targeted) activity models.

Each resource model shall be represented as an appropriate profile.

6.2.2.4 Manufacturing information model and its profile

The manufacturing information model represents the data types of the events and the data exchanged between resources in the process model representing the specific scope of activities in the application activity model.

The information model shall be named and labeled appropriately using the taxonomy registered and the sequence and layer number of the related (targeted) activity models.

Each resource model shall be represented as an appropriate profile.

6.2.3 Computational model and its associated class

The computational model is a model representing the mapping of the process model, the resource model, and the information model described in 6.2.2.

6.2.3.1 Class representation of the software unit

6.2.3.1.1 Class name

The information model shall be named and labeled appropriately using the taxonomy registered and the sequence and layer number of the related (targeted) activity models.

6.2.3.1.2 Class attributes

The derived attributes of the class shall be listed with its data type and capability for external access.

6.2.3.1.3 Class operations

The derived operations of the class shall be listed with its signature and capability for the external service.

The software unit may be a package consisting of multiple subsequent classes. In such a case, all the included classes shall be listed.

6.2.3.2 Associated software architecture, software design pattern class used

The typical property of the software architecture as well as the software design pattern to be used for the software unit's framework shall be listed along with the role the software unit performs.

Architecture design patterns and examples[1)] of its structure and role are listed below.

a) Layering architecture

EXAMPLE Structure: applications that can be decomposed into groups of sub-tasks in which each group of sub-tasks is at a particular level of abstraction. Role: N layer entity with a role to service for N+1 layer entity.

b) Broker architecture

EXAMPLE Structure: distributed software systems with decoupled components that interact by remote service invocations. Role: clients, servers, brokers, bridges, client- side proxies, server-side proxies.

c) Model-View-Controller architecture

EXAMPLE Structure: the model contains the core functionality and data, views display information to the user, controllers handle user input. Role: Model, Observer, View, Controller.

d) Master-Slave

EXAMPLE Structure: a master component distributes work to identical slave components and computes a final result from the results these slaves return. Role: Client, Master, Slave1, Slave 2,…, Slave N.

e) Proxy

EXAMPLE Structure: makes the clients of a component communicate with representative rather than to the component itself. Role: Client, Proxy, Original.

f) Publisher-Subscriber

EXAMPLE Structure: one publisher notifies any number of subscribers about changes to its state. Role: Publisher, Subscriber.

1) The examples are taken from F. Buschmann et al, "Pattern Oriented Software Architecture," John Wiley & Sons, June 2000.

6.2.3.3 Service or protocol class

The interfaces of the software unit shall be described as the service (for example, in layering architecture, the N layer entity shall serve the N+1 layer entity) or protocol (for example, in client server architecture, the clients interface with a specific protocol to the server) with its data type.

6.2.4 Non-functional properties of the software unit

Contrary to 6.2.2 and 6.2.3 which prescribe classification from a functional viewpoint, the following properties of the software unit are taken from a non-functional view of the software unit.

6.2.4.1 Vendor, version and history of the unit

The capability profile of the software unit should include vendor (supplier) name and contact address, the newest version of the software and its revision history.

6.2.4.2 Computing facilities to be used

The following information should be included in the capability profile of the software unit:

a) processor — the processor type is of key importance, but its performance should also be considered, as poor performance may seriously hamper effective and timely execution of the software;

b) operating system and required options — the appropriate operating system and release version required to run the software component. Any feature to support upward compatibility may be included in this information;

c) language — the source language for the software component, including the versions of the editor, compiler, linker, and debugger used in generating the component. Any feature to support upward compatibility may be included in this information;

d) run-time memory — the type and amount of memory needed to run the software component along with any other runtime support;

e) disk space — the type and amount of media needed to store the runtime and source forms of the software component. This information shall include any data store, such as disk storage, required for operating variables, processing results, and fault recovery mechanisms;

f) multi-user support — the ability of the software component to handle multiple users, clients, or subscribers;

g) remote access — the ability of the software component, as well as any other software components that are downloaded or uploaded prior to its execution, to support remote access, control, and management;

h) add-ons and plug-ins — software extensions required to support the runtime behaviour of the software capability, e.g. interpreting imported images or filtering incoming data in a non-native application format.

6.2.4.3 Measured performance of the unit

The capability profile of the software unit should include performance data for a specific computing facility needed for real time (time critical) usage of the software unit. This performance data should include:

a) elapsed time (execution time) with specific input data and its constraints (fundamental performance data);

b) number of specific transactions per unit time (integrated performance data).

6.2.4.4 Reliability data of the unit

The software unit's capability profile should include usage history, number of shipments of the software unit, its intended safety integrity level, and whether the safety integrity level was self-determined or determined by a third party.

6.2.4.5 Competency

The software unit's capability profile should include information concerning the competency of each vendor, such as: authority, commitment, and policies in applying the software unit; license requirements; shop floor policies, and; operator training requirements.

6.2.4.6 Price data

The software unit's capability profile should include the initial and operating costs of the software unit.

6.3 Capability templates and rules

A software unit that enables or supports an activity with an associated capability class is concisely described in a capability template. The structure of a software capability template shall follow the structure of a manufacturing capability class.

NOTE In a hierarchical structure, a capability template is associated with each capability defined at each level of the structure. In a nested structure, a similar association exists between each capability class and a template at each level of the structure.

Figure 6 shows an example of a conceptual structure of a template. The structure shall consist of a part that is common to all templates and another part that is specific to capability class. The formal structure of a template is defined in ISO 16100-3.

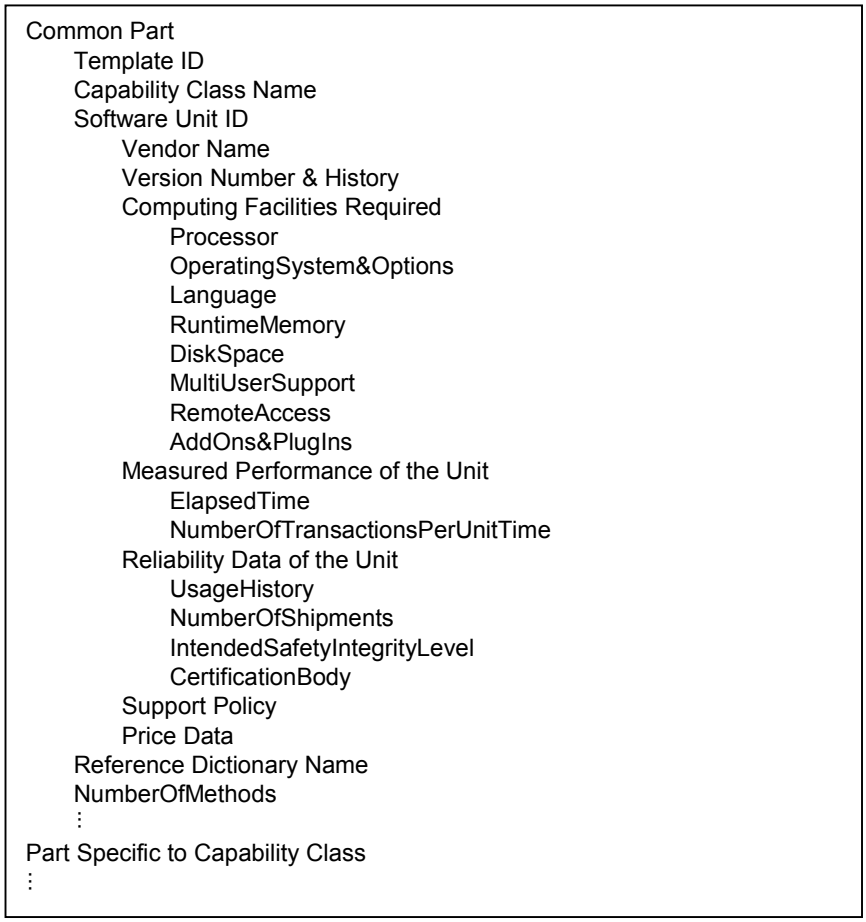


Figure 6 — Example template structure

In the common part, the following set of elements shall be included:

a) template ID — an identifier of the object template;

b) software unit ID — an identifier of the manufacturing software unit that enables manufacturing function;

c) reference dictionary name — the name of a dictionary that contains the definitions of the capability classes;

d) capability class name — the name of a referenced capability class;

e) number of profile attributes — the number of attributes inherited from the corresponding capability class;

f) number of methods — the number of methods provided by the software unit;

g) number of resources — the number of resources required in the software environment;

h) number of constraints — the number of conditions required to run a software unit;

i) number of extensions — the number of other software unit aspects as provided by the units supplier;

j) number of lower levels — the extent of nesting or the deepest level in the hierarchy of the reference capability class structure;

k) number of subtemplates at next lower level — the number of templates associated with subcapabilities comprising the target capability associated with the template at one level below the current level in the hierarchy or nesting.

In the part specific to the capability class, the following set of elements shall be included:

a) list of attributes;

b) list of methods;

c) list of resources, e.g. type of operating system;

d) list of constraints, e.g. type of architecture, design pattern;

e) list of extensions;

f) list of lower levels;

g) list of subtemplates.

Capability templates shall be defined using XML conventions for creating XML Schemas (see REC-xmlschema-1-20010502 and REC-xmlschema-2-20010502). Relationships between capability templates shall be denoted using XML conventions for transformation of XML Schemas and XML files. When a capability class is specified in a template and such a class has been instantiated, then the instantiated class represents an object. Two capability templates are identical if their respective attributes and operations are identical. When the attributes of one template form a subset of the attributes of another and the operations of one template form a subset of the operations of another, then the two capability templates are considered to be compatible and have a match.

6.4 Capability profiles and rules

Capability profiles are capability templates with, at a minimum, the profiled software unit name instantiated. Other items are fulfilled according to specification level.

Capability profiles shall be defined using XML conventions for creating XML files. Relationships between capability profiles shall be denoted using XML conventions for transformation of XML files. When a capability template is referenced in a capability profile and such a template has been filled, then the filled template represents a profile object.

6.5 Software unit profile database

The following elements, each distinguished by the dictionary name and described in 6.1 to 6.4, are stored in databases:

a) a set of taxonomies;

b) a set of capability classes;

c) a set of capability templates;

d) a set of capability profiles.

Databases may be structured as a free combination of the above four elements to provide necessary services.

A taxonomy shall be unique when entered in a taxonomy dictionary. A capability class shall be unique when entered in a class dictionary. A capability template shall be unique when entered in a template dictionary. A capability profile shall be unique when entered in a profile dictionary.

6.6 Rules for matching capability profiles

Matching capability profiles are used in the following processes:

a) the analysis of software unit in the *capability profiling* process (see Figure 2);

b) the decomposition of requirements in the *software requirements analysis* process (see Figure 3);

c) the database search for each profile in the *software unit selection and verification, or creation* process (see Figure 4).

Matching is attempted between software unit descriptions, manufacturing software requirements, or required software unit capability profiles in these processes and that of capability profiles in the database. Software unit and manufacturing software requirements shall be described using XML conventions by referring to the taxonomy prepared for the database, contents of the capability classes, and existing templates. Descriptions of generated software units, manufacturing software requirements, or required software unit capability profiles are compared with the contents of capability classes in the database described in 6.2. If at least a part of both match, then the capability class description in the database is the result of this matching. Templates in the database are searched by using the output contents of capability classes.

6.7 Interoperability criteria

Interoperability criteria are used to perform the verification process in 5.4.

7 Conformance

Conformance to this part of ISO 16100 is covered in ISO 16100-4.

Annex A
(informative)
Reference methods

A.1 Extensible Markup Language (XML)

The eXtensible Markup Language (XML) (Schuldt, 1998) possesses some features which may be employed either directly or indirectly in the software capability profiling work. XML is a language for expressing the lexical elements of a “document” as a directed graph, particularly for distribution on the web. The lexical elements can be user defined. XML is a practical subset of SGML, and provides tagging like HTML. Any XML document can also be checked for XML validity. For the purpose of using XML in software capability profiling, however, it should be noted that XML has XML Namespaces for reconciling or registering of name spaces.

A.2 Vocabularies, definitions and interchange formats for software packages: Open Software Description (OSD) and Channel Definition Format (CDF)

The Open Software Description[2)] (OSD) is an XML based vocabulary for describing software packages and their inter-dependencies. As such OSD can be useful in software distribution environments, either in user-initiated ("pulled") or automatic ("pushed") situations. OSD could be a potential standard used for distributing software on the Web in either one of these two modes.

Pull-based software distribution involves user action to discover, download, and update software. While OSD makes it easy to automate the download and installation of required software components, Web users are still required to browse to an HTML page that initiates the software installation process. The "OBJECT" tag from the HTML 4.0 specification is used to "advertise" the presence of new software on the Web. Upon detecting an OBJECT at an OSD resource, an OSD-aware user agent can automatically download and update the necessary software components.

The Channel Definition Format (CDF), also based on XML, provides a meta-data vocabulary for describing inter-relationships between HTML pages and other Web resources. CDF-aware clients may use "smart-pull" techniques to automatically download Web content, and CDF-aware servers may implement "true-push" mechanisms for automatic distribution of content from client to server. CDF thus provides a language for content "push," providing the ideal leverage point for enabling software "push," or automatic distribution of software. In order for CDF to activate software "push," a CDF file needs to include references to OSD-based software packages.

The OSD vocabulary includes an extensive vocabulary with which to describe the elements of software. These include the following:

SOFTPKG:	defines a general software package
IMPLEMENTATION:	used to describe an implementation of a software package
DEPENDENCY:	used to indicate a dependency between software distributions, or components thereof
TITLE:	provides the title, or "friendly name," of the software package

2) Van Hoff et al, 1997.

ABSTRACT:	provides a short description summarizing the nature and purpose of a software distribution
LICENSE:	indicates the location where a license agreement or copyright notice can be retrieved
CODEBASE:	indicates a location (usually on the network) where an archive of the software distribution exists
OS:	indicates the required operating system
OSVERSION:	indicates the required operating system version
PROCESSOR:	indicates the required natural language in the software's user interface
LANGUAGE:	provides the title, or "friendly name," of the software package
VM:	indicates the required virtual machine
MEMSIZE:	indicates the required amount of run-time memory
DISKSIZE:	indicates the required amount of disk space
IMPLTYPE:	indicates the type of the implementation

A.3 Distributing software services: Open Distributed Processing (ODP) and Common Object Request Broker Architecture (CORBA)

The ODP reference model describes the important attributes of an information system on which constraints may be defined in order for that system to be both open and distributed. The reference model is the basis for a series of standards that defines these requirements in detail.

The Object Management Group (OMG) is a non-profit consortium of software vendors, software developers and end-users, formed in May 1989. It seeks to provide a common architectural framework for distributed object-oriented applications based on widely available interface specifications such as that provided by ODP. The Object Management Architecture (OMA) is the center of all the activity undertaken by OMG and consists of a reference model (published in 1992) that identifies and characterizes the components, interfaces, and protocols that compose the OMA but does not in itself define them in detail.

CORBA, developed by the Object Management Group, is an architecture and protocol for dynamically binding distributed objects. Binding is syntactical and lexical only, limited by capabilities of IDL.

There are five components to the reference model:

a) object request broker — provides an infrastructure allowing objects to converse, independent of the specific platforms and techniques used to implement the objects;

b) object services — standardize the life-cycle management of objects. Interfaces are provided to create objects, to control access to objects and to keep track of relocated objects. Object services provide for application consistency;

c) common facilities — provide a set of generic application functions that can be configured to the specific requirements of a particular configuration. These are facilities that are more readily recognizable by the end-user, such as printing and electronic mail;

d) domain interfaces — represent the vertical areas that provide functionality of direct interest to end-users in particular application domains such as manufacturing, finance and healthcare for example;

e) application objects — are not part of the OMG standardization activity, but it is recognized by OMG that these are crucial to the development of successful applications.

The OMG approach defines interfaces to the distributed objects using the Interface Definition Language (IDL). OMG has identified a need for a more semantically rich description language. As a result, a number of RFPs are about to be issued which cover the business object domain and in particular focus on the need for a more semantically rich language. Recently, OMG has started the standardization of a number of business objects; for example, task currency, which could in the future lead to vendors building software using some base standardized objects and hence facilitating interoperability.

Bibliography

[1] ISO/IEC TR 10000-1:1998, *Information technology — Framework and taxonomy of International Standardized Profiles — Part 1: General principles and documentation framework*

[2] ISO/IEC 10746-2:1996, *Information technology — Open Distributed Processing — Reference Model: Foundations*

[3] ISO 18629-1:—[3)], *Industrial automation systems and integration — Process specification language — Part 1: Overview and basic principles*

[4] ISO/IEC 19501-1:—[3)], *Information technology — Unified Modeling Language (UML) — Part 1: Specification*

[5] IEEE 1320-1:1998, *Standard for Functional Modeling Language — Syntax and Semantics for IDEF0*

[6] REC-xml-20001006, Extensible Markup Language (XML) 1.0 Second Edition — W3C Recommendation 6 October 2000

3) To be published.

ICS 25.040.01

Price based on 17 pages