INTERNATIONAL STANDARD

ISO 13281-2

First edition
2000-03-15

Industrial automation systems and integration — Manufacturing Automation Programming Environment (MAPLE) —

Part 2: Services and interfaces

Systèmes d'automatisation industrielle et intégration — Environnement de programmation d'automatisation de fabrication (MAPLE) —

Partie 2: Services et interfaces

Reference number
ISO 13281-2:2000(E)

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

Printed in Switzerland

Contents

Page

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 3.

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 part of ISO 13281 may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

International Standard ISO 13281-2 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 5, *Architecture, communications and integration frameworks*.

ISO 13281 consists of the following parts, under the general title *Industrial automation systems and integration — Manufacturing Automation Programming Environment (MAPLE)*:

— *Part 1: Functional architecture*

— *Part 2: Services and interfaces*

Annex A forms a normative part of this part of ISO 13281.

Introduction

Over the recent past, manufacturing systems have become considerably more flexible and have acquired greater functionality. The numbers and types of component devices of manufacturing systems, such as NC machines, robots, automated guided vehicles, programmable controllers and manufacturing cells have increased. Manufacturing engineers are thus required to develop and update programs not only for many kinds of individual devices but also for combinations of devices. Due to this fact, the difficulty of integrating and programming the control of manufacturing operations has increased.

Manufacturing programs have an intense need for a large variety of manufacturing data, including product oriented data, process oriented data, operation oriented data and management oriented data. This diversity means that manufacturing data has a much more complicated and varied schema than the processing data encountered in other systems, e.g., business systems. Therefore, the use and management of manufacturing databases requires a manufacturing oriented approach. The concept of MAPLE is intended to provide assistance to address this need.

MAPLE assists program developers, planners and operators in a manufacturing automation environment to generate programs and prepare them for their execution.

MAPLE will assist in the following activities:

a) generation of programs to control devices, cells, shop floors and factories, either manually or with computer assisted tools;

b) manufacturing and process planning;

c) checking and preparation of resources;

d) preparation of manufacturing data sets for execution (e.g., post processing).

The outcomes of these activities are:

a) manufacturing data sets (e.g., geometry, tools, technology, sequence of operations, setups, measuring, testing, handling);

b) cell, shop floor and factory monitoring and control programs.

This standard for MAPLE services and interfaces builds upon the functional architecture as specified in ISO 13281-1. The functional architecture provides a manufacturing data dictionary and a manufacturing software dictionary that facilitate the separation of the underlying data sources and I/O requirements, in whatever format, from the executing manufacturing task. Hence, MAPLE provides a mechanism by which a number of diverse data sources and software tools can be integrated seamlessly.

ISO 13281-1 and 13281-2 are intended to guide software developers of MAPLE environments as well as system integrators and software tool developers. The standard for MAPLE services and interfaces relies on ISO/IEC DIS 14750 for the interface description language, and ISO/IEC 10746 for presenting Open Distributed Processing (ODP) view points.

Other relevant work such as aspects of STEP (ISO 10303, Product data representation and exchange) data and the content of data files used in the NC machining environment for example, will be addressed in a potential new work item on the MAPLE data dictionary and software program dictionary.

ISO 13281-1 provides an overview of the MAPLE functional architecture in order to aid in the understanding of how MAPLE services might be provided through a number of functional components within MAPLE, and their internal and external interfaces.

MAPLE is a building block that can be applied at any level within a manufacturing enterprise. Separate MAPLE implementations can be configured and connected within an enterprise as required.

Industrial automation systems and integration — Manufacturing Automation Programming Environment (MAPLE) —

Part 2: Services and interfaces

1 Scope

This part of ISO 13281 specifies a minimum set of services to be provided and interface requirements for creating a MAPLE. The specifications in this part of ISO 13281 are specifically for software developers of MAPLE environments, system integrators, and software tool developers. Specifications that address the needs of users such as program developers, planners and operators in a manufacturing automation environment are outside the scope of this document.

This part of ISO 13281 only specifies the interface at the application layer between MAPLE and software programs.

The creation or deletion of a Manufacturing Database, as well as specifications for the MAPLE Data Dictionary and MAPLE Software Dictionary beyond the dictionary definition schema, are outside the scope of this part of ISO 13281.

2 Conformance

To be conformant with this part of ISO 13281, an implementation shall use the concepts and rules of this part of ISO 13281.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO 13281. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO 13281 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 9579-1:1993, *Information technology — Open Systems Interconnection — Remote Database Access — Part 1: Generic Model, Service and Protocol.*

ISO/IEC 9579-2:1998, *Information technology — Open Systems Interconnection — Remote Database Access — Part 2: SQL Specialization.*

ISO/IEC 9579-3:1996, *Information technology — Open Systems Interconnection — Remote Database Access — Part 3: SQL specialization Protocol Implementation Conformance Statement (PICS) proforma.*

ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.*

ISO/IEC 10746-3:1996, *Information technology — Open Distributed Processing — Reference Model: Architecture.*

ISO 13281:1997, *Industrial automation systems — Manufacturing Automation Programming Environment (MAPLE) — Functional architecture.*

ISO/IEC 14750:1999, *Information technology — Open Distributed Processing — Interface Definition Language.*

4 Terms and definitions

For the purposes of this part of ISO 13281, the following terms and definitions apply. Italicized words in the definitions are terms that appear elsewhere in this clause. Components of the MAPLE architecture appear capitalized.

4.1
action
elemental description of a work request to external *Manufacturing Software Programs* or other *MAPLE*s

4.2
data classification
identifier of a computer-readable representation of data for a specific application

EXAMPLES Bill of materials, CL file format, STEP format.

4.3
data classification catalogue
user-specified or vendor-specified unique collection of *data classifications* used within *MAPLE*

4.4
data source
description of the data schema, data location and access method to the data for a particular data format that is registered in *MAPLE*

4.5
data storage type
particular type of data storage system used

EXAMPLES Directory files, Oracle database, Access database, ONTOS, ODBC data source.

4.6
data storage type catalogue
user-specified or vendor-specified unique collection of *data storage types* of the manufacturing databases connected to *MAPLE*

4.7
Data Translator
manufacturing software program for converting the representation of data

4.8
Dictionary Manager
manufacturing software program to facilitate the manipulation of the *Manufacturing Data Dictionary* and the *Manufacturing Software Dictionary*

4.9
Execution Manager
software that controls the sequence of execution of internal processes of *MAPLE* and the external *Manufacturing Software Programs*

4.10
logical-to-physical mapping
relationship between the *data sources* and the physical data stored in the *Manufacturing Data Dictionary*

4.11
Manufacturing Database
data repository, possibly distributed, containing product, process, facility and management oriented data

4.12
Manufacturing Data Dictionary
collection of storage types of the *Manufacturing Databases* and the description of the *data sources*

4.13
Manufacturing Data Manager
Manufacturing Software enabling access to the *Manufacturing Databases*

4.14
Manufacturing Software Dictionary
collection of descriptions of the capabilities and invocations of *Manufacturing Software Programs*

4.15
Manufacturing Software Programs
software, registered in the *Manufacturing Software Dictionary*, which have command and information connections with *MAPLE* through the *MAPLE* interface

4.16
MAPLE
Manufacturing Automation Programming Environment, a common vendor-independent neutral support facility for the programming of multiple manufacturing devices and controls

4.17
MAPLE Engine
function for receiving and handling requests to *MAPLE*, initializing and shutting down the *MAPLE* environment

4.18
software capability catalogue
user-specified or vendor-specified unique collection of software capabilities within *MAPLE*

4.19
software program capability
classification of the functionalities of software programs connected to *MAPLE*

4.20
Software Tool Linker
manufacturing software to analyse, select and sequence other *Manufacturing Software Programs* to fulfil *MAPLE Engine* requests

4.21
task
elemental description of a work item internal to *MAPLE*

4.22
task list
sequenced set of tasks that may include complex sequences of concurrent tasks

NOTE Depending on the instance, some entries of a task list may refer to either an internal task or an external action. In that case the word task is being used.

4.23
Task Planner
function to analyse a request from the *MAPLE Engine* and to select and sequence *Manufacturing Software Programs* into a *task list* to be executed by the *Execution Manager*

5 Enterprise viewpoint of interfaces between MAPLEs and Manufacturing Software Programs

The classification of the interfaces between MAPLE and Manufacturing Software Programs using the interface description language given in ISO/IEC 14750 is based on objects which have related operations. From an enterprise viewpoint in ODP (ISO/IEC 10746-3) the interface objects can be described as follows.

a) There are three relevant objects: MAPLE; ManufacturingSoftwareProgram, and; MAPLEFinder with respect to these interfaces.

b) The roles of these relevant objects are:

   1) MAPLE is a supplier of MAPLE services;

   2) ManufacturingSoftwareProgram is a user or consumer of MAPLE services. MAPLE itself may also be a user;

   3) MAPLEFinder is used to find a MAPLE which can provide a service request forwarded by another MAPLE.

c) MAPLE has a structure, which is composed of the MAPLE Engine, Dictionary Manager, Manufacturing Data Manager, Software Tool Linker and the Execution Manager. (See ISO 13281-1 for the MAPLE functional architecture.)

d) MAPLE services are defined in clause 8.

e) MAPLE users can access a MAPLE only through an interface defined in clause 9.

6 Symbols and abbreviations

a) DST_ data_storage_type;

b) DS_ data_source;

c) SP_ software program.

7 Dictionary Definition Schema

7.1 General

This clause corresponds to the ODP-Information viewpoint, as in ISO/IEC 10746-3.

The dictionaries allow the handling of both object-based and non-object-based data sets and programs.

7.2 Catalogues

There shall be these three catalogues:

7.2.1 Data Classification Catalogue

Examples of entries in this catalogue are NC program, tool data, setup data and product data.

7.2.2 Software Capability Catalogue

Examples of entries in this catalogue are post processing, monitoring and simulation.

7.2.3 Data Storage Type Catalogue

Examples of entries in this catalogue are file, relational database, object database and image database.

7.3 Manufacturing Data Dictionary

The Manufacturing Data Dictionary is related to the Data Storage Type Catalogue, and contains two main parts: description of data sources, and Logical-to-physical Mapping. The description of data sources contains information about how to handle related manufacturing data.

Examples of entries in this dictionary are an NC program for one specific machine, and STEP product data.

7.4 Manufacturing Software Dictionary

The Manufacturing Software Dictionary is related to the Data Storage Type Catalogue and the Software Capability Catalogue, and contains a description of software programs. This description consists of the capabilities and the invocation of the particular software programs.

Examples of entries in the Manufacturing Software Dictionary are NC program editor, post processor for vendor-specific control, and simulator for CL files.

8 MAPLE services

8.1 General

This clause corresponds to the ODP-Engineering viewpoint, as in ISO/IEC 10746-3.

8.2 Required services

A MAPLE shall, at a minimum, provide the following services:

a) plan task (see 8.5);

b) register data source (see 8.6.2);

c) list data source (see 8.6.3);

d) deregister data source (see 8.6.4);

e) modify data source (see 8.6.5);

f) register software program capability (see 8.6.6);

g) list software program capability (see 8.6.7);

h) deregister software program capability (see 8.6.8);

i) modify software program capability (see 8.6.9);

j) register catalog entry (see 8.6.11);

k) list catalog (see 8.6.12);

l) deregister catalog entry (see 8.6.13);

m) modify catalog entry (see 8.6.14);

n) insert data (see 8.7.2);

o) list data (see 8.7.3);

p) delete data (see 8.7.4);

q) update data (see 8.7.5);

r) retrieve data (see 8.7.6);

s) check out data (see 8.7.7);

t) check in data (see 8.7.8);

u) pass through query (see 8.7.9);

v) translate data (see 8.8);

w) transfer intermediate data (see 8.9);

x) invoke software program capability (see 8.10);

y) start manufacturing software program (see 8.11.1);

z) execute task list (see 8.11.2);

    1) start task list (see 8.11.2.1);

    2) terminate task list (see 8.11.2.2);

    3) suspend task list (see 8.11.2.3);

    4) suspend after (see 8.11.2.4);

    5) resume task list (see 8.11.2.5);

    6) StepMode (see 8.11.2.6);

    7) StepTask (see 8.11.2.7);

    8) StepTo (see 8.11.2.8);

    9) OutOfStep (see 8.11.2.9);

    10) SkipN (see 8.11.2.10);

    11) SkipTo (see 8.11.2.11);

aa) show task list (see 8.11.3.1);

bb) delete task list (see 8.11.3.2);

cc) create task list (see 8.11.3.3);

dd) start trace (see 8.11.4.1);

ee) Stop trace (see 8.11.4.2);

ff) Check task list status (see 8.11.4.3);

gg) Initialize MAPLE (see 8.12.2);

hh) Shut down MAPLE (see 8.12.3);

ii) Administrate system (see 8.12.4);

jj) Communicate MAPLE to MAPLE (see 8.12.5);

kk) Access MAPLE (see 8.13).

The services listed can be extended with vendor-specific services.

8.3 Status return

All services shall return a status.

8.4 Input and output type definitions

The following general type definitions shall apply:

```
typedef int  status

typedef wstring   DataSource

typedef wstring   SPCapability

typedef wstring  CatalogEntry

typedef sequence<wstring>   AttributeList

typedef sequence<any>   ValueList

typedef sequence<wstring>   ArgumentList

typedef sequence<wstring>   TaskList

typedef sequence<CatalogEntry> CatalogEntryList

enum IdentifierType {logical, physical};

struct DataIdentifier {
        IdentifierType   logical_or_physical;
        wstring data_identifier;
};

enum CheckOutStatus {true, false};
```

8.5 Plan task

This service shall provide pre-processing of requests passed from the MAPLE Engine before being executed by the Execution Manager service. The plan task service shall expand the given request into a task list which can be performed directly by the Execution Manager service.

The Task Planner service shall support the following functions:

a) build a task list by replacing parameters and input/output;

b) search in the Software Dictionary for the requested capability;

c) resolve multiple search results by selection;

d) retrieve the dictionary information;

e) ensure that the requested capability can be resolved;

f) references with their actual values;

g) check for format compatibility between inputs and outputs, and insert translation task where necessary;

h) if not able to meet the requested capability, issue an error message to the MAPLE Engine.

The inputs to this service are:

a) SP_capability of type SPCapability;

b) SP_capability_input argument_list of type ArgumentList.

The output from this service is a TaskList.

8.6 Data and Software Dictionary maintenance

8.6.1 Hierarchy

Data source is considered to have a hierarchy, such that a data set is made up of data records. The services that need to be supported to implement the maintenance of the dictionaries and of their associated catalogues shall be as specified in subclauses 8.6.2 to 8.6.14.

8.6.2 Register data source

This service provides the ability for the MAPLE user to register a data source with the Manufacturing Data Dictionary.

The inputs to this service are:

a) data_source_name of type wstring;

b) DST_name of type wstring;

c) DST_parameter_set of type wstring;

d) data_source_command_set of type wstring.

8.6.3 List data source

This service provides the ability for the MAPLE user to query the Manufacturing Data Dictionary and find those data sources currently registered with MAPLE.

The input to this service is the data_source_filter of type wstring, which defines a selection of the existing data_sources based on the value of its attributes.

The output from this service is a set_of_data_sources with the relevant attributes of type DataSourceList.

8.6.4 Deregister data source

This service provides the ability for the MAPLE user to deregister a data source from the Manufacturing Data Dictionary.

The input to this service is the data_source_name of type wstring.

8.6.5 Modify data source

This service provides the ability for the MAPLE user to modify a data source in the Manufacturing Data Dictionary.

The inputs to this service are:

a) data_source_name of type wstring;

b) data_source_attributes of type AttributeList (defining those attributes to be modified);

c) data_source_attribute_values of type ValueList (provides the corresponding values of the attributes to be modified).

8.6.6 Register software program capability

This service provides the ability for the MAPLE user to register a capability of a software program with the Manufacturing Software Dictionary.

The inputs to this service are:

a) SP_capability_name of type wstring;

b) SP_name of type wstring;

c) SP_location of type wstring;

d) SP_argument_template of type wstring;

e) SP_capability of type SPCapability;

f) SP_argument_set of type ArgumentList.

8.6.7 List software program capability

This service provides the ability for the MAPLE user to query the Manufacturing Software Dictionary and find those software program capabilities currently registered with MAPLE.

The input to this service is the SP_capability_filter of type SPCapability which defines a selection of the existing SP_capabilities based on the value of its attributes.

The output from this service is a set_of_SP_capabilities of type SPCapabilityList with the relevant attributes SP_capability_attributes of type AttributeList.

8.6.8 Deregister software program capability

This service provides the ability for the MAPLE user to deregister a SP_capability from the Manufacturing Software Dictionary.

The input to this service is the SP_capability_name of type wstring.

8.6.9 Modify software program capability

This service provides the ability for the MAPLE user to modify a software program capability in the Manufacturing Software Dictionary.

The inputs to this service are:

a) SP_capability_name of type wstring;

b) SP_capability_attributes of type AttributeList (defining those attributes to be modified);

c) SP_capability_attribute_values of type ValueList (providing the corresponding values of the attributes to be modified).

8.6.10 Logical-to-physical Mapping maintenance

This service is supported through services specified in subclauses 8.6.11 to 8.6.14.

8.6.11 Register catalogue entry

This service provides the ability for the MAPLE user to register an entry to a specified catalogue.

The inputs to this service are:

a) catalogue_name of type wstring;

b) catalogue_entry_identifier of type wstring (defining a user unique identifier for the catalogue entry);

c) catalogue_entry_attributes of type AttributeList (the parameters required by the catalogue);

d) catalog_entry_attribute_values of type ValueList.

8.6.12 List catalogue

This service provides the ability for the MAPLE user to query a specified catalogue.

The inputs to this service are:

a) catalogue_name of type wstring;

b) catalogue_entry_filter of type AttributeList (defining a selection of the existing catalogue entries based on the value of its attributes).

The output from this service is a set of catalogue entries with the relevant catalogue_entry_attributes of type CatalogEntryList.

8.6.13 Deregister catalogue entry

This service provides the ability for the MAPLE user to deregister a catalogue entry from a specified catalogue.

The inputs to this service are:

a) catalogue_name of type wstring;

b) catalogue_entry_identifier of type wstring.

8.6.14 Modify catalogue entry

This service provides the ability for the MAPLE user to modify a catalogue entry in a specified catalogue.

The inputs to this service are:

a) catalogue_name of type wstring;

b) catalogue_entry_identifier of type wstring;

c) catalogue_entry_attributes of type AttributeList (defining those attributes to be modified);

d) catalogue_entry_attribute_values of type ValueList.

8.7 Manufacturing data access

8.7.1 Required services

The services that need to be supported to implement manufacturing data access shall be as specified in subclauses 8.7.2 to 8.7.9. These services shall apply to both data sets and data records. Both types of use of these services shall require specific registration of the data types in the Data Dictionary.

8.7.2 Insert data

This service allows the MAPLE user to insert a new data item into a Manufacturing Database.

The inputs to this service are:

a) data_source_name of type wstring;

b) logical_data_identifier of type wstring (a user unique identifier within the MAPLE environment denoting a specific data item);

c) physical_data_identifier of type wstring (an identifier denoting a specific data item within a data_source);

d) data_item of type wstring (the actual data item being inserted into the data_source).

8.7.3 List data

This service allows the MAPLE user to get a list of all the data items in the Manufacturing Databases for a specific data_classification.

The input to this service is the data_classification of type wstring.

The outputs from this service are:

a) logical_data_identifier of type wstring (containing the logical names and data_source names);

b) physical_data_identifier of type wstring (containing the corresponding physical identifier names).

8.7.4 Delete data

This service allows the MAPLE user to delete a particular data item provided the data item has not been checked out by another user.

The inputs to this service are:

a) data_source_name of type wstring;

b) logical_data_identifier of type DataIdentifier or physical_data_identifier of type DataIdentifier.

8.7.5 Update data

This service allows the MAPLE user to update existing data items provided the data item has already been checked out by the same user.

The inputs to this service are:

a) data_source_name of type wstring;

b) logical_data_identifier of type DataIdentifier or physical_data_identifier of type DataIdentifier;

c) data_item_attributes of type AttributeList (defining those attributes to be updated);

d) data_item_attribute_values of type ValueList (providing the corresponding values of the attributes to be updated).

8.7.6 Retrieve data

This service allows the MAPLE user to retrieve a particular data item.

The inputs to this service are:

a) data_source_name of type wstring;

b) logical_data_identifier of type DataIdentifier or physical_data_identifier of type DataIdentifier.

The outputs from this service are:

a) data_item of type wstring;

b) data_check_out_status of type CheckOutStatus.

NOTE This is true if the data has been checked out by another user.

8.7.7 Check out data

This service allows the MAPLE user to reserve the right to exclusively update a specific data set or data record until the data is checked in (see 8.7.8).

The inputs to this service are:

a) data_source_name of type wstring;

b) logical_data_identifier of type DataIdentifier or physical_data_identifier of type DataIdentifier.

The output from this service is the check_out_identifier of typewstring, a system generated identifier of type wstring for the check out.

8.7.8 Check in data

This service allows the MAPLE user to release the right to exclusively update a specific data set or data record.

The input to this service is the check_out_identifier of type wstring.

8.7.9 Pass through query

This service allows the MAPLE user to pass to the Data Manager any data retrieval query related to a specific data source.

The inputs to this service are:

a) data_source_name of type wstring (needed reference to the database to be queried);

b) query_string of type wstring.

The output from this service is the query_output of type wstring.

8.8 Translate data

This service allows the MAPLE user to execute a particular data translation process.

The inputs to this service are:

a) translation_name of type wstring (identifier for the translation);

b) input_schema of type wstring (schema of the input file);

c) output_schema of type wstring (schema of the output file);

d) output_location of type wstring (location for the resulting translation process).

8.9 Transfer intermediate data

This service allows data to be sent from one manufacturing software program to another manufacturing software program without having to store the data in the Manufacturing Database.

The inputs to this service are:

a) target_software_program_name of type wstring (SP_name of type wstring of the target Manufacturing Software Program);

b) data_item of type wstring.

8.10 Invoke software program capability

This service allows making use of the capability of a Manufacturing Software Program registered in the Manufacturing Software Dictionary.

The inputs to this service are:

a) SP_capability_name of type wstring;

b) SP_capability_input_argument_list of type ArgumentList.

The output from this service is the SP_capability_output_argument_list of type ArgumentList.

8.11 Program execution management

These services are intended to support development and maintenance activities.

The Program Execution Manager services shall also provide the following capabilities:

a) execution of a task list which can be stepwise, terminated or temporarily suspended by the originator of the command;

b) provision of the task list and the execution history to the command originator;

c) modification of the task list by the originator.

8.11.1 Start Manufacturing Software Program

This service allows the Execution Manager to request the MAPLE Engine to start a Manufacturing Software Program registered in the Manufacturing Software Dictionary.

Input: SP_start_up_script of type wstring.

8.11.2 Execute task list

This service allows the Execution Manager to request the MAPLE Engine to execute a task list. It shall provide the following options.

For all options the inputs are:

a) task_list_id of type wstring;

b) task_id of type wstring.

8.11.2.1 Start task list

This service allows for the execution of a task list.

8.11.2.2 Terminate task list

This service allows for the termination of a task list.

8.11.2.3 Suspend task list

This service causes the suspension of a task list.

8.11.2.4 Suspend after

This service causes the suspension of a task list after the execution of a specific task (or external action).

8.11.2.5 Resume task list

This service causes the execution of the task list to be resumed.

8.11.2.6 StepMode

This service causes the execution to go into step mode rather than sequential mode.

8.11.2.7 StepTask

This service causes the next task (or action) to be executed then stop.

8.11.2.8 StepTo

This service causes the execution of the task list from the current task to a particular future point in the task list.

8.11.2.9 OutOfStep

This service causes the task list to be executed in normal mode.

8.11.2.10 SkipN

This service allows the execution process to be started n tasks from the current place within the task list.

An additional input is: n of type wstring.

8.11.2.11 SkipTo

This service allows the execution process to be restarted at a particular task within the task list.

8.11.3 Task list support

8.11.3.1 Show task list

This service allows for a particular task list to be displayed.

The inputs to this service are:

a) task_list_id of type wstring;

b) output_destination of type wstring.

The output from this service is task_list_information of type ValueList.

8.11.3.2 Delete task list

This service allows for a particular task list to be deleted.

The input to this service is: task_list_id of type wstring.

8.11.3.3 Create task list

This service allows for the creation of a task list.

The input to this service is: task_list_name of type wstring.

8.11.4 Status management

8.11.4.1 Start trace

This service creates a trace of the execution of a particular task list.

The input for this service is: task_list_id of type wstring.

8.11.4.2 Stop trace

This service allows a trace to be stopped on the execution of a particular task list.

The input to this service is: task_list_id of type wstring.

8.11.4.3 Check task list status

This service allows the user the ability to check on what task is being executed.

The input to this service is: task_list_id of type wstring.

The output from this service is: task_id of type wstring.

8.12 MAPLE system services

8.12.1 Required services

The services that shall be required are specified in subclauses 8.12.2 to 8.12.5.

8.12.2 Initialize MAPLE

This service initializes and starts a MAPLE.

The input to this service is: MAPLE_name of type wstring.

8.12.3 Shut down MAPLE

This service provides an orderly shut down of a MAPLE.

The input to this service is: MAPLE_name of type wstring.

8.12.4 Administrate system

This service provides a limited environment to system administrators to handle system critical operations.

The input to this service is: MAPLE_name of type wstring.

8.12.5 Communicate MAPLE to MAPLE

This service provides communication between one MAPLE and another MAPLE.

The input to this service is: other_MAPLE_name of type wstring.

8.13 Access MAPLE

This service shall provide access control to MAPLE.

The input to this service is: MAPLE_name of type wstring.

The output from this service is: return_value of type status.

9 Interfaces with MAPLE

9.1 General

This clause includes the ODP-Computational viewpoint and the ODP-Informational viewpoint, as in ISO/IEC 10746-3. The specific correspondences to the respective viewpoints will be indicated in the relevant subclauses.

The interface messages shall be transferred in a 16-bit character representation according to ISO/IEC 10646-1, and the specification for "wstring" according to CORBA 2.2.

NOTE Since this document only specifies the interface at the application layer between MAPLE and software programs, MAPLE developers will have to use existing solutions, e.g., DCOM, CORBA, for the lower communication layers.

9.2 Summary of interfaces

A MAPLE shall, at a minimum, provide interface points to the following:

a) Manufacturing Software Programs or other MAPLEs (see subclause 9.3.2 and annex A) comprised of

   1) interface MAPLE,

   2) interface ManufacturingSoftwareProgram,

   3) interface MAPLEFinder;

b) Manufacturing Databases (see subclause 9.4);

c) Data Translator (see subclause 9.5).

9.3 Interfaces to Manufacturing Software Programs or other MAPLEs

9.3.1 General

These interfaces consist of the interface to MAPLE, the interface to Manufacturing Software Programs and the interface to the MAPLE Finder, which will support eleven types of actions:

a) Request by Manufacturing Software Program for a New Session (see subclause 9.3.4.1);

b) Initiation of a New Session with a Manufacturing Software Program by MAPLE (see subclause 9.3.4.2);

c) Request MAPLE service (see subclause 9.3.4.3);

d) Invoke Manufacturing Software Program capability by MAPLE (see subclause 9.3.4.4);

e) Retrieve data from MAPLE (see subclause 9.3.4.5);

f) Send data to MAPLE (see subclause 9.3.4.6);

g) Retrieve data (see subclause 9.3.4.7);

h) Send data to a Manufacturing Software Program from another MAPLE (see subclause 9.3.4.8);

i) Broadcast to other MAPLEs (see subclause 9.3.4.9);

j) Terminate session from Manufacturing Software Program (see subclause 9.3.4.10);

k) Terminate session from MAPLE (see subclause 9.3.4.11).

The interfaces are defined in 9.3.2 and annex A. The semantics for the common action parameters are given in subclause 9.3.3, and the interface action semantics and syntax are given in subclause 9.3.4.

Interfaces described between MAPLE and a Manufacturing Software Program apply also to the interface between MAPLE and another MAPLE.

The specification of exception handling for the MAPLE interface over and above the provision of status information is left to the MAPLE implementor.

9.3.2 Interface definitions

The interfaces are defined in annex A, using ISO/IEC 14750.

9.3.3 Common action parameters

The semantics for action parameters used in the interface definitions are given below:

session_identifier: a MAPLE-defined data type to recognize a unique session with MAPLE;

request_identifier: a MAPLE-defined data type to recognize a unique request with MAPLE;

originator_identifier: a MAPLE-defined data type to uniquely recognize the originator of a command;

application_program_identifier: a MAPLE-defined data type to recognize a unique application program;

data_send_request_identifier: a MAPLE-defined data type to recognize a unique request for sending data;

requested_data_identifier: a MAPLE-defined data type to recognize a unique data item that had been requested;

data_retrieve_identifier: a MAPLE-defined data type to recognize a unique data item to be retrieved;

MAPLE_identifier: a MAPLE-defined data type to uniquely recognize a MAPLE;

requested_service: a MAPLE-defined structure which invokes a MAPLE service;

service_name: a MAPLE defined key word string to specify one of the MAPLE services described in clause 8;

service_req_parameter_list: a parameter list for the input to a requested MAPLE service;

service_results: a MAPLE defined structure which provides output parameters from a MAPLE service;

service_rsp_parameter_list: a parameter list for the output from a requested MAPLE service;

requested_SP_capability: a MAPLE-defined structure which invokes a capability of a manufacturing software program;

SP_capability_name: a key word string to specify a desired capability provided by a registered Manufacturing Software Program;

SP_capability_parameter_list: a parameter list for the input to a software program capability;

SP_capability_results: a parameter list for the output from a software program capability;

new_session_status: a MAPLE defined data type to describe the status of a new session request command;

request_status: a MAPLE defined data type to describe the status of an interface request;

termination_status: a MAPLE defined data type to describe the status of a command termination;

data_receive_status: a MAPLE defined data type to describe the status of a data receiving action;

receive_status: a MAPLE defined data type to describe the status of a data receive action;

data_to_be_sent: application specific data being provided by MAPLE or a manufacturing software program;

data_requested: application specific data that has been demanded by MAPLE or a manufacturing software program;

application_program_data_type: defines the type of data a manufacturing software program requires from MAPLE.

9.3.4 Interface action semantics and syntax

This subclause corresponds to the ODP - Computational viewpoint, as in ISO/IEC 10746-3.

9.3.4.1 Request by Manufacturing Software Program for a new session

This action allows a Manufacturing Software Program to establish a communication session with MAPLE with the sequence shown in Figure 1.

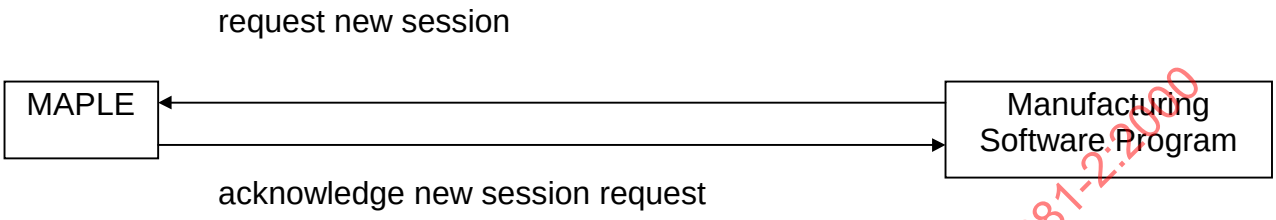

Figure 1 — Request by Manufacturing Program for a new session

request_new_session is an operation invoked by a manufacturing software program to establish a communication session with MAPLE.

void request_new_session(

in Identifier originator_identifier,
in Identifier request_identifier
);

acknowledge_new_session_request is an operation invoked by MAPLE to the requesting manufacturing software program to confirm creation of, and identify, a new session.
void acknowledge_new_session_request(
in Identifier originator_identifier,
in Identifier request_identifier
in Identifier session_identifier
in Status new_session_status
);

9.3.4.2 Initiation of a new session with a Manufacturing Software Program by MAPLE

This action allows MAPLE to establish a communication session with a Manufacturing Software Program in the Manufacturing Software Dictionary with the sequence shown in Figure 2.

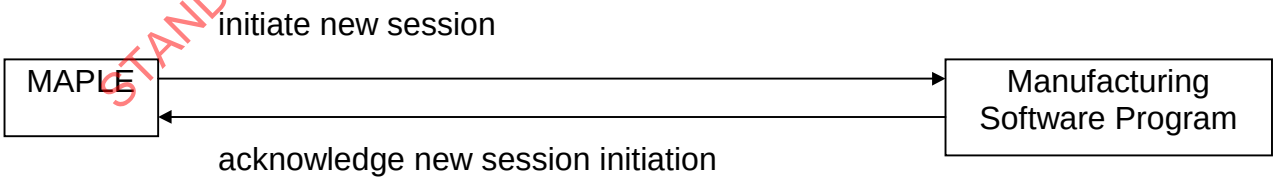

Figure 2 — Initiation of a new session with a Manufacturing Software Program by MAPLE

initiate_new_session is an operation invoked by MAPLE to establish a communication session with a manufacturing software program.

```
void initiate_new_session(
                in Identifier application_program_identifier,
                in Identifier session_identifier
        );
```

acknowledge_new_session_initiation is an operation invoked by a manufacturing software program to MAPLE to confirm the establishment of a new session.

```
void acknowledge_new_session_initiation(
                in Identifier session_identifier,
                in Identifier new_session_status
        );
```

9.3.4.3 Request MAPLE Service

This action allows a Manufacturing Software Program to invoke a service provided by MAPLE with the sequence shown in Figure 3.

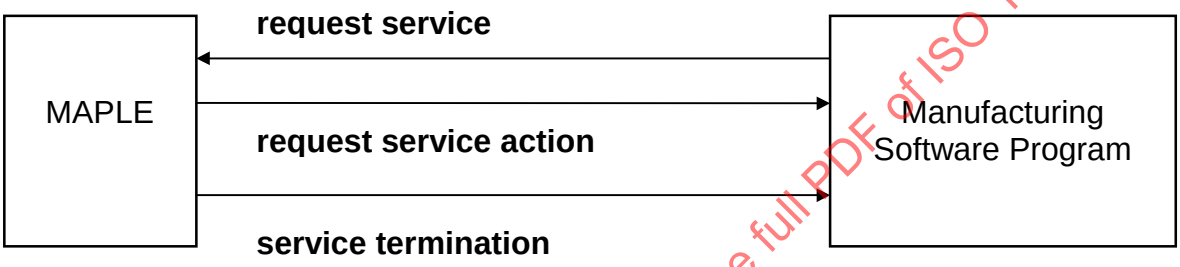

Figure 3 — Request MAPLE service

request_service is an operation invoked by a manufacturing software program to invoke an external service of MAPLE.

```
void request_service(
                in Identifier session_identifier,
                in Identifier request_identifier,
                in RequestedService requested_service
        );
```

request_service_action_response is an operation invoked by MAPLE to the requesting manufacturing software program to confirm receipt of a service request.

```
void request_service_action_response(
                in Identifier session_identifier,
                in Identifier request_identifier,
                in RequestStatus request_status
        );
```

service_termination is an operation invoked by MAPLE to the requesting manufacturing software program to confirm termination of a service request.

```
void service_termination(
                in Identifier session_identifier,
                in Identifier request_identifier,
                in Status termination_status,
                in ServiceResults service_results
        );
```

Requested service could be a basic service (refer to clause 8) or a service registered in the Manufacturing Software Dictionary.

9.3.4.4 Invoke Manufacturing Software Program Capability by MAPLE

This action allows MAPLE to invoke a capability of a Manufacturing Software Program with the sequence shown in Figure 4.

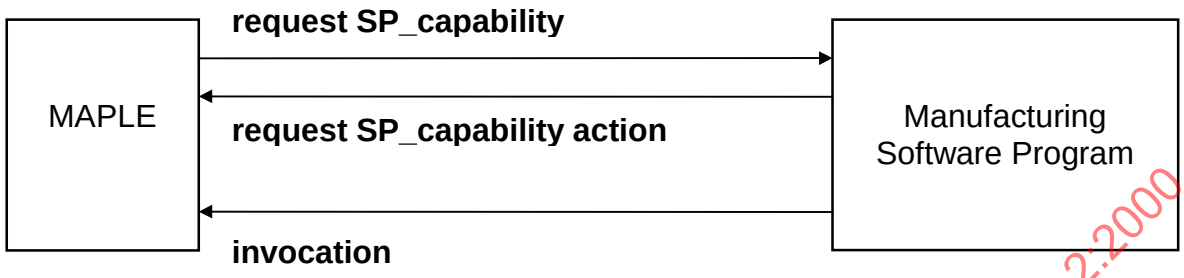

Figure 4 — Invoke Manufacturing Software Program capability by MAPLE

request_SP_capability is an operation invoked by MAPLE to a manufacturing software program to invoke an external capability.

```
void request_SP_capability(
                in Identifier session_identifier,
                in Identifier request_identifier,
                in RequestedCapability requested_SP_capability
        );
```

request_SP_capability_action_response is an operation invoked by a manufacturing software program to MAPLE to confirm receipt of a request for a capability.

```
void request_SP_capability_action_response(
                in Identifier session_identifier,
                in Identifier request_identifier,
                in RequestStatus request_status
        );
```

invocation_termination is an operation invoked by a manufacturing software program to MAPLE to confirm termination of a request for a capability.

```
        void invocation_termination(
                in Identifier session_identifier,
                in Identifier request_identifier,
                in Status termination_status,
                in CapabilityParameterList SP_capability_results
        );
```

9.3.4.5 Retrieve Data from MAPLE

This action allows a Manufacturing Software Program to retrieve data from MAPLE with the sequence shown in Figure 5.

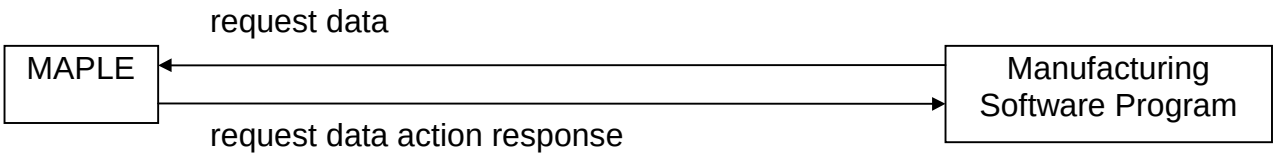

Figure 5 — Retrieve data from MAPLE

request_data is an operation invoked by a manufacturing software program to MAPLE to retrieve data from MAPLE.

```
void request_data(
        in Identifier session_identifier,
        in Identifier request_identifier,
        in RequestedService requested_service
    );
```

request_data_action_response is an operation invoked by MAPLE to a manufacturing software program to provide the data requested.

```
void request_data_action_response(
        in Identifier session_identifier,
        in Identifier request_identifier,
        in ParameterList data_requested,
        in RequestStatus request_status
    );
```

9.3.4.6 Send Data to MAPLE

This action allows a Manufacturing Software Program to send data to MAPLE with the sequence shown in Figure 6.

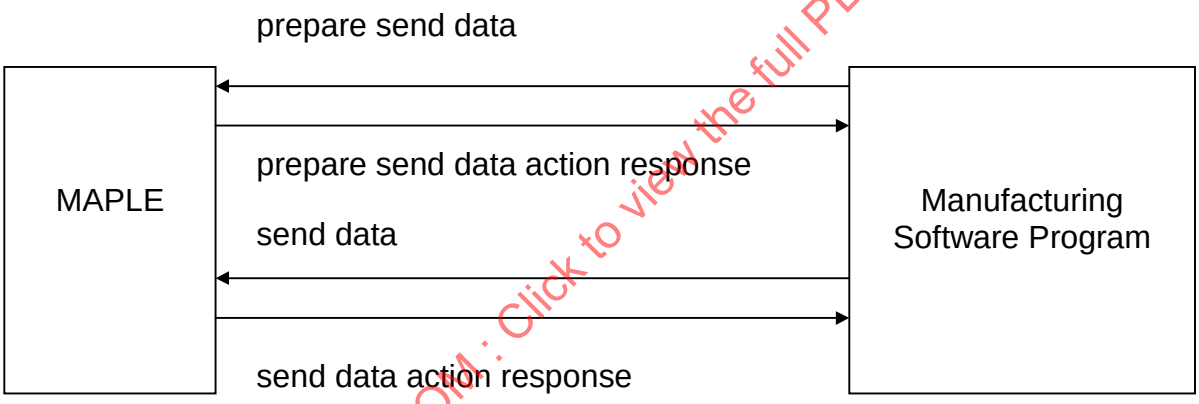

Figure 6 — Send data to MAPLE

prepare_data_send is an operation invoked by a manufacturing software program to MAPLE in preparation to send data to MAPLE.

```
void prepare_data_send(
        in Identifier session_identifier,
        in Identifier request_identifier,
        in RequestedService requested_service
    );
```

prepare_data_send_action_response is an operation invoked by MAPLE to a manufacturing software program to indicate its readiness to receive data.

```
void prepare_data_send_action_response(
        in Identifier session_identifier,
        in Identifier request_identifier,
        in Identifier data_send_request_identifier,
        in RequestStatus request_status
    );
```

send_data is an operation invoked by a manufacturing software program to MAPLE to send data to MAPLE.

void send_data_from_MAPLE(
in Identifier data_send_request_identifier,
in ParameterList data_to_be_sent
);

send_data_action_response is an operation invoked by MAPLE to a manufacturing software program to indicate receipt of data.

void send_data_action_response(
in Identifier data_send_request_identifier,
in Status data_receive_status
);

9.3.4.7 Retrieve Data from another MAPLE

This action allows MAPLE to retrieve data from another MAPLE with the sequence shown in Figure 7.

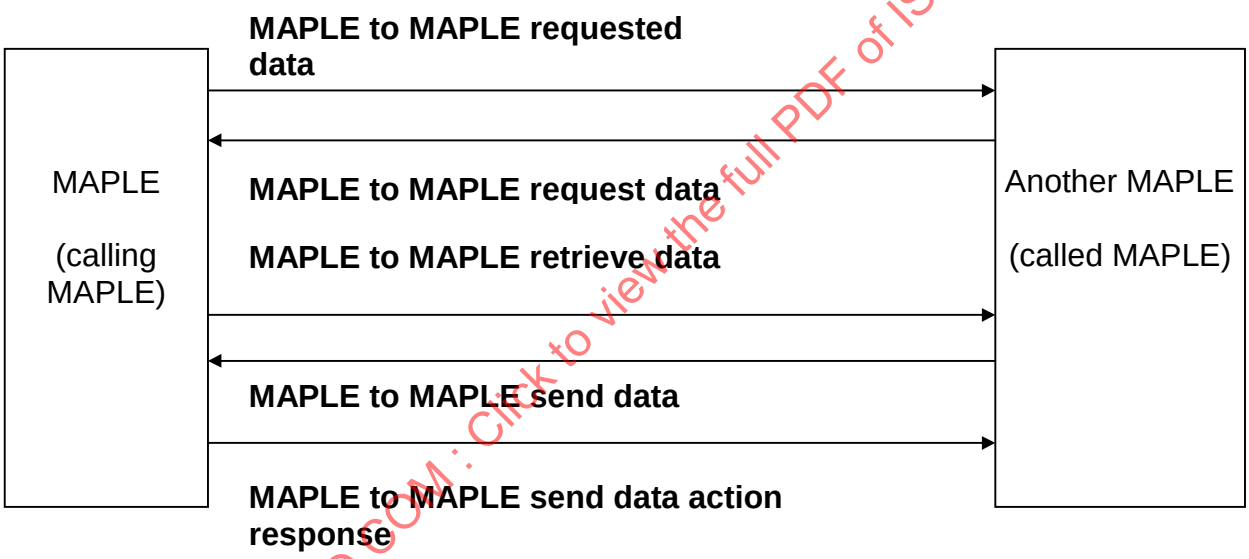

Figure 7 — Retrieve data from another MAPLE

MAPLE_to_MAPLE_requested_data is an operation invoked by a one MAPLE (the calling MAPLE) to another MAPLE (the called MAPLE) to request retrieval of data.

void MAPLE_to_MAPLE_requested_data(
in Identifier session_identifier,
in Identifier request_identifier,
in Identifier requested_data_identifier
);

MAPLE_to_MAPLE_request_data_action_response is an operation invoked by a called MAPLE to a calling MAPLE to indicate its readiness to send data.

void MAPLE_to_MAPLE_requested_data_action_response(
in Identifier session_identifier,
in Identifier request_identifier,
in Identifier data_retrieve_identifier,
in RequestStatus request_status
);

MAPLE_to_MAPLE_retrieve_data is an operation invoked by a calling MAPLE to a called MAPLE to indicate its readiness to receive data.

```
void MAPLE_to_MAPLE_retrieve_data(
                in Identifier data_retrieve_identifier
        );
```

MAPLE_to_MAPLE_send_data is an operation invoked by a called MAPLE to a calling MAPLE to send the requested data.

```
void MAPLE_to_MAPLE_send_data(
                in Identifier data_retrieve_identifier,
                in ParameterList data_requested
        );
```

MAPLE_to_MAPLE_send_data_action_response is an operation invoked by a calling MAPLE to a called MAPLE to indicate receipt of data.

```
void MAPLE_to_MAPLE_send_data_action_response(
                in Identifier data_retrieve_identifier,
                in Status data_receive_status
        );
```

9.3.4.8 Send Data to a Manufacturing Software Program

This action allows MAPLE to send data to a Manufacturing Software Program with the sequence shown in Figure 8.

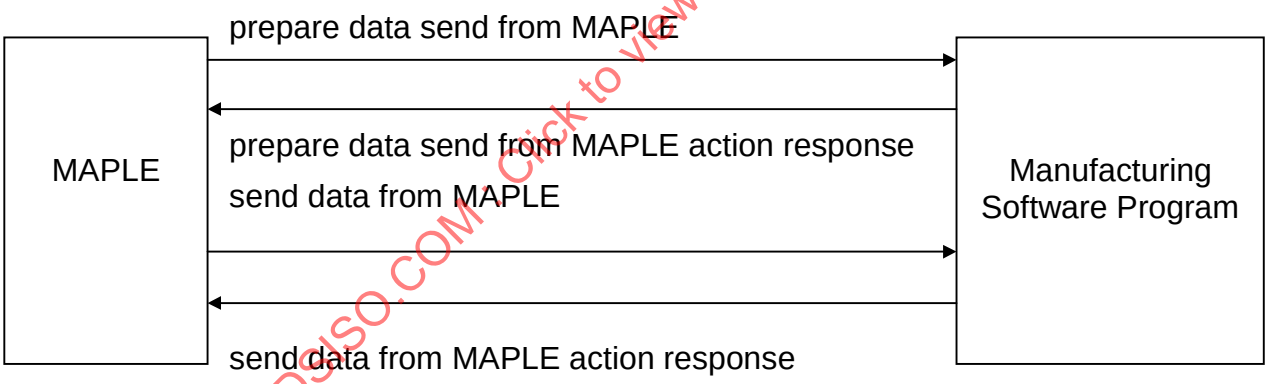

Figure 8 — Send data to a Manufacturing Software Program

prepare_data_send_from_MAPLE is an operation invoked by MAPLE to a manufacturing software program in preparation to send data to the manufacturing software program.

```
void prepare_data_send_from_MAPLE(
                in Identifier session_identifier,
                in Identifier data_send_request_identifier,
                in wstring application_program_data_type
        );
```

prepare_data_send_from_MAPLE_action_response is an operation invoked by a manufacturing software program to MAPLE to indicate readiness to receive data from MAPLE.

```
void prepare_data_send_from_MAPLE_action_response(
                in Identifier session_identifier,
                in Identifier data_send_request_identifier,
```

```
            in RequestStatus request_status
    );
```

send_data_from_MAPLE is an operation invoked by MAPLE to a manufacturing software program to send data to the manufacturing software program.

```
void send_data_from_MAPLE(
            in Identifier data_send_request_identifier,
            in ParameterList data_to_be_sent
    );
```

send_data_from_MAPLE_action_response is an operation invoked by a manufacturing software program to MAPLE to confirm receipt of data from MAPLE.

```
void send_data_from_MAPLE_action_response(
            in Identifier data_send_request_identifier,
            in Status data_receive_status
    );
```

9.3.4.9 Broadcast to other MAPLEs

Whenever MAPLE comes across a task or action in its task list that it cannot perform, it will attempt to send the action out to another MAPLE. In the suite of software programs, there is one whose specific function is to go and find another MAPLE to perform a given action. It is called the “MAPLE Finder”, and it will find the “Other MAPLE” either by selecting one, subject to a given set of rules, or by broadcasting to all other MAPLEs to query their suitability to perform the action in question.

This action allows MAPLE to find another MAPLE to perform a specific task that cannot be performed by MAPLE with the sequence shown in Figure 9.

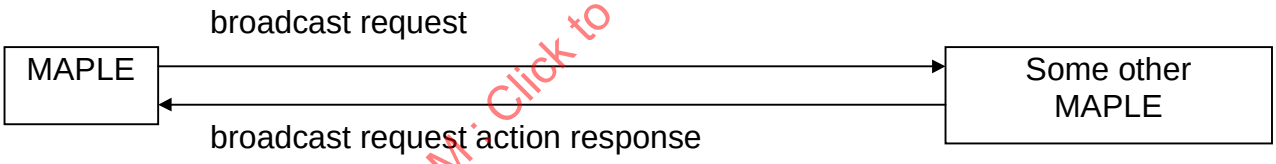

Figure 9 — Broadcast to other MAPLEs

broadcast_request is an operation invoked by one MAPLE to all other MAPLE’s in order to find another MAPLE which can perform a specific action that cannot be performed by the broadcasting MAPLE.

```
void broadcast_request(
            in Identifier request_identifier,
            in Identifier MAPLE_identifier,
            in RequestedCapability requested_SP_capability
    );
```

broadcast_request_action_response is an operation by each MAPLE to the broadcasting MAPLE in response to a broadcast command invoked, to indicate its ability or inability to offer the requested capability.

```
void broadcast_request_action_response(
            in Identifier request_identifier,
            in Identifier MAPLE_identifier,
            in RequestStatus request_status
    );
```

9.3.4.10 Terminate Session from Manufacturing Software Program

This action allows a Manufacturing Software Program to terminate a session with MAPLE with the sequence shown in Figure 10.

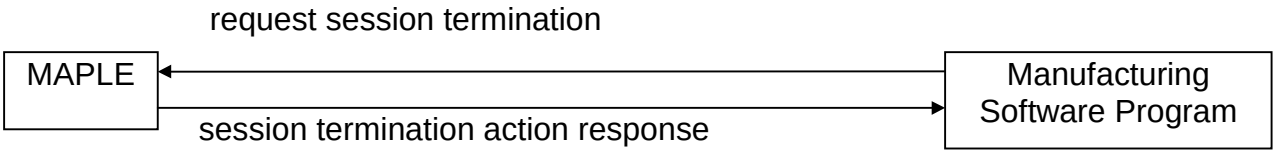

Figure 10 — Terminate session from Manufacturing Software Program

request_session_termination is an operation invoked by a manufacturing software program to terminate its session with a MAPLE.

```
void request_session_termination(
                in Identifier session_identifier
        );
```

session_termination_action_response is an operation invoked by a MAPLE to a manufacturing software program to confirm the requested session termination with this manufacturing software program.

```
void request_session_termination_action_response(
                in Identifier session_identifier,
                in Status termination_status
        );
```

The termination_status returned by MAPLE can be negative only if MAPLE is the initiator of the session.

9.3.4.11 Terminate Session from MAPLE

This action allows a MAPLE to terminate a session with a Manufacturing Software Program or another MAPLE with the sequence shown in Figure 11.

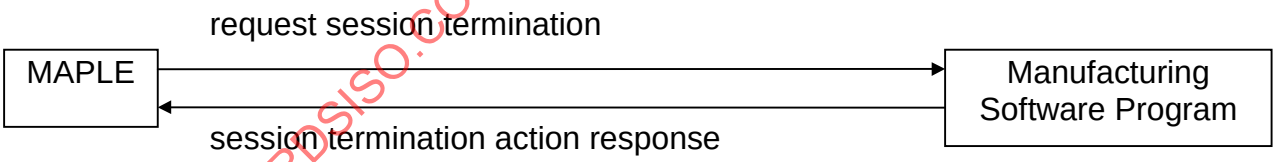

Figure 11 — Terminate session from MAPLE

request_session_termination is an operation invoked by a MAPLE to terminate its session with a manufacturing software program or another MAPLE.

```
void request_session_termination(
                in Identifier session_identifier
        );
```

session_termination_action_response is an operation invoked by a manufacturing software program to MAPLE to confirm the requested session termination by MAPLE.

```
void request_session_termination_action_response(
                in Identifier session_identifier,
                in Status termination_status
        );
```

The termination_status returned by the Manufacturing Software Program can be negative only if the Manufacturing Software Program is the initiator of the session.

9.4 Interface to Manufacturing Databases

This interface shall be called the manufacturing_databases_interface. It shall allow data access to Manufacturing Databases. This comprises the data access actions (insert, delete, update and retrieve), multi-user access and database security issues.

NOTE So that MAPLE implementation is independent of the databases of a given organization, this interface should allow access to most available database systems. There are a number of available standards and universal access tools to fulfil this objective. For example, SQL-3 as described in ISO/IEC 9579 parts 1, 2 and 3, ODBC and Internet standards.

9.5 Interface to Data Translator

A data translator can be considered as a Manufacturing Software Program. This interface shall be called the data_translator_interface. The interface actions with MAPLE shall be the same as stated in 9.3.4.2, 9.3.4.4, 9.3.4.6, 9.3.4.8 and 9.3.4.11.

Annex A
(normative)

Interface description

The interfaces are defined using ISO/IEC 14750 for the interface definition language.

```
module MAPLEs_and_Manufacturing_Software_Programs {

// Identifier
typedef wstring Identifier;

// Status
enum Status {success, failure};
enum RequestStatus {request_success, request_denied, invalid_request};
enum CheckOutStatus {true, false};

// Parameters for MAPLE

// General Parameter List
union Constant switch (long) {
        case 1: long long_const;
        case 2: unsigned long ulong_const;
        case 3: double double_const;
};
enum ParameterAlternative {identifier, constant, other_params};
union Parameter switch (ParameterAlternative) {
        case identifier: Identifier identifier;
        case constant: Constant constant;
        default: any other_params;
};
typedef sequence<Parameter> ParameterList;

// Parameters for MAPLE Services
// Primary Type Definitions for Input and Output Parameters
typedef long ServiceStatus;
typedef wstring DataSource;
typedef wstring SPCapability;
typedef wstring CatalogueEntry;
typedef sequence<wstring> AttributeList;
typedef sequence<any> ValueList;
typedef sequence<wstring> ArgumentList;
typedef sequence<wstring> TaskList;
typedef sequence<DataSource> DataSourceList;
typedef sequence<SPCapability> SPCapabilityList;
typedef sequence<CatalogueEntry> CatalogueEntryList;
enum IdentifierType {logical, physical};
struct DataIdentifier {
        IdentifierType   logical_or_physical;
        wstring data_identifier;
};

// Plan Task
struct InPlanTask {
        SPCapability    SP_capability;
        ArgumentList    sp_capability_input_argument_list;
};
```

```
struct OutPlanTask {
        ServiceStatus   status;
        TaskList        task_list;
};

// Register Data Source
struct InRegisterDataSource {
        wstring data_source_name;
        wstring DST_name;
        wstring DST_parameter_set;
        wstring data_source_command_set;
};
struct OutRegisterDataSource {
        ServiceStatus   status;
};

// List Data Source
struct InListDataSource {
        wstring data_source_filter;
};
struct OutListDataSource {
        ServiceStatus   status;
        DataSourceList set_of_data_sources;
};

// Deregister Data Source
struct InDeregisterDataSource {
        wstring data_source_name;
};
struct OutDeregisterDataSource {
        ServiceStatus   status;
};

// Modify Data Source
struct InModifyDataSource {
        wstring data_source_name;
        AttributeList   data_source_attributes;
        ValueList       data_source_attribute_values;
};
struct OutModifyDataSource {
        ServiceStatus   status;
};

// Register Software Program Capability
struct InRegisterSoftwareProgramCapability {
        wstring SP_capability_name;
        wstring SP_name;
        wstring SP_location;
        wstring SP_argument_template;
        SPCapability    SP_capability;
        ArgumentList    SP_argument_set;
};
struct OutRegisterSoftwareProgramCapability {
        ServiceStatus   status;
};

// List Software Program Capability
struct InListSoftwareProgramCapability {
        SPCapability    SP_capability_filter;
};
```

```
struct OutListSoftwareProgramCapability {
        ServiceStatus   status;
        SPCapabilityList        set_of_SP_capabilities;
};

// Deregister Software Program Capability
struct InDeregisterSoftwareProgramCapability {
        wstring SP_capability_name;
};
struct OutDeregisterSoftwareProgramCapability {
        ServiceStatus   status;
};

// Modify Software Program Capability
struct InModifySoftwareProgramCapability {
        wstring SP_capability_name;
        AttributeList   SP_capability_attributes;
        ValueList       SP_capability_attribute_values;
};
struct OutModifySoftwareProgramCapability {
        ServiceStatus   status;
};

// Register Catalogue Entry
struct InRegisterCatalogueEntry {
        wstring catalogue_name;
        wstring catalogue_entry_identifier;
        AttributeList   catalogue_entry_attributes;
        ValueList       catalogue_entry_attribute_values;
};
struct OutRegisterCatalogueEntry {
        ServiceStatus   status;
};

// List Catalogue
struct InListCatalogue {
        wstring catalogue_name;
        AttributeList   catalogue_entry_filter;
};
struct OutListCatalogue {
        ServiceStatus   status;
        CatalogueEntryList      set_of_catalogue_entries;
};

// Deregister Catalogue Entry
struct InDeregisterCatalogueEntry {
        wstring catalogue_name;
        wstring catalogue_entry_identifier;
};
struct OutDeregisterCatalogueEntry {
        ServiceStatus   status;
};

// Modify Catalogue Entry
struct InModifyCatalogueEntry {
        wstring catalogue_name;
        wstring catalogue_entry_identifier;
        AttributeList   catalogue_entry_attributes;
        ValueList       catalogue_entry_attribute_values;
};
```

```
struct OutModifyCatalogueEntry {
        ServiceStatus   status;
};

// Insert Data
struct InInsertData {
        wstring data_source_name;
        wstring logical_data_identifier;
        wstring physical_data_idenfifier;
        wstring data_item;
};
struct OutInsertData {
        ServiceStatus   status;
};

// List Data
struct InListData {
        wstring data_classification;
};
struct OutListData {
        ServiceStatus   status;
        wstring logical_data_identifier;
        wstring physical_data_idenfifier;
};

// Delete Data
struct InDeleteData {
        wstring data_source_name;
        DataIdentifier    logical_or_physical_data_identifier;
};
struct OutDeleteData {
        ServiceStatus   status;
};

// Update Data
struct InUpdateData {
        wstring data_source_name;
        DataIdentifier    logical_or_physical_data_identifier;
        AttributeList     data_item_attributes;
        ValueList         data_item_attribute_values;
};
struct OutUpdateData {
        ServiceStatus   status;
};

// Retrieve Data
struct InRetrieveData {
        wstring data_source_name;
        DataIdentifier    logical_or_physical_data_identifier;
};
struct OutRetrieveData {
        ServiceStatus   status;
        wstring data_item;
        CheckOutStatus          data_check_out_status; // true or false ?
};

// Check Out Data
struct InCheckOutData {
        wstring data_source_name;
        DataIdentifier    logical_or_physical_data_identifier;
};
```

```
struct OutCheckOutData {
        ServiceStatus   status;
        wstring check_out_identifier;
};

// Check In Data
struct InCheckInData {
        wstring check_out_identifier;
};
struct OutCheckInData {
        ServiceStatus   status;
};

// Pass Through Query
struct InPassThroughQuery {
        wstring data_source_name;
        wstring query_string;
};
struct OutPassThroughQuery {
        ServiceStatus   status;
        wstring query_output;
};

// Translate Data
struct InTraslateData {
        wstring translation_name;
        wstring input_schema;
        wstring output_schema;
        wstring output_location;
};
struct OutTranslateData {
        ServiceStatus   status;
};

// Transfer Intermediate Data
struct InTransferIntermediateData {
        wstring target_software_program_name;
        wstring data_item;
};
struct OutTransferIntermediateData {
        ServiceStatus   status;
};

// Invoke Software Program Capability
struct InInvokeSoftwareProgramCapability {
        wstring SP_capability_name;
        ArgumentList    SP_capability_input_argument_list;
};
struct OutInvokeSoftwareProgramCapability {
        ServiceStatus   status;
        ArgumentList    SP_capability_output_argument_list;
};

// Start Manufacturing Software Program
struct InStartManufacturingSoftwareProgram {
        wstring SP_start_up_script;
};
struct OutStartManufacturingSoftwareProgram {
        ServiceStatus   status;
};
```

```
// Start Task List
struct InStartTaskList {
        wstring task_list_id;
        wstring task_id;
};
struct OutStartTaskList {
        ServiceStatus   status;
};

// Terminate Task List
struct InTerminateTaskList {
        wstring task_list_id;
        wstring task_id;
};
struct OutTerminateTaskList {
        ServiceStatus   status;
};

// Suspend Task List
struct InSuspendTaskList {
        wstring task_list_id;
        wstring task_id;
};
struct OutSuspendTaskList {
        ServiceStatus   status;
};

// Suspend After
struct InSuspendAfter {
        wstring task_list_id;
        wstring task_id;
};
struct OutSuspendAfter {
        ServiceStatus   status;
};

// Resume Task List
struct InResumeTaskList {
        wstring task_list_id;
        wstring task_id;
};
struct OutResumeTaskList {
        ServiceStatus   status;
};

// StepMode
struct InStepMode {
        wstring task_list_id;
        wstring task_id;
};
struct OutStepMode {
        ServiceStatus   status;
};

// StepTask
struct InStepTask {
        wstring task_list_id;
        wstring task_id;
};
```

```
struct OutStepTask {
        ServiceStatus   status;
};

// StepTo
struct InStepTo {
        wstring task_list_id;
        wstring task_id;
};
struct OutStepTo {
        ServiceStatus   status;
};

// OutOfStep
struct InOutOfStep {
        wstring task_list_id;
        wstring task_id;
};
struct OutOutOfStep {
        ServiceStatus   status;
};

// SkipN
struct InSkipN {
        wstring task_list_id;
        wstring task_id;
        wstring n;
};

struct OutSkipN {
        ServiceStatus   status;
};

// SkipTo
struct InSkipTo {
        wstring task_list_id;
        wstring task_id;
};
struct OutSkipTo {
        ServiceStatus   status;
};

// Show Task List
struct InShowTaskList {
        wstring task_list_id;
        wstring output_destination;
};
struct OutShowTaskList {
        ServiceStatus   status;
        ValueList       task_list_information;
};

// Delete Task List
struct InDeleteTaskList {
        wstring task_list_id;
};
struct OutDeleteTaskList {
        ServiceStatus   status;
};
```

```
// Create Task List
struct InCreateTaskList {
        wstring task_list_id;
};
struct OutCreateTaskList {
        ServiceStatus   status;
};

// Start Trace
struct InStartTrace {
        wstring task_list_id;
};
struct OutStartTrace {
        ServiceStatus   status;
};

// Stop Trace
struct InStopTrace {
        wstring task_list_id;
};
struct OutStopTrace {
        ServiceStatus   status;
};

// Check Task List Status
struct InCheckTaskListStatus {
        wstring task_list_id;
};
struct OutCheckTaskListStatus {
        ServiceStatus   status;
        wstring         task_id;
};

// Initialize MAPLE
struct InInitializeMAPLE {
        wstring MAPLE_name;
};
struct OutInitializeMAPLE {
        ServiceStatus   status;
};

// Shut Down MAPLE
struct InShutDownMAPLE {
        wstring MAPLE_name;
};
struct OutShutDownMAPLE {
        ServiceStatus   status;
};

// Administrate System
struct InAdministrateSystem {
        wstring MAPLE_name;
};
struct OutAdministrateSystem {
        ServiceStatus   status;
};

// Communicate MAPLE to MAPLE
struct InCommunicateMAPLEToMAPLE {
        wstring other_MAPLE_name;
};
```

```
struct OutCommunicateMAPLEToMAPLE {
        ServiceStatus   status;
};

// Access MAPLE
struct InAccessMAPLE {
        wstring MAPLE_name;
};
struct OutAccessMAPLE {
        ServiceStatus   status;
};

// Service Name
enum ServiceName {
        plan_task, register_data_source, list_data_source,
        deregister_data_source, modify_data_source,
        register_software_program_capability,
        list_software_program_capability,
        deregister_software_program_capability,
        modify_software_program_capability,
        register_catalogue_entry, list_catalogue,
        deregister_catalogue_entry, modify_catalogue_entry,
        insert_data, list_data, delete_data, update_data,
        retrieve_data, checkout_data, checkin_data,
        pass_through_query, translate_data, transfer_intermediate_data,
        invoke_software_program_capability, start_manufacturing_software_program,
        start_task_list, terminate_task_list, suspend_task_list,
        suspend_after, resume_task_list, step_mode, step_task, step_to,
        out_of_step, skip_n, skip_to, show_task_list, delete_task_list,
        create_task_list, start_trace, stop_trace, check_task_list_status,
        initialize_MAPLE, shut_down_MAPLE, administrate_system,
        communicate_MAPLE_to_MAPLE, access_MAPLE, other_services
};

// Service Parameter List

// Parameters for MAPLE Service Requests
union ServiceReqParameterList switch(ServiceName) {
        case plan_task:
                InPlanTask plan_task;
        case register_data_source:
                InRegisterDataSource register_data_source;
        case list_data_source:
                InListDataSource list_data_source;
        case deregister_data_source:
                InDeregisterDataSource deregister_data_source;
        case modify_data_source:
                InModifyDataSource modify_data_source;
        case register_software_program_capability:
                InRegisterSoftwareProgramCapability register_software_program_capability;
        case list_software_program_capability:
                InListSoftwareProgramCapability list_software_program_capability;
        case deregister_software_program_capability:
                InDeregisterSoftwareProgramCapability deregister_software_program_capability;
        case modify_software_program_capability:
                InModifySoftwareProgramCapability modify_software_program_capability;
        case register_catalogue_entry:
                InRegisterCatalogueEntry register_catalogue_entry;
        case list_catalogue:
                InListCatalogue list_catalogue;
```

```
case deregister_catalogue_entry:
        InDeregisterCatalogueEntry deregister_catalogue_entry;
case modify_catalogue_entry:
        InModifyCatalogueEntry modify_catalogue_entry;
case insert_data:
        InInsertData insert_data;
case list_data:
        InListData list_data;
case delete_data:
        InDeleteData delete_data;
case update_data:
        InUpdateData update_data;
case retrieve_data:
        InRetrieveData retrieve_data;
case checkout_data:
        InCheckOutData checkout_data;
case checkin_data:
        InCheckInData checkin_data;
case pass_through_query:
        InPassThroughQuery pass_through_query;
case translate_data:
        InTraslateData translate_data;
case transfer_intermediate_data:
        InTransferIntermediateData transfer_intermediate_data;
case invoke_software_program_capability:
        InInvokeSoftwareProgramCapability invoke_software_program_capability;
case start_manufacturing_software_program:
        InStartManufacturingSoftwareProgram start_manufacturing_software_program;
case start_task_list:
        InStartTaskList start_task_list;
case terminate_task_list:
        InTerminateTaskList terminate_task_list;
case suspend_task_list:
        InSuspendTaskList suspend_task_list;
case suspend_after:
        InSuspendAfter suspend_after;
case resume_task_list:
        InResumeTaskList resume_task_list;
case step_mode:
        InStepMode step_mode;
case step_task:
        InStepTask step_task;
case step_to:
        InStepTo step_to;
case out_of_step:
        InOutOfStep out_of_step;
case skip_n:
        InSkipN skip_n;
case skip_to:
        InSkipTo skip_to;
case show_task_list:
        InShowTaskList show_task_list;
case delete_task_list:
        InDeleteTaskList delete_task_list;
case create_task_list:
        InCreateTaskList create_task_list;
case start_trace:
        InStartTrace start_trace;
case stop_trace:
        InStopTrace stop_trace;
```