INTERNATIONAL STANDARD

ISO 16100-3

First edition
2005-12-15

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

Part 3: Interface services, protocols and capability templates

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

Partie 3: Services d'interface, protocoles et gabarits d'aptitude

Reference number
ISO 16100-3:2005(E)

PDF disclaimer

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

Adobe is a trademark of Adobe Systems Incorporated.

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

ISO 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 16100-3 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 services, protocols and capability templates*

In addition, the following part is envisaged:

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

Introduction

The motivation for ISO 16100 stems from the industrial and economic environment, in particular:

a) a growing base of vendor-specific software intensive solutions;

b) increasing user difficulty in applying independently-developed standards;

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

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

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

Certain diagrams in this part of ISO 16100 are constructed following UML conventions. Because not all concepts embodied in these diagrams are explained in the text, some familiarity with UML on the part of the reader is assumed.

In this part of the ISO 16100, references to classes (objects) and services use a specific naming convention as shown in the following examples:

ServiceAccessPoint a service access point object

registerProfile a service primitive for profile registration

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

Part 3:
Interface services, protocols and capability templates

1 Scope

This part of ISO 16100 specifies requirements for interface services and protocols used to access and edit capability profiles and associated templates used in the capability profiling method defined in Clause 5 of ISO 16100-2.

The detailed services for accessing capability profiles and performing the matching process on these profiles are defined in this part of ISO 16100.

This part of ISO 16100 is applicable only for the interoperability of software units used in the manufacturing domain. Concerns regarding interchangeability of manufacturing software units are outside the scope of this standard.

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 16100-1:2002	*Industrial automation systems and integration — Manufacturing software capability profiling for interoperability — Part 1: Framework*
ISO 16100-2:2003	*Industrial automation systems and integration — Manufacturing software capability profiling for interoperability — Part 2: Profiling methodology*
IEEE 1320.1-1998	*Standard for Functional Modeling Language — Syntax and Semantics for IDEF0*
OMG ad/2003-04-01	*Unified Modeling Language; Superstructure v2.0*
REC-xml-19980210	*Extensible Markup Language (XML) 1.0 W3C Recommendation*
REC-soap12-20021219	*SOAP Version 1.2 — Part 1: Messaging Framework*
REC-xmlschema-1-20010502	*XML Schema Part 1: Structures*

3 Terms and definitions

For the purposes of this part of ISO 16100, the following terms and definitions apply.

3.1 ISO 16100-3 definitions

3.1.1
capability profile interface
functional (implementation-independent) service access point that provides a set of services described in 5.4 of this part of ISO 16100 to handle capability profiles

NOTE In some implementations as noted in ISO 16100-2 the CPI can be implemented by a database server.

3.1.2
capability profile service provider
software that implements the capability profile interface

3.1.3
cluster
set of manufacturing resource units

3.1.4
component
part of a manufacturing software unit, including manufacturing software components.

3.1.5
consumer
<profile>
user of profile or Matching Level result

3.1.6
matcher
mechanism to compare an offered capability profile with a required capability profile.

3.1.7
matching level
<profile>
qualitative measure of how closely a capability profile of a MSU meets the software functional requirements of a manufacturing activity

3.1.8
MSU interoperability
capability of a MSU to support a particular usage of an interface specification in exchanging a set of application information with another MSU

3.1.9
MSU interchangeability
capability of a MSU to replace another MSU in performing a required function within a particular manufacturing activity

3.1.10
producer
<profile>
generator of profile or Matching Level result for consumption

3.1.11
reference capability class structure
schema representing a hierarchy of capability classes to be used for capability profiling.

3.1.12
reference dictionary
list of capability classes used in the reference capability class structure

3.1.13
schema
XML meta-data definition

3.1.14
template
schema for a manufacturing software capability profile

3.1.15
type I matcher
matcher that can process profiles derived from the same capability class structure

3.1.16
type II matcher
matcher that can process profiles whether they are derived from the same or from different capability class structures

3.2 Applicable definitions from ISO 16100-1

For the purposes of this document, the following terms and definitions from ISO 16100-1 apply. The reference to the specific subclause in ISO 16100-1 appears in brackets after the definition. Following clause C.1.4 of ISO / IEC directives, part 2 some definitions are repeated here with notes added as required.

3.2.1
capability
<software>
set of functions and services with a set of criteria for evaluating the performance of a capability provider

[3.3]

3.2.2
capability profiling
selection of a set of offered services defined by a particular interface within a software interoperability framework

[3.4]

3.2.3
manufacturing software
type of software resource within an automation system that provides value to a manufacturing application (e.g. CAD/PDM) by enabling the flow of control and information among the automation system components involved in the manufacturing processes, between these components and other enterprise resources, and between enterprises in a supply chain or demand chain

[3.10]

3.2.4
manufacturing software capability
set of manufacturing software functions and services against a set of criteria for evaluating performance under a given set of manufacturing conditions

[3.14]

3.2.5
manufacturing software capability profile
concise representation of a manufacturing software capability to meet a requirement of a manufacturing application

[3.15]

3.2.6
manufacturing software component
class of manufacturing software resource intended to support the execution of a particular manufacturing task

[3.11]

3.2.7
manufacturing software unit
class of software resource, consisting of one or more manufacturing software components, performing a definite function or role within a manufacturing activity while supporting a common information exchange mechanism with other units

[3.12]

3.3 Applicable definitions from ISO 16100-2

For the purposes of this document, the following terms and definitions from ISO 16100-2 apply. The reference to the specific subclause in ISO 16100-2 appears in brackets after the definition.

3.3.1
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

[3.3]

3.3.2
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

[3.4]

3.3.3
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

[3.8]

3.3.4
profile
set of one or more base specifications or sub-profiles or both, 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

[3.10]

4 Abbreviated terms

CPI	Capability Profile Interface
DTD	Document Type Definition
ML	Matching Level
MSU	Manufacturing Software Unit
UML	Unified Modeling Language
XML	eXtensible Markup Language

5 Manufacturing software information model and profile

5.1 Manufacturing activity and information exchange model

A manufacturing application shall be modeled as a set of manufacturing processes that are enabled, controlled and automated by a set of manufacturing resources through a series of information exchanges, along with transfers of materials and energy. This is shown in Figure 4 of ISO 16100-1.

A manufacturing process shall be modeled as a sequence of scheduled manufacturing activities, where each manufacturing activity is associated with a set of manufacturing functions (see 5.3 of ISO 16100-1 and Annex C of this part).

In order to meet the requirements of a manufacturing application, a set of MSUs shall be sequenced and scheduled to accomplish the combined set(s) of required manufacturing functions for all the manufacturing activities associated with the set of manufacturing processes that constitute a manufacturing application.

Per the manufacturing software interoperability framework in Clause 6 of ISO 16100-1, each manufacturing activity shall be associated with a set of MSUs. As shown in Annex A of ISO 16100-1 that is based on IEC 62264, a complex manufacturing activity can be modeled as a combination of a set of simpler manufacturing activities. A simple manufacturing activity shall correspond to a single function and the activity shall be enabled by a single MSU.

Each manufacturing function can be accomplished by a set of manufacturing software units (MSUs). A set of manufacturing functions may be accomplished by one MSU (see 6.2 of ISO 16100-1).

EXAMPLE A simple manufacturing application (e.g. pick-and-place) can be modeled as a set of three manufacturing process (e.g. load an item, move an item, unload an item). Each manufacturing process can be associated with a single activity composed of a particular sequence of functions from the following set – locate item, identify item, identify place, go to item, acquire item, locate place, go to place, release item to place, notify process coordinators. In one case, two classes of MSUs (load/unload, move) with two capability templates can be profiled into three MSU instances (one for each activity). In another case, there maybe four lower level activities or MSU classes (locate, identify/notify, acquire/release, go to target) and nine MSU instances.

As shown in Figure 1 a MSU provides several interfaces to its capabilities, including its capability profile. The capability profile can be accessed at a Capability Profile Interface (CPI). Information about the other interfaces is included in the capability profile and therefore the information is accessible via the CPI.

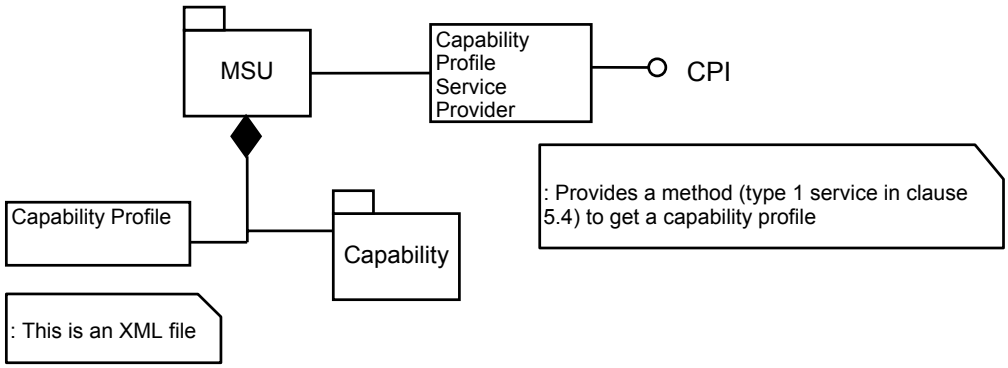

Figure 1 — MSU with its capability and corresponding capability interfaces, especially the Capability Profile Interface

5.2 Manufacturing software unit

A manufacturing software unit (MSU) shall be modeled as a type of manufacturing resource that can satisfy a set of interoperability criteria. These criteria shall be determined by the required sequence and timing of the specific set of manufacturing functions that have to be accomplished by a MSU and the information exchanges that it has to support.

As shown in Figure 4 of ISO 16100-1 a MSU shall be associated with a manufacturing activity and its corresponding capability. A software component shall not be associated with a capability profile.

A manufacturing function associated with a capability class of a manufacturing activity shall be modeled as being enabled by one or more MSUs.

EXAMPLE A manufacturing function associated with manufacturing activity N in Figure 2 is enabled by MSU 3. On the other hand, a manufacturing function associated with manufacturing activity M is enabled by both MSU 1 and MSU 2.

The manufacturing capability classes supported by a set of MSUs shall be determined by the manufacturing function of a manufacturing activity and the related information exchanges among the other manufacturing resources deployed to enable the manufacturing process.

A particular class of MSU may be used in different activities. Each MSU shall provide a set of interfaces. The interoperability criteria between MSUs shall be determined only by the requirements of the interoperable activities. Interoperability criteria between manufacturing processes shall not be considered in this International Standard. Interoperability criteria involving groups of MSUs associated with manufacturing processes shall not be considered in this part of ISO 16100.

At each level, the manufacturing software requirements can be modeled as a set of capability classes organized in a similar structure as shown in Figure B.1.

NOTE 1 In Figure 4 of ISO 16100-1, a manufacturing process is composed from a set of manufacturing activities. A manufacturing process can have a nested or hierarchical structure of manufacturing activities. The interoperability of MSUs only applies to the latter set (i.e. activities).

When two or more MSUs provide the required manufacturing software function within a manufacturing activity, these MSUs shall satisfy a set of interoperability criteria. The required interface(s) for interoperability of a set of MSUs within a particular activity shall be designated in a software capability profile of that activity.

NOTE 2 In Figure 2, an interface A provided by MSU 1 from vendor A interoperates with interface B provided by MSU 2 from vendor B. The interoperability criteria is denoted by interoperability I based on requirements of activity M. The capability profile for interface A should match the capability profile for interface B to support interoperability I. This profile can differ for different manufacturing activities.

NOTE 3 In Figure 2, when two activities such as M and N have to cooperate, another set of interoperability criteria can be used. The interoperability criteria denoted by interoperability J is based on common requirements of activities M and N. The set of MSUs that enable both activities have capability profiles that support interoperability J.

A combined behaviour of multiple MSUs shall be equivalent to the situation in which the manufacturing software requirements of the activity were being provided by a single MSU. This combined behaviour (of a single equivalent MSU) depends on the compatible use of an interface specification common to a set of MSUs. Conversely, a MSU can be composed of a set of MSUs to reflect a decomposition of a single activity into a set of activities.

When a MSU is modelled as a set of manufacturing software components, this MSU shall not contain another MSU. These MSU components shall be considered to belong only to that MSU. The information exchange and associated interfaces among the components within a MSU are outside the scope of this standard.

NOTE 4 In Figure 2, a MSU 2 provided by vendor C may replace a MSU 2 provided by vendor B in providing the manufacturing function required within manufacturing activity M. Although interface A provided by MSU 1 from vendor A interoperates with interface C provided by MSU 2 from vendor C, the full interchangeability of both MSU 2s cannot be realized. The capability profile for interface B matches the capability profile for interface C to support interoperability but not their interchangeability.

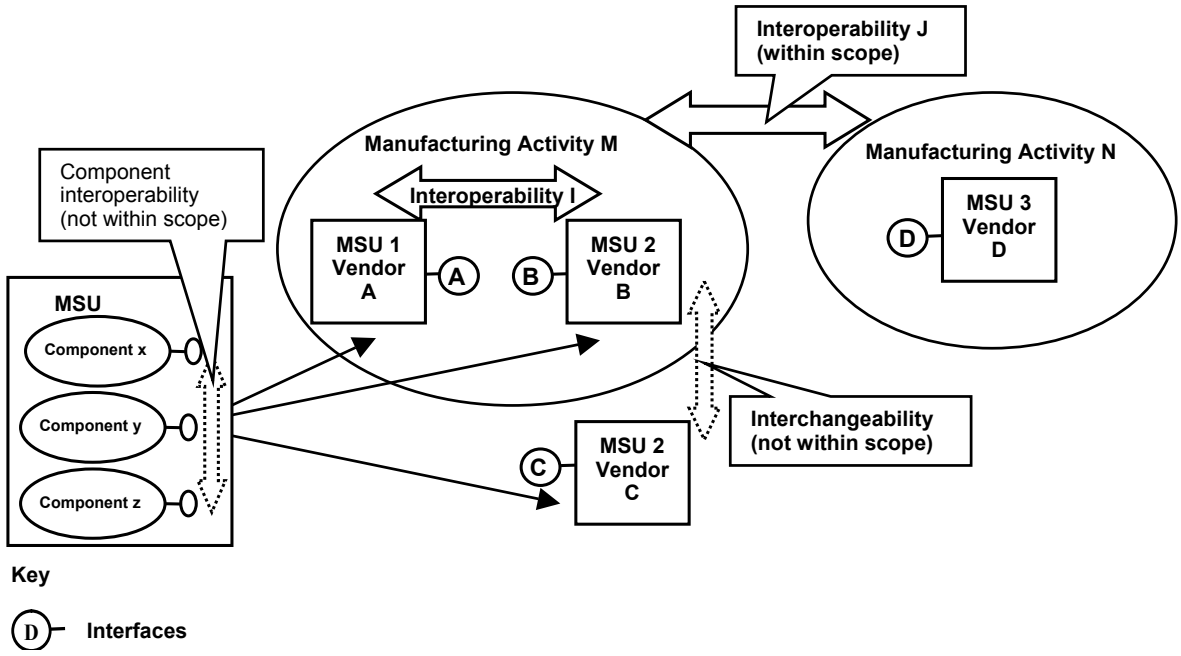

Figure 2 — MSUs within a manufacturing activity

5.3 Matching capability profiles

5.3.1 General

The structure of a MSU capability template shall be derived from the manufacturing capability class structure (see 6.3 of ISO 16100-2). Capability profiles pertaining to a MSU or a capability requirement for a manufacturing activity shall be matched according to the rules of 6.3 of ISO 16100-2. Capability profiles are capability templates with, at a minimum, the profiled software unit name instantiated; other items in the capability template shall be filled according to the desired Matching Level (see 6.4 of ISO 16100-2).

The attribute of a manufacturing capability class are defined in 6.2.1 of ISO 16100-2, while a conceptual structure is defined in 6.2.3 and 6.2.4 of ISO 16100-2.

NOTE 1 See Figure 4 of ISO 16100-1 regarding relationships between manufacturing application, manufacturing activities, and manufacturing resources. See 6.2 of ISO 16100-2 regarding the relationship between capability classes and manufacturing activities.

NOTE 2 Reference capability class structure is represented as a reference schema. See Annex A for “Reference Class Structure”.

A matching MSU capability profile shall represent the following required aspects of a manufacturing activity:

a) handling the input and output information associated with the required manufacturing function;

NOTE 3 The elements in the “part specific to the capability profile” portion of the template can be categorized as input or output elements of the manufacturing function associated with the activity (see 6.3 of ISO 16100-2). A match between the MSU profile and the required capability implies that these elements are included in the MSU profile.

b) corresponding MSUs with compatible interfaces, expressed by their capability profiles, where two activities are interoperable.

NOTE 4 A MSU’s interface services and protocol settings that match the corresponding settings of the interfaces of the other MSU(s) associated with the other manufacturing activities enables the MSUs to be interoperable, as well as their respective manufacturing activities.

Figure 3 models the matching of a MSU’s capability profile to a required capability profile of a manufacturing activity.

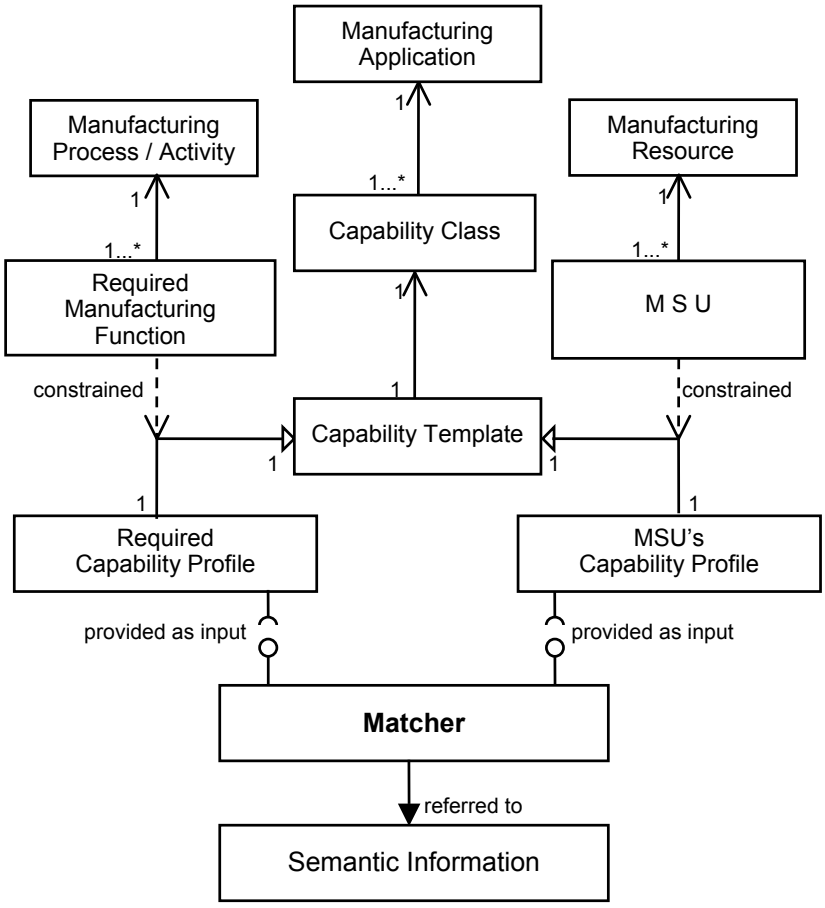

Figure 3 — Matching capability profiles

All matchers use additional semantic information about application specific domain. This semantic information can include identity information that is generated on the basis of a set of taxonomies that can be stored in the Software Unit Capability Profile Database.

5.3.2 Type 1 Matcher

A Type 1 Matcher is distinguished from a Type 2 Matcher (see (5.3.3) in that a Type 1 Matcher can only match capability profiles that were generated from a capability template derived from the same capability class.

5.3.3 Type 2 Matcher

A Type 2 Matcher can match capability profiles that were generated from templates derived from the same capability class or derived from different capability classes. In the case of different capability classes, the classes can either be from the same manufacturing application or from different manufacturing applications.

Different capability classes exist because different enterprises classify manufacturing functions on the basis of different activity domains and various functional boundaries as shown in ISO 16100-1, annex A.1. In this case, the same manufacturing functions belong to different capability classes.

The Type 2 Matcher uses semantic information that includes the identity information about the same manufacturing functions in different capability classes to perform the matching process.

In the case of a Type 2 Matcher matching profiles generated from templates derived from two different capability classes both of the same manufacturing application, a customer describes a certain manufacturing function by completing a capability template derived from a certain capability class to produce the required capability profile. A supplier describes the same manufacturing function by completing a capability template derived from a different capability class to produce the MSU's capability profile. The Type 2 Matcher then matches the generated required capability profiles to generated MSU capability profiles (see Figure 4).

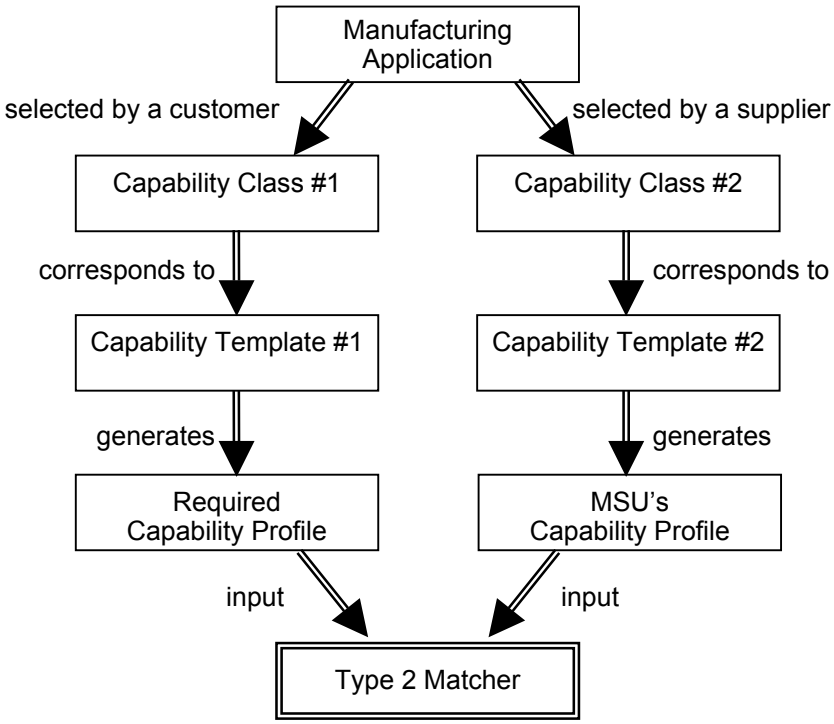

Figure 4 — Different capability classes from same manufacturing application

In other cases a Type 2 Matcher handles profiles generated from templates derived from two different capability classes from different manufacturing applications.

5.4 Interface service definition

The following services are required to support the use of the capability profile concept for software interoperability as described in ISO 16100-2:

a) capability profiling of a new MSU;

b) matching of a proposed MSU capability profile to a required capability;

c) checking of a capability profile against the rules of construction for profiles;

d) editing capability profiles and templates;

e) creating new templates.

The capability profile service provider in Figure 1 provides the following types of services through a CPI, which encompass services a) through e).

Type 1 service set allowing access to a capability profile of either a MSU or that required by a manufacturing activity, and allowing two profiles to be matched.

Type 2 service set allowing creation and modification of a capability profile, and used in capability profiling process of a new MSU (see Figures 1 and 2 of ISO 16100-2);

Type 3 service set allowing creation and modification of a capability profiling template and used to create templates (see Figure 3 in);

In general, all types of services can be available at the same MSU interface or at different interfaces.

In this part of ISO 16100 only Type 1 services protocol definitions are described (see 6.2).

6 Capability profile interface, service, and protocol

6.1 Capability profile service usage

6.1.1 Capability profile access

A Type 1 service set shall be used to access and match capability profiles.

A capability profile can be accessed from either a MSU or a resource distinct from the MSU.

6.1.2 Matching of two capability profiles

A MSU selection process starts with a required profile for a given activity. A desired set of MSUs shall have a corresponding set of capability profiles that match the capabilities required for a given activity. The required profile for a given activity contains a set of mandatory and optional capabilities. The desired set of MSUs shall, at a minimum, provide the complete set of mandatory capabilities for a given activity.

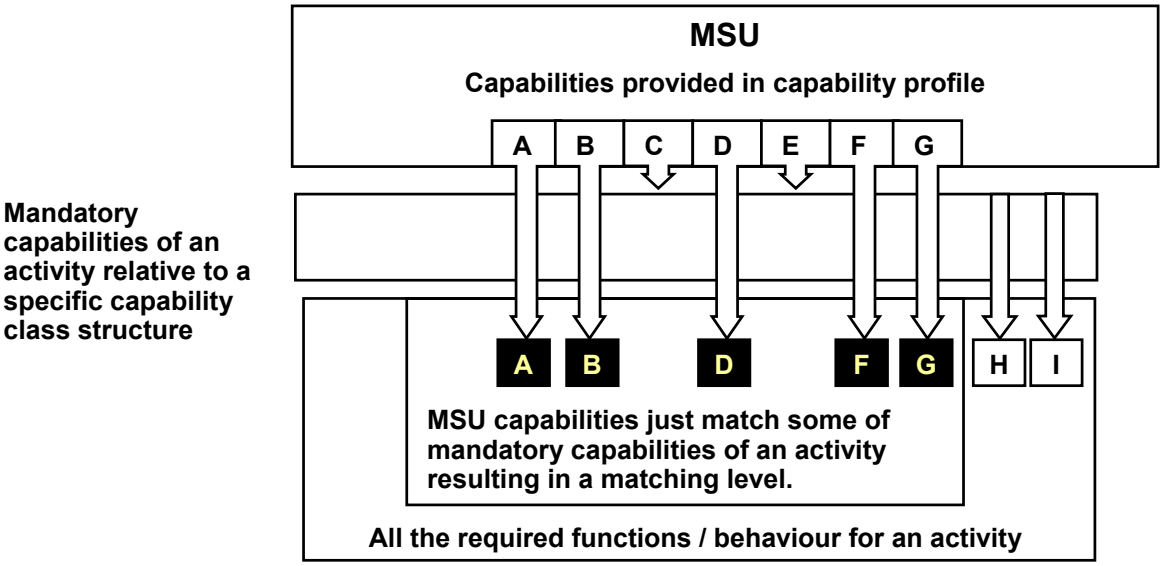

Figure 5 — Obtaining the Matching Level for a MSU relative to an activity

In each step of a selection process the mandatory and optional capabilities contained in a MSU capability profile shall be matched with those of the activity's required profile. The matching level shall denote the result of a filtering step; see Figure 5.

The filtered behaviour of a MSU for a particular activity compared with the required behaviour of the activity shall be used as a measure for the Matching Level (ML).

The Matching Level shall assume one of the following values, as illustrated in Figure 6:

a) Complete Match — all of the mandatory and optional functions of the activity capability profile are matched by the MSU profile.

b) All Mandatory Match — all the mandatory functions of the activity capability profile are matched by the MSU profile.

c) Some Mandatory Match — some of the mandatory functions of the activity capability profile are matched by the MSU profile.

d) No Mandatory Match — none of the mandatory functions of the activity capability profile are matched by the MSU capability profile

NOTE 1 The "Some Mandatory Match" level can be used to iterate the matching process throughout the full capability class structure.

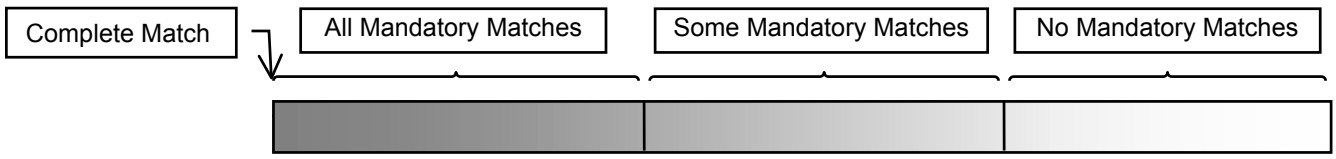

Figure 6 — Matching Level

NOTE 2 The quality of the matcher determines the extent and usefulness of reasons provided for partial matching and of hints provided on how to obtain a higher Matching Level. (refer clause 7.4)

The MSU selection process shall be considered successful when a set of MSU capability profiles match a required capability profile with a matching level of "Complete Match" or "All Mandatory Match".

6.1.3 Service set Type 1 primitives

6.1.3.1 General

The service sequences to access and match a capability profile shall support the following cases, see Figure 7.

Case 1 Request for either requirement profile or MSU profile, where capability profile is known to registry; response is the profile itself.

Case 2 Request for MSU profile, where capability profile is resident with MSU; response is the profile itself.

Case 3 Request for two profiles to be matched by matching service, normally one requirement profile and one MSU profile; in request parameter, either both profiles' information are sent or optionally, profile ID of one or both can be sent; response is the matching result that consists of the Matching Level and, at a minimum, a list of mandatory functions matched when Matching Level is "Some Mandatory Match".

Case 4 Request to a MSU to match its profile with an input profile. MSU can use a matching service similar to case 3, where the input profile and the MSU profile are provided to the matching service.

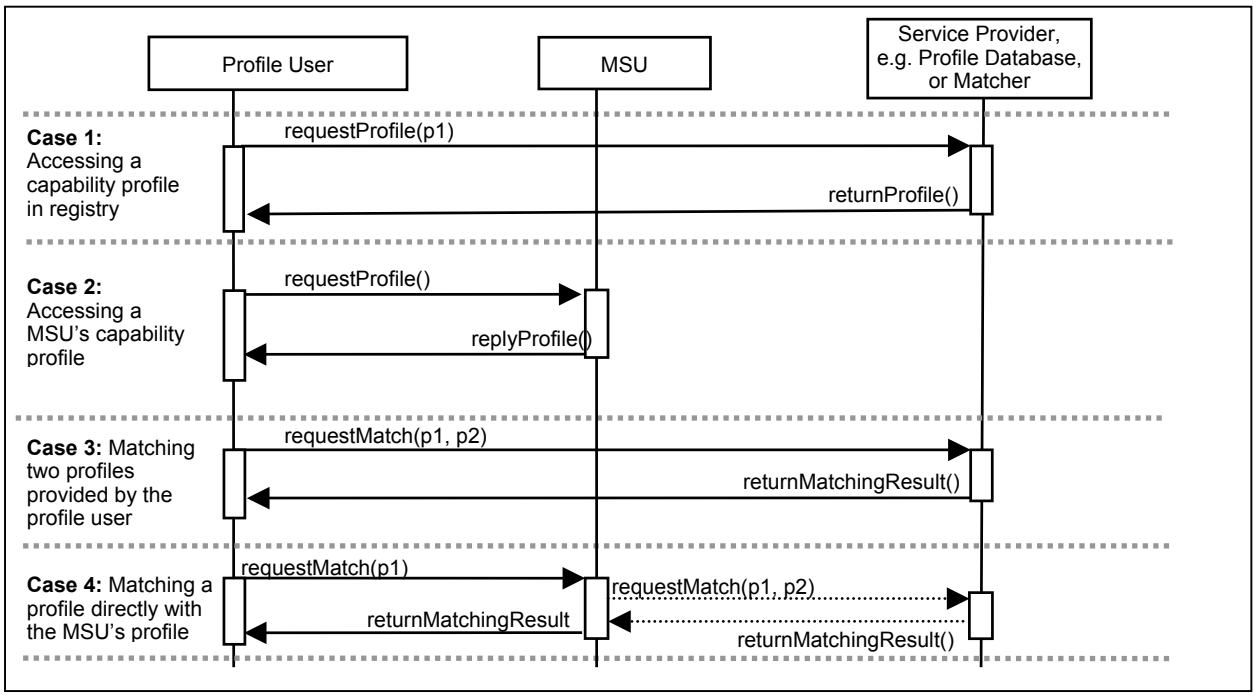

Figure 7 — Profile access service set Type 1

A service access point object, *ServiceAccessPoint*, shall be provided by a source of a capability profile or a source of a Matching Level evaluation. The source providing the profile or the matching result acts as a producer and the destination of the profile or matching result acts as a consumer.

In addition to the described parameters an additional parameter with a path to the requesting *ServiceAccessPoint* object shall be provided.

6.1.3.2 Accessing a capability profile via the Service Provider (Case 1)

The *requestProfile* service-requesting mechanism shall consist of the following steps.

a) Profile user invokes the *requestProfile* service of the *ServiceAccessPoint* object.
   1) Parameter of the service request call is: (i) capability profile ID.
   2) The behaviour of the service responder follows an asynchronous manner.

b) Service Provider invokes the *returnProfile* service of the *ServiceAccessPoint* object.
   1) Parameters of the service response are:
      (i) capability profile ID; (ii) profile contents; (iii) error status on access.
   2) The behaviour of the service requester follows an asynchronous manner.
   3) Another service request is not invoked until a service response has been received.

The capability profile name can refer to a requirement capability profile or a MSU capability profile.

6.1.3.3 Accessing a MSU's capability profile (Case 2)

The *requestProfile* service requesting mechanism shall consist of the following steps.

a) Profile user invokes the *requestProfile* service of the MSU object.
   1) Parameters of the service request call are: none.
   2) The behaviour of the service requester follows an asynchronous manner.

b) MSU invokes the *returnProfile* service of the *ServiceAccessPoint* object.
   1) Parameters of the service response are:
      (i) capability profile name; (ii) profile contents; (iii) error status on access;
   2) The behaviour of the service responder follows an asynchronous manner.
   3) Another service request is not invoked until a service response has been received.

The returned profile is the requested MSU's own capability profile.

6.1.3.4 Matching two profiles via the matcher (Case 3)

The *requestMatch* service requesting mechanism shall consist of the following steps:

a) Profile user invokes the *requestMatch* service of the *ServiceProvider* object.
   1) Parameters of the service request call are:
      (i) ID of first capability profile; (ii) ID of second capability profile.
   2) The behaviour of the service requester follows an asynchronous manner.

b) Producer invokes the *returnMatchingResult* service of the *ServiceAccessPoint* object.
   1) Parameters of the service response are:
      (i) ID of capability profile 1; (ii) ID of capability profile 2;
      (iii) the matching result; (iv) error status on access.
   2) The behaviour of the service responder follows an asynchronous manner.
   3) Another service request is not invoked until a service response has been received.

The capability profile names may refer to a requirement capability profile or a MSU capability profile.

6.1.3.5 Matching a profile directly with the MSU's profile (Case 4)

The *requestMatch* service requesting mechanism shall consist of the following steps.

a) Profile user invokes *requestMatch* service of the *ServiceAccessPoint* object.

1) Parameter of the service request call is:
(i) name of capability.

2) The behaviour of the service requester follows an asynchronous manner.

b) Producer invokes *returnMatchingResult* service of the *ServiceAccessPoint* object.

1) Parameters of the service response are:
(i) name of capability profile 1; (ii) name of MSU's capability profile;
(iii) the matching result; (iv) error status on access.

2) The behaviour of the service responder follows an asynchronous manner.

3) Another service request is not invoked until a service response has been received.

The capability profile name may refer to a requirement capability profile or a MSU capability profile.

6.1.4 Common management services for the capability profiling and analysis process

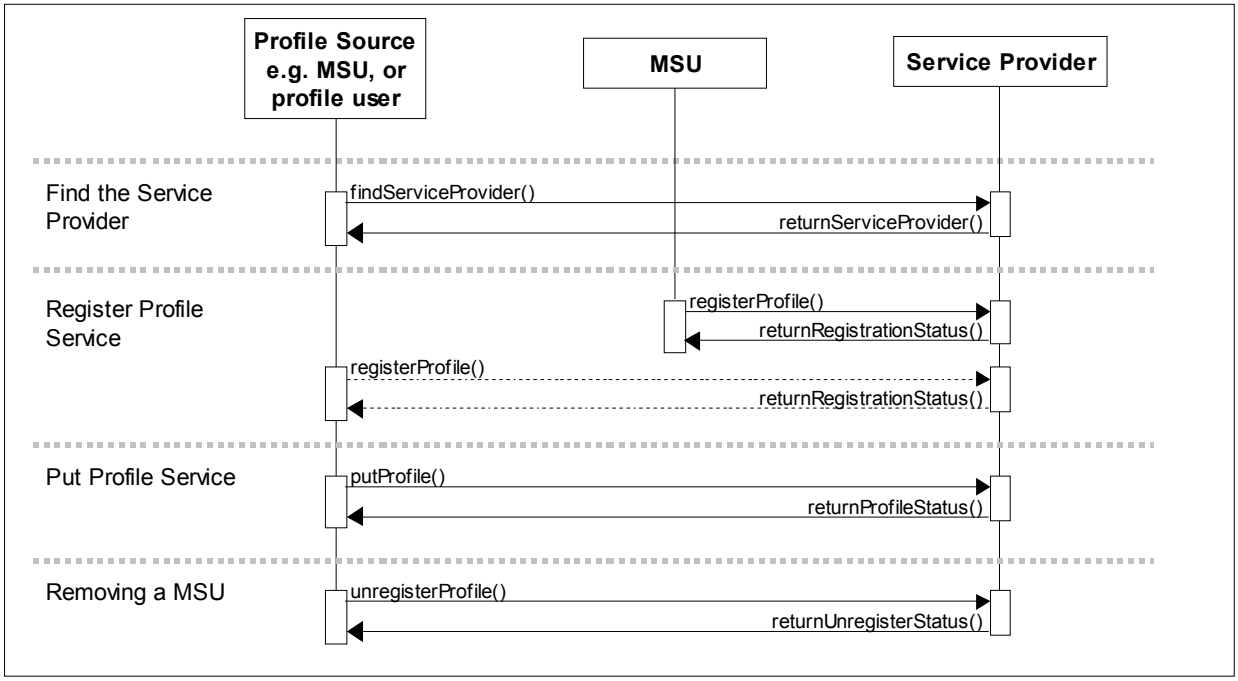

Figure 8 — Common management services

6.1.4.1 General

The common management services provided by the CPI are described in 6.1.4. These services support the Type 1 service set to get and match profiles.

A MSU shall offer its capability and discover other MSU capabilities in order to support the interactions in a common manufacturing activity. All MSUs involved in the profiling process shall use these common management services provided by the CPI.

The MSUs or other profile sources interact with the service provider using these services.

The common management services enable interoperability of handling:

a) the structure of profiles;

b) the content of profiles.

6.1.4.2 Finding the Service Provider

The findServiceProvider service requesting mechanism shall consist of the following steps.

a) Profile user invokes the findServiceProvider service of the ServiceAccessPoint object.
   1) Parameters of the service request call are: none.
   2) The behaviour of the service requester follows an asynchronous manner.

b) Producer invokes the returnServiceProvider service of ServiceAccessPoint object.
   1) Parameters of the service response are:
      (i) reference to service provider as a URI;
      (ii) error status on access.
   2) The behaviour of the service responder follows an asynchronous manner.
   3) Another service request is not invoked until a service response has been received.

6.1.4.3 Registering a Profile at the Service Provider

The registerProfile service shall enable a profile source such as a MSU to send its unambiguous and distinguishable profile ID. The registerProfile service shall perform the following tasks with the following parameters.

a) Profile user invokes the registerProfile service of the ServiceAccessPoint object.
   1) Parameter of the service request call is: (i) capability profile ID.
   2) The behaviour of the service requester follows an asynchronous manner.

b) Producer invokes the returnRegistrationStatus service of the ServiceAccessPoint object.
   1) Parameters of the registration response are: (i) status of the registration; (ii) error status on access.
   2) The behaviour of the service responder follows an asynchronous manner.
   3) Another service request is not invoked until a service response has been received.

Even though a registration of a profile is normally done by a MSU, any profile user can also register a profile.

6.1.4.4 Sending a profile to the Service Provider

The putProfile service shall enable a profile user to send a profile to the service provider. The putProfile service shall perform the following tasks with the following parameters.

a) Profile user invokes the putProfile service of the ServiceAccessPoint object.
   1) Parameters of the service request call are: (i) the capability profile ID; (ii) the capability profile.
   2) The behaviour of the service requester follows an asynchronous manner.

b) Producer invokes the returnPutProfileStatus service of the ServiceAccessPoint object.
   1) Parameters of the putProfile response are: (i) status of the profile transmission; (ii) error status on access.
   2) The behaviour of the service responder follows an asynchronous manner.
   3) Another service request is not invoked until a service response has been received.

The putProfile service is the “push” action to the corresponding “pull” action of requestProfile.

6.1.4.5 Unregistering a profile at the Service Provider

The unregisterProfile service shall be used by a profile source such as a MSU to unregister itself before removing the corresponding MSU. The unregisterProfile service shall perform the following tasks with the following parameters.

a) Profile user invokes the unregisterProfile service of the ServiceAccessPoint object.
   1) Parameter of the service request call is: (i) capability profile ID.
   2) The behaviour of the service requester follows an asynchronous manner.

b) Producer invokes the returnUnregistrationStatus service of the ServiceAccessPoint object.
   1) Parameters of the registration response are: (i) status of the unregistration; (ii) error status on access.
   2) The behaviour of the service responder follows an asynchronous manner.
   3) Another service request is not invoked until a service response has been received.

6.1.5 Validation of capability profiles

Validation for transmitted capability profiles [e.g. by putProfile(), or requestProfile()] is under the responsibility of a Type 2 or Type 3 service provider, which offers a *CommonValidationService*.

The capability profile shall always consist of a common part and a specific part. The *CommonValidationService* shall validate the capability profile against the common part of the schema defined in 7.2 and the schema of the specific part of the capability profile.

All strings used in capability profiles shall be encoded in UTF-8.

6.2 Protocol specifications

6.2.1 Service URL syntax

The protocols allow the direct issuing of requests to services. A service request is handled by the service that responds with a service reply.

A service URL starts with the string `"service:"`. The service URL includes the service type followed by the relevant service access point up to but not including the final `":"` where the address specification starts. The service-specific attribute information follows the address specification encoded according to the URL grammar.

The complete service URL shall be of the following type:

`"service:<service-type>:<service-access-point>://<address>;<attribute-list>"`

The attribute list consists of a list of semicolon ";" separated attribute assignments. The attribute assignments shall have the form

`<attribute-id>=<attribute-value>`

In addition, for keyword attributes the following form shall be used:

`<attribute-id>`

6.2.2 Type 1 service protocol

6.2.2.1 Request capability profile

The requestProfile service returns a requirement capability profile or a MSU capability profile, and shall have the service type

```
<service-type> = "requestCapabilityProfile"
```

The corresponding attribute shall be

```
capability_profile_ID="the_capability_profile_id"
```

The returnProfile service returns the requirement capability profile or a MSU capability profile, and shall have the service type

```
<service-type> = "returnCapabilityProfile"
```

The corresponding attributes shall be

```
capability_profile_ID="the_capability_profile_id"
capability_profile_content="the_capability_profile_content"
access_status="the_access_status"
```

6.2.2.2 Accessing a MSU's capability profile

The requestProfile service of a MSU returns its capability profile, and shall have the service type

```
<service-type> = "requestCapabilityProfile"
```

There are no attributes.

The returnProfile service returns the MSU capability profile, and shall have the service type

```
<service-type> = "returnCapabilityProfile"
```

The corresponding attributes shall be

```
capability_profile_ID="the_capability_profile_id"
capability_profile_content="the_capability_profile_content"
access_status="the_access_status"
```

6.2.2.3 Matching two profiles

The requestMatch service returns a matching result, and shall have the service type

```
<service-type> = "requestCapabilityProfileMatch"
```

The corresponding attributes shall be

```
capability_profile_1_ID="the_first_capability_profile_id"

capability_profile_2_ID="the_second_capability_profile_id"
```

The returnMatchingResult service returns the matching result, and shall have the service type

```
<service-type> = "returnCapabilityProfileMatchResult"
```

The corresponding attributes shall be

```
capability_profile_1_ID="the_first_capability_profile_id"
```

```
capability_profile_2_ID="the_second_capability_profile_id"
matching_result="the_matching_level;the_matching_comment"
access_status="the_access_status"
```

6.2.2.4 Matching a profile directly with the MSU's profile

The requestMatch service returns a matching result, and shall have the service type

```
<service-type> = "requestCapabilityProfileMatch"
```

The corresponding attribute shall be

```
capability_profile_ID="the_capability_profile_id"
```

The returnMatchingResult service returns the matching result, and shall have the service type

```
<service-type> = "returnCapabilityProfileMatchResult"
```

The corresponding attributes shall be

```
capability_profile_1_ID="the_capability_profile_id"
capability_profile_2_ID="the_MSU_capability_profile_id"
matching_result="the_matching_level;the_matching_comment"
access_status="the_access_status"
```

6.2.3 Common management service protocol

6.2.3.1 Finding the Service Provider

The findServiceProvider service returns the service provider, and shall have the service type

```
<service-type> = "requestServiceProvider"
```

There are no attributes.

The returnServiceProvider service returns the required service provider, and shall have the service type

```
<service-type> = "returnServiceProvider"
```

The corresponding attributes shall be

```
service_provider_URI="the_service_provider_URI"
access_status="the_access_status"
```

6.2.3.2 Registering capability profile

The registerProfile service registers a capability profile with its ID, and shall have the service type

```
<service-type> = "registerCapabilityProfile"
```

The corresponding attribute shall be

```
capability_profile_ID="the_capability_profile_id"
```

The returnRegistrationStatus service returns the registration status of the capability profile, and shall have the service type

```
<service-type> = "returnRegistrationStatus"
```

The corresponding attributes shall be

```
registration_status="the_registration_status"
acsess_status="the_access_status"
```

6.2.3.3 Sending a profile to the Service Provider

The putProfile service accepts a capability profile with its ID, and shall have the service type

```
<service-type> = "putCapabilityProfile"
```

The corresponding attributes shall be

```
capability_profile_ID="the_capability_profile_id"
capability_profile_content="the_capability_profile_content"
```

The returnPutProfileStatus service returns the transmission status of the capability profile, and shall have the service type

```
<service-type> = "returnPutProfileStatus"
```

The corresponding attributes shall be

```
transmission_status="the_transmission_status"
acsess_status="the_access_status"
```

6.2.3.4 Unregistering a capability profile

The unregisterProfile service unregisters a capability profile with its ID, and shall have the service type

```
<service-type> = "unRegisterCapabilityProfile"
```

The corresponding attribute shall be

```
capability_profile_ID="the_capability_profile_id"
```

The returnUnregistrationStatus service returns the unregistration status of the capability profile, and shall have the service type

```
<service-type> = "returnUnRegistrationStatus"
```

The corresponding attributes shall be

```
unregistration_status="the_unregistration_status"
access_status="the_access_status"
```

6.2.4 Type 2 and Type 3 service protocols

Specification of the Type 2 and Type 3 service protocols will be considered for a future edition of ISO 16100.

7 Templates

7.1 Overall structure

7.1.1 General

The structure of any capability template shall be described using a XML schema. Using the XML schema definition language, elements as well as attributes can be defined.

This template structure consists of two parts:

a) the common part of type `commonPartType`;

b) the specific part of type `specificPartType`.

7.1.2 Formal structure

The formal template structure is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="ISO16100CapabilityProfileSchema"
xmlns:cpsc="ISO16100CommonSchema" xmlns="ISO16100CapabilityProfileSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema" id="CapabilityProfiling" xml:lang="EN">
  <xs:element name="CapabilityProfiling">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="type">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="CapabilityProfile">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="pkgtype">
                <xs:complexType>
                  <xs:attribute name="version" type="xs:string" form="unqualified"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="Common" type="CommonPartType"/>
              <xs:element name="Specific" type="SpecificPartType"/>
            </xs:sequence>
            <xs:attribute name="date" type="xs:string" form="unqualified"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

7.2 Common part

7.2.1 General

The common part of the template shall consist of the following main schema elements (see Figure 6 of ISO 16100-2):

A choice of either a requirement or a MSU's profile shall be done by the identifier

`Requirement` or `MSU_Capability`

In both cases an identifier `ID` shall be added as an attribute.

`ID`	a string that distinguishes a MSU from other MSUs with differing matching levels

An unbounded sequence of pairs of `ReferenceCapabilityClassStructure` and `TemplateID` shall follow.

`ReferenceCapabilityClassStructure`	indicates the activity classes
`TemplateID`	identifier to distinguish the starting class within a reference capability class structure. Typically, the value is null if matching is required for the full capability class structure

More than one reference capability class structure can be referenced in the common part for different application areas. For each reference capability class structure name an appropriate profile shall be defined in the specific part.

Using the descriptions of the template content in 6.2.1 of ISO 16100-2 and the conceptual template structure in 6.3 of ISO 16100-2 a complete common part structure is described in 7.2.2.

7.2.2 Formal structure

The formal template structure of the `CommonPartType` is as follows:

```
<xs:complexType name="CommonPartType">
 <xs:sequence>
  <xs:choice>
   <xs:element name="Requirement">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
     </xs:sequence>
     <xs:attribute name="id" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="MSU_Capability">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
     </xs:sequence>
     <xs:attribute name="id" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
  </xs:choice>
  <xs:sequence maxOccurs="unbounded">
   <xs:element name="ReferenceCapabilityClassStructure">
    <xs:complexType>
     <xs:attribute name="id" type="xs:string" form="unqualified"/>
     <xs:attribute name="name" type="xs:string" form="unqualified"/>
     <xs:attribute name="version" type="xs:string" form="unqualified"/>
     <xs:attribute name="url" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="TemplateID">
    <xs:complexType>
     <xs:attribute name="ID" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:element name="Version">
   <xs:complexType>
    <xs:attribute name="major" type="xs:string" form="unqualified"/>
    <xs:attribute name="minor" type="xs:string" form="unqualified"/>
   </xs:complexType>
  </xs:element>
  <xs:element name="Owner">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="name" type="xs:string" minOccurs="0"/>
```

```
      <xs:element name="street" type="xs:string" minOccurs="0"/>
      <xs:element name="city" type="xs:string" minOccurs="0"/>
      <xs:element name="zip" type="xs:string" minOccurs="0"/>
      <xs:element name="state" type="xs:string" minOccurs="0"/>
      <xs:element name="country" type="xs:string" minOccurs="0"/>
      <xs:element name="comment" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="ComputingFacilities" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Processor0" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="type" type="xs:string" form="unqualified"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="OperatingSystem0" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="type" type="xs:string" form="unqualified"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Language" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="name" type="xs:string" form="unqualified"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Memory" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="size" type="xs:string" form="unqualified"/>
          <xs:attribute name="unit" type="xs:string" form="unqualified"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="DiskSpace" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="size" type="xs:string" form="unqualified"/>
          <xs:attribute name="unit" type="xs:string" form="unqualified"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="type" type="xs:string" form="unqualified"/>
  </xs:complexType>
</xs:element>
<xs:element name="Performance" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:attribute name="ElapsedTime" type="xs:string" form="unqualified"/>
    <xs:attribute name="TransactionsPerUnitTime" type="xs:string" form="unqualified"/>
  </xs:complexType>
</xs:element>
<xs:element name="ReliabilityData" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="UsageHistory" type="xs:string" minOccurs="0"/>
      <xs:element name="Shipments" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="number" type="xs:string" form="unqualified"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="IntendedSafetyIntegrity" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="level" type="xs:string" form="unqualified"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Certification" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="no1" type="xs:string" form="unqualified"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="SupportPolicy" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:attribute name="index" type="xs:string" form="unqualified"/>
  </xs:complexType>
</xs:element>
<xs:element name="PriceData" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
```

```
        <xs:attribute name="invest" type="xs:string" form="unqualified"/>
        <xs:attribute name="annualSupport" type="xs:string" form="unqualified"/>
        <xs:attribute name="unit" type="xs:string" form="unqualified"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

EXAMPLE See A.1.

7.3 Specific part

The specific part in a template shall be expressed as an XML-schema. The specific part can consist of one specific complex element or several specific complex elements, as many as described in the common part. Each of the specific complex elements of the specific part shall refer to one reference capability class structure.

This allows defining and expressing the class structure coming from the activity model.

Each template shall be derived by its reference capability class structure. The template shall describe the complete or a part of the reference capability class structure. A reference capability class structure can have more than one associated templates covering parts of the complete structure.

Both the required capability profile of an activity as well as the capability profile of a MSU shall be described using templates, expressed as an XML schema. These templates shall be derived from the same reference capability class structure.

EXAMPLE See A.2 and A.3.

7.4 Usage of Templates

Templates are used to create requirement capability profiles as well as capability profiles of MSUs.

In a matching process two profiles are compared. This needs a common base structure that is defined by the structure of the template expressed as a XML-schema.

The matching rules depend on the type of function. More or less sophisticated matcher use additional semantic information about application specific domain.

A matcher shall return the Matching Level as defined in clause 6.1.2 and a comment to the matching process.

EXAMPLE For a matching process, see A.2.

8 Conformance

The concepts and rules for conformity assessment of capability profiles are detailed in ISO 16100-4.

Annex A
(informative)

Capability profile template

A.1 General capability profile template

A.1.1 Filled template

A filled template is the capability profile. A sample is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ... -->
<CapabilityProfiling xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\...\ISO16100-General.xsd">
  <Type id="ReqProf_0012"/>
  <CapabilityProfile date="2003-12-11">
    <pkgtype version="1,0,0,2"/>
    <Common>
      <!-- Fill in the common part of the capability profile -->
      <!-- sample in A.1.2 -->
    </Common>
    <Specific>
      <!-- Fill in the specific part of the capability profile -->
      <!-- two samples, one in A.2.2 and one in A.3.2 -->
    </Specific>
  </CapabilityProfile>
</CapabilityProfiling>
```

A.1.2 Common part sample

A sample of the common part is shown below; two samples for the specific part are shown in A.2.2 and A.3.2.

```
   <Common>
     <Requirement ID="abc-Req-672726"/>
     <ReferenceCapabilityClassStructure id="CRCS_001" name="CEA_ReferenceCapabilitySchema"
version="1.0.1" url="www.asam.net/..."/>
     <TemplateID ID=" "/>
     <Version major="1" minor="1"/>
     <Owner>
       <name>ABC Inc.</name>
       <street>Waterstreet 56</street>
       <city>Softcity</city>
       <zip>734562</zip>
       <state>Alabama</state>
       <country>USA</country>
       <comment>This is a comment</comment>
     </Owner>
     <ComputingFacilities type="PC">
       <Processor type="Intel"/>
       <OperatingSystem type="WindowsXP"/>
       <Language name="en"/>
       <Memory size="138" unit="kByte"/>
       <DiskSpace size="23" unit="MByte"/>
     </ComputingFacilities>
     <Performance ElapsedTime="14ms" TransactionsPerUnitTime="743"/>
     <ReliabilityData>
       <UsageHistory>history discription</UsageHistory>
       <Shipments number="65"/>
       <IntendedSafetyIntegrity level="high"/>
       <Certification no="ISO9001"/>
     </ReliabilityData>
     <SupportPolicy index="String"/>
     <PriceData invest="10000" annualSupport="5000" unit="$"/>
   </Common>
```

A.2 Manufacturing capability class structure

A.2.1 Sample of a reference class structure using XML syntax

In accordance with 6.2.1 and 6.3 of ISO 16100-2, a reference capability class structure is built. The reference class structure shown below is not a full schema description (not all nodes are described), but rather illustrates the main principle. A filled template is described in A.2.2 and A.2.3.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Reference Capability Class Structure -->
<ReferenceCapabilityClassStructure id="CRCS_001" name=" DiscreteManufacturingActivity "
version="1.0.1" url="www.xxx.net/..."/>
<TemplateID ID="861"/>
<Version major="1" minor="1"/>
  <ManufacturingActivity level="0" ID="A">
    <DevelopProducts level="1" ID="AA">
      <EngineeringProcess level="2" ID="AA2">
        <DevelopDetailedProcessPlan level="3" ID="AA22">
          <GenerateProcessSequence level="4" ID="AA221">
            <ScheduleSetUp level="5" ID="AA2211">
            <!-- see Fig.B10 of part 1 -->
            </ScheduleSetUp>
            <ScheduleHandling level="5" ID="AA2212"></ScheduleHandling>
            <ScheduleProcessing level="5" ID="AA2213">
              <ScheduleInspection level="6" ID="AA22131"></ScheduleInspection>
              <SchedulePartMaking level="6" ID="AA22132">
                <ScheduleShaping level="7" ID="AA221321">
                  <ScheduleMaterialRemoving level="8" ID="AA2213211">
                    <!-- see Annex B of part 1 -->
                    <ScheduleMechanicalRemoving level="9" ID="AA22132111">
                      <ScheduleMachining level="10" ID="AA221321111">
                        <ScheduleSinglePointCutting level="11" ID="AA2213211111">
                          <Boring level="12" ID="AA22132111111">
                          </Boring>
                          <Threading_Sp level="12" ID="AA22132111112">
                          </Threading_Sp>
                          <Turning level="12" ID="AA22132111113">
                          </Turning>
                        </ScheduleSinglePointCutting>
                        <ScheduleMultiplePointCutting level="11" ID="AA2213211112">
                          <Drilling level="12" ID="AA22132111121">
                          </Drilling>
                          <Milling level="12" ID="AA22132111122">
                          </Milling>
                          <Threading_Mp level="12" ID="AA22132111123">
                          </Threading_Mp>
                        </ScheduleMultiplePointCutting>
                      </ScheduleMachining>
                    </ScheduleMechanicalRemoving>
                    <ScheduleChemicalRemoving>
                      <!-- continue -->
                    </ScheduleChemicalRemoving>
                  </ScheduleMaterialRemoving>
                </ScheduleShaping>
              </SchedulePartMaking>
            </ScheduleProcessing>
            <ScheduleLoadUnload></ScheduleLoadUnload>
            <ScheduleIdling></ScheduleIdling>
          </GenerateProcessSequence>
          <GenerateOperations level="4" ID="AA222"></GenerateOperations>
        </DevelopDetailedProcessPlan>
      </EngineeringProcess>
    </DevelopProducts>
  </ManufacturingActivity>
  <ManufacturingInformationExchange level="0" ID="C">
    <ComputingFacilitiesRequired level="1" ID="CC">
      <Processor level="2" ID="CC1">
        <RISC level="3" ID="CC11"/>
        <CISC level="3" ID="CC12"/>
        <DSP level="3" ID="CC13"/>
        <ASIC level="3" ID="CC14"/>
      </Processor>
      <OperatingSystem level="2" ID="CC2">
        <Windows level="3" ID="CC21">
          <XP level="4" ID="CC211">
```

```
        </XP>
      </Windows>
      <JavaVM level="3" ID="CC22">
      </JavaVM>
      <Linux level="3" ID="CC23">
      </Linux>
      <Unix level="3" ID="CC24">
      </Unix>
      <GenericPLC_OS level="3" ID="CC25">
      </GenericPLC_OS>
    </OperatingSystem>
    <Language level="2" ID="CC3">
      <!-- refer to proper standard in ISO -->
    </Language>
    <Memory level="2" ID="CC4">
      <Volatile level="3" ID="CC41">
      </Volatile>
      <NonVolatile level="3" ID="CC42">
      </NonVolatile>
    </Memory>
    <DiskSpace level="2" ID="CC5">
      <capacity>40GB</capacity>
      <partitioning>fixed</partitioning>
    </DiskSpace>
  </ComputingFacilitiesRequired>
 </ManufacturingInformationExchange>
</ReferenceCapabilityClassStructure>
```

A.2.2 Example of a requirement capability profile

The following sample shows a requirement capability profile using the template (schema) of Annex B according to the activity model from ISO 16100-2, B.2 (Develop Product). For more details see the capability collaboration diagram in Figure B.11.

```
<?xml version="1.0" encoding="UTF-8"?>
<CapabilityProfiling xmlns="http://tempuri.org/CapabilityProfileTemplate.xsd">
  <Type id="ReqProf_786z7" />
  <CapabilityProfile date="2003-09-11">
  <pkgtype version="1,0,0,2"></pkgtype>
    <Common>
      <Requirement ID="81-0001"/>
      <ReferenceCapabilityClassStructure id="rcs_1001" name="DiscreteManufacturingActivity"
version="001" url="" />
     <TemplateID ID="manuAct32" />
     <Version major="7" minor="3" />
     <Owner>
       <name>MM Production Inc.</name>
       <street>Summer Ave.7</street>
       <city>Softcity</city>
       <zip>4711</zip>
       <state>CA</state>
       <country>USA</country>
       <comment>Only best experiences!</comment>
     </Owner>
     <ComputingFacilities type="required">
       <Processor type="INTEL" />
       <OperatingSystem type="LINUX" />
       <Language name="EN" />
       <Memory size="28" unit="MB" />
       <DiskSpace size="30" unit="GB" />
     </ComputingFacilities>
     <Performance ElapsedTime="61ms" TransactionsPerUnitTime="621" />
     <ReliabilityData>
       <UsageHistory>
          abc1
          abc2
        </UsageHistory>
       <Shipments number="55" />
       <IntendedSafetyIntegrity level="3" />
       <Certification no1="ISO9001" />
     </ReliabilityData>
     <SupportPolicy index="23" />
     <PriceData invest="12000" annualSupport="2400" unit="USD" />
     <TemplateID ID_Number="Ex_A22_DevelopProdcuct_ISO-DIS16100-01" />
```

```
    <!-- gives the uppest level, relative root; e.g. to level 4-->
    <CapabilityClassName id="ccn_1001"> GenerateProcessSequence </CapabilityClassName>
    <CapabilityClassName id="ccn_1002"> GenerateControlPrograms </CapabilityClassName>
   </Common>
   <Specific>
    <ReferenceCapabilityClassStructure id="rcs_1001">
     <ManufacturingActivity>
      <!-- for standard matching rules -->
      <Activity id="act_2001" level="12" mandatoryLevel="10">
       <Boring>
       </Boring>
      </Activity>
      <Activity id="act_2002" level="12" mandatoryLevel="10">
       <Threading_sp>
       </Threading_sp>
      </Activity>
      <Activity id="act_2003" level="12" mandatoryLevel="10">
       <Threading_Mp>
       </Threading_Mp>
      </Activity>
      <Activity id="act_2004" level="12" mandatoryLevel="10">
       <Drilling attr1="abc">
        <Material type="Copper">
         <Depth> 100mm </Depth>
         <Speed> 5mm/s </Speed>
        </Material>
       </Drilling>
      </Activity>
      <!-- end of standard matching rules -->
     </ManufacturingActivity>
     <ManufacturingInformationExchange>
      <Processor id="111" level="3" mandatoryLevel="3">
       <RISC />
      </Processor>
      <OperatingSystem id="112" level="3" mandatoryLevel="2">
       <XP />
      </OperatingSystem>
     </ManufacturingInformationExchange>
    </ReferenceCapabilityClassStructure>
   </Specific>
  </CapabilityProfile>
</CapabilityProfiling>
```

A.2.3 Example of a capability profile of a MSU

The following sample describes the capability of a MSU according to the activity model from ISO 16100-1, B.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<CapabilityProfiling xmlns="http://tempuri.org/CapabilityProfileTemplate.xsd">
  <Type id="MSU_Profile" />
  <CapabilityProfile date="2003-09-11">
  <pkgtype version="1,0,0,2"></pkgtype>
   <Common>
    <MSU_Capability ID="xyz-MSU-101010"/>
    <ReferenceCapabilityClassStructure id="rcs_1001" name="DiscreteManufacturingActivity"
version="001" url="" />
    <ReferenceCapabilityClassStructure id="rcs_1002" name="DiscreteEngineeringActivity"
version="001" url="" />
    <Requirement ID="81-0001"></Requirement>
    <Version major="2" minor="3" />
    <Owner>
     <name>MSU Developer Inc.</name>
     <street>Winter Ave.7</street>
     <city>Softcity</city>
     <zip>4712</zip>
     <state>CA</state>
     <country>USA</country>
     <comment>Only best experiences!</comment>
    </Owner>
    <ComputingFacilities type="required">
     <!-- see Fig 6 part 2 -->
     <Processor type="INTEL" />
     <OperatingSystem type="LINUX" />
     <Language name="EN" />
```

```
      <Memory size="28" unit="MB" />
      <DiskSpace size="30" unit="GB" />
    </ComputingFacilities>
    <Performance ElapsedTime="61ms" TransactionsPerUnitTime="621" />
    <TemplateID ID_Number="Ex_A22_DevelopProdcuct_ISO-DIS16100-01" />
    <!-- gives the uppest level, relative root; e.g. to level 4-->
    <CapabilityClassName id="ccn_1001"> GenerateProcessSequence </CapabilityClassName>
    <CapabilityClassName id="ccn_1002"> GenerateControlPrograms </CapabilityClassName>
  </Common>
  <Specific>
    <ReferenceCapabilityClassStructure id="rcs_2001">
      <ManufacturingActivity>
        <!-- for standard matching rules -->
        <Activity id="act_2001" level="12" mandatoryLevel="10">
          <Boring>
          </Boring>
        </Activity>
        <Activity id="act_2002" level="12" mandatoryLevel="10">
          <Threading_sp>
          </Threading_sp>
        </Activity>
        <Activity id="act_2003" level="12" mandatoryLevel="10">
          <Threading_Mp>
          </Threading_Mp>
        </Activity>
        <Activity id="act_2004" level="12" mandatoryLevel="10">
          <Drilling attr1="abc">
            <Material type="Copper">
              <Depth> 200mm </Depth>
              <Speed> 8mm/s </Speed>
            </Material>
            <Material type="Steel">
              <Depth> 120mm </Depth>
              <Speed> 4mm/s </Speed>
            </Material>
            <Material type="Aluminium">
              <Depth> 250mm </Depth>
              <Speed> 7mm/s </Speed>
            </Material>
          </Drilling>
        </Activity>
        <!-- end of standard matching rules -->
      </ManufacturingActivity>
      <ManufacturingInformationExchange>
        <Processor id="111" level="3" mandatoryLevel="3">
          <RISC />
        </Processor>
        <OperatingSystem id="112" level="3" mandatoryLevel="2">
          <XP />
        </OperatingSystem>
      </ManufacturingInformationExchange>
    </ReferenceCapabilityClassStructure>
    <ReferenceCapabilityClassStructure id="rcs_3002">
      <ManufacturingActivity>
        <!-- for standard matching rules -->
        <Activity id="act_3001" level="7" mandatoryLevel="6">
          <ControlFunctionBlock>
          </ControlFunctionBlock>
        </Activity>
        <Activity id="act_3002" level="7" mandatoryLevel="6">
          <IOFunctionBlock>
          </IOFunctionBlock>
        </Activity>
        <!-- end of standard matching rules -->
      </ManufacturingActivity>
      <ManufacturingInformationExchange>
        <Processor id="115" level="3" mandatoryLevel="3">
          <ARM />
        </Processor>
        <OperatingSystem2 id="1" level="3" mandatoryLevel="2">
          <GenericPLC_OS />
        </OperatingSystem2>
      </ManufacturingInformationExchange>
    </ReferenceCapabilityClassStructure>
  </Specific>
 </CapabilityProfile>
</CapabilityProfiling>
```

A.2.4 Matching a required capability profile with one of a MSU

The matching process has to take into account all items of the underlying template for the activity “drilling” in A.2.3.

The MSU offers drilling of three different materials with three different characteristics as speed and depth. Looking into the requirement, only one type of drilling is required: “Copper”. Both attributes for “Copper” speed and depth are satisfied with the MSU profile. So there exists a full match for this item.

The matching process has to continue for all the other items to get a total Matching Level.

A.3 Capability class structure for a test unit

A.3.1 Sample of a reference class structure using XML syntax

Sample of a reference dictionary for a test unit.

```
<?xml version="1.0"?>
<!-- edited with . . .  -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="qualified" id="CapabilityProfiling">
  <xs:element name="CapabilityProfiling">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="type">
          <xs:complexType>
            <xs:attribute name="id" type="xs:string" form="unqualified"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="ReferenceCapabilityClassStructure">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="TestActivities">
                <xs:complexType>
                  <xs:choice>
                    <xs:element name="Importer">
                      <xs:complexType>
                        <xs:choice>
                          <xs:element name="File">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="SingleFile" minOccurs="0"
maxOccurs="unbounded">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="ASCII" minOccurs="0"
maxOccurs="unbounded">
                                        <xs:complexType>
                                          <xs:sequence>
                                            <xs:element name="FeatureS" minOccurs="0"
maxOccurs="unbounded">
                                              <xs:complexType>
                                                <xs:sequence>
                                                  <xs:element name="Feature" nillable="true"
minOccurs="0" maxOccurs="unbounded">
                                                    <xs:complexType>
                                                      <xs:simpleContent>
                                                        <xs:extension base="xs:string"/>
                                                      </xs:simpleContent>
                                                    </xs:complexType>
                                                  </xs:element>
                                                </xs:sequence>
                                              </xs:complexType>
                                            </xs:element>
                                          </xs:sequence>
                                          <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                          <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
```

```
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="EDAS" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                      <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="Excel" minOccurs="0"
maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="Excel-Binary" minOccurs="0"
maxOccurs="unbounded">
                          <xs:complexType>
                            <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                            <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="Excel-Text" minOccurs="0"
maxOccurs="unbounded">
                          <xs:complexType>
                            <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                            <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                      <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                      <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="level" type="xs:int" form="unqualified"/>
                <xs:attribute name="ID" type="xs:string" form="unqualified"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="MultipleFile" minOccurs="0"
maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Diadem" minOccurs="0"
maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                      <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="level" type="xs:int" form="unqualified"/>
                <xs:attribute name="ID" type="xs:string" form="unqualified"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="Folder" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="ISO13499" minOccurs="0"
maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                      <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="level" type="xs:int" form="unqualified"/>
                <xs:attribute name="ID" type="xs:string" form="unqualified"/>
              </xs:complexType>
```

```
                        </xs:element>
                      </xs:sequence>
                      <xs:attribute name="level" type="xs:int" form="unqualified"/>
                      <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="Database">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="SQL" minOccurs="0" maxOccurs="unbounded">
                          <xs:complexType>
                            <xs:attribute name="level" type="xs:int" form="unqualified"/>
                            <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="ODS" minOccurs="0" maxOccurs="unbounded">
                          <xs:complexType>
                            <xs:attribute name="level" type="xs:int" form="unqualified"/>
                            <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                      <xs:attribute name="level" type="xs:int" form="unqualified"/>
                      <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                    </xs:complexType>
                  </xs:element>
                </xs:choice>
                <xs:attribute name="level" type="xs:int" form="unqualified"/>
                <xs:attribute name="ID" type="xs:string" form="unqualified"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="Worker">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="Calculation">
                    <xs:complexType>
                      <xs:choice>
                        <xs:element name="Formula">
                          <xs:complexType>
                            <xs:choice>
                              <xs:element name="FormulaEditor">
                                <xs:complexType>
                                  <xs:sequence>
                                    <xs:element name="ArithmetikOperators" minOccurs="0"
maxOccurs="unbounded">
                                      <xs:complexType>
                                        <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                        <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                                      </xs:complexType>
                                    </xs:element>
                                    <xs:element name="ArithmeticFunctions" minOccurs="0"
maxOccurs="unbounded">
                                      <xs:complexType>
                                        <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                        <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                                      </xs:complexType>
                                    </xs:element>
                                    <xs:element name="StatisticalFunctions" minOccurs="0"
maxOccurs="unbounded">
                                      <xs:complexType>
                                        <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                        <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                                      </xs:complexType>
                                    </xs:element>
                                    <xs:element name="TrigonometricFunctions" minOccurs="0"
maxOccurs="unbounded">
                                      <xs:complexType>
                                        <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                        <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                                      </xs:complexType>
```

```
                                </xs:element>
                                <xs:element name="HyperbolicFunctions" minOccurs="0"
maxOccurs="unbounded">
                                  <xs:complexType>
                                    <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                    <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                                  </xs:complexType>
                                </xs:element>
                              </xs:sequence>
                              <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                              <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="FixedFormula">
                            <xs:complexType>
                              <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                              <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                            </xs:complexType>
                          </xs:element>
                        </xs:choice>
                        <xs:attribute name="level" type="xs:int" form="unqualified"/>
                        <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="FrequenyAnalysis">
                      <xs:complexType>
                        <xs:choice>
                          <xs:element name="FFT">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="TwoToTwoUnLimited" type="xs:string"
minOccurs="0"/>
                              </xs:sequence>
                              <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                              <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="DFT">
                            <xs:complexType>
                              <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                              <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                            </xs:complexType>
                          </xs:element>
                        </xs:choice>
                        <xs:attribute name="level" type="xs:int" form="unqualified"/>
                        <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="Filtering">
                      <xs:complexType>
                        <xs:choice>
                          <xs:element name="FIR-Filter">
                            <xs:complexType>
                              <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                              <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="FFT-Filter">
                            <xs:complexType>
                              <xs:choice>
                                <xs:element name="CFC">
                                  <xs:complexType>
                                    <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                    <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
```

```
                                    </xs:complexType>
                                  </xs:element>
                                  <xs:element name="xyz">
                                    <xs:complexType>
                                      <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                      <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                                    </xs:complexType>
                                  </xs:element>
                                </xs:choice>
                                <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                              </xs:complexType>
                            </xs:element>
                          </xs:choice>
                          <xs:attribute name="level" type="xs:int" form="unqualified"/>
                          <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="Statistic">
                        <xs:complexType>
                          <xs:choice>
                            <xs:element name="HIC">
                              <xs:complexType>
                                <xs:attribute name="level" type="xs:int"
form="unqualified"/>
                                <xs:attribute name="ID" type="xs:string"
form="unqualified"/>
                              </xs:complexType>
                            </xs:element>
                          </xs:choice>
                          <xs:attribute name="level" type="xs:int" form="unqualified"/>
                          <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                        </xs:complexType>
                      </xs:element>
                    </xs:choice>
                    <xs:attribute name="level" type="xs:int" form="unqualified"/>
                    <xs:attribute name="ID" type="xs:string" form="unqualified"/>
                  </xs:complexType>
                </xs:element>
              </xs:choice>
              <xs:attribute name="level" type="xs:int" form="unqualified"/>
              <xs:attribute name="ID" type="xs:string" form="unqualified"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="Exporter">
            <xs:complexType>
              <xs:attribute name="level" type="xs:int" form="unqualified"/>
              <xs:attribute name="ID" type="xs:string" form="unqualified"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="Viewer">
            <xs:complexType>
              <xs:attribute name="level" type="xs:int" form="unqualified"/>
              <xs:attribute name="ID" type="xs:string" form="unqualified"/>
            </xs:complexType>
          </xs:element>
        </xs:choice>
        <xs:attribute name="level" type="xs:int" form="unqualified"/>
        <xs:attribute name="ID" type="xs:string" form="unqualified"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="InformationExchange">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="InputDataTypes" type="DataTypes"/>
          <xs:element name="OutputDataTypes" type="DataTypes"/>
        </xs:sequence>
        <xs:attribute name="level" type="xs:int" form="unqualified"/>
        <xs:attribute name="ID" type="xs:string" form="unqualified"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" form="unqualified"/>
</xs:complexType>
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="DataTypes">
  <xs:sequence>
   <xs:element name="Numerical" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Zero-Dimensional" type="BasicNumeric"/>
      <xs:element name="One-Dimensional" type="BasicNumeric"/>
      <xs:element name="Two-Dimensional" type="BasicNumeric"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="MultiMedia" type="BasicMultiMedia" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="BasicNumeric">
  <xs:sequence>
   <xs:element name="double" minOccurs="0">
    <xs:complexType>
     <xs:attribute name="level" type="xs:int" form="unqualified"/>
     <xs:attribute name="ID" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="int" minOccurs="0">
    <xs:complexType>
     <xs:attribute name="level" type="xs:int" form="unqualified"/>
     <xs:attribute name="ID" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="string" minOccurs="0">
    <xs:complexType>
     <xs:attribute name="level" type="xs:int" form="unqualified"/>
     <xs:attribute name="ID" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="time" minOccurs="0">
    <xs:complexType>
     <xs:attribute name="level" type="xs:int" form="unqualified"/>
     <xs:attribute name="ID" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="BasicMultiMedia">
  <xs:sequence>
   <xs:element name="Video" minOccurs="0">
    <xs:complexType>
     <xs:attribute name="level" type="xs:int" form="unqualified"/>
     <xs:attribute name="ID" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="Audio" minOccurs="0">
    <xs:complexType>
     <xs:attribute name="level" type="xs:int" form="unqualified"/>
     <xs:attribute name="ID" type="xs:string" form="unqualified"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

A.3.2 Example of a requirement capability profile

The following sample shows a requirement capability profile using the template (schema) of Annex A.3.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<CapabilityProfiling>
 <Type id="Requirement Profile"/>
 <CapabilityProfile date="2003-09-11">
  <!-- ISO 8601 defined date yyyy-mm-dd -->
  <pkgtype version="V01.01.03a"/>
```

```
   <!--DTD-NAME= V<Application Profile Nr>.<Version Nr>.<Revision Nr>[<Patch Level>]-->
   <Common>
     <Requirement id="AMS-r701-0001" version="V01.01.01a"/>
     <Owner>
       <name>MSU User Ltd.</name>
       <street>Spring Blv.2</street>
       <city>Softcity</city>
       <zip>0915</zip>
       <state/>
       <country>France</country>
       <comment>Never seen before</comment>
     </Owner>
     <ComputingFacilities type="offered">
       <!-- see Fig 6 part 2 -->
       <Processor type="INTEL"/>
       <OperatingSystem>LINUX</OperatingSystem>
       <Language name="EN"/>
       <Memory size="28" unit="MB"/>
       <DiskSpace size="30" unit="GB"/>
     </ComputingFacilities>
     <Performance>
       <ElapsedTime value="200ms"/>
       <TransactionsPerSecond value="200"/>
     </Performance>
     <TemplateID ID_Number="Cea_A1_Evaluation_ISO-DIS16100-01" />
     <!-- gives the uppest level, relative root; e.g. to level 4 -->
     <CapabilityClassName id="ccn_1001"> Calculate_FFT </CapabilityClassName>
   </Common>
   <Specific>
     <ReferenceCapabilityClassStructure id="cea_1001" name="CEA_TestActivity"
version="001" url="">
       <TestActivities>
         <FormulaEditor level="4" ID="ABAAA" status="mandatory"/>
       </TestActivities>
       <InformationExchange>
         <InputDataTypes level="1" ID="BA">
           <DoubleValue level="4" ID="BAAAA" status="optional"/>
           <IntegerValue level="4" ID="BAAAB" status="optional"/>
           <DoubleVector level="4" ID="BAABA" status="mandatory"/>
           <IntegerVector level="4" ID="BAABB" status="optional"/>
         </InputDataTypes>
         <OutputDataTypes level="1" ID="BB">
           <Vector level="3" ID="BBAB" status="mandatory"/>
         </OutputDataTypes>
       </InformationExchange>
     </ReferenceCapabilityClassStructure>
   </Specific>
  </CapabilityProfile>
</CapabilityProfiling>
```

A.3.3 Example of a capability profile of a MSU

The following sample describes a capability profile of a CEA_TestActivity which can perform a FFT analysis where double values as well as integer values are accepted as input. The calculated result will be offered as double values.

```
<?xml version="1.0" encoding="UTF-8"?>
<CapabilityProfiling>
  <Type id="Capability Profile"/>
  <CapabilityProfile date="2003-11-13">
    <!-- ISO 8601 defined date yyyy-mm-dd -->
    <pkgtype version="V01.01.01a"/>
    <!--DTD-NAME= V<Application Profile Nr>.<Version Nr>.<Revision Nr[<Patch Level>]-->
    <Common>
      <MSU_Capability id="AMS-101-0001" name="SampleCeaWorker" version="V01.02.01a"
uri=""/>
      <Owner>
        <name>MSU Developer Inc.</name>
        <street>Winter Ave.7</street>
        <city>Softcity</city>
        <zip>4712</zip>
        <state>CA</state>
        <country>USA</country>
        <comment>Only best experiences!</comment>
      </Owner>
```

```
    <ComputingFacilities type="required">  <!-- see Fig 6 part 2 -->
      <Processor type="INTEL"/>
      <OperatingSystemS>
        <OperatingSystem>LINUX</OperatingSystem>
        <OperatingSystem>JAVA</OperatingSystem>
      </OperatingSystemS>
      <Language name="EN"/>
      <Memory size="28" unit="MB"/>
      <DiskSpace size="30" unit="GB"/>
    </ComputingFacilities>
    <Performance>
      <ElapsedTime value="61ms"/>
      <TransactionsPerSecond value="621"/>
    </Performance>
    <TemplateID ID_Number="Cea_A1_Evaluation_ISO-DIS16100-01" />
    <!-- gives the uppest level, relative root; e.g. to level 4-->
    <CapabilityClassName id="ccn_1001"> Calculate_FFT </CapabilityClassName>
   </Common>
   <Specific>
    <ReferenceCapabilityClassStructure id="cea_1001" name="CEA_TestActivity"
version="001" url="">
      <TestActivities>
        <Worker>
          <Calculation>
            <FrequencyAnalysis>
              <FFT/>
            </FrequencyAnalysis>
          </Calculation>
        </Worker>
      </TestActivities>
      <InformationExchange>
        <InputDataTypes level="1" ID="BA">
          <Numerical level="2" ID="BAA">
            <One-Dimensional level="3" ID="BAAA">
              <double level="3" ID="BAAAA"/>
              <int level="3" ID="BAAAB"/>
            </One-Dimensional>
          </Numerical>
        </InputDataTypes>
        <OutputDataTypeslevel="1" ID="BB">
          <Numerical level="2" ID="BAA">
            <One-Dimensional level="3" ID="BAAA">
              <double level="3" ID="BAAAA"/>
            </One-Dimensional>
          </Numerical>
        </OutputDataTypes>
      </InformationExchange>
    </ReferenceCapabilityClassStructure>
   </Specific>
  </CapabilityProfile>
</CapabilityProfiling>
```

Annex B
(informative)

Capability reference models

B.1 Capability class diagram and object model

An information sharing and exchange model for manufacturing process shall include an object model for classes used in process. The object model shall describe the data and functions used in the design and manufacturing process. The main purpose of the model development is to facilitate the creation of standard information content specifications that are necessary for capability profiling.

For example, to achieve truly collaborative design and manufacturing, information representations of both product design and manufacturing processes must support multiple levels of abstraction for bi-directional (or multi-directional) communication between the application processes. During the early conceptual product design phase, it is important to understand the trade-offs and implications of high-level design decisions. Symbolic descriptions of product designs, which are not yet defined geometrically, can yield enough input to determine many characteristics of the manufacturing process with underlying cost estimates. The formal representation of such early design descriptions can enable input to conceptual process planning and manufacturing applications. The formal representations can be used to develop interface specifications for integrating product design and manufacturing engineering activities.

A class is an information construct to represent a concept or physical object. It consists of attributes and functions. A class can extend another class by inherence with all the attributes from that class. The information model has the following key words: class, extend, string, integer, and double.

The information exchange between product design and manufacturing process planning applications (and other applications as well) occur at more than one stage. Figure B.1 illustrates many stages of communication that can exist when establishing interoperability between conceptual design and conceptual process planning.

Conceptual design is an activity in the early design stage in which the concept of a product is formulated. The concept of a product includes product requirements, functions, possible behaviours, form/structure (layout), and associated properties.

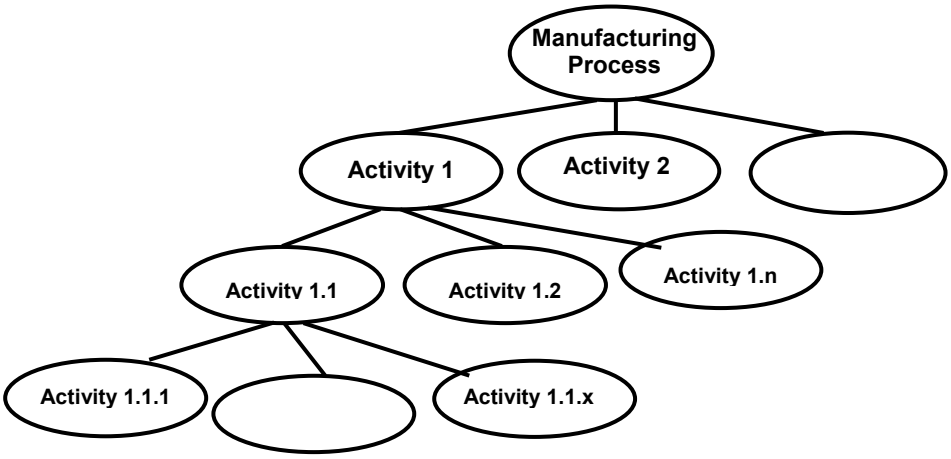

Figure B.1 — Hierarchy or nesting of activities within a manufacturing process

Properties include material, assembly-level tolerance, critical surface roughness and hardness parameters, and critical dimensions. Functional product design generates product major functions and decomposes these functions into detailed functions from requirements. Behavioural design maps detailed functions into behaviour. Embodiment design specifies product form and structure based on functions and behaviours. Detailed product

information is derived from the conceptual design. Conceptual design information is used in the detailed design process, such as geometry, topology, tolerance, and dimension specification.

A Conceptual Design activity (B1) is decomposed into five subactivities, A11 to A15, shown with the data flow in Figure B.2. Activity A11 is to define product functions and constraints based on the input, engineering requirements. This activity is called functional design. Activity A12 is to generate product behaviours based on product functions and constraints — the output from A11. This activity is to generate behaviours. For those products that do not have behaviours, such as static structural objects, this activity should be skipped. Activity A13 is to decompose functions, constraints, and behaviours so that each part, subassembly, and assembly of a product has its own functions, constraints and, if applicable, behaviours. With decomposed functions, constraints, and behaviours, parts can be designed and product can be configured by these parts in A14. Activity A14 can be decomposed into two subactivities: A141 to specify product structure based on function, and A142 to specify detailed information about the artefact to be manufactured.

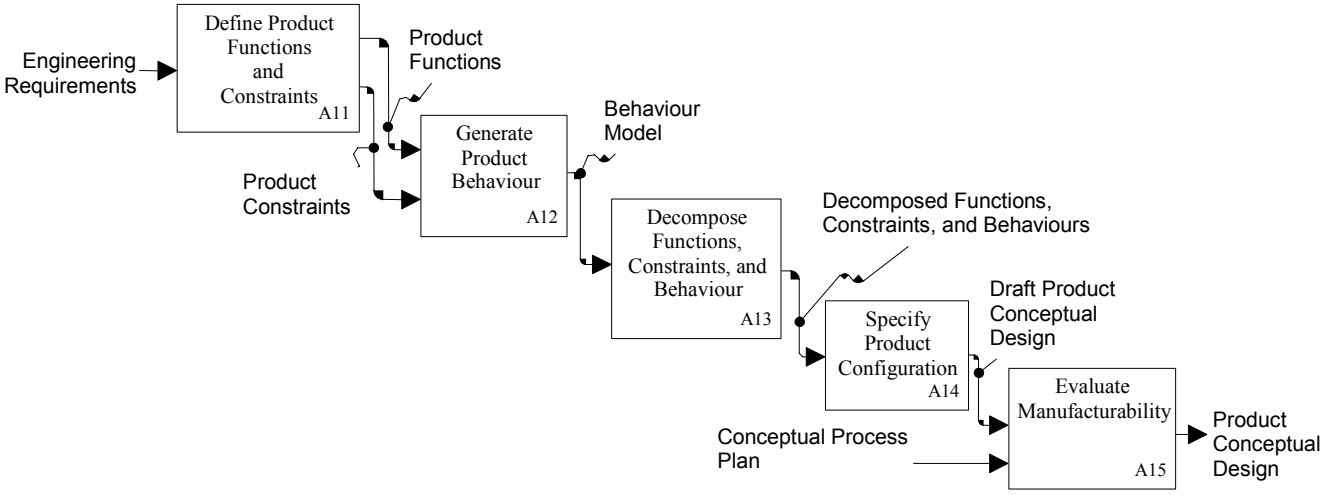

Figure B.2 — Functional decomposition of Conceptual Design activity

The output of A14 is the concept of a product that is the product conceptual design. Finally, this conceptual design is evaluated for manufacturability in A15. The manufacturability analysis takes input from both conceptual process planning and conceptual design. The form and structure of the product is conceptualized in this subactivity. The detailed information generated is necessary for process planning such as manufacturing process selection, resource selection, and cost estimating.

The activity model describes functions and their input and output data in conceptual design, based on the concept of integrated design and process planning. The activity model provides a context in which object model is developed.

The object model contains many aspects of conceptual design described in the activity model. The whole model of an artefact, its function, and the assembly relationships is described below. The classes pertaining to product requirements are not described because conceptual design starts from functional design. The model is represented using the class diagrams of the Unified Modelling Language (UML). The concept of the following classes is adopted from the core: Artefact, Function, and Constraint. The following classes are derived from the core: Behaviour, Material, Requirement, and Constraint. The rest of the classes are specific to the conceptual design integrated with conceptual process planning. The intended uses of this model are as follows:

to be a reference architecture for the next-generation conceptual design;

to serve as a basis for developing standard interfaces;

to be used as database schema.

Artefact is a general term for referring to an individual component in a product hierarchical structure, such as a part, subassembly, assembly, or the product. Specific information about an artefact includes material parameters, tolerances, dimensions, surface roughness parameters, surface hardness parameters, and texture parameters.

Figure B-3 shows the class diagram for Artefact . The Artefact class has a recursive definition. Artefacts are in a hierarchical structure, which can capture the product structure in a hierarchical structure. Artefact has two derived classes: ArtefactToBeMade and ArtefactToBeBought. Some components (artefact s) can be purchased from suppliers. Others have to be made by the company in which the product is designed. ArtefactToBeMade has a sequence of manufacturing processes. ArtefactToBeBought can be gotten from suppliers. The Supplier class describes the information about suppliers.

Artefact has the following associated classes: EngineeringRequirement, ArtefactConstraint, Function, Behaviour, Form, Material, SurfaceCondition, and Tolerance.

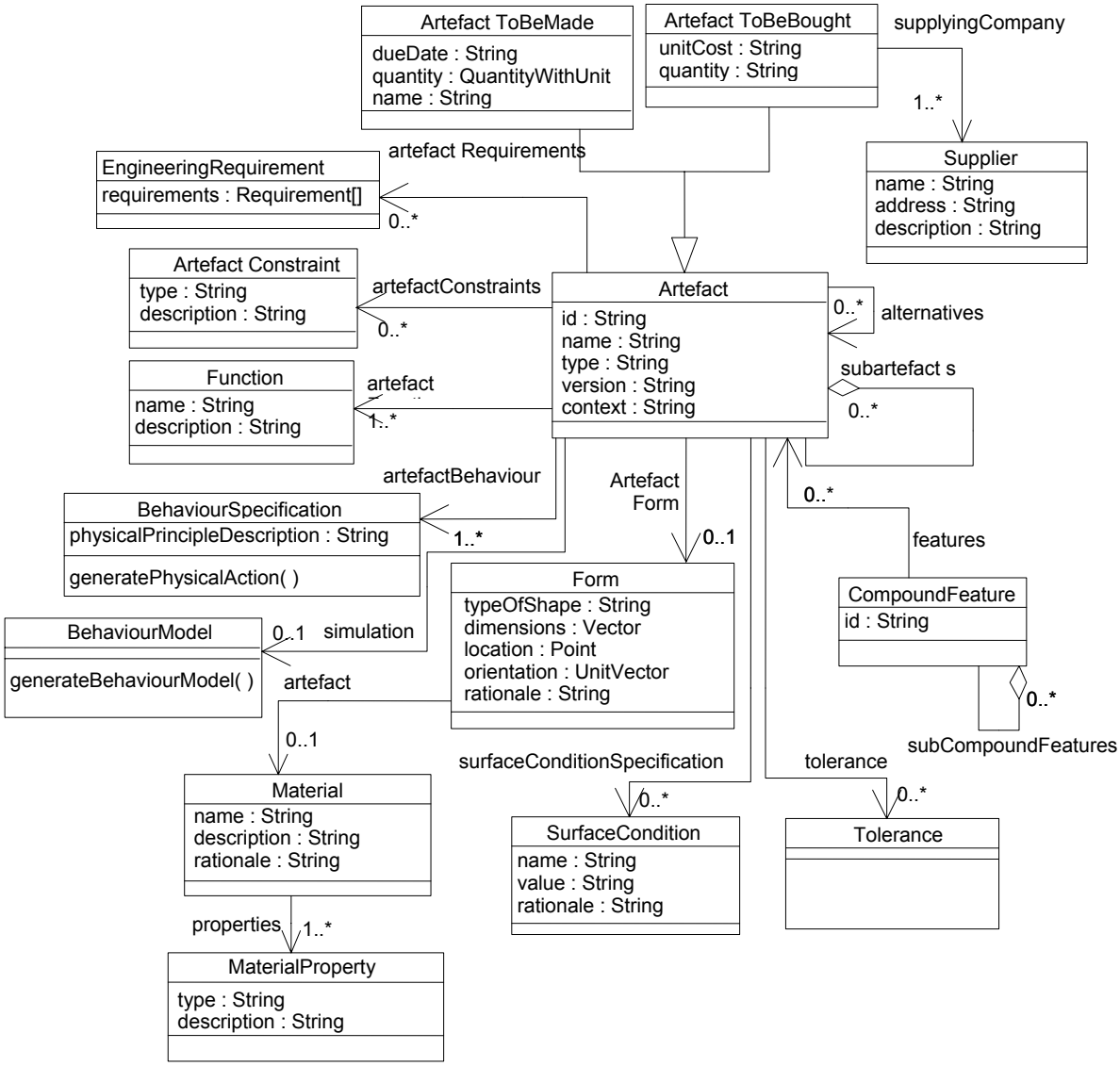

Figure B.3 — Class diagram for Artefact

The EngineeringRequirement class captures the requirements related to engineering, such as motion requirements, power requirements and tolerance requirements.

The ArtefactConstraint class relates to those factors that prohibit product to function, such as physical limitations, environmental concerns, and safety regulations.

The Tolerance class defines the limits within which a feature can vary. Tolerance includes form tolerance, location tolerance, profile tolerance, and runout tolerance. Figure B.4 illustrates classes that relate to tolerance. Rationale captures the reasons that tolerance is defined, material is selected, and form and feature are designed. The Tolerance class has three subclasses: ShapeTolerance, PositionTolerance, RunoutTolerance.

The ShapeTolerance Class has two subclasses: ProfileTolerance and FormTolerance. The PositionTolerance class has two subclasses: LocationTolerance and OrientationTolerance.

The Function class captures product functions, translated from engineering requirements. Function is the intended use or purpose of device, it represents the transformation relationship between input and output, i.e., energy, material and information flow. Transfer function, as in the Function model in Figure B.5, is a derived class from Function. It captures those transferring functions that have input and output of information, material, and/or energy. Input has a type of input, quantity, and the source. Similarly, Output has a type, quantity, and the destination. The functions of manufactured parts do vary and have to be constrained by AllowableVariation, which sets the limits for which a function of an artefact can vary. The allowable functional variation can be converted into tolerances on certain feature parameters using, for instance, Taguchi methods.

The Form class has the conceptual or sketched shape of the product and its components. Form describes the type of shape, dimension, location, orientation and material of artefact s. Feature is a portion of the form and is also in recursive definition. A feature can consist of many subfeatures, which are in a hierarchical structure. For example, a screw hole consists of a countersink feature and a hole feature, and the hole feature consists of a cylindrical feature (hole) and a thread feature. The class can also represent composite features, composed features from different artefact s.

The Material class represents the information of material properties relating to the realization of product shape.

The SurfaceCondition class specifies the hardness, roughness, and texture requirements on a feature of a form and referenceDatum.

The Behaviour class (see Figure B.6) captures the motion of a product. The motion is calculated or simulated in the method generateBehaviourModel() in the class. Behaviour is physical actions of objects under specified environments or conditions. Environment or condition means input efforts and flows of artefact , for example, acted external forces. Physical actions mean output efforts and flows of artefact , for instance, the motion or deformation states of artefact acted by external forces. Behaviour can be described with its attributes, behaviour variables of input and output.

Behavioural specification (design) is the mapping process from product functions to product behaviours. Behaviour can be mapped to form based on physical principles and prior knowledge. The BehaviourSpecification class has the following attributes: physicalPrincipleDescription, inputflowVariable, inputEffortVariable, outputFlowVariable, and OutputEffortVariable. GeneratePhysicalAction() is a method of the Behaviour class to generate physical action. The InputFlow class has four attributes: inputFlowName, inputFlowType, inputFlowDirection, and inputFlowPosition. The InputEffort class has four attributes: inputEffortName, inputEffortType, inputEffortDirection, and inputEffortPosition. The OutputFlow class has four attributes: OutputFlowName, OutputFlowType, OutputFlowDirection, and OutputFlowPosition. The OutputEffort class has four attributes: Output-EffortName, OutputEffortType, OutputEffortDirection, and OutputEffortPosition.

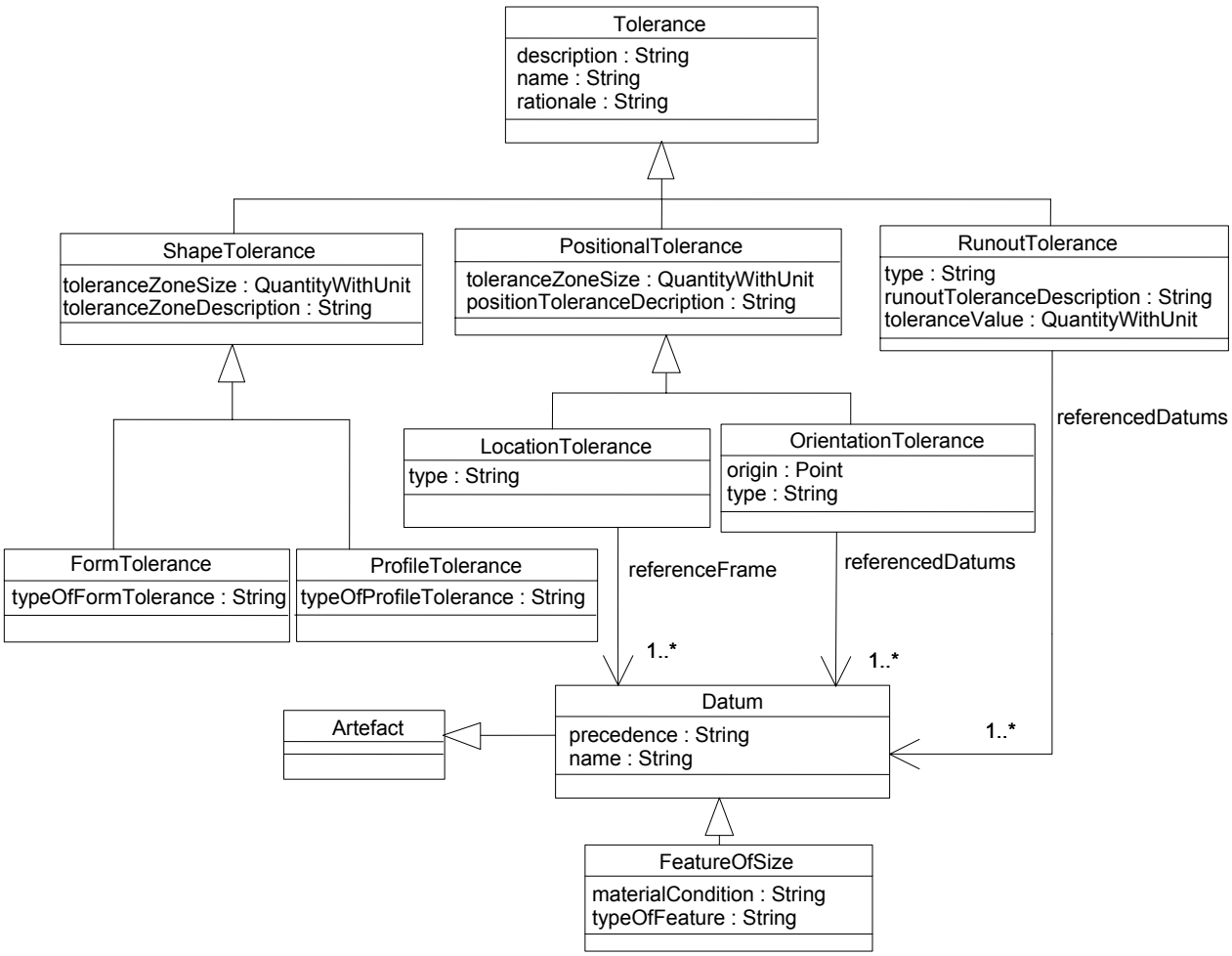

Figure B.4 — Class diagram for Tolerance

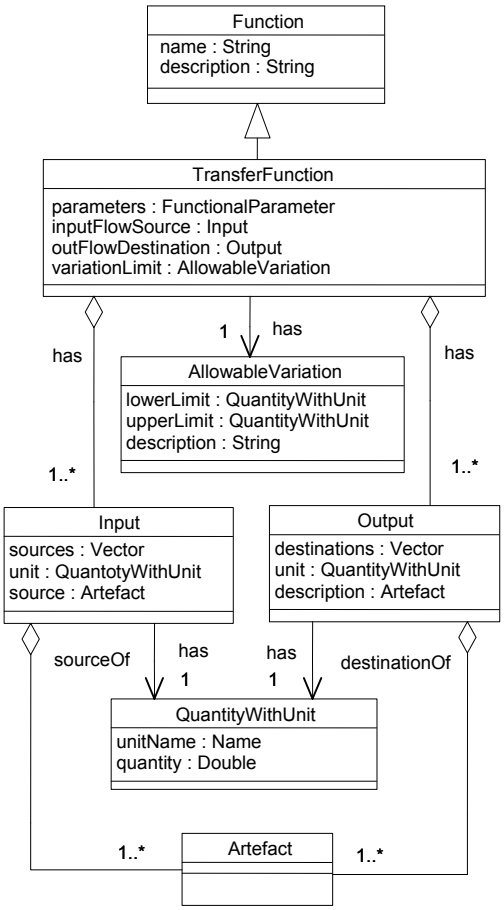

Figure B.5 — Class diagram for Function

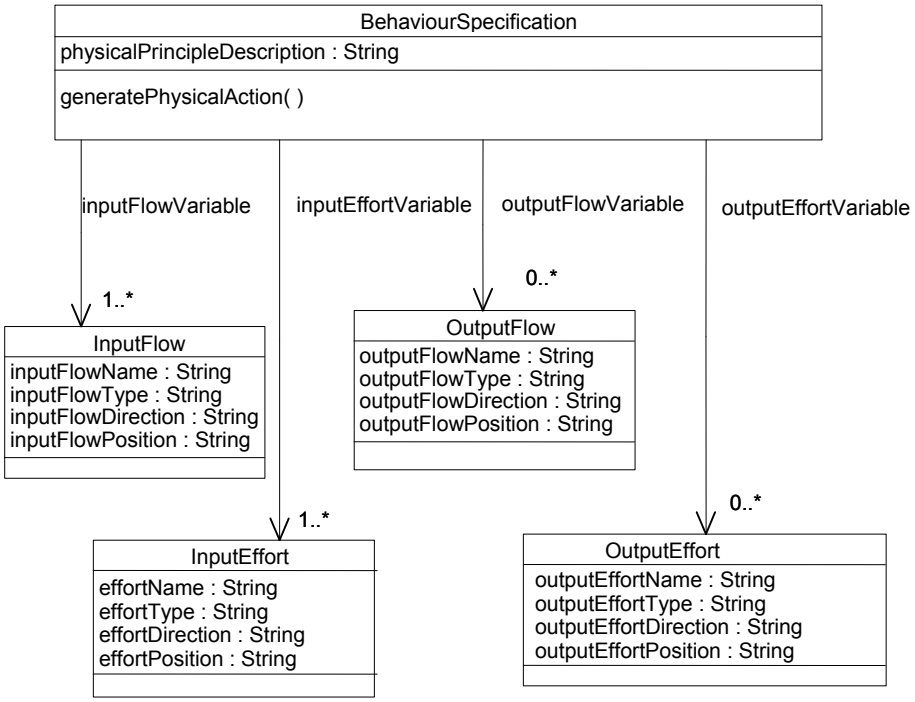

Figure B.6 — Class diagram for Behaviour

The artefact joining, also referred as assembly, model describes how two artefacts are assembled, as shown in Figure B.7. ArtefactJoint describes the joining relationship of two artefacts. The class has recursive definition. This generates an assembly hierarchy, the lower pair in the hierarchy the earlier they should be joined with each other. Thus, the hierarchy indicates the assembly sequence. Each pair of joined artefacts is connected with each other in some portions, features, of the artefacts. FeatureJoint captures assembly relationship of two features and type of joint, rigid or kinematics. This class is also in a recursive structure to describe the joining relationships of features of a feature. Like in ArtefactJoint, the feature assembly sequence is captured in the hierarchical structure.

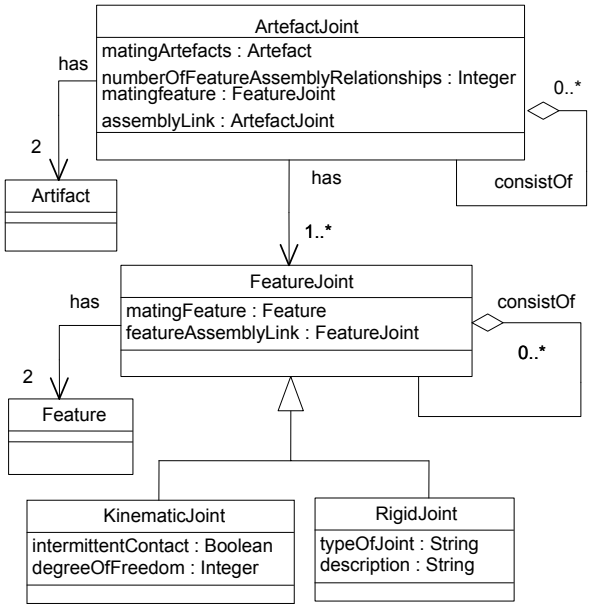

Figure B.7 — Class diagram for Joining

B.2 Capability collaboration diagram

Conceptual manufacturing process planning is an activity to select preliminary manufacturing processes, initially assess the manufacturability, and roughly estimate the cost of a conceptual design. Conceptual process planning information is used in the detailed process planning activity, such as operation sequences, process parameters, and setup specification.

In this example, an information model of the conceptual design is constructed to facilitate a conceptual process planning. In order to set the context in which the information is used, an activity model of a conceptual process planning is developed. This model decomposes the main idea into more detailed levels.

Conceptual Process Planning (CPP) is an activity of selecting preliminary manufacturing processes, assessing the manufacturability and estimating the cost of conceptual design in the early product design stage. This activity aims at determining manufacturing processes, selecting resources and equipment, and estimating manufacturing costs and time. The purpose of conceptual process planning is to support product design in optimizing product form, configuration, and material selection to minimize the manufacturing cost.

One of closely related activities to conceptual process planning is detailed process planning. In contrast to CPP, detailed process planning is an activity based on a detailed design and the results from conceptual process planning to specify operations, determine operation sequences, select machines and tools to be used, depict setups, define process parameters, and estimate process time and manufacturing cost. Based on the definition of CPP, an activity model using IDEF0 methodology is developed to describe the functions and data flow in the CPP activity in detail.

A Conceptual Process Planning activity (A2; activity A1 is Conceptual Design) is decomposed into three subactivities, A21 to A23, which are shown with the data flow in Figure B.8. Activity A21 is to determine manufacturing processes. Depending on conceptual product information, such as material, form, structure, and tolerances, primary manufacturing processes, such as casting, forging, molding, and machining, are select. This activity also includes the subsequence of processes to complete the manufacture of the product.

Activity A22 is to select manufacturing resources. Based on the selected manufacturing processes, appropriate manufacturing resources including both physical resources and human resources are chosen. Resources include machines, tools, and labor skills. Activity A23 is to estimate manufacturing cost. Manufacturing cost covers material, purchased parts, labor, tooling, capital, machine usage, and overhead.

Activity A22 covers a series of resource selection functions and can be further decomposed into three subactivities shown in Figure B.9. Activity A221 is to select machines. Based on the selected manufacturing processes, machines available in factories for manufacturing the designed product are selected. Machines include machines tools, forging machines, casting machines, material handling and assembly machines, and measuring machines. Activity A222 is to select tools and fixtures. Based on the selected machines, tools and fixtures are selected that are necessary for supporting the selected manufacturing processes. Activity A223 is to select labor skills. Based on the selected machines and tools, labor skills are selected to operate the machines and to use tools for production.

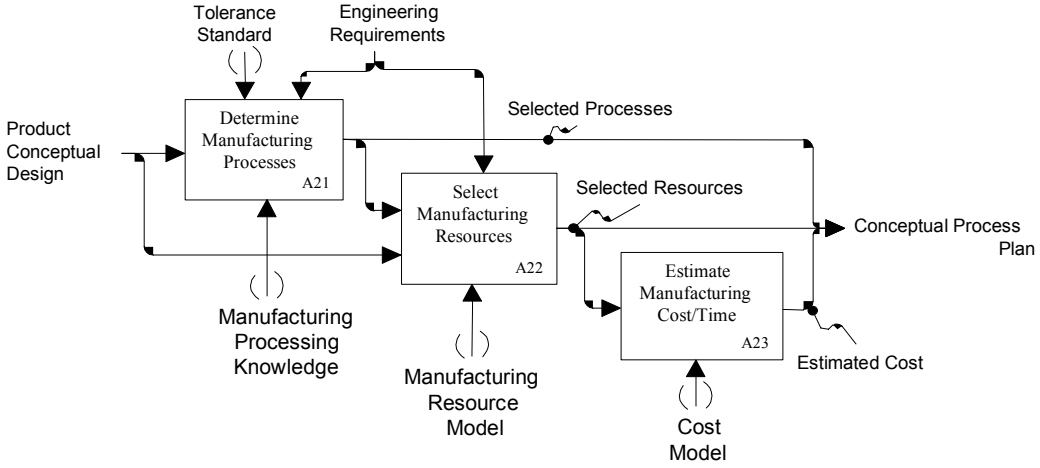

Figure B.8 — Functional decomposition of Conceptual Process Planning

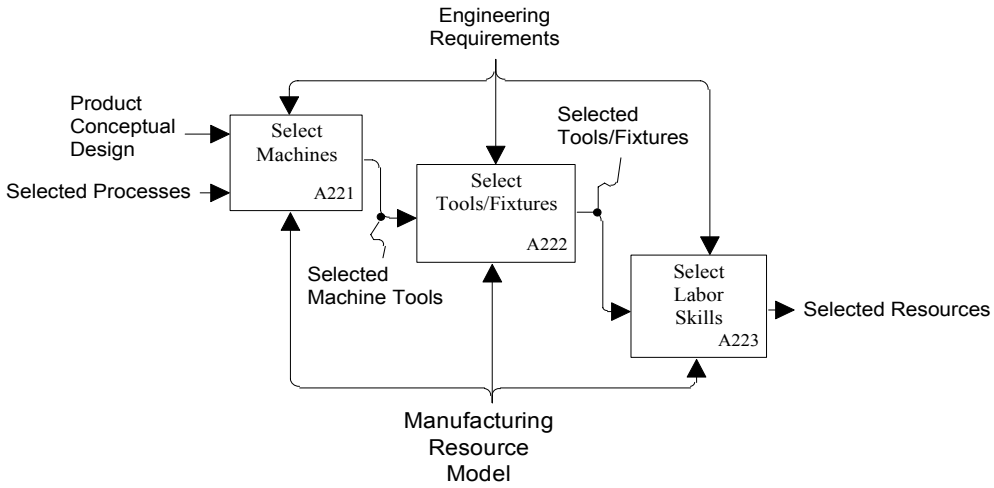

Figure B.9 — Manufacturing Resource Selection

The object model contains information on the data flow in Conceptual Process Planning, described in the activity model above. The object model of processes, resources, and cost structure for a conceptual design is described below. The model is represented using the class diagrams of UML.

Figure B.10 shows classes related to manufacturing processes. ArtefactToBeMade has a sequence of ManufacturingProcess objects. ManufacturingProcess represents a generic process. The manufacturing activity class has a recursive definition, which generates a hierarchical structure. The levels in the hierarchy can be routing, workstation, operation, step, and feature. The type of manufacturing activity indicates the level. Additionally, manufacturing activity has the following attributes: manufacturing parameters, manufacturing resource, estimated cost/time, alternatives, branch, and joint. Branch and joint are used together to form the

structure of concurrent and parallel activities. The first level specialization of manufacturing activity includes Setup, Handling, Processing, LoadUnload, and Idling. Setup represents the activity of machine setup, tool setup, or workpiece setup. Handling captures the information on material handling including transferring materials or tools from one place to another. LoadUnload represents the activity of loading and unloading workpieces or tools onto a machine. Idling represents the idling time between two activities. Processing represents a manufacturing activity, which includes Inspection, PartMaking, and Assembly.

PartMaking represents a part fabrication activity, which can be further specialized. The derived classes from PartMaking are SurfaceTreatment and Shaping. SurfaceTreatment is an activity of treating surfaces of a workpiece for meeting tolerance, surface roughness, and surface hardness specifications. Three derived classes from SurfaceTreatment are Finishing, Coating, and Hardening. Shaping represents an activity of transforming a workpiece into a certain shape. This class has two derived classes: MaterialRemoving and Forming. Forming is activity of molding, casting, or forging material into a certain shape. MaterialRemoving is a shaping activity that removes material from a workpiece to get the anticipated shape. It can be ChemicalRemoving, ThermalRemoving, or MechanicalRemvoing. ChemicalRemoving can be photochemical milling, electrochemical machining (ECM), or chemical milling. *ThermalRemoving* can be high energy beam machining, electrical discharge machining (EDM), or torch cutting.

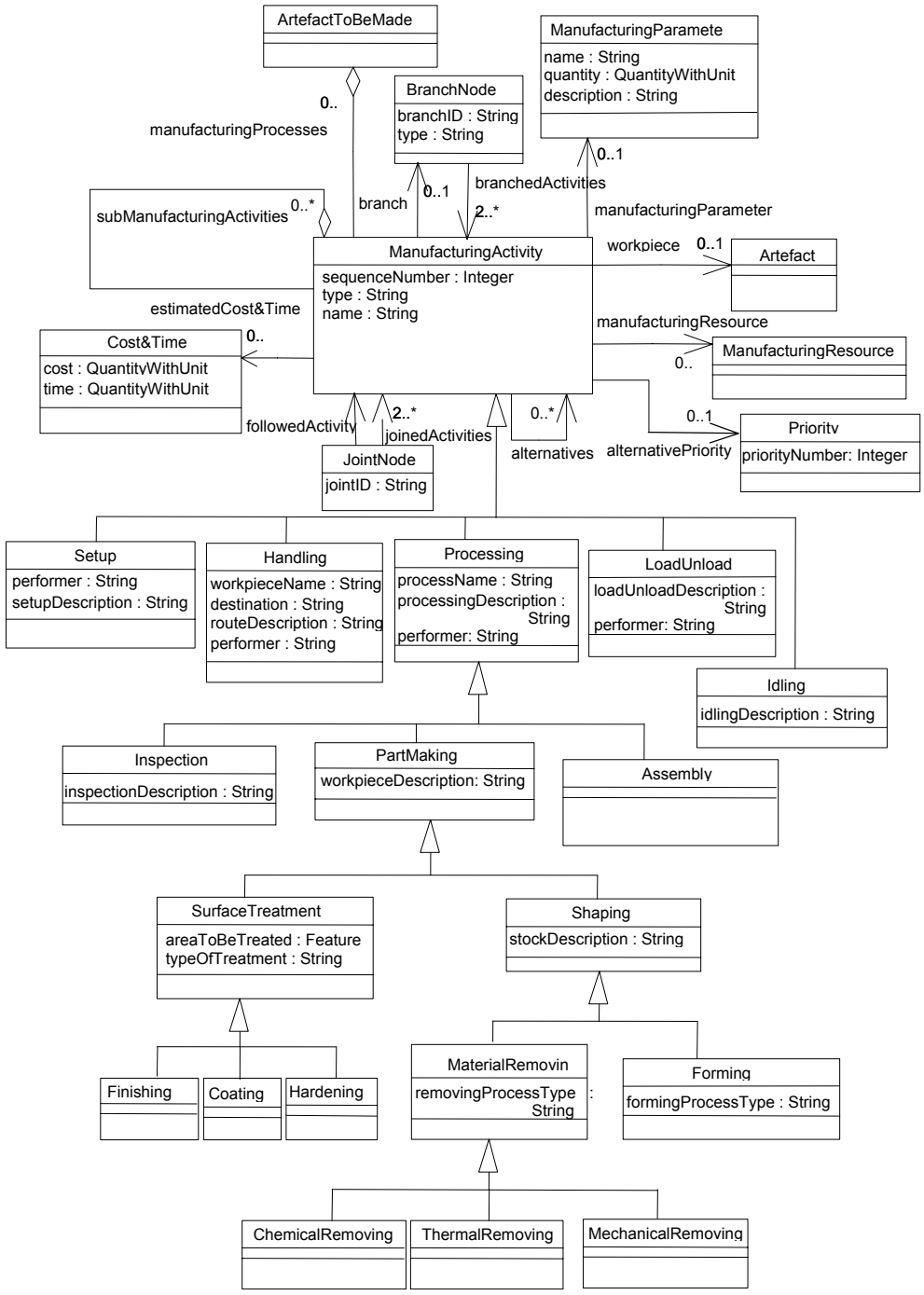

Figure B.10 — Class diagram for Manufacturing Process

Figure B.11 shows the mechanical removing related classes. MechanicalRemoving is an activity of removing material from workpiece using mechanical methods, such as machining and shearing. There are two derived classes: SheetSeperating and Machining. SheetSeperating uses shearing methods to cut metal sheets. Specific methods are piercing, punching, and shearing. Machining is a metal cutting activity using a machine tool and has three derived classes: SinglePointCutting, MultiplePointCutting, and Abrasive. SinglePointCutting includes boring, turning, grooving, and single-point threading. MultiplePointCutting includes drilling, reaming, milling, multiple-point threading, broaching, gear cutting, and sawing. Abrasive includes grinding, honing, ultrasonic machining, super finishing, and lapping.

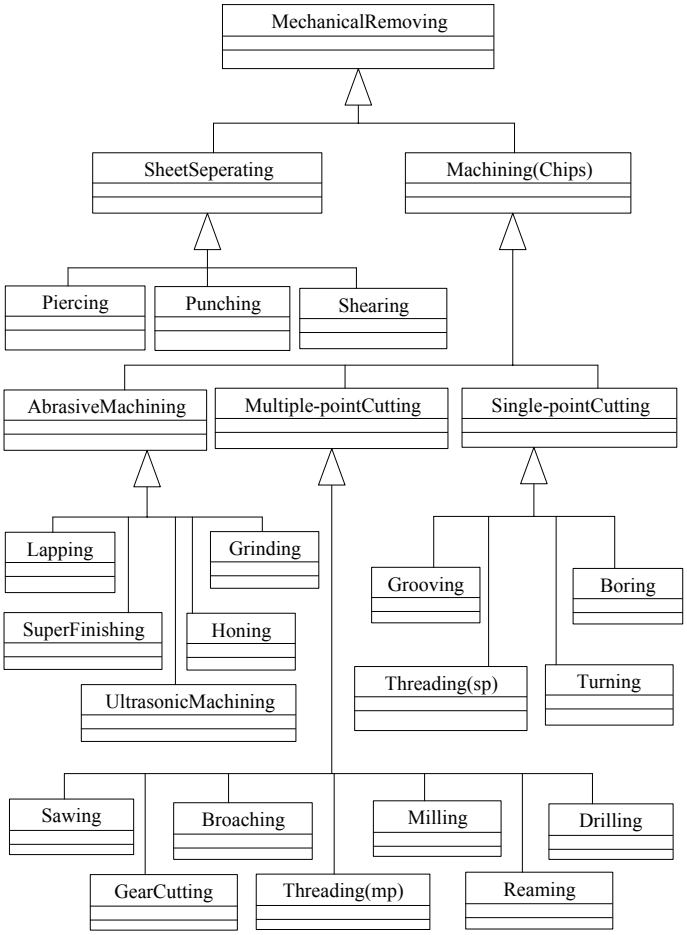

Figure B.11 — Object model of Mechanical Removing

Figure B.12 shows the forming-related classes. Forming has two derived classes: Deformation and Consolidation. Deformation is a class that represents the activity of changing the shape of a workpiece. Specific processes are forging, extrusion, drawing, rolling, knurling, and hobbing. Consolidation is a class that represents the activity of shape forming with mold or die. Specific processes are casting, polymer molding, ceramic molding, compacting, and laminating.

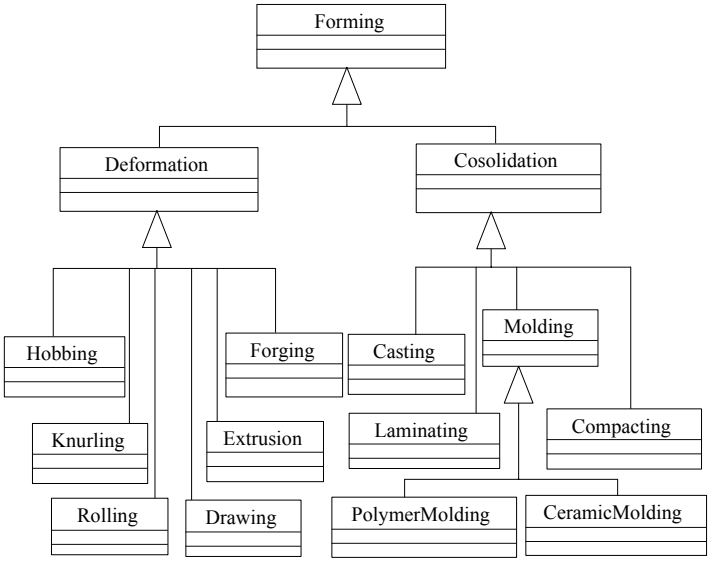

Figure B.12 — Object model of Forming

AssemblyProcess, as in Figure B.13, is a class represents the activity of joining components into a product. The derived classes are MechanicalJoining, ThermalJoining, and ChemicalJoining. ChemicalJoining is a class that represents the activity of joining components using chemical methods, such as adhesive bonding. ThermalJoining is a class that represents the activity of joining components using thermal bonding methods, such as brazing, thermal welding, and soldering. MechanicalJoining is a class that represents the activity of joining components using mechanical methods, such as fit, attachment, and mechanical welding. Fit can be either force fit or loose fit of two components in a product. Attachment can be riveting or using keys, pins, or screws. Mechanical welding includes friction welding, explosive welding, ultrasonic welding, and pressure welding.

ManufacturingResource, as in Figure B.14, is a class that represents a physical object or a labor skill that is used in a manufacturing process. ManufacturingEquipment is a class that represents a piece of equipment (a physical entity) that is used in manufacturing processing. Examples are machine, tool, fixture, and gauge. A piece of equipment has a set of parameters that describe the piece of equipment. EquipmentParameter is a class that represents parameters. The derived classes from ManufacturingEquipment are Machine, ToolForMachining, Mold, and Die. Machine can be a machining center, forging machine, EDM machining, etc. ToolForMachining is a tool used in machining process, such as a cutter, extender, holder, or gauge.

Conceptual Process Planning is a manufacturability analysis activity, which includes selecting manufacturing processes based on conceptual design, selecting manufacturing resources, and estimating manufacturing cost and time. Incorporating manufacturing analysis into design can ensure that the design is manufacturable and within cost limits. Manufacturability analysis and cost estimation are important to minimize production cost. However, there currently lack software tools in conceptual process planning. Manufacturers need new software tools that will effectively support translating conceptual design into manufacturing process and resource selection and, then, to estimate manufacturing time and cost. To develop these new tools, information models are necessary to support the tool development and the integration of the tools.

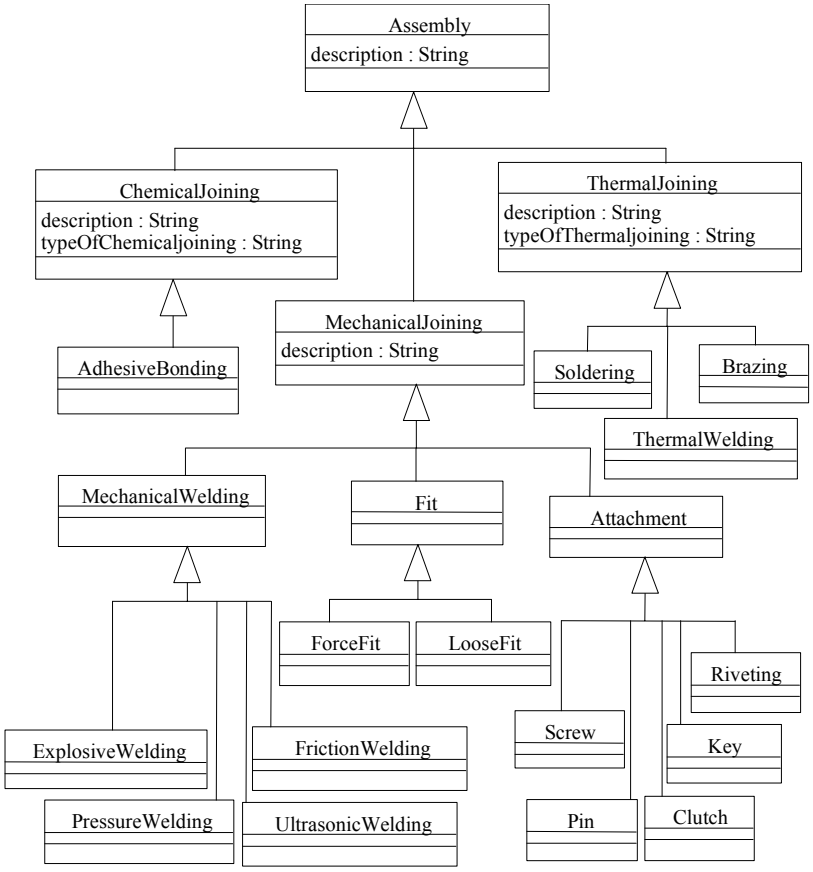

Figure B.13 — Object model of Assembly Process

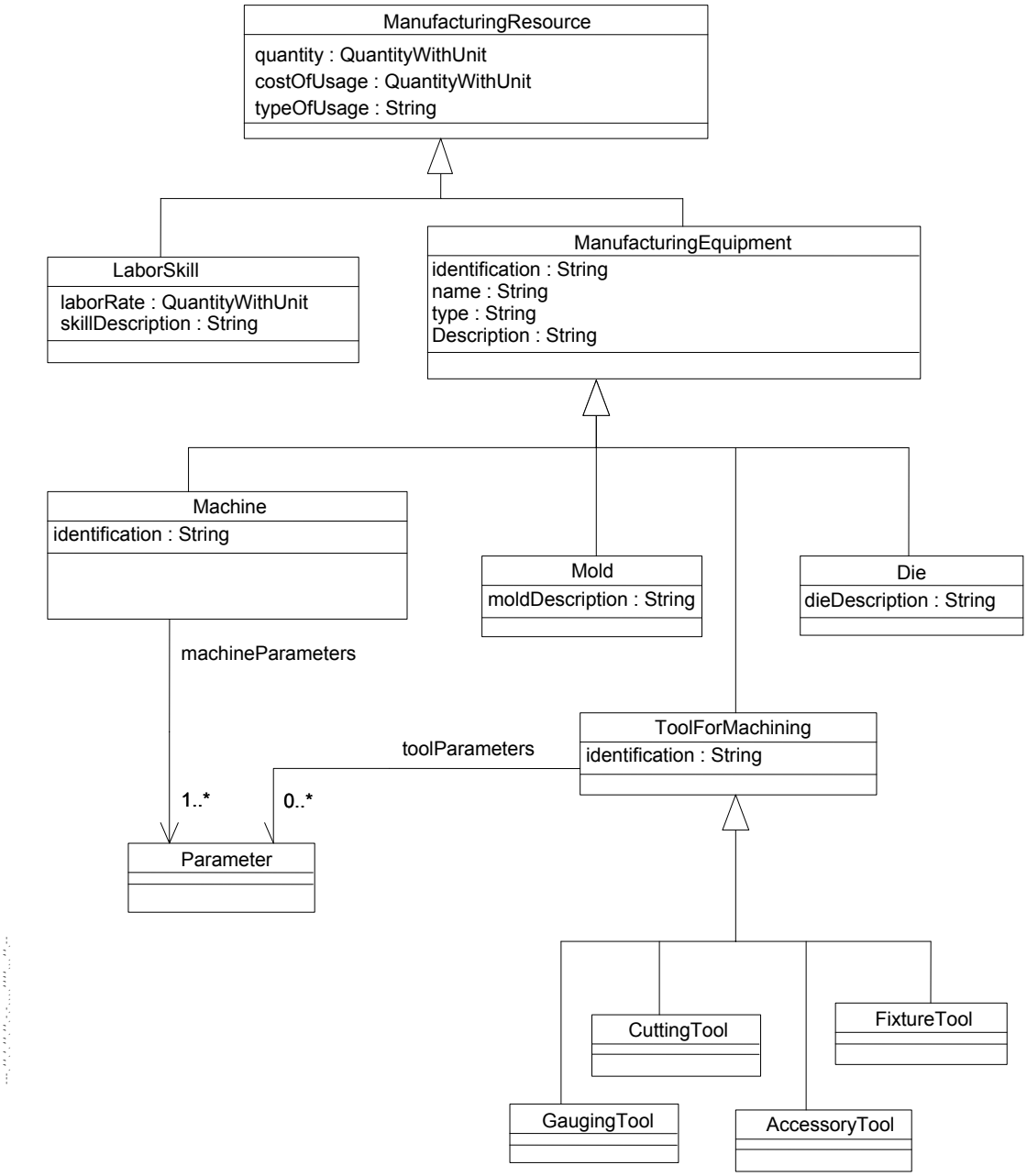

Figure B.14 — Object model of Manufacturing Resource

Annex C
(informative)

Examples of a software unit information model

C.1 Software unit for Data Analysis and Visualization (DAV)

The software unit for Data Analysis and Visualization (DAV) defines architecture as well as basic functions which should enable a flexible assembly of evaluation applications out of different functional components. The main technical objective is to be able to combine the software components of several different manufacturers to a specific manufacturing software unit (application) without having to know how a particular component is implemented or even modifying the component's implementation.

Furthermore, the DAV architecture should provide a possibility to customize such a software unit in a consistent way, e.g. providing a harmonious feel — and, perhaps, even a look — for all participating components. Reusable modules are built for customized, project-specific manufacturing solutions. They deal with the defined task exclusively without overtaxing the user with unnecessary functionality. Regarding the reusability of components, it is not necessary to start from scratch every time. Moreover, the manufacturer is not compelled to cover all aspects of an evaluation. Common developments of software components will provide practical solutions. The main features of model and thinking behind the DAV architecture are outlined below in a compact manner to provide a common understanding of what constitutes a DAV software unit and how it is used. The overall DAV architecture is illustrated in Figure C.1.

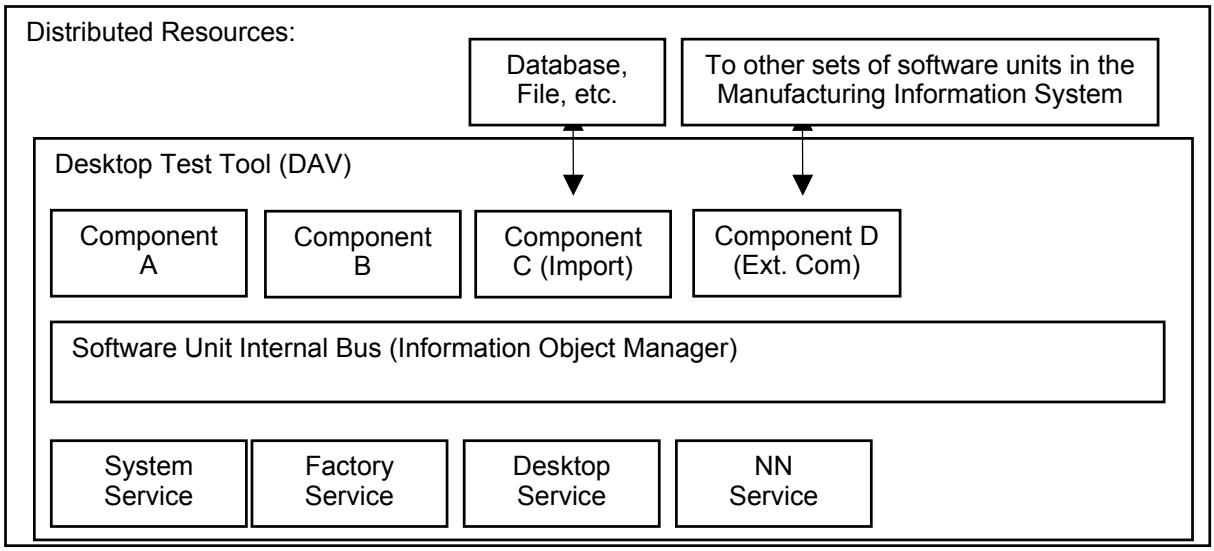

Figure C.1 — Structure of a component-based DAV software unit

This software unit description has its focus upon applications for data (e.g. statistical process data) observation, analyzing and visualizing typically performed on a single desktop computer (of course with access to other databases, files, information systems, etc. that might be located elsewhere). The details and implementations of such a software unit are not specified here. Instead, to support such applications in an interchangeable and non-proprietary way, a complete communication infrastructure and suitable component interfaces are specified to allow the development of standard components.

The main advantage for end users is better independence from single software-vendors as well as closing the gap between the competing "standards vs. individual programming" by offering the best from both points of view. To realize these ideas a "communication bus" is defined that allows building a "backbone" of an individual software unit decoupling the single functional software components from each other. Some amount

of standardization is necessary to let individually and independently developed components work together via such a bus. The bus itself at first is a purely local — again oriented to a desktop computer — concept, since a typical application is a conglomerate of local components tied together to build a meaningful information flow for data evaluation. Only single components might comprise some remote communication with resources outside the software unit itself, mainly to access the distributed IT-environment of the Manufacturing Information System.

C.2 Services — Offering common functions

Aside from the bus the system service acts as the “backbone” of a single software unit. It offers functionality fundamental to components. More general, the concept of such “services” to the components of a software unit is used to offer specific functionality of general interest to a software unit and its building blocks. Figure C.1 shows some of these services. Additional services or extensions to existing service functionality might be defined in future editions of this part of ISO 16100.

For now, a Help Service gives standardized access to help information with regard to a component. The Desktop Service in Figure C.1 offers access to windows and a common graphics context as an abstraction to concrete desktop GUI environments such as Microsoft Windows, UNIX X-Windows or Java Swing. A Logging and Tracing Service offers simple operations to let a component write different kinds of log messages monitoring its execution. This helps in debugging and assuring repeatable calculation results. A Scripting Service provides a simple interface to write out single configuration and execution steps of a software unit to build up a complete script that allows exact repetition or later recalculation of the same application (that is, the same conglomerate of components with the same connections to each other and the same parameters). A Property Service gives a generic abstraction to a component’s properties (also called attributes or features). It also allows building sets of properties common to more than one component. The Factory Service in Figure C.1 allows a component to create new, empty data objects that can be used for inter-component communication.

C.3 Items — The communicated objects

Information (data) as well as services is communicated between components by a single homogeneous concept called "items". In fact, the bus is basically an “item manager”, that is, it manages the announcement of new data available in the form of an item or its destruction if it is no longer valid or needed. Items can show up in different "flavors", that is, different kinds (or “types”) representing different underlying data, each with a specific meaning to an application scenario. A component that offers data (for example being a facade to a database or file access to retrieve measurement data) to a software unit does so by creating an item of a suitable type, “fills” it with the real data values and announces (“publishes”) it to the bus. In turn, the bus informs all components registered to it about the availability of a new item. Components can retrieve information about the item (its name, its flavor, etc.) to decide if it matches what they need as an input to their own functionality, e.g. do some calculations on the data.

Therefore, during a component’s configuration, the user associates the needed items (typically by name) with a component’s input. Indeed, by doing so, a software unit is build up (“programmed”), since this association of items to components really “connects” the functional building blocks. To transfer items between components without corrupting the idea of loose coupling between components, the concept of "events" is used. An event — itself being a programmatic object from an implementer’s point of view — “carries” an item and signals its availability, change or deletion. As a net result, “connections” of components to each other and a bus are really made of the registration of event sources and sinks. To assure flexibility and a high degree of decoupling most events are exchanged via the bus and not exchanged directly between components. Different kinds of events, e.g. a change or revocation of data, are distinguished by respective labels carried by each event. Aside from bus, service, item and event concepts the last main pillar on which DAV’s architecture is based is the notion of a "component".

C.4 Software components — The functional modules of a software unit

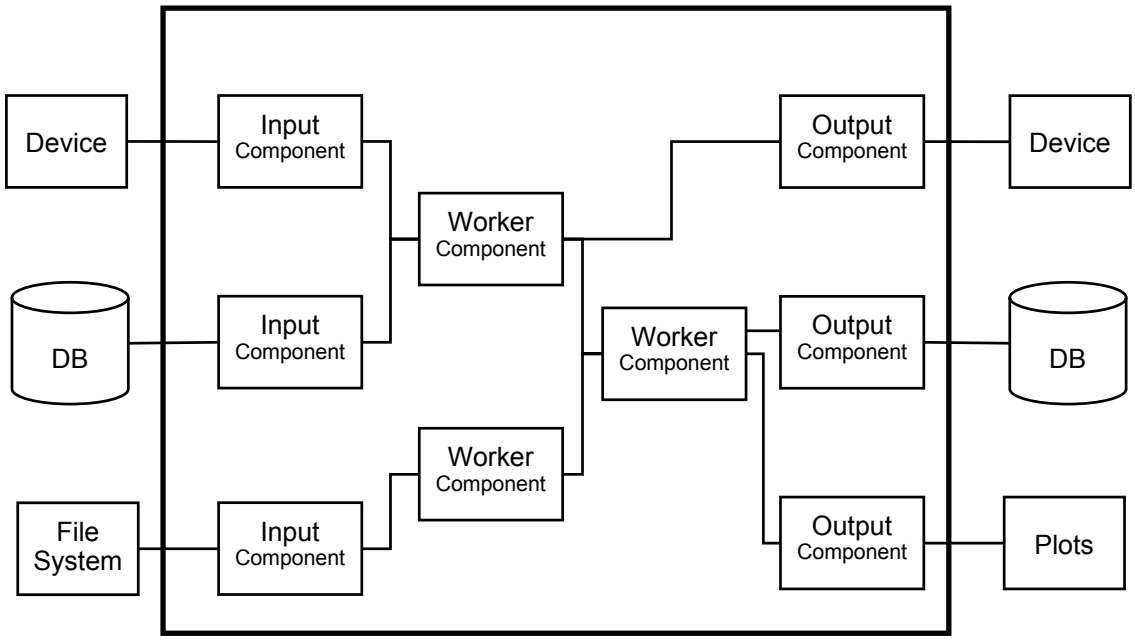

Figure C.2 — Logical view to a DAV software unit's structure and information flow

The term "software unit" denotes an aggregate ("composition") of functional modules (software components) each offering a specific and complete, self-contained functionality rich enough to justify its encapsulation into a single, individual building block and small enough to be usefully composable with additional functionality (in form of other such building blocks) to build complete computation chains. Obeying this rule of thumb, an independent software vendor can offer a single unit of computation that comprises enough functionality to distribute is as a product on its own. Then, each building block is called a component - and if its programmatic interfaces obey the definitions of this part of ISO 16100, it is guaranteed to work with other components (possibly distributed by other vendors) also adhering to this part of ISO 16100.

The reasons for this component based architecture are as follows.

a) Components can be produced by many different suppliers.

1) There is no dependency on one specific component provider.
2) A multitude of components for a general or specific type of problem can be produced.
3) Users may choose from larger sets of varying components.
4) Projects easily may integrate third party components, thus improving development time (and maybe cost).
5) The production of components is less complex than producing whole applications, thus even small, highly-specialized companies may contribute to the component market.
6) The component developer does not necessarily have to know the final application(s). The developer concentrates on the efficiency, performance, and output of a partial problem in which he is the expert.

b) Components are reusable.

1) Components can be created or bought separately and then be used together or in varying combinations with other components in different applications or different application contexts.
2) Single concepts can be implemented isolated from existing ones.
3) To be reusable in as many situations as possible components usually come with their own "customizers", typically configuration dialogs that allow the test engineer to set characteristics and attributes of a component within ranges predefined by the component developer. Thus,

componentized applications typically are more error prone and predictable than arbitrarily modifiable software. Configuration dialogs may range from simple multiple choice checkboxes to complete “wizards” that guide and assist the engineer through complex configuration.

c) The composition of different components allows the quick development of new, highly flexible applications that reduce maintenance and administration cost.

   1) Lessons learned from building hardware systems, e.g. applying integrated circuits, are introduced to software development - which is still more of an art than an engineering discipline.

   2) A component can be replaced by another component (if necessary from another provider) as well as new components may be added without touching or changing other parts of an application.

   3) Existing systems and functionality can be wrapped into a component model and more easily integrated into new developments.

To aid the application assembly from various components, they have to be delivered as self-contained packages not only with regard to their technical functionality, but also including help, version, national language, and certification level information.

To make these promises of the component paradigm become reality, a standardization of an underlying and possibly application domain specific component model - which is mainly about interfaces, communication and, perhaps, supporting services - is necessary.

C.5 Setting up a software unit

While C.4 looked at a software unit from a user’s overall point of view, the technical level of operations called during the set-up and execution of a software unit is detailed below. Sequence diagrams are used to give the reader an idea of the necessary steps. Each diagram shows respective operation calls. Since some details, particularly the loading of a component, intentionally are not specified, italic font or annotation is used for “pseudo operations” like instantiating an object.

Of course, there is some level of variation for some of the illustrated interaction, e.g. a test tool might do things in parallel such as loading component descriptions and instantiating services, or a component might wait for itself being configured by the user before it looks for items. Figure C.3 depicts a quite typical scenario for the initial setup of a software unit and the loading of a first software component (of course, a real-world scenario would involve firstly the loading of a more specific component, probably a producer).

The basic pattern of a software component’s lifecycle comprises the steps

- load
- initialize
- configure
- execute
- shutdown

probably looping between execution and configuration for a while.

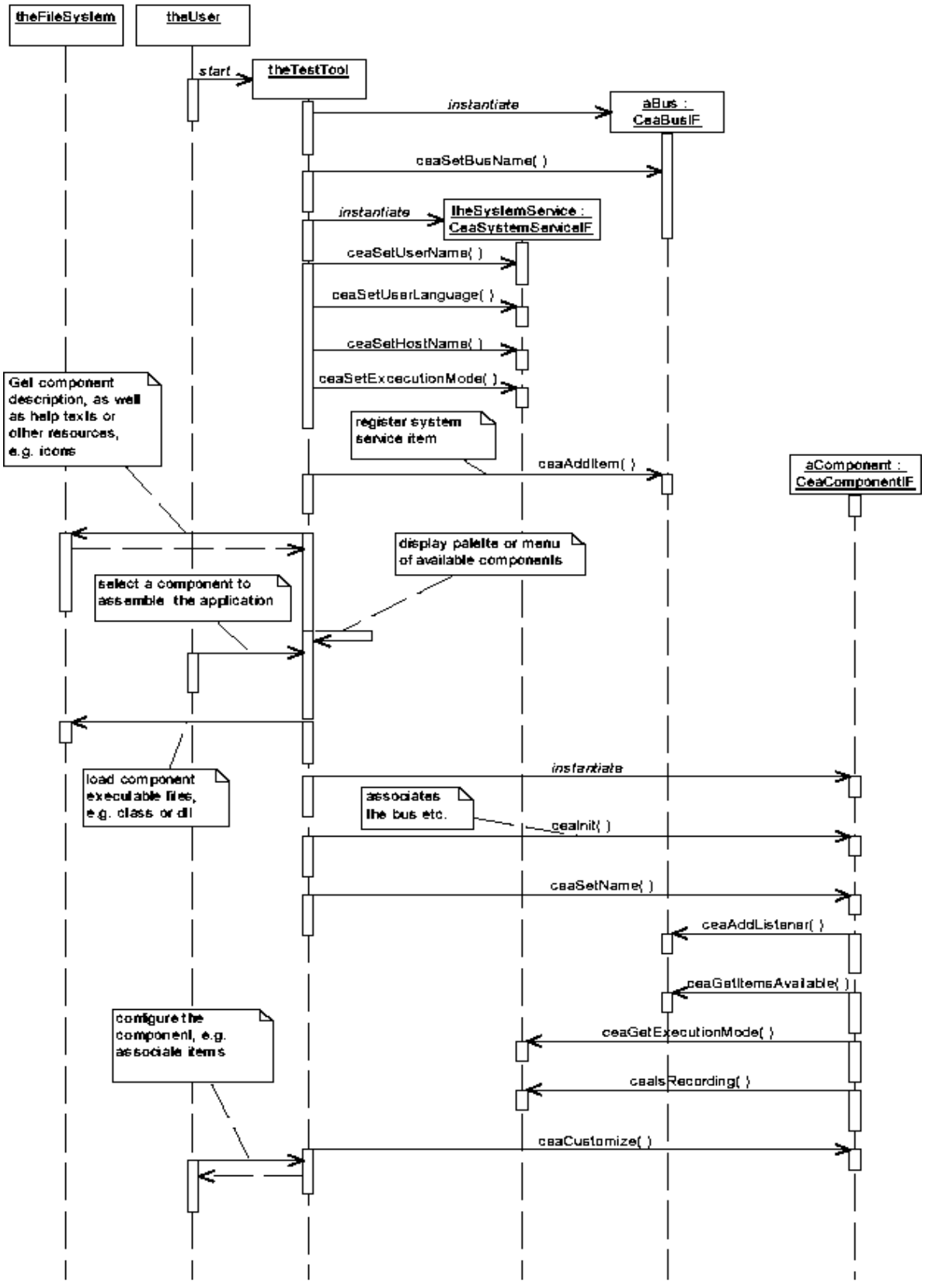

Figure C.3 — DAV software unit's lifecycle: adding a first component

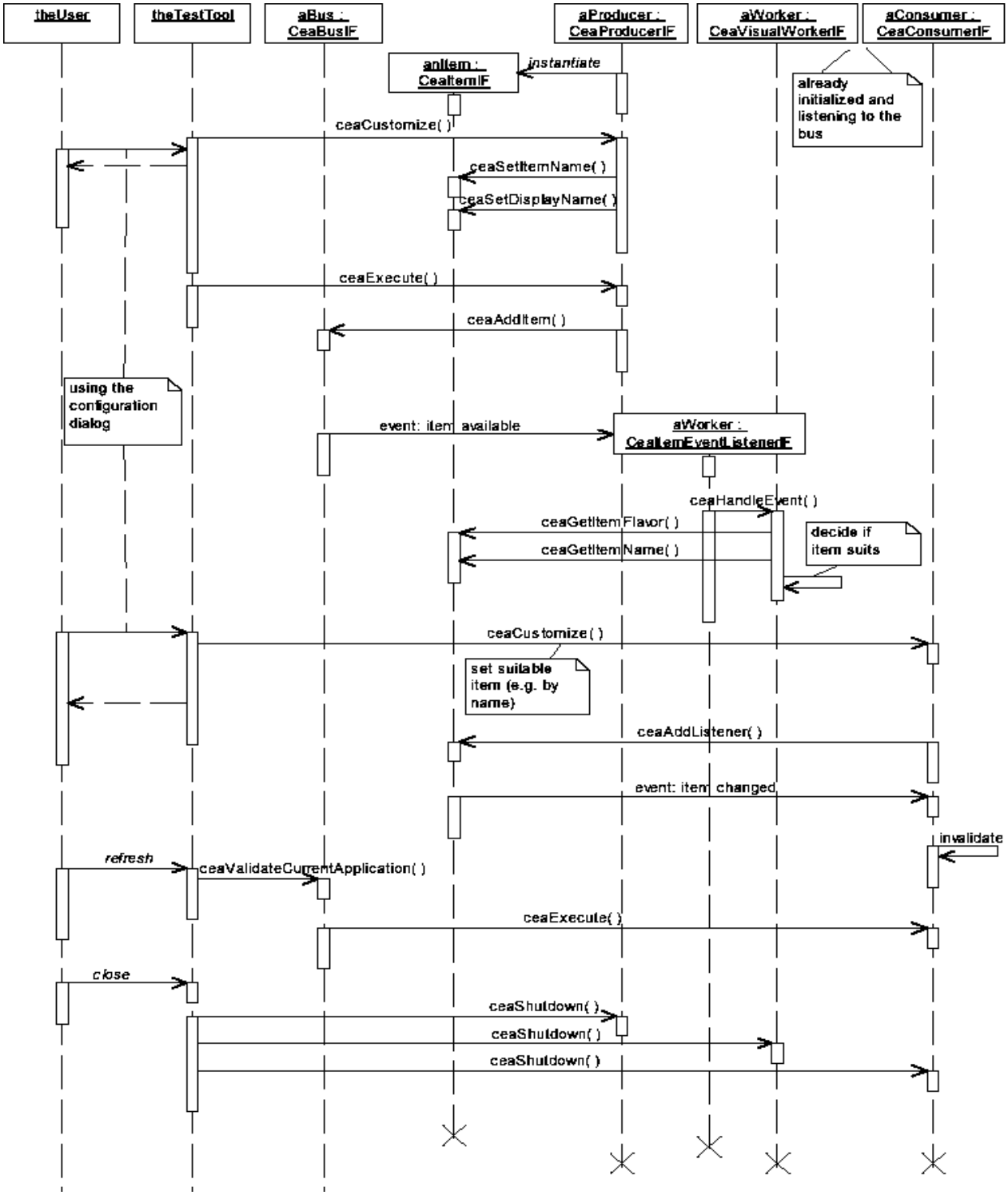

Figure C.4 — DAV software unit's lifecycle: typical interaction while running

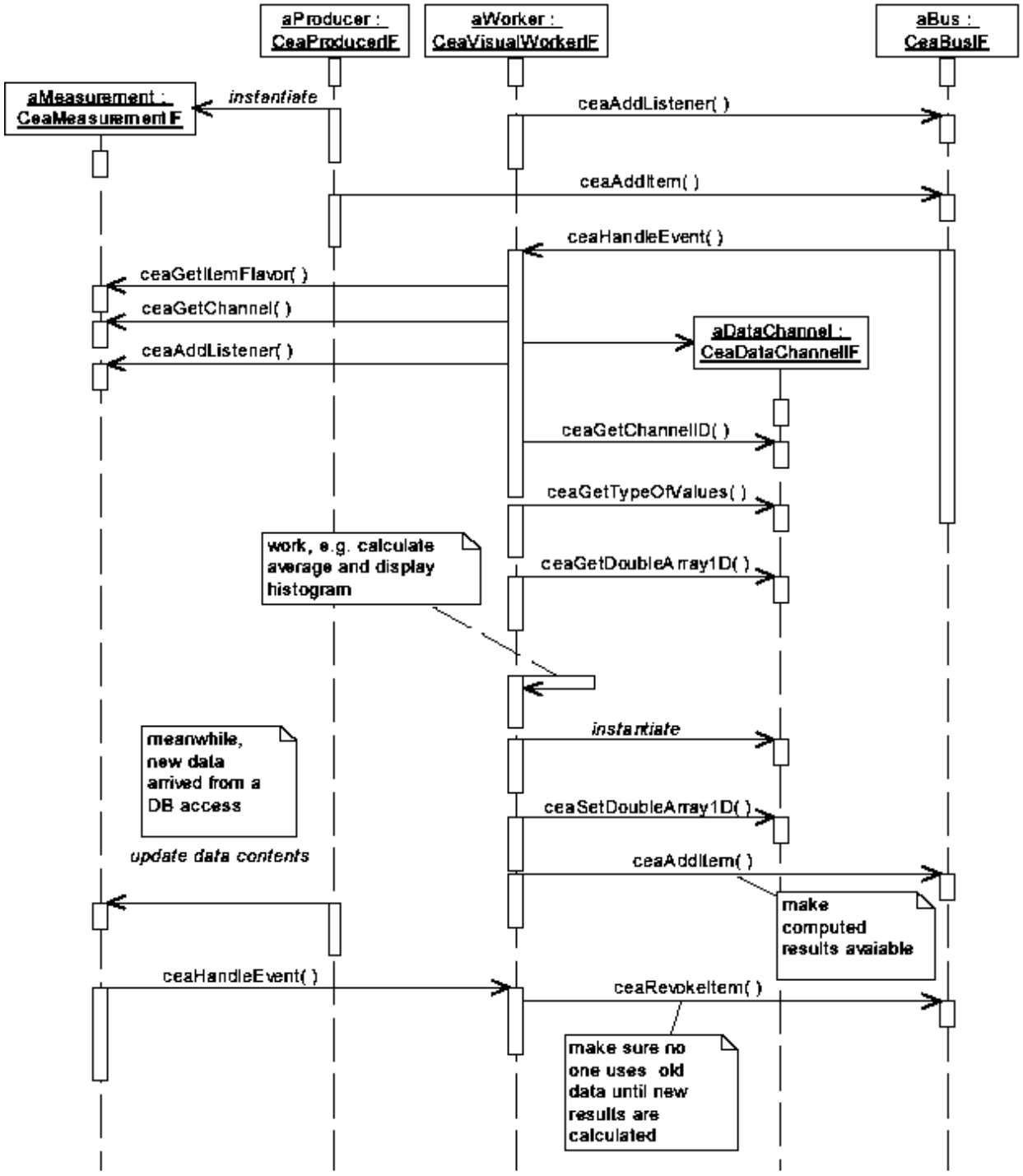

Figure C.5 — DAV software unit's lifecycle: Typical interactions related to items

C.6 Example of communicated objects

Figure C.6 shows an example of the communicated objects describing a class diagram of a Manufacturing Execution System.

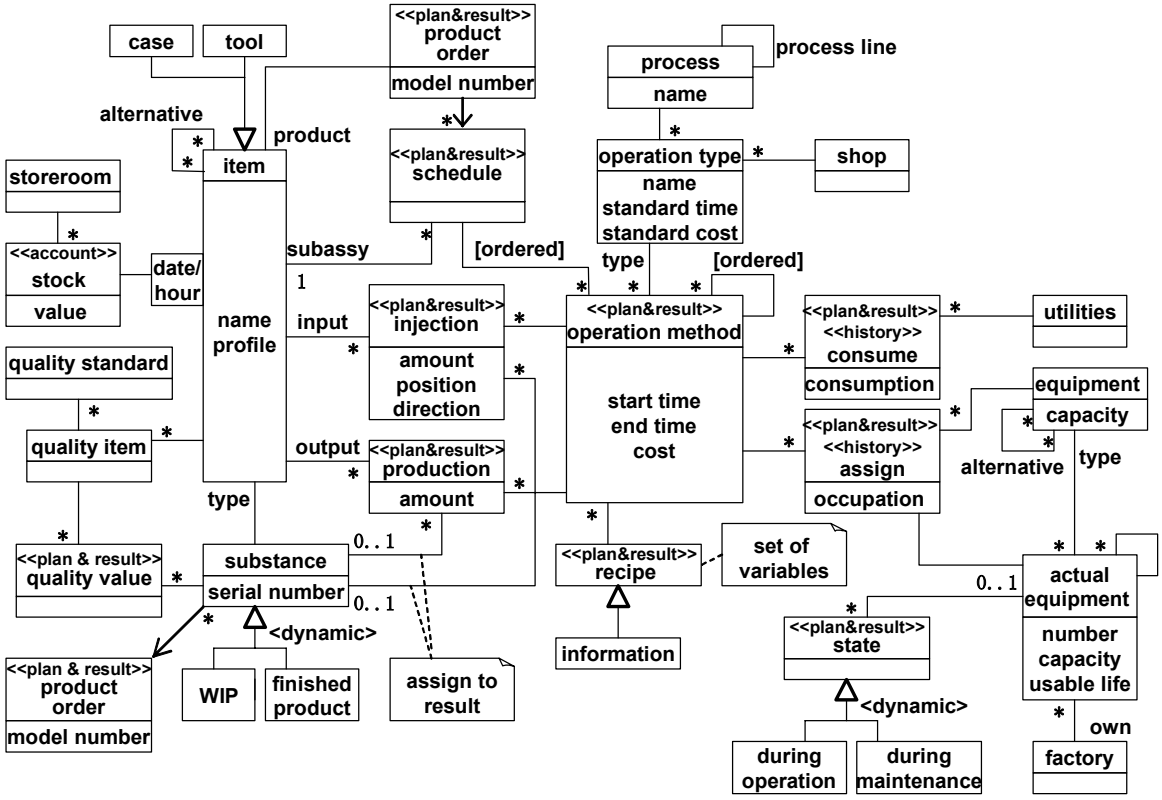

Figure C.6 — Example of a class diagram of a Manufacturing Execution System

In this example, a software unit judging the quality of a substance from the measured quality value receives the following communicated objects (see left hand side of Figure C.6):

a) the serial number attribute of the substance;

b) the quality value of the substance measured by equipment;

c) quality items from the quality standard database.

The software unit compares these received values and outputs the judgments of the quality of the substance as communicated objects to a software unit such as a MES at the upper layers of the hierarchy shown in Figure B.1.

ICS 25.040.01

Price based on 58 pages