Laboratory Equipment Control Interface Specification

This OMG document replaces the draft adopted specification (dtc/01-11-06). It is an OMG Final Adopted Specification, which has been approved by the OMG board and technical plenaries, and is currently in the finalization phase. Comments on the content of this document are welcomed, and should be directed to *issues@omg.org* by June 7, 2002.

You may view the pending issues for this specification from the OMG revision issues web page *http://cgi.omg.org/issues*; however, at the time of this writing there were no pending issues.

The FTF Recommendation and Report for this specification will be published on July 5, 2002. If you are reading this after that date, please download the available specification from the OMG formal specifications web page.

OMG Final Adopted Specification
December 2001

Laboratory Equipment Control Interface Specification

Final Adopted Specification
December 2001

PATENT

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

NOTICE

The information contained in this document is subject to change without notice. The material in this document details an Object Management Group specification in accordance with the license and notices set forth on this page. This document does not represent a commitment to implement any portion of this specification in any company's products.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIEDWARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR PARTICULAR PURPOSE OR USE. In no event shall The Object Management Group or any of the companies listed above be liable for errors contained herein or for indirect, incidental, special, consequential, reliance or cover damages, including loss of profits, revenue, data or use, incurred by any user or any third party. The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents & Specifications, Report a Bug/Issue.

Contents

Listings

Preface

About the Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 800 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

What is CORBA?

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA 1.1 was introduced in 1991 by Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can interoperate.

Associated OMG Documents

The CORBA documentation is organized as follows:

- *Object Management Architecture Guide* defines the OMG's technical objectives and terminology and describes the conceptual models upon which OMG standards are based. It defines the umbrella architecture for the OMG standards. It also provides information about the policies and procedures of OMG, such as how standards are proposed, evaluated, and accepted.
- *CORBA: Common Object Request Broker Architecture and Specification* contains the architecture and specifications for the Object Request Broker.
- *CORBA Services: Common Object Services Specification* contains specifications for OMG's Object Services.
- *CORBA Common Facilities*: contains services that many applications may share, but which are not as fundamental as the Object Services.
- CORBA domain specifications are comprised of stand-alone documents for each specification; however, they are listed under the domain headings, such as Telecoms, Finance, Med, etc.

OMG collects information for each book in the documentation set by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote.

To obtain OMG publications, contact the Object Management Group, Inc. at:

OMG Headquarters
250 First Avenue, Suite 201
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Helvetica bold - OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier bold` - Programming language elements.

Helvetica - Exceptions

If applicable, terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Acknowledgments

The following companies submitted and/or supported parts of this specification:

- Advanced Chemistry Development, Inc.
- CREON·LAB·CONTROL AG

Overview 1

1.1 Proof of Concept

The specification reflects experience with a prototype implementation of a device simulation in a CORBA-based environment. The resulting prototype implementation provides a proof of concept.

1.2 Problem Statement

The labor-intensive process of integrating different equipment into an automated system is a primary problem in laboratory automation today. Hardware and software standards are needed to facilitate equipment integration and thereby significantly reduce the cost and effort to develop fully automated laboratories. Recent market surveys showed that, in the pharmaceutical, biotechnology, and healthcare worlds, the trend in device control is towards component-oriented, distributed approaches.

The integration of different laboratory devices into a working automation environment has long been a difficult task. In a fully automated laboratory, integration issues become even more demanding. Too many proprietary device interfaces, protocols, and concepts are involved, and, worse yet, the vendors in this market have not been able to agree upon a common control standard. Such a standard would enable a user to configure his/her lab as a heterogeneous environment that is simpler to configure, easier to extend, and much less expensive to maintain.

LECIS (Laboratory Equipment Control Interface Specification, ASTM (American Society for Testing and Materials) Standard E1989-98) is an initial approach to solve these problems. LECIS defines a discrete state and communication model for laboratory devices.

Bringing LECIS concepts into the powerful CORBA environment promises to be a big step towards trouble-free laboratory device integration. The outcome of this specification should be easy to implement and use, as well as platform and programming language independent. Vendors and users should not be forced to adopt a new IT environment.

The RFP lifesci/00-06-09 also specifies the inclusion of a descriptive representation of device functionality and properties known as a Device Capability Dataset (DCD). The DCD concept allows a system controller to learn about a device on the fly—thereby making Plug and Play a reality. The DCD concept was initially defined at the National Institute of Standards and Technology (NIST) [6].

In an effort to address the lack of an open, object-oriented, component model-based standard for distributed instrument control, a request is made for a proposal defining a generic device control interface that will allow deterministic, hardware-independent remote control of laboratory equipment. This standard is intended to define the command and event communications between instrument controllers and instruments in a distributed environment.

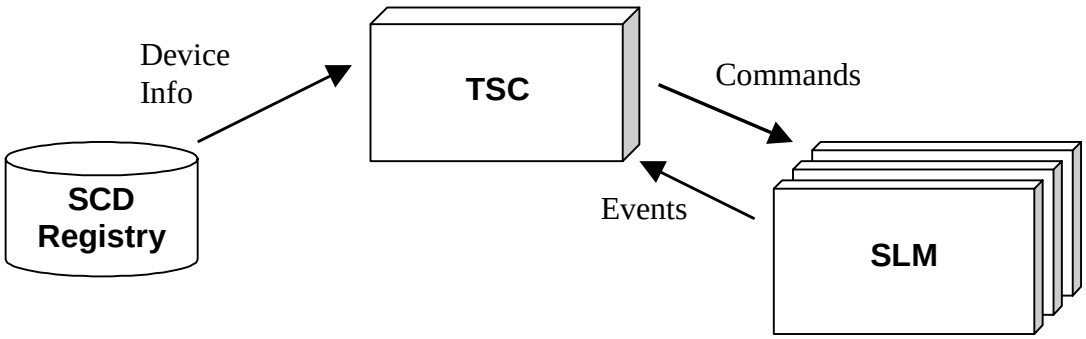

Figure 1-1 Basic Control Concept

Specific problem areas related to laboratory equipment control that would benefit from interface standardization are: LIMS (Laboratory Information Management System), laboratory equipment data acquisition, laboratory equipment scheduler, laboratory system capability dataset, laboratory equipment capability dataset, and generic error and event handling.

1.3 Scope

The scope of this proposal is limited to generic specification of control interfaces, events, state model, and device property definitions.

- Control interfaces allow for device independent control of laboratory equipment.
- Events are asynchronous responses to the requests sent to the device.
- A state model enables the controlling software to handle command requests to the device deterministically.

- Device properties include such things as its device type, its command set, etc.

The scope of this specification does not extend to the standardization of equipment software user interfaces and equipment result data standardization.

Focus has been placed on interface definition rather than on implementation details. The interfaces are applicable to a wide variety of different equipment types and allows for uniform control of smart as well as “dumb” laboratory equipment.

In 1998, the American Society for Testing and Materials (ASTM) issued the Standard E1989-98 [5], the first version of LECIS. ASTM LECIS is a string based message protocol for the communication between TSC and SLM. Some companies related to laboratory automation implemented ASTM LECIS for customer projects. The concept of ASTM LECIS has been proven to work. Nevertheless, some issues concerning the design have been raised in implementations of the ASTM LECIS standard.

Torsten Staab from the Los Alamos National Laboratories (LANL), who led the ASTM LECIS development, also initiated the development of CORBA LECIS. CORBA LECIS should move the existing ASTM specification to the powerful CORBA architecture and resolve the issues raised during the implementation of ASTM LECIS.

It was a requirement of the LECIS RFP (lifesci/00-06-09) to make only a minimum of changes to the original specification. There are two reasons for this.

- First, LECIS is already a well known name in the laboratory automation community. CORBA LECIS should use this background to provide a device interface standard that can be accepted by all parties involved.
- Second, existing implementations can be adopted without causing too many costs.

In the near future, a new ASTM working group might be formed to adopt the changes made in this specification to ASTM LECIS. This would finally provide a consistent standard, that would support different implementations of the same design concept, dependent on the requirements of the laboratory or equipment.

Conflicts between ASTM LECIS and CORBA LECIS implementations are not expected, because most people who where working with ASTM LECIS were also involved in the development of CORBA LECIS. For detailed information about all LECIS developments take a look at the LECIS website: http://www.lecis.org.

1.4 Related Documents and Standards

[1] Interoperable Naming Service, December 1999, Object Management Group,(http://www.omg.org/cgi-bin/doc?ptc/99-12-03)

[2] Security Service, December 1998, Object Management Group, (http://www.omg.org/cgi-bin/doc?formal/98-12-09)

[3] Time Service, June 2000, Object Management Group, (http://www.omg.org/cgi-bin/doc?formal/00-06-26)

[4] Minimum CORBA, August 1998, Object Management Group, (http://www.omg.org/cgi-bin/doc?orbos/98-08-04)

[5] E1989-98 Standard Specification for Laboratory Equipment Control Interface (LECIS), November 1998, American Society for Testing and Materials (http://www.astm.org , http://www.lecis.org) .

[6] Initial CAALS Device Capability Dataset (NISTIR 6294). Version 1.0.7., December 1998, National Institute of Standards and Technology (http://www.lecis.org)

[7] COM/CORBA Interworking Specification, October 1997, Object Management Group, (http://www.omg.org/cgi-bin/doc?formal/97-10-20)

1.5 Relationship to Existing OMG Specifications

In order to create the greatest possibility for extensibility and interoperability of the specified system with other laboratory and information resources within the target enterprise, the following CORBA specifications have been considered to be applied within this proposal:

- *Interoperable Naming Service* - The Interoperable Naming Service has been applied for registering a SLM within the laboratory system.
- *Event or Notification Service* - This specification does not make use of the Event or Notification Service, but employs other strategies for handling device communication, and system-system communication, which are believed to be more robust for this particular application.
- *Minimum CORBA* - This specification adheres to Minimum CORBA.

Specification 2

2.1 Introduction

The SLM or Standard Laboratory Module exists as a wrapper for each hardware laboratory device defined and in use by the system.

Using the DCD, the Task Sequence Controller (TSC) has detailed information about the commands supported by this SLM. The SLM is translating the TSC requests to physical instrument commands and sends status messages from the instrument to the TSC. The SLM acquires raw data from the device hardware and passes it to the TSC for storage and subsequent processing.

Figure 2-1 on page 2-1 provides an abstract view of an SLM. It consists of one Main-Unit with one or more Sub-Units. Each Sub-Unit has its own thread of control, which enables a Sub-Unit to execute a request independently of the other Sub-Units. The Sub-Units provide the actual device specific functionality. The Main-Unit primarily supervises the function of its Sub-Units. In addition it can execute SLM macro commands, which include atomic commands of more than one Sub-Unit.

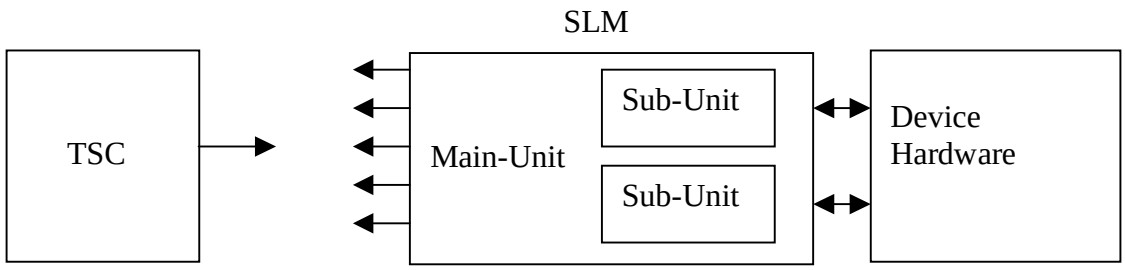

Figure 2-1 Abstract Laboratory Device View

The Main-Unit contains a state model for the global state management of the SLM. Each Sub-Unit contains a sub state model.

Main-Unit and Sub-Units provide a single CORBA interface. Main- and Sub-Units are addressed through unique identifiers.

How the SLM is connected to the device hardware is not part of this specification.

2.2 The Control State Models

2.2.1 Main-Unit State Model

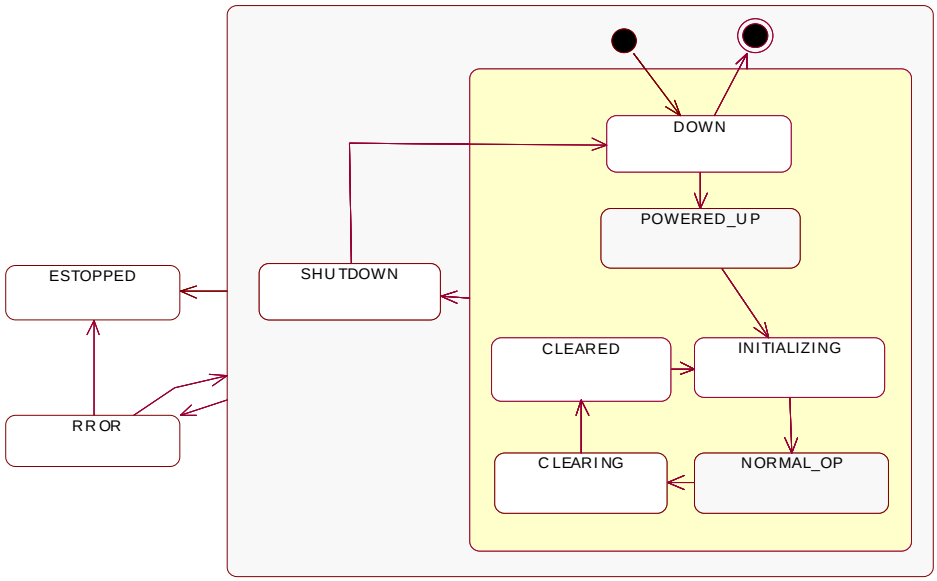

Figure 2-2 Main-Unit State Model

Listing 1: The IDL SLM Control States

```
enum EMainCtrlState {
    POWERED_UP,
    INITIALIZING,
    NORMAL_OP,
    ERROR,
    CLEARING,
    CLEARED,
    DOWN,
    SHUTDOWN,
    ESTOPPED
};
```

After turning on the SLM, it is in the state **DOWN**. This indicates that it is not ready to do anything useful.

First, it has to be powered up. The SLM is executing a device specific startup routine and is connecting itself to the network. The SLM is transferred into the state **POWERED_UP**. If the Main-Unit is powered up, all its Sub-Units are powered up as well. An SLM is powering up itself automatically. Is the power up routine failing the SLM is transferred back into the state **DOWN**.

If the SLM is powered up, it can be initialized. The initialization is initiated from the TSC. During the initialization the Main-Unit is in the state **INITIALIZING**. The TSC can provide the SLM with additional initialization parameters. If the Main-Unit is initialized, all Sub-Units are initialized with their default initialization parameters as well. Each Sub-Unit can be initialized separately, after the Main-Unit has been initialized.

After initializing, the Main-Unit is in the state **NORMAL_OP**. This is the normal operation mode. Only in this Main-Unit state, the Sub-Units are allowed to accept device specific command requests.

Clearing the Main-Unit is transferring the Main-Unit into the state **CLEARING**. Clearing the Main-Unit also clears all its Sub-Units, which are in the state **SUB_IDLE**, **SUB_ERROR**, or **SUB_ABORTED**. After clearing, the SLM is in the state **CLEARED** and has to be reinitialized to process new command requests from TSC. Clearing can be used to remove an internal configuration, or to resolve an error condition. This behavior is implementation-specific.

In an emergency condition, the Main-Unit and all its Sub-Units must terminate all actions immediately. The Main-Unit is transferred into the state **ESTOPPED**. This is a final state. An emergency stop (e-stop) cannot be resolved without turning the device offline. This is an essential requirement to prevent any damage to the user or the equipment.

To turn off the device under normal conditions, the user would have to shut down the device. The Main-Unit is transferred into the state **SHUTDOWN**. After shutting it down, the SLM is in the state **DOWN**, from which it can be powered up again or is switched off. This behavior is implementation specific.

2.2.2 Sub-Unit State Model

To ensure that each Sub-Unit behaves predictably and unambiguously, it is necessary to define a state model that is implemented by each Sub-Unit. (Figure 2-3).

Since each Sub-Unit has its own thread of control, a laboratory device is able to handle several command requests concurrently. For example, a Gas Chromatograph (GC) might have a heating unit (Sub-Unit 1) and a detector (Sub-Unit 2). While the heating unit is processing Sample 1, the detector can already process Sample 2. This ensures the most efficient sample throughput. The TSC has explicit control over each Sub-Unit and its functionality.

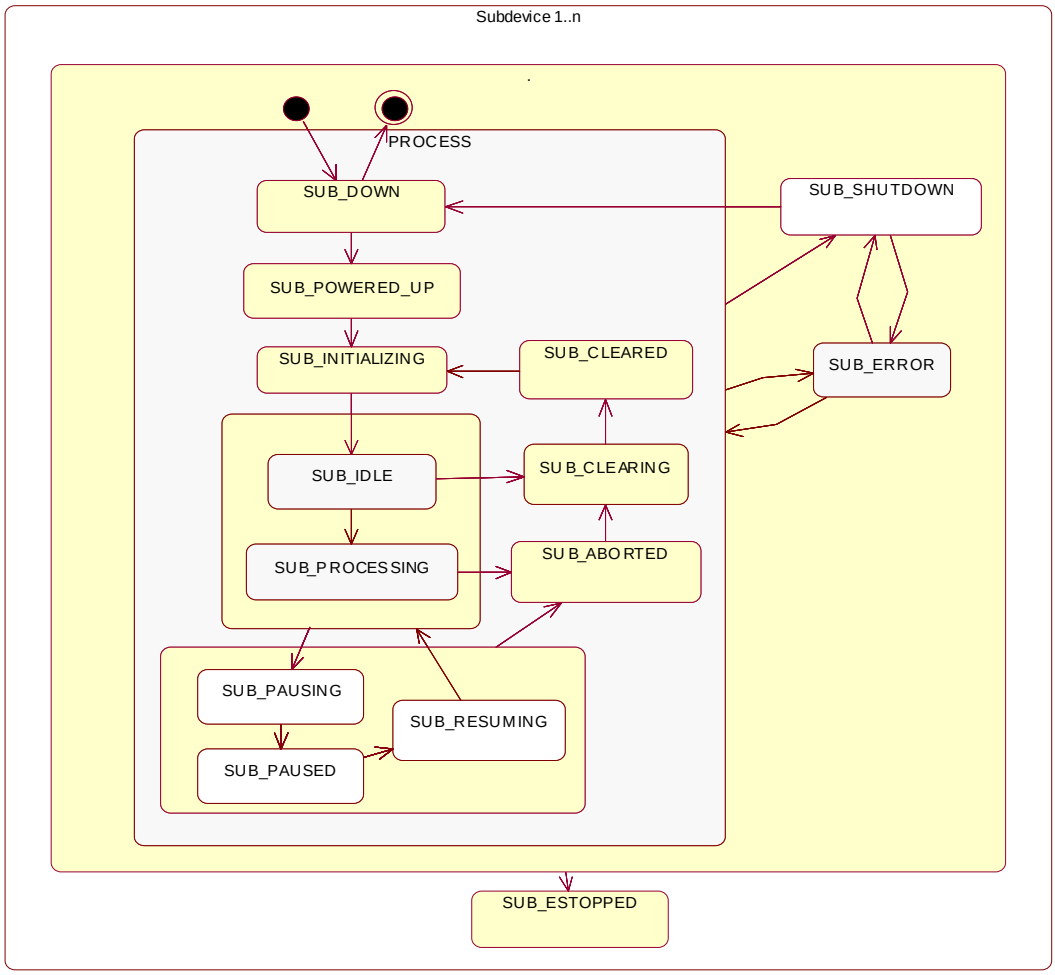

Figure 2-3 Sub-Unit State Model

The following list gives an overview of the Sub-Unit control states and their function:

- After turning on the Sub-Unit, it is in the state **SUB_DOWN**.
- A Sub-Unit is powered up, when the Main-Unit is powered up. A Sub-Unit can power up itself independently of the Main-Unit, if the Main-Unit is already in the state **NORMAL_OP**. After powering up the Sub-Unit, it is in the state **SUB_POWERED_UP**.
- After a Sub-Unit has been powered up it has to be initialized. Sub-Units can be initialized individually, but only in the Main-Unit state **NORMAL_OP**. If the Main-Unit is initialized, all Sub-Units, that are in the state **SUB_POWERED_UP, SUB_ERROR**, or **SUB_CLEARED** are initialized as well. During the initialization, the Sub-Unit is in the state **SUB_INITIALZING**.

- After the initialization, the Sub-Unit is in the state **SUB_IDLE**. In this state the Sub-Unit is waiting for a device specific command request (e.g., a sample measurement) from the TSC. Such a call is made through the **run_op()** operation of the SLM CORBA interface (Listing 22). To accept such a command request the Main-Unit must be in the state **NORMAL_OP**. If the Main-Unit is in any other state, the command request is denied. During the execution of a device specific command request, the Sub-Unit is in the state **SUB_PROCESSING**.
- Each Sub-Unit operation can be aborted independently. An operation can be aborted, if it is currently processed or paused. The Sub-Unit is transferred into the state **SUB_ABORTED**. To accept a new command request the Sub-Unit must be cleared and initialized first.
- If an error occurs during an command execution, the Sub-Unit is transferred into the state **SUB_ERROR**. The Sub-Unit should be cleared and initialized before it can accept a new command request. Can the error be resolved without clearing and initializing, the last operation can be completed or the Sub-Unit is transferred directly in the state **SUB_IDLE**. This behavior is implementation specific. An error in one Sub-Unit does not necessarily affect other Sub-Units. (Important runtime information for scheduler and error management).
- In an emergency condition, the Sub-Unit must terminate all actions immediately and is transferred into the state **SUB_ESTOPPED**. In this case the Main-Unit and all other Sub-Units are stopped as well. This is a final state and cannot be resolved without turning the device offline.
- A shutdown of the complete SLM is possible only, if no Sub-Unit operation is running. A Sub-Unit can be shut down individually. Whether a shut down request is granted or denied is implementation specific. During the shut down, the Sub-Unit is in the state **SUB_SHUTDOWN**.
- Pausing an action might be useful for some devices, whereas for others it is not applicable. An example is a shaking device with multiple sample ports. While the device is shaking, a new sample could be added to a free port. Therefore it would be advantageous to pause the current shaking operation, add the sample, and resume the shaking operation. Other operations may not be paused, because they would destroy the sample. For this reason, whether a device can grant a pause request or not is implementation specific. If a pause request is granted, the device pauses the current operation and is then in the state **SUB_PAUSED**. Otherwise the device returns the result code **PAUSE_REQUEST_DENIED**.
- After calling the **resume()** operation of the Sub-Unit, the device is in the same state it was in before pausing the operation. While the last operation is resumed the SUB-Unit is in the state **SUB_RESUMING**.
- If the SLM is executing a SLM-based macro-operation, the Sub-Units of this SLM cannot process other command requests during this time. Any other command request through the **run_op()** operation during this time is denied. All Sub-Units are transferred into the state **SUB_PROCESSING** during the execution of a SLM based macro command.

- The operations of one Sub-Unit should not run concurrently. If the Sub-Unit is in the state **SUB_PROCESSING**, the TSC must wait until the SLM is idle again to execute the following command. The operations of different Sub-Units may run concurrently. Exception: Each operation in the DCD contains an exclusion list of all SLM operations that should not run concurrently to this operation.

Listing 2: The IDL-Sub-Unit States

```
enum ESubCtrlState {
    SUB_DOWN,
    SUB_POWERED_UP,
    SUB_INITIALIZING,
    SUB_CLEARING,
    SUB_CLEARED,
    SUB_SHUTDOWN,
    SUB_ERROR,
    SUB_IDLE,
    SUB_PROCESSING,
    SUB_ABORTED,
    SUB_PAUSING,
    SUB_PAUSED,
    SUB_RESUMING,
    SUB_ESTOPPED
};
```

2.3 The CORBA IDL Interfaces

2.3.1 Result Values

Each SLM interface operation returns a structure of the type **SLM_RESULT** (Listing 3). The structure provides the following information.

- A **result_code** of the type **EResultCode** (Listing 4). This result code must always be provided.
- The **minor_code** is a device specific extension to the error code and may reference detailed information about the result.
- **SLM_RESULT** also contains the current state of the Main-Unit (**main_state**) as well as all Sub-Unit states (**sub_states**).
- **lr_mode** returns the current local/remote state of the SLM.
- Finally, the SLM may provide a human readable string **message**.

Listing 3: SML_RESULT

```
struct SLM_RESULT {
    EResultCode result_code;
    string minor_code;
    EMainCtrlState main_state;
```

```
    SLM_INTERFACE::SeqSubStates sub_states;
    ELocalRemote lr_mode;
    string message;
};
```

Listing 4: Result Code

```
enum EResultCode {
    SUCCESS,
    REMOTE_CTRL_REQ_DENIED,
    LOCAL_CTRL_REQ_DENIED,
    FORCE_LOCAL_CTRL_FAILED,
    RELEASE_REMOTE_CTRL_FAILED,
    READ_DCD_FAILED,
    WRITE_DCD_FAILED,
    DCD_NOT_AVAILABLE,
    SUBUNIT_UNKOWN,
    DEVICE_HARDWARE_ERROR,
    COMMUNICATION_ERROR,
    TIMEOUT,
    EXECUTE_MACRO,
    SUB_STATE_INCORRECT,
    MAIN_STATE_INCORRECT,
    PAUSE_REQUEST_DENIED,
    UNKNOWN_COMMAND,
    TIME_SYNCHRONIZATION_FAILED,
    TIME_SYNCHRONIZATION_NOT_AVAILABLE,
    WRONG_ARGUMENT_LIST,
    DATA_ID_UNKNOWN,
    INVALID_DATA,
    EXECUTING_MACRO,
    ACCESS_DENIED,
    UNSPECIFIED_ERROR,
    EXECUTION_STOPPED
};
```

2.3.2 The SLM Interface ILECI

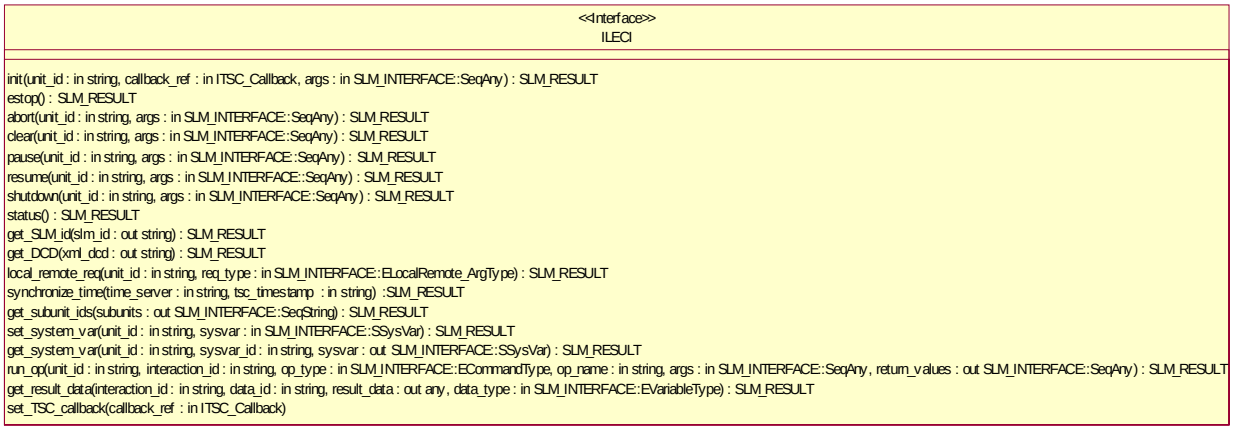

Figure 2-4 SLM Interface

Each SLM provides the CORBA control interface ILECI (Figure 2-4). This interface provides all means to control the SLM Main-Unit as well as all its Sub-Units.

The ILECI interface also provides operations for the remote/local control, time synchronization and the System Variable management.

2.3.3 Unit IDs

Most operations of the ILECI interface contain an argument **unit_id**. This value determines which unit of the SLM is addressed. If this value set to “0,” the Main-Unit is addressed, otherwise it is the corresponding Sub-unit. The Sub-Unit IDs are defined in the DCD or can be obtained via the interface operation **get_subunit_ids()** (Listing 10).

2.3.4 SLM Startup

After the device has been turned on, it is powering up itself and is registering its ILECI control interface in the CORBA Naming Service (see Section 2.6.1, “Registering the SLM Reference in the CORBA Naming Service,” on page 2-24. If the power up sequence is failing, the SLM is returning to the state **DOWN**, otherwise it is transferred into the state **POWERED_UP**.

The TSC has to resolve the reference to the SLM’s ILECI interface in the CORBA Naming Service before it can communicate with the SLM.

After the TSC has resolved the reference, it does not know the current state of the SLM. To get the SLM state, the SLM must know the valid reference to the TSC callback interface to distribute its events (see Section 2.6, “Device Responses,” on page 2-20 for details).

Thus, the TSC is calling the operation **set_TSC_callback()** (Figure 2-5) to provide the SLM with a reference to this interface. If the SLM is already online, the operation returns successfully. The **SLM_RESULT** structure returned with the call contains the complete state information of the SLM.

If the SLM is powered up but not initialized, the TSC calls the operation **init()** (Listing 13) of the ILECI interface to initialize the SLM. After initialization the SLM is ready to process SLM specific command requests with the **run_op()** (Listing 22) operation.

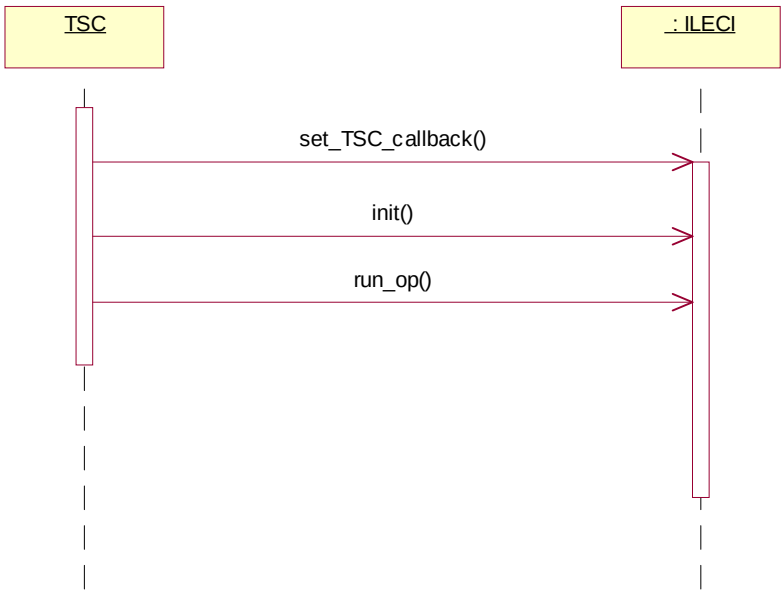

Figure 2-5 Startup

2.3.5 *Primary Commands*

The ILECI interface contains several primary commands that influence the LECIS control states.

Each SLM must provide the following primary commands:

- init
- clear
- abort
- estop
- pause
- resume
- shutdown

The primary commands are defined in the CORBA IDL of the ILECI interface.

To give the device manufacturer more freedom about the functionality provided with these commands, all commands except **estop()** (Listing 17) have an additional argument ***args***, which is of the type *sequence of any* (see example in Listing 5).

estop() does not provide arguments, because an emergency stop immediately stops the SLM completely.

Listing 5: Example for Primary Command

```
SLM_RESULT  clear (in string unit_id, in SeqAny args);
```

An argument contained in ***args*** can be of the following CORBA type: **long**, **float**, **boolean, string, octet, SLM_Interface::SeqLong, SLM_Interface::SeqFloat,** and **SLM_Interface::SeqOctet**.

All primary commands (except **estop()**) are defined in the DCD, to provide the TSC with the required information about the argument types and default values in the parameter **args**. In the DCD they are defined like device specific commands that are called through the **run_op()** operation of the ILECI interface, with the restriction, that only the variable arguments included in **args** are defined. The fixed arguments like the **unit_id** are not defined in the DCD.

2.3.6 Remote/Local Control

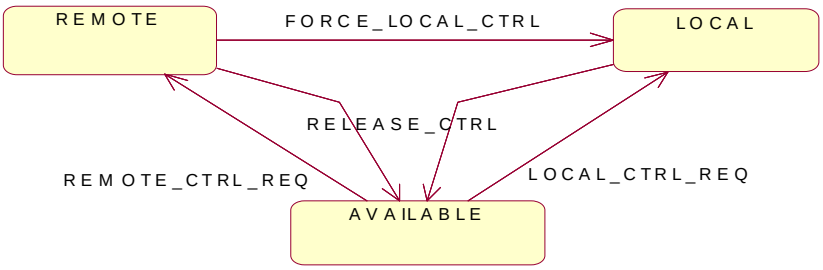

Figure 2-6 Remote/Local Control States

Listing 6: IDL Local-Remote States

```
enum ELocalRemote {
    LOCAL,
    REMOTE,
    AVAILABLE
};
```

If a user operates a device at the device panel (local control), the automated system should not try to use it concurrently.

Similarly, if the automated system is operating the device (remote control), the local user should not intervene, except if he/she has a good reason to do so.

Hence, before using the device it is necessary to check if the device is available either for local or remote control.

ILECI provides the operation **local_remote_req()** (Listing 8). The argument **req_type** of this operation specifies the type of the request. (Listing 7)

Listing 7: ELocalRemote_ArgType

```
enum ELocalRemote_ArgType {
    LOCAL_CTRL_REQ,
    REMOTE_CTRL_REQ,
    FORCE_LOCAL_CTRL,
    RELEASE_CONTROL
};
```

Listing 8: Local/Remote Control Request

```
SLM_RESULT local_remote_req ( ELocalRemote_ArgType req_type);
```

If the TSC sends a **REMOTE_CTRL_REQ** to the ILECI interface, the device may grant or deny the request, depending on the current control mode (Figure 2-6 on page 2-10). If the device is in the **AVAILABLE** state, or is already in the **REMOTE** state, the request is granted and the device's control mode is set to **REMOTE**. Now the TSC can use the device. If the device is in **LOCAL** use, the remote control request is denied with a result code of **REMOTE_CTRL_REQ_DENIED**.

To control the device from the device panel, the local user sends an implicit local control request (**ELocalRemote_ArgType = LOCAL_CTRL_REQ**) to the ILECI interface, as soon as he/she presses a button at the panel. If the SLM is in the **AVAILABLE** state, or is already in the **LOCAL** state, then the request is granted and the device's control mode is set to **LOCAL**. If the device is under **REMOTE** control, the local control request is denied with a result code of **LOCAL_CTRL_REQ_DENIED**.

If a device is not in use anymore, control should be released (**ELocalRemote_ArgType = RELEASE_CONTROL**), so that it can be used by another party (remotely or locally).

If a user has the immediate need to use the device locally, even though it is under remote control, he/she can force the device into local control (**ELocalRemote_ArgType = FORCE_LOCAL_CTRL**). For example, if the system control software crashes, or the current measurement is invalid, it is advantageous to be able to force the device into local control.

If the device is under local control, but has not been used in the last 10 minutes, the control is set back to the state **AVAILABLE** automatically. This timeout ensures that the SLM can be used, even if the local user leaves the SLM without returning the control mode to **AVAILABLE**.

Caution: The SLM does not check if a command request is actually remote or local. Therefore it is the responsibility of the client software to verify the control mode.

2.3.7 Time Synchronization

The SLM interface provides an operation to synchronize system and device time.

Time synchronization is done through an SNTP time server, or a simple timestamp.

To synchronize itself with a SNTP time server, the SLM requires the IP address of this server. This IP address could be provided in a configuration file, as well as with the ILECI operation **synchronize_time()** (Listing 9) of the SLM.

Calling this method would trigger the SLM to synchronize its internal clock with the given Server. If the SLM does not provide an SNTP client, but an internal system clock, it can synchronize itself with the **tsc_timestamp** (which is somewhat imprecise, but good enough for most purposes), which is generated from the TSC.

Format of tsc_timestamp: yyyymmddhhmmssmmm (year/ month/ day/ hour/ minutes/ seconds/ milliseconds)

Example: “20010615093023456”

Listing 9: Time Synchronization

```
SLM_RESULT synchronize_time (
    in string time_server,
    in string tsc_timestamp
);
```

If the SLM does not provide an internal system clock, it returns the result code **TIME_SYNCHRONIZATION_NOT_AVAILABLE**. If the SLM cannot connect to the SNTP time server, it returns the result code **TIME_SYNCHRONIZATION_FAILED**.

2.3.8 Sub-Unit Handling

After turning a laboratory device on, the Main-Unit may register an undetermined number of Sub-Units.

The main reason for multiple Sub-Units is to support a concurrent, but deterministic access to different features of the equipment.

Furthermore, it is possible to extend or restrict the functionality of the equipment by adding or removing Sub-Units.

Since Sub-Units do not provide their own CORBA interface, the Sub-Unit handling is completely up to the SLM manufacturer.

2.3.8.1 Registration Process is Implementation Specific

The specification does not define how Sub-Units are added and where the required information is stored. This is not possible, because the equipment that might implement OMG LECIS ranges from embedded controllers to fully featured workstations. It is up to the manufacturer to implement the Sub-Unit registration depending on the preconditions.

An embedded controller could store the information about the available Sub-Unit in the BIOS, or hardcode the information, whereas a workstation could store the information in a file or database. The same applies when a new Sub-Unit must somehow notify the Main-Unit of its availability. When the user plugs in an additional Sub-Unit (hardware or software) into the system, it would be nice to implicitly register the new Sub-Unit with the Main-Unit. Therefore the manufacturer is responsible for implementing this depending on the given software architecture.

2.3.8.2 Unique Identification of Sub-Units

Each Sub-Unit must be uniquely identified. To differentiate between a Main-Unit that has two Sub-Units of the same type, it is necessary that each Sub-Unit is assigned a unique identifier.

This unique identifier must be stored inside the device, e.g. in the BIOS or on hard-disc. In addition the manufacturer should print a label with the unique identifier on the device case.

2.3.8.3 Additional Operations for Sub-Unit Management

To determine which Sub-Units are already registered, the interface provides the operation **get_subunit_ids()**.

Listing 10: get_subunit_ids()

```
SLM_RESULT get_subunit_ids (out SeqString subunits);
```

It returns a sequence with all registered Sub-Unit IDs. Through this operation, the TSC can verify that all Sub-Units that are defined in the DCD are physically available.

2.3.9 System Variables

A System Variable is a property value of the SLM that can always be set or retrieved, regardless of other running operations (Listing 12).

Setting/getting a System Variable must not influence the LECIS control state of the SLM. System Variables can only be set when the SLM Main-Unit is in the control state **NORMAL_OP**.

Listing 11: System Variable

```
struct SSysVar {
    string variable_id;
    string description;
    string category;
    any value;
};
```

A System Variable includes a **variable_id**, a **description**, a device specific **category**, and a **value**.

The value can be of the following CORBA type: **long, float, boolean, string, octet, SLM_Interface::SeqLong, SLM_Interface::SeqFloat,** and **SLM_Interface::SeqOctet**.

The DCD should contain a definition of all existing System Variables for this SLM. This enables the controlling system to handle the System Variable values correctly.

Examples of System Variables are global measurement parameters and device specific property values, e.g. “Is shutter closed?” .

Listing 12: System Variables

```
SLM_RESULT get_system_var (
    in string unit_id,
    in string sysvar_id,
    out SLM_INTERFACE::SSysVar sysvar);

SLM_RESULT set_system_var (
    in string unit_id,
    in SLM_INTERFACE::SSysVar sysvar);
```

If the SLM is not in the state **NORMAL_OP** while calling **set_system_var()** the operation is returning with the result code **MAIN_STATE_INCORRECT**.

If the Sub-Unit is not able to set the System Variable at this time, the result code is **SUB_STATE_INCORRECT**.

System Variables must not substitute SLM **run_op()** operations and may not change the current SLM state. They should primarily be used to change/obtain device specific property values.

2.3.10 Control State Handling

All operations that influence the current control state of the SLM have an argument **unit_id**. If the value of this unit_id=”0”, the Main-Unit of the SLM is referenced. Otherwise the Sub-Unit with the specified ID **unit_id** is referenced. If the Sub-Unit does not exist the result code **SUBUNIT_UNKNOWN** is returned.

Listing 13: init()

```
SLM_RESULT init (
    in string unit_id,
    in SLM_INTERFACE::ITSC_Callback callback_ref,
    in SLM_INTERFACE::SeqAny args);
```

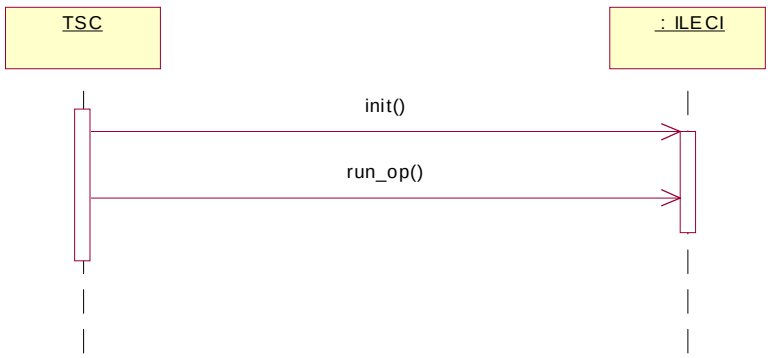

Figure 2-7 init()

After the SLM or a Sub-Unit is powered up or cleared, it may be initialized. Upon initialization the device is ready to accept command requests. Initializing the Main-Unit also initializes all Sub-Units. If a Sub-Unit is in a state that does not allow re-initializing, the Sub-Unit state is not changed and no error message is returned. The initialization of the other Sub-Units is not affected. The TSC can always obtain the current state of all sub-units. If a single Sub-Unit is reinitialized, but is in a state that does not allow this, a result code of **SUB_STATE_INCORRECT** is returned.

If the Main-Unit is reinitialized, but is in a state that does not allow this, a result code of **MAIN_STATE_INCORRECT** is returned.

During initialization the TSC is providing its callback interface to the SLM. The parameter **callback_ref** contains a reference to the **ITSC_Callback** interface.

Has the TSC interface changed while the SLM is already initialized, the TSC may call the operation **set_TSC_callback()** (Listing 14) to provide the SLM with its interface.

Listing 14: set_TSC_callback()

```
SLM_RESULT set_TSC_callback (
    in SLM_INTERFACE::ITSC_Callback callback_ref
    );
```

Listing 15: clear()

```
SLM_RESULT  clear (in string unit_id, in SeqAny args);
```

If the SLM or a Sub-Unit is in an error state, the last operation has been aborted, or an internal state must be reset, the SLM or Sub-Unit state can be cleared. After the clearing, the device must be reinitialized. Clearing the Main-Unit also clears all Sub-Units, if the Sub-Unit is in a state that allows clearing. Otherwise the Sub-Unit state is not changed, and no error message is returned. If a single Sub-Unit is cleared, but in a

state that does not allow this, a result code of **SUB_STATE_INCORRECT** is returned. If the Main-Unit is cleared, but in a state that does not allow this, a result code of **MAIN_STATE_INCORRECT** is returned.

Listing 16: abort()

```
SLM_RESULT  abort (in string unit_id,in SeqAny args);
```

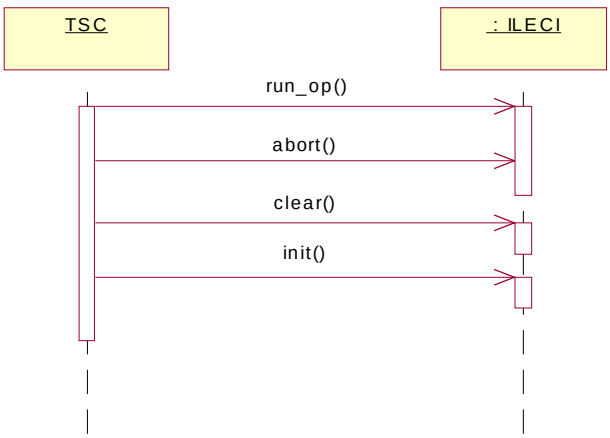

Figure 2-8 abort control flow

The operation **abort()** aborts the currently running operation of this Sub-Unit or all running SLM operations. Aborting the Main-Unit aborts all Sub-Units that are in the state **SUB_PROCESSING**. If the Sub-Unit is not in the state **SUB_PROCESSING**, the Sub-Unit state is not changed, and no error message is returned. If a single Sub-Unit is aborted, but not in the state **SUB_PROCESSING**, a result code of **SUB_STATE_INCORRECT** is returned. An operation that has been aborted returns with the result code **EXECUTION_STOPPED**. If the Main-Unit is aborted, but not in the state **NORMAL_OP**, a result code of **MAIN_STATE_INCORRECT** is returned.

Listing 17: estop()

```
SLM_RESULT  estop ();
```

estop() causes an immediate stop of all SLM actions. The SLM is transferred into the final state **ESTOPPED**. All Sub-Units are transferred into the final state **SUB_ESTOPPED**. An estop request cannot be denied.

An operation that has been e-stopped returns with the result code **EXECUTION_STOPPED**.

Listing 18: pause()

```
SLM_RESULT pause (in string unit_id, in SeqAny args);
```

pause() is a request to pause the currently running operation. The request can be granted or denied. If the request is denied **SLM_RESULT** returns the result code **PAUSE_REQUEST_DENIED**. If the Main-Unit is paused, all Sub-Unit operations that support pausing are paused. The operation does not provide an error message in the result code if a Sub-Unit does not pause the current operation, because the pausing ability is not a requirement. The TSC should check which Sub-Units have been paused after the pause request is returned. The Sub-Units, that have not been paused because of a running command execution are not paused after the processed command is finished. Paused operations do not return to the TSC.

Listing 19: resume()

```
SLM_RESULT  resume (in string unit_id,in SeqAny args);
```

resume() resumes an operation of a specified Sub-Unit that has been paused before. If the Main-Unit is addressed, all currently paused Sub-Units are resumed. A resume request cannot be denied.

Listing 20: shutdown()

```
SLM_RESULT shutdown (in string unit_id,in SeqAny args);
```

shutdown() shuts down the specified Sub-Unit or the complete SLM. After shutting down the SLM, it is in the state **DOWN** and is either turned off or powers up itself again. If a Sub-Unit is shut down, it is in the state **SUB_DOWN** and is either turned off or powers up itself again. This behavior is implementation specific.

If the complete device is to be shut down, no Sub-Unit may still process commands. If a Sub-Unit is executing a command, the shutdown request would fail with a **SUB_STATE_INCORRECT** result code.

To request the current SLM state, the operation **status()** is provided. It returns the structure **SLM_RESULT** that contains the complete state information of the SLM and its Sub-Units. (Listing 21).

Listing 21: status()

```
SLM_RESULT  status ();
```

2.3.11 Device Specific Functionality

The most important feature of an SLM is its ability to provide its device specific functionality to the system.

The TSC calls the operation **runOp()** (Listing 22) of the ILECI interface, if it would like to execute a device specific command request at the SLM.

Listing 22: runOp()

```
SLM_RESULT run_op (
    in string unit_id,
    in string interaction_id,
    in ECommandType op_type,
    in string op_name,
    in SeqAny args,
    out SeqAny return_values
);
```

All valid SLM specific commands are stored in the SCDRegistry (Listing 50). The TSC uses this information to call the operation **run_op()** with the correct arguments.

The **unit_id** determines which Sub-Unit should execute this command request.

The **interaction_id** is generated by the TSC, and is returned by the device if it provides asynchronous events to the TSC. With this **interaction_id**, the TSC can relate an SLM event to an operation it has called with this id. This is essential to enable result data to be correlated with the existing workflow.

If the command name is not supported by the SLM, a result code of **UNKNOWN_COMMAND** is returned. If the argument list is incorrect, a result code of **WRONG_ARGUMENT_LIST** is returned.

Today's devices may provide a tool to define command macros after the device has been installed in the laboratory. **run_op()** provides the parameter **op_type** of type **ECommandType** to differentiate between atomic commands and SLM based macros. This is especially useful for existing devices that do not support a CORBA LECIS interface and which need to be wrapped with a CORBA-layer. Since it is not possible to extend the CORBA interface each time a new macro is defined, **run_op()** can be called with an **op_type** of **MACRO**. Whether a command is a macro or an atomic command is defined in a device's DCD.

The **op_name** specifies the name of the operation to be called.

Input arguments can be specified in the parameter **args**. Output values are specified in the parameter **return_values**.

A value contained in **args** or **return_values** can be of the following CORBA type: **long, float, boolean, string, octet, SLM_Interface::SeqLong, SLM_Interface::SeqFloat,** and **SLM_Interface::SeqOctet**.

The event model of a CORBA LECIS device allows the sending of a data event that contains the data values or a link to the actual data. This link can refer to a file, a database or the ILECI interface operation **get_result_data()**. If the controller receives a link to this function, it can call **get_result_data()** and will receive a **CORBA::any** value containing the expected results. The data is identified by the **data_id** that has been received with a data event.

get_result_data() (Listing 23) provides a standard way to retrieve asynchronous result data from the device. The **interaction_id** specifies the operation, which has initiated the data collection. The **data_id** is a unique identifier, for this data packet. If

the measurement is distributed over more than one data packet, it can be linked with the **data_id**. The **data_id** is created by the SLM. The result data is of the type **any**. The included data type must be specified in the **data_type** enumeration. It can be one of the following CORBA types: **long, float, boolean, string, octet, SLM_Interface::SeqLong, SLM_Interface::SeqFloat,** and **SLM_Interface::SeqOctet**.

If the **data_id** provided is not known by the SLM, it returns with a result code of **DATA_ID_UNKNOWN**. If the data is not available at the SLM for any reason, the operation returns the result code **INVALID_DATA**.

Listing 23: get_result_data()

```
SLM_RESULT get_result_data (
    in string interaction_id,
    in string data_id,
    out any result_data,
    in SLM_INTERFACE::EVariableType data_type);
```

2.4 Relation between Main-Unit and Sub-Units

A Main-Unit and its Sub-Units do not act independently. The Sub-Units depend on the Main-Unit states and vice versa (e.g., if the Main-Unit is e-stopped, the Sub-Units are e-stopped as well).

2.4.1 The Dependencies of the Control States

The control states of the Sub-Unit always depend on the control states of the Main-Unit. In the following chapter, the states and their dependencies are defined:

If a Sub-Unit is executing a command request depends on its current state. If it is not in the correct control state, the Sub-Unit returns with a result code **SUB_STATE_INCORRECT**.

To execute a **run_op()** request, the Sub-Unit has to be in the **SUB_IDLE** state.

Only when the Main-Unit is in the state **NORMAL_OP** will the Sub-Unit accept a **run_op()** command request. If the Main-Unit is in any other state, the request is denied. In this case, **run_op()** returns with the result code **MAIN_STATE_INCORRECT**.

If a Sub-Unit is in the state **SUB_ESTOPPED**, the Main-Unit and all other Sub-Units are stopped as well. This is a final state and can be recovered only through turning the device offline. An emergency stop request cannot be denied.

If a global emergency stop is initiated (through the TSC or the local emergency stop), the complete device is stopped immediately.

If a Sub-Unit request is aborted (**SUB_ABORTED**), the current task is stopped. In such circumstances it cannot be guaranteed that the sample is not damaged.

If a Sub-Unit is running in an error condition (**SUB_ERROR**), the consequence is implementation specific, as follows:

- If other Sub-Units depend on this Sub-Unit, they are transferred into the state **SUB_ABORTED**. Thus the controller knows which Sub-Unit has caused the error. No new command requests are accepted at the affected Sub-Units until the error is resolved.
- If the Sub-Unit is running independently of all other Sub-Units nothing else happens. This Sub-Unit stays in the state **SUB_ERROR** until this error is cleared, or the device is shut down.

To shut down the complete device, no Sub-Units should be processing a command request at this time. If a Sub-Unit is still processing a command the shutdown request is denied with the result code **SUB_STATE_INCORRECT**.

2.5 Multithreaded Server Implementation

All IDL operations of the ILECI interface are defined synchronously: The TSC is blocking at a request until the SLM has finished its execution.

The TSC can invoke operations concurrently, e.g. it invokes an emergency stop while another operation is still running. The emergency stop request cannot wait until the device is finished with the other task. The device must stop all actions immediately.

These requirements lead to a multithreading architecture of the laboratory equipment interface. Because of its platform independence, the CORBA specification does not define how multithreaded servers must be implemented by an ORB vendor. It only distinguishes between a single threaded and ORB controlled model.

The ORB vendors are free to implement the ORB controlled model. Four thread models are commonly used: *Thread pool, Thread per Client, Thread per Object, and Thread per Request.*

The LECIS CORBA interfaces are required to support the *Thread per Request* model. This allows calling all operations of one interface concurrently, independent of the client that is making the request.

Thread per request can lead to memory, resource, and timing problems if too many threads are created at the same time. But this should not be an issue with laboratory equipment. In most cases there will not be more than maybe three or four requests simultaneously. In general the device is likely to execute only one request at a time.

Therefore, *Tread per Request* is the choice for the LECIS CORBA interfaces. This complicates the implementation of the SLM. Since every operation can be called at any time, it is the task of the SLM to synchronize the access to its resources.

2.6 Device Responses

One requirement of the LECIS RFP is an asynchronous messaging scheme from the SLM to the TSC. Several options have been evaluated.

The OMG provides two specifications for this requirement: the Event Service and Notification Service. The Notification Service is based on the architecture of the Event Service but provides additional features. Both services allow the creation of event channels with multiple event-suppliers and event-consumers.

Though the Notification Service provides a technically convincing solution, it implies a configuration and implementation overhead that makes it not applicable for this situation.

Therefore the TSC is providing the callback interface **ITSC_Callback** (Figure 2-9). The SLM is notifying the TSC through this interface. The TSC is responsible for distributing SLM events to the system.

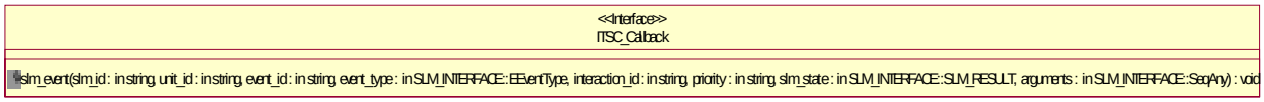

Figure 2-9 TSC Callback Interface

There are three reasons for an SLM to notify the controller:

- The device has changed an internal state.
- The device has changed a System Variable.
- The device is providing data to the system.

The TSC callback interface provides only the operation **slm_event()**. (Listing 24).

Listing 24: slm_event()

```
void slm_event (
    in string slm_id,
    in string unit_id,
    in string event_id,
    in SLM_INTERFACE::EEventType event_type,
    in string interaction_id,
    in string priority,
    in SLM_INTERFACE::SLM_RESULT slm_state,
    in SLM_INTERFACE::SeqAny arguments
);
```

An SLM event is identified through the **slm_id**, the **unit_id** and the **event_id**. The **slm_id** must always be set. The **unit_id** is “0” if the Main-Unit of the SLM was causing the event.

The device may provide additional event data through the function parameter e **arguments** a . The data types of the arguments depend on the **event_type** (Figure 2-10) and are defined in the DCD.

```
enum EEventType {
    ALARM,
    MESSAGE,
    DATA_DIRECT,
    DATA_LINK,
    SYSVAR_CHANGED,
    CONTROL_STATE_CHANGE,
    DEVICE_STATE_CHANGE
};
```

Figure 2-10 Event Type

The different event types in detail:

- The argument list must be implemented in the sequence as specified.

ALARM

The SLM is in an error condition and transfers the error code to the TSC. The argument list:

- A **string** with the error code
- A **string** with a user readable message.

MESSAGE

The device would like to send some user information to the TSC. The TSC would open a message box and display the SLM message.

The argument list:

- A **string** with a user readable message.

DATA_DIRECT

If the SLM is providing result data directly to the TSC, it transfers the data as a sequence of octets. The TSC interprets the octet stream with the help of the **dataformatID**. The **dataformatID** is a string value that is provided by the device manufacturer. The system data handler needs a mapping table (which is implementation specific and not part of this specification), to map this proprietary identifier to a format identifier known to the system. The **dataID** identifies this data stream or a sequence of data streams. This id is created by the SLM. The **seqCounter** determines the sequence of incoming data with the same **dataID**.

The argument list:

- A **Sequence of Octet** containing the data values.
- A string with the **dataformatID**
- A string with the **dataID**
- A long value with the sequence counter

DATA_LINK

If the device provides huge volumes of data, it might be reasonable to only provide the controller with a link to the actual data. The controller can then decide whether to access that data or not. The link type is either a link to a file, to the SLM operation **get_result_data()**, or a database. The link value contains the information needed to retrieve the data from the link target.

The argument list:

- A string with the data ID.
- A string with a sequence counter.
- A string with the dataformatID.
- An **EDataLinkType** defining the link type (Listing 25).
- A string with the link value.

Listing 25: EDataLinkType

```
enum EDataLinkType {
    FILE,
    DB,
    OPERATION
};
```

If the **linkType** is **OPERATION**, the **linkValue** is empty, because the SLM is providing the data with the operation **get_result_data()**.

If the **linkType** is **FILE**, the **linkValue** contains the UNC path to the file.

If the **linkType** is **DB**, the **linkValue** contains a string with the database URL (**DBURL**) and the database table name (**TABLENAME**).

The string must be generated like the following example:

```
DBURL=jdbc:oracle:thin:@192.9.444.99:1521:ABCD
TABLENAME=spectrum
```

It contains two lines. The first line is the standard URL to the database, and the second line specifies the table name.

SYS_VAR

The value of a SLM System Variable has changed. The argument list:

- A sequence (**SeqSysVar)** of System Variable structures **SSysVar**, that include the new data values.

CONTROL_STATE_CHANGE

A LECIS control state has changed. This event is sent only, if the state change is not caused by a TSC operation. Has the TSC caused the state change, the return value of this operation will already provide the SLM state to the TSC.

The argument list (**arguments**) of this event is empty because the current state is already transferred via the **SLM_RESULT** structure of the **event** operation.

DEVICE_STATE_CHANGE

An SLM specific state has changed. These events and their possible values are defined in the DCD. The argument list:

- A string with the device specific event id.
- A string with new event value.
- A string with the previous value.

All device specific SLM events and their event IDs should be defined in the DCD. Through the event ID, the TSC knows how to handle the event.

The **SLM_RESULT** argument of the operation **slm_event()** provides the TSC with the current SLM control state information.

The **interaction_id** is set, if the event is in reaction to a command request from the TSC. Each command request is providing an interaction identifier. This id is used to relate SLM events to the system workflow. Is the event not in reaction to a TSC command the **interaction_id** is set to "0."

The **priority** of the event handling. This can be value from 1-10. 1 is the highest priority.

2.6.1 Registering the SLM Reference in the CORBA Naming Service

An SLM must provide its object reference to the system. The TSC must know the SLM's object reference to call operations to the ILECI interface. Therefore the SLM registers its IOR in the CORBA Naming Service [1]. A system implementing OMG LECIS must provide an implementation of the Naming Service. The SLM's ORB must be configured with the Naming Service URL. How this is done is implementation specific.

Each time the SLM is powered up, it registers itself in the Naming Service with its unique ID. This ID must be equal to the SLM ID stored in the DCD. Then the TSC can look up the ID in the Naming service to connect to the correct SLM. The naming context, under which the SLM references are stored, is specified as **automation/LECIS/SLM**.

2.7 Typical Control Flow

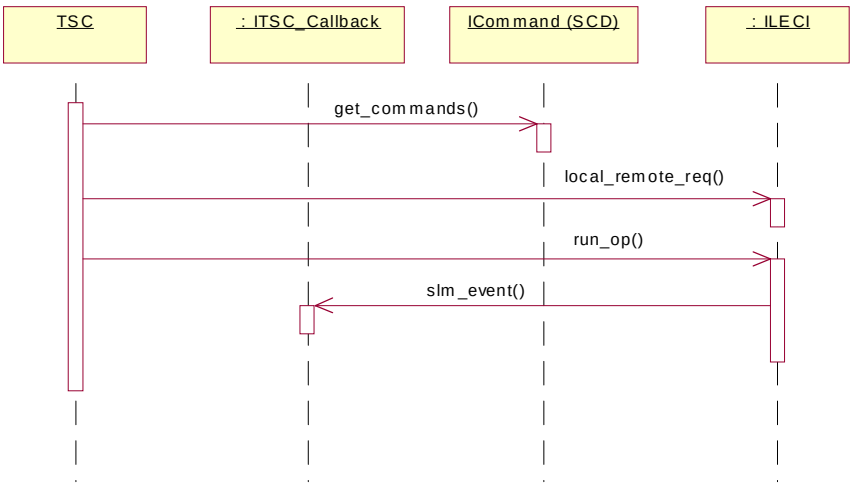

Figure 2-11 Typical Control Flow

The sequence diagram (Figure 2-11) shows a typical control flow between TSC, SCD, and SLM.

1. The TSC obtains the command information about the SLM from the SCD.
2. The TSC requests remote control for the SLM.
3. The TSC calls the **run_op()** operation of the SLM to execute a command request. The TSC uses the information it obtained from the SCD.

If the device is sending an event to the TSC. For example, providing asynchronous data values, the SLM calls the **ITSC_Callback** interface operation **slm_event()**.

SCD and DCD 3

3.1 Overview

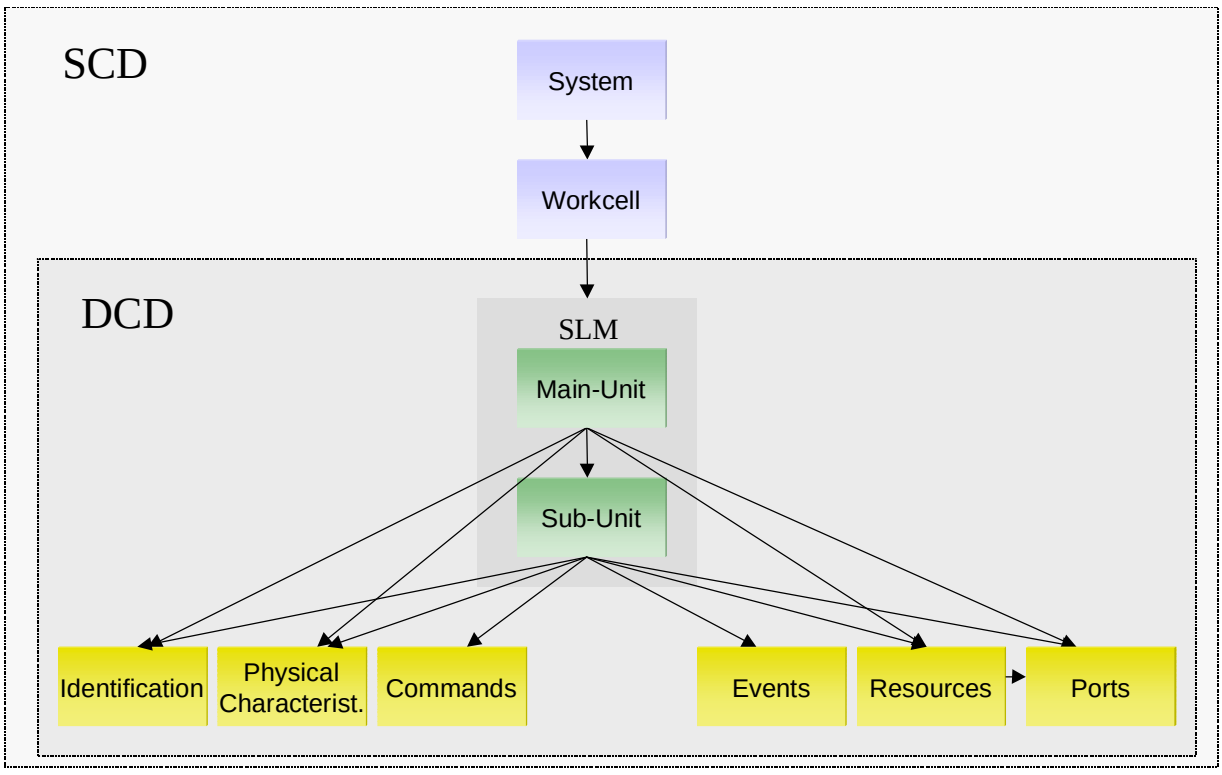

Figure 3-1 DCD Structure

The System Capability Dataset (SCD) is a descriptive representation of the laboratory environment. It describes the system, work cells as well as all SLMs on these work cells. Figure 3-1 gives an (incomplete) overview of the SCD structure.

The Device Capability Dataset (DCD) is a subset of the SCD. A DCD provides the system controller with all information it needs to control a single SLM. A DCD contains the equipment's identification, physical dimensions, supported command set, generated events, status and error representations, resources, and input-output ports.

The DCD definition in this document is based on the CAALS DCD specification, developed at NIST [6].

A device can be installed in an automated system by physically connecting it to the system and incorporating its device information into the system database. It can eliminate custom programming for the dissimilar interfaces found on many devices and facilitate integrating different types of devices into laboratory automation systems. It enables centralized control and error handling. Because the concept does not restrict the functionality of devices, the DCD still permits product differentiation among manufacturers.

To construct an SCD/DCD that is easily reusable from system to system without significant reconstruction, we must describe the device characteristics as abstract as possible. The representational scheme we choose should be independent of the device being described, and the device description must be completely independent from the implementation or the platform used. Once created in this way, device information can be used as long as the device itself is used. Even when the implementation changes, the device description is not lost. To achieve the implementation independence of the capability dataset we need a neutral mechanism that is capable of describing these data.

As a result the SCD and DCD are defined in XML (eXtensible Markup Language). XML is also the exchange format for DCDs. The device manufacturer would provide the DCD in an XML file. The XML-DCDs can then be mapped into implementation specific databases for advanced data handling options.

One DCD always contains only one SLM with its Sub-Units. An SCD may contain several DCDs.

3.2 DCD Interface Operations

The SLM interface **ILECI** supports an operation to provide the XML-DCD to the TSC. This is the standard DCD provided by the manufacturer.

Listing 26: Get DCD

```
SLM_RESULT get_DCD (out string xml_dcd );
```

If an SLM does not provide a DCD, then the operation returns the result code **DCD_NOT_AVAILABLE**.

If only a Sub-Unit does not provide its part of the DCD, the operation delivers the existing DCD parts to the TSC, but returns with the result code **DCD_NOT_AVAILABLE** and a minor code with a comma separated list of all Sub-Units that did not provide their part of the DCD.

3.3 Device ID Mapping

It is difficult for a device manufacturer to provide an XML-DCD for each *device instance* with the correct SLM IDs already included. It would be easier for the manufacturer to provide one XML-DCD for each *device type* that is shipped.

Thus, the system administrator has to replace the generic IDs with the unique SLM IDs. All IDs in the SCD that must be globally unique are marked with the label (**unique**).

3.4 Why WC3 Schema representation?

XML Schemas provide a means for defining the structure, content and semantics of XML documents.

Schemas provide functionality beyond that which is provided by Document Type Definitions (DTDs).

DTDs can be used to define content models and, to a limited extent, the data types of attributes, but they have a number of limitations:

- They are written in a different (non-XML) syntax.
- They have no support for namespaces.
- They only offer extremely limited data typing.
- DTDs have no ability to express the data type of character data in elements.

XML Schemas overcome these limitations and are much more expressive than DTDs.

3.5 XML Definition

XML is the exchange format for SCDs and DCDs. They can be mapped into an implementation specific database, which is used by the automated system.

During the mapping into the database, the correct IDs are assigned to the DCD entities, because the manufacturer delivers a generic XML-file for all devices of the same type.

From the database an XML-DCD file can be reconstructed to exchange updated device information.

SCD and DCD use the same XML Schema file.

3.5.1 SCD Definition

The SCD is only a superset of the DCD. It extends the Device Capability Dataset with the work cell and the system definition. Each SCD may contain only one system definition.

The SCD and the DCD share several type definitions. This chapter shows the types that are defined only in the SCD. All shared types are defined in the following DCD specification.

IDs that must be unique in the SCD are labeled **"(unique)".** All other IDs are only unique within the scope of their parent entity.

The root of the SCD is the XML-element **SCD**.

To create an SCD XML-data file, the SCD element is selected in the XML-editor as root element.

Listing 27: SCD root element

```
<xsd:element name="SCD">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="SYSTEM" type="SYSTEM_TYPE"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
```

An SCD has exactly one SYSTEM of type SYSTEM_TYPE.

Listing 28: SCD SYSTEM_TYPE

```
<xsd:complexType name="SYSTEM_TYPE">
    <xsd:sequence>
        <xsd:element name="NAME" type="xsd:string"/>
        <xsd:element name="LOCATION" type="xsd:string"/>
        <xsd:element name="WORKCELLS" type="WORKCELL_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="RESOURCES" type="RESOURCE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="DOMAIN" type="ESYSTEM_DOMAIN"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

The NAME is a meaningful string for the system.

The LOCATION specifies where the system is located.

A system can be assigned to a DOMAIN.

There is only one system per SCD. A system may contain a number of WORKCELLS and system wide RESOURCES, for example, samples.

The DESCRIPTION is a user readable description of the system.

Listing 29: SCD WORKCELL_TYPE

```
<xsd:complexType name="WORKCELL_TYPE">
        <xsd:sequence>
            <xsd:element name="WORKCELL_ID" type="xsd:ID"/>
            <xsd:element name="LOCATION" type="xsd:string"/>
            <xsd:element name="SLMS" type="SLM_TYPE" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="RESOURCES" type="RESOURCE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="SUBCELLS" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="SUPERCELL" type="xsd:IDREF" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
            <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
```

```
        </xsd:sequence>
    </xsd:complexType>
```

The WORKCELL_ID is a unique identifier for the work cell. **(unique)**

The LOCATION describes the location of the work cell as a user readable string.

A WORKCELL_TYPE has SLMs and RESOURCES.

Furthermore, a work cell has a physical dimension and location (PHYSICAL_CHARACTERISTICS).

Listing 30: SCD ESYSTEM_DOMAIN

```
<xsd:simpleType name="ESYSTEM_DOMAIN">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="COUNTRY"/>
            <xsd:enumeration value="DEPARTMENT"/>
            <xsd:enumeration value="SUBDIVISION"/>
            <xsd:enumeration value="LABORATORY"/>
            <xsd:enumeration value="ROOM"/>
        </xsd:restriction>
    </xsd:simpleType>
```

ESYSTEM_DOMAIN defines with which domain this system is associated.

- COUNTRY: The system includes several sites in the world/country.
- DEPARTMENT: The system belongs into one department.
- SUBDIVISION: The system belongs only to one sub-division.
- LABORATORY: The system specifies only parts of one laboratory.
- ROOM: The system is restricted to one room.

3.5.2 DCD Definition

The root of the DCD is the XML-element **DCD**.

To create a DCD XML-data file, the DCD element is selected in the XML-editor as root element.

Listing 31: DCD root element

```
<xsd:element name="DCD">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="SLM" type="SLM_TYPE"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
```

Each DCD may contain only one SLM of type SLM_TYPE, whereas an SCD can contain many SLMs.

Listing 32: DCD - SLM_TYPE

```
<xsd:complexType name="SLM_TYPE">
    <xsd:sequence>
        <xsd:element name="SLM_ID" type="xsd:ID"/>
        <xsd:element name="ADMINISTRATIVE" type="ADMINISTRATIVE_TYPE"/>
        <xsd:element name="FUNCTIONALITY" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
        <xsd:element name="SUBUNITS" type="SUBUNIT_TYPE" maxOccurs="unbounded"/>
        <xsd:element name="RESOURCES" type="RESOURCE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PORTS" type="PORT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PRIMARY_COMMANDS" type="COMMAND_TYPE"
         maxOccurs="unbounded"/>
        <xsd:element name="EXT_MACROS" type="EXT_MACRO_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="DOWNTIME" type="DOWNTIME_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="SYSTEM_VARIABLES" type="SYSTEM_VARIABLE_TYPE" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="EVENTS" type="EVENT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
```

The SLM_TYPE contains all the elements that identify a Main-Device entity.

The SLM_ID is the unique identifier for this device. **(unique)**

The ADMINISTRATIVE contains additional information about the device itself, the manufacturer, etc.

The FUNCTIONALITY is a list of keywords that describe the functionality of this device. For example a balance could contain the keyword “weigh”. The keywords are not defined in this specification.

The PHYSICAL_CHARACTERISTICS describe the dimension, location and weight of the device.

An SLM has a number of SUBUNITS.

PROPERTIES are a user definable list of item-value pairs to specify additional information not defined in this specification.

The RESOURCE list defines all resources, e.g. a well or a reagent, that belong to this device.

EXT_MACROS is a list of pre-configured commands of this device. This enables the manufacturer to create complex procedures for this device in advance.

An SLM may have a number of PORTS.

The PRIMARY_COMMANDs init, estop, clear, pause, resume, abort and shutdown are specified with their argument list.

DOWNTIME information is necessary to plan scheduled downtimes of the SLM.

SYSTEM_VARIABLES define device specific values.

All device EVENTS must be defined in the DCD.

The SLM must have at least one Sub-Unit. The Sub-Units provide the functionality of the equipment

Listing 33: DCD - SUBUNIT_TYPE

```
<xsd:complexType name="SUBUNIT_TYPE">
    <xsd:sequence>
        <xsd:element name="UNIT_ID" type="xsd:ID"/>
        <xsd:element name="ADMINISTRATIVE" type="ADMINISTRATIVE_TYPE"/>
        <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
        <xsd:element name="COMMANDS" type="COMMAND_TYPE" maxOccurs="unbounded"/>
        <xsd:element name="PRIMARY_COMMANDS" type="COMMAND_TYPE"
        maxOccurs="unbounded"/>
        <xsd:element name="EXT_MACROS" type="EXT_MACRO_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="FUNCTIONALITY" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="RESOURCES" type="RESOURCE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PORTS" type="PORT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="SYSTEM_VARIABLES" type="SYSTEM_VARIABLE_TYPE" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="EVENTS" type="EVENT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="DOWNTIME" type="DOWNTIME_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
```

A SUBUNIT_TYPE describes a Sub-Unit entity.

The UNIT_ID is the unique identifier for this Sub-Unit. **(unique)**

The UNIT_ID should be unique, because it could be possible to move a Sub-Unit from one SLM to another.

The ADMINISTRATIVE contains additional information about the device itself, the manufacturer etc.

A COMMAND defines the functionality of the device. This command definition is used by the system controller to manage the device through the CORBA interface.

The FUNCTIONALITY is a list of keywords that describe the functionality of this device. For example a balance could contain the keyword “weigh”. The keywords are not defined in this specification.

The PHYSICAL_CHARACTERISTICS describe the dimension, location and weight of the device.

The PROPERTIES is a user definable list of item-value pairs to define additional information not defined in this specification.

The RESOURCE list defines all resources, e.g. a well or a reagent, that belongs to this device.

EXT_MACROS is a list of pre-configured commands of this device. This enables the manufacturer to create complex procedures for this device in advance.

An SLM may have a number of PORTS.

DOWNTIME information is necessary to plan downtimes of the SLM.

SYSTEM_VARIABLES define device specific values.

All device EVENTS must be defined in the DCD.

The XML-schema contains a number of enumeration types.

ECOMMAND_CATEGORY categorizes a command. This eases the command handling. Only commands of the user-defined interface are categorized. The categories in detail:

- INIT: A command that is initializing parts of the device.
- CONTROL: A command that influences a device control state.
- FUNCTION: A command that specifies the functionality of the device, e.g. a measurement.
- CONFIGURE: A device configuration command, e.g. a robot changes its hand.
- RECOVERY: A command to recover from a specific error or malfunction.
- STATUSREQ: A status request to the device.
- MAINTAIN: A command for maintenance purposes.
- CALIBRATE: A command to calibrate the device.
- ADMIN: A command for device administration tasks.
- RESULT: A command that provides result data.

Listing 34: DCD - ECOMMAND_CATEGORY

```
<xsd:simpleType name="ECOMMAND_CATEGORY">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="INIT"/>
        <xsd:enumeration value="CONTROL"/>
        <xsd:enumeration value="FUNCTION"/>
        <xsd:enumeration value="CONFIGURE"/>
        <xsd:enumeration value="RECOVERY"/>
        <xsd:enumeration value="STATUSREQ"/>
        <xsd:enumeration value="MAINTAIN"/>
        <xsd:enumeration value="CALIBRATE"/>
        <xsd:enumeration value="ADMIN"/>
        <xsd:enumeration value="RESULT"/>
    </xsd:restriction>
</xsd:simpleType>
```

ECOMMAND_TYPE defines whether a command is an atomic command or an SLM based macro command.

Listing 35: ECOMMAND_TYPE

```
<xsd:simpleType name="ECOMMAND_TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ATOMIC"/>
        <xsd:enumeration value="MACRO"/>
    </xsd:restriction>
</xsd:simpleType>
```

EEVENT_CATEGORY defines the device event categories.

- ALARM: An error condition of the SLM.
- MESSAGE: The SLM is providing a user message.
- DATA_DIRECT: the SLM provides the result data within the data event.
- DATA_LINK: The SLM provides only a link to the data within the event.
- SYSVAR_CHANGED: A System Variable value has changed.
- CONTROL_STATE_CHANGED: A LECIS control state of the SLM has changed.
- SLM_STATE_CHANGED: An SLM specific state has changed.

Listing 36: DCD EEVENT_CATEGORY

```
<xsd:simpleType name="EEVENT_CATEGORY">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ALARM"/>
            <xsd:enumeration value="MESSAGE"/>
            <xsd:enumeration value="DATA_DIRECT"/>
            <xsd:enumeration value="DATA_LINK"/>
            <xsd:enumeration value="SYSVAR_CHANGED"/>
            <xsd:enumeration value="CONTROL_STATE_CHANGED"/>
            <xsd:enumeration value="SLM_STATE_CHANGED"/>
        </xsd:restriction>
    </xsd:simpleType>
```

ENUMBER_TYPE specifies if a number is of the type “long” or of the type “float.”

Listing 37: DCD - ENUMBER_TYPE

```
<xsd:simpleType name="ENUMBER_TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="LONG_NTYPE"/>
            <xsd:enumeration value="FLOAT_NTYPE "/>
        </xsd:restriction>
    </xsd:simpleType>
```

EDOWNTIME_CATEGORY specifies the reason for a scheduled downtime.

- CLEANING: The SLM is to be cleaned.
- CALIBRATION: The SLM is calibrated.
- SOFTWARE_UPDATE: The SLM software is updated.
- HARDWARE_UPDATE: The SLM hardware is updated.

Listing 38: DCD - EDOWNTIME_CATEGORY

```
<xsd:simpleType name="EDOWNTIME_CATEGORY">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="CLEANING"/>
            <xsd:enumeration value="CALIBRATION"/>
            <xsd:enumeration value="SOFTWARE_UPDATE"/>
            <xsd:enumeration value="HARDWARE_UPDATE"/>
        </xsd:restriction>
    </xsd:simpleType>
```

EDOWNTIME_TYPE specifies if the scheduled downtime is estimated through an interval time, or if the system administrator defines a fixed start time for a scheduled downtime.

Listing 39: DCD - EDOWNTIME_TYPE

```
<xsd:simpleType name="EDOWNTIME_TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ESTIMATED"/>
            <xsd:enumeration value="ACTUAL"/>
        </xsd:restriction>
    </xsd:simpleType>
```

ECAPACITY_CATEGORY specifies if a resource has a finite or an infinite capacity.

Listing 40: DCD ECAPACITY_CATEGORY

```
<xsd:simpleType name="ECAPACITY_CATEGORY">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FINITE"/>
            <xsd:enumeration value="INFINITE"/>
        </xsd:restriction>
    </xsd:simpleType>
```

ECOMPONENT_CATEGORY defines the category of a component.

Listing 41: DCD - ECOMPONENTCATEGORY

```
<xsd:simpleType name="ECOMPONENT_CATEGORY">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="SYSTEM"/>
            <xsd:enumeration value="WORKCELL"/>
            <xsd:enumeration value="SLM"/>
            <xsd:enumeration value="SUBUNIT"/>
            <xsd:enumeration value="RESOURCE"/>
        </xsd:restriction>
    </xsd:simpleType>
```

ERESOURCE_CATEGORY specifies the category of a resource.

- HARDWARE: All hardware on the workbench except the devices, e.g. vials and pipettes.
- SAMPLE: The measurement samples, e.g. a blood sample
- REAGENT: A substance that is needed for an analytical reaction.

- WASTE: Waste.
- SPACE: Space can be resource, e.g. a robot is moving through space.
- BUFFER: A special kind of space that is used as a resource buffer.
- UNDEFINED: A resource that does not belong in the previous categories.

Listing 42: DCD - ERESOURCE_CATEGORY

```
<xsd:simpleType name="ERESOURCE_CATEGORY" base="xsd:string">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="HARDWARE"/>
            <xsd:enumeration value="SAMPLE"/>
            <xsd:enumeration value="REAGENT"/>
            <xsd:enumeration value="WASTE"/>
            <xsd:enumeration value="SPACE"/>
            <xsd:enumeration value="BUFFER"/>
            <xsd:enumeration value="UNDEFINED"/>
        </xsd:restriction>
    </xsd:simpleType>
```

EPORT_TYPE specifies if an SLM port is a data port or a material port. This enumeration is only provided for backward compatibility with ASTM LECIS. OMG LECIS does not define data ports. In OMG LECIS, data is provided through device events.

```
<xsd:simpleType name="EPORT_TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="DATA"/>
            <xsd:enumeration value="MATERIAL"/>
        </xsd:restriction>
    </xsd:simpleType>
```

ETRANSFER_TYPE specifies, if a CORBA command argument is defined as “in,” “out,” or “inout.”

Listing 43: DCD - ETRANSFER_TYPE

```
<xsd:simpleType name="ETRANSFER_TYPE" base="xsd:string">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="INTRANSFER"/>
            <xsd:enumeration value="OUTTRANSFER"/>
            <xsd:enumeration value="INOUTTRANSFER"/>
        </xsd:restriction>
    </xsd:simpleType>
```

EVARIABLE_TYPE specifies the valid CORBA IDL types of a LECIS variable. This includes basic types and IDL types defined in this specification.

- LONG_TYPE: long
- FLOAT_TYPE: float
- BOOLEAN_TYPE: boolean
- STRING_TYPE: string

- OCTET_TYPE: octet
- SEQ_OCTET_TYPE: SLM_INTERFACE :SeqOctet
- SEQ_FLOAT_TYPE: SLM_INTERFACE :SeqFloat
- SEQ_LONG_TYPE: SLM_INTERFACE :SeqLong

Listing 44: DCD - EVARIABLE_TYPE

```
<xsd:simpleType name="EVARIABLE_TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="LONG_TYPE "/>
            <xsd:enumeration value="FLOAT_TYPE "/>
            <xsd:enumeration value="BOOLEAN_TYPE "/>
            <xsd:enumeration value="STRING_TYPE "/>
            <xsd:enumeration value="OCTET_TYPE "/>
            <xsd:enumeration value="SEQ_OCTET_TYPE "/>
            <xsd:enumeration value="SEQ_FLOAT_TYPE "/>
            <xsd:enumeration value="SEQ_LONG_TYPE "/>
        </xsd:restriction>
    </xsd:simpleType>
```

EACCESSTYPE defines how a resource can be transferred to or from a port.

- INLET: A resource goes only in the port.
- OUTLET: A resource comes only out of the port.
- INOUTLET: A resource can go in or out.
- TRANSFER: A resource is only transferred through this port, e.g. a tube.

Listing 45: DCD - EACCESS_TYPE

```
<xsd:simpleType name="EACCESS_TYPE" base="xsd:string">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="INLET"/>
            <xsd:enumeration value="OUTLET"/>
            <xsd:enumeration value="INOUTLET"/>
            <xsd:enumeration value="TRANSFER"/>
        </xsd:restriction>
    </xsd:simpleType>
```

EOWNERSTATUS specifies the ownerstatus of a resource or port.

- PRIVATE_OWNER: A private resource or port cannot change its owner.
- UNLOCKED: An unlocked resource or port can be obtained for use by another component.
- LOCKED:A locked resource or port is in use by a component.

Listing 46: DCD - EOWNER_STATUS

```
<xsd:simpleType name="EOWNER_STATUS">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="PRIVATE_OWNER"/>
            <xsd:enumeration value="UNLOCKED"/>
```

```
            <xsd:enumeration value="LOCKED"/>
        </xsd:restriction>
    </xsd:simpleType>
```

RANGE_TYPE defines the range of a value. The range is limited by a low and a high limit, e.g. a temperature value has a low limit of 20 degrees Celsius and a high limit of 50 degrees Celsius.

Listing 47: DCD - RANGE_TYPE

```
    <xsd:complexType name="RANGE_TYPE">
        <xsd:sequence>
            <xsd:element name="LOW_LIMIT" type="LIMIT_TYPE" minOccurs="0">
            </xsd:element>
            <xsd:element name="HIGH_LIMIT" type="LIMIT_TYPE" minOccurs="0">
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
```

The ITEM_VALUE_TYPE defines an item value pair of strings.

Listing 48: DCD - ITEM_VALUE_TYPE

```
<xsd:complexType name="ITEM_VALUE_TYPE">
        <xsd:sequence>
            <xsd:element name="ITEM" type="xsd:string">
            </xsd:element>
            <xsd:element name="VALUE" type="xsd:string">
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
```

An ARGUMENT_TYPE defines a command argument. The argument has a NAME, an ARGUMENT_TYPE, a DEFAULT_VALUE, a TRANSFER_TYPE, a DESCRIPTION, user definable PROPERTIES and an ARGUMENT_RANGE.

Listing 49: DCD - ARGUMENT_TYPE

```
<xsd:complexType name="ARGUMENT_TYPE">
    <xsd:sequence>
        <xsd:element name="NAME" type="xsd:string"/>
        <xsd:element name="ARGUMENT_TYPE" type="EVARIABLE_TYPE"/>
        <xsd:element name="DEFAULT_VALUE" type="xsd:string" minOccurs="0"/>
        <xsd:element name="TRANSFER_TYPE" type="ETRANSFER_TYPE"/>
        <xsd:element name="RANGE" type="RANGE_TYPE" minOccurs="0"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

The COMMAND_TYPE defines a functional command of the device.

The COMMAND_ID identifies the command within the Sub-Unit.

The command has a NAME. If the name is difficult to read by humans, it is possible to give it an ALIASNAME. This alias name replaces the name where the user may read the command name.

The DURATION specifies the maximum execution time of this command. Time dependent commands that cannot specify this time have a duration of 0.

The command CATEGORY categorizes the command.

The DESCRIPTION is a user readable text, describing the functionality of this command.

A command may have a list of ARGUMENTS, the “in” parameters of this command.

The EXCLUSION_LIST defines a list of commands that cannot run concurrently to this command. Each item in the list contains the unit_id and the command_id of the excluded SLM command.

SYNC_RESPONSE_DATA defines the “out” parameters of this command.

PROPERTIES is a user definable list of item-value pairs to define additional information not defined in this specification.

CONFIGURATION_COMMANDS specifies configuration commands that should be called before this command is called.

The REQUIRED_RESOURCES and PRODUCED_RESOURCES reference resources in the system that are required for this command execution, or which are produced during the command execution.

INPUT_PORTS and OUTPUT_PORTS specify which ports are used during the command execution.

Listing 50: DCD - COMMAND_TYPE

```
<xsd:complexType name="COMMAND_TYPE">
    <xsd:sequence>
        <xsd:element name="COMMAND_ID" type="xsd:ID"/>
        <xsd:element name="NAME" type="xsd:string"/>
        <xsd:element name="ALIAS_NAME" type="xsd:string" minOccurs="0"/>
        <xsd:element name="DURATION" type="xsd:long"/>
        <xsd:element name="CATEGORY" type="ECOMMAND_CATEGORY"/>
        <xsd:element name="TYPE" type="ECOMMAND_TYPE"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="FORMAL_ARGUMENTS" type="ARGUMENT_TYPE" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="EXCLUSION_LIST" type="ITEM_VALUE_TYPE" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="SYNC_RESPONSE_DATA" type="ARGUMENT_TYPE" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="CONFIGURATION_COMMANDS" type="xsd:IDREF" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="REQUIRED_RESOURCES" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PRODUCED_RESCOURCES" type="xsd:IDREF" minOccurs="0"
```

```
                maxOccurs="unbounded"/>
                <xsd:element name="OUTPUT_PORTs" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="INPUT_PORTs" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
```

The VALUE_TYPE specifies a VALUE, its data-TYPE, UNIT and EXPONENT (10-based). This is required to describe values like 20,43 $*10^{-3}$ kg.

The UNIT is specified as a string value. Since this specification cannot provide all possible units that might be used in the world, we provide only a list of the ISO base units in the appendix. The string representations of these values should be used as the UNIT identifier if possible. Additional units are system specific and must be defined externally.

Listing 51: DCD - VALUE_TYPE

```
<xsd:complexType name="VALUE_TYPE">
        <xsd:sequence>
            <xsd:element name="VALUE" type="xsd:string"/>
            <xsd:element name="TYPE" type="ENUMBER_TYPE"/>
            <xsd:element name="EXPONENT" type="xsd:string"/>
            <xsd:element name="UNIT" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
```

The CAPACITY_TYPE defines the capacity of a resource, for example a vial. It has a minimum and maximum capacity. The fill steps defines in which steps something can be filled in this resource, e.g. only in steps of 10ml.

Listing 52: DCD - CAPACITY_TYPE

```
<xsd:complexType name="CAPACITY_TYPE">
        <xsd:sequence>
            <xsd:element name="MAX_CAPACITY" type="VALUE_TYPE"/>
            <xsd:element name="MIN_CAPACITY" type="VALUE_TYPE"/>
            <xsd:element name="FILL_STEPS" type="VALUE_TYPE" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
```

The COMPONENT_ID_TYPE specifies the ID and category (system, workcell, slm, subunit, or resource) of a component.

The category determines which ID fields are required to identify the component unambiguously.

If the category is SYSTEM, no ID is required, because an SCD contains only one system.

If the category is WORKCELL, only the WORKCELL_ID is required.

If the category is SLM, only the SLM_ID is required, because the SLM_ID is unique within the SCD.

If the category is SUBUNIT, the SLM_ID and SUBUNIT_ID is required.

If the category is RESOURCE, the RESOURCE_ID and the ID of associated parent of the resource is required.

Listing 53: DCD - COMPONENT_ID_TYPE

```
<xsd:complexType name="COMPONENT_ID_TYPE">
        <xsd:sequence>
            <xsd:element name="WORKCELL_ID" type="xsd:string">
            </xsd:element>
            <xsd:element name="SLM_ID" type="xsd:string"/>
            <xsd:element name="SUBUNIT_ID" type="xsd:string"/>
            <xsd:element name="RESOURCE_ID" type="xsd:string"/>
            <xsd:element name="COMPONENT_CATEGORY" type="ECOMPONENT_CATEGORY">
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
```

The OWNERSHIP_TYPE defines who is owner of a resource or port. It contains the component ID and the current ownership status.

Listing 54: DCD - OWNERSHIP_TYPE

```
<xsd:complexType name="OWNERSHIP_TYPE">
        <xsd:sequence>
            <xsd:element name="COMPONENT_ID" type="COMPONENT_ID_TYPE">
            </xsd:element>
            <xsd:element name="OWNER_STATUS" type="EOWNER_STATUS">
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
```

The PORT_TYPE defines an access port.

The port identifier consists of four parts. The PORT_ID and the X,Y,Z values. X,Y,Z identify a port in a multidimensional array. The port ID must only be unique within the scope of its parent entity, because a port always belongs to a resource or an SLM. To uniquely identify a port, the parent identifier is concatenated with the PORT_ID.

The ACCESS_TYPE defines how a port can be accessed.

A port usually has a CAPACITY and can be owned by different components. The ownership is defined in the element OWNERSHIP.

A port may have physical properties (dimension, location) as well as user definable properties.

The DESCRIPTION is a user readable text.

A port may contain a resource (CONTENT_RESCOURCE).

Listing 55: DCD - PORT_TYPE

```
<xsd:complexType name="PORT_TYPE">
    <xsd:sequence>
        <xsd:element name="PORT_ID" type="xsd:ID"/>
```

```
        <xsd:element name="X" type="xsd:string"/>
        <xsd:element name="Y" type="xsd:string"/>
        <xsd:element name="Z" type="xsd:string"/>
        <xsd:element name="CONTENT_RESOURCE" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PORT_TYPE" type="EPORT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="ACCESS_TYPE" type="EACCESS_TYPE"/>
        <xsd:element name="CAPACITY" type="CAPACITY_TYPE"/>
        <xsd:element name="OWNERSHIP" type="OWNERSHIP_TYPE"/>
        <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

Resources may be consumed, created, owned, or transformed. For example, auto sampler vials or pipette tips are resources. Resources are distinguished from devices in that the latter communicate with the system and accept commands. Resources however, are essential aids to completing the task at hand.

A resource is identified through its RESOURCE_ID. If a resource belongs to an SLM, the RESOURCE_ID is unique only within the scope of the SLM. In this case the RESOURCE_ID must be concatenated with the SLM_ID and/or SUBUNIT_ID.

If a resource belongs to the system or a work cell, the RESOURCE_ID must be unique within the complete SCD. In this case the resource can change its parent entity without changing its unique identifier.

A resource is categorized with its RESOURCE_CATEGORY.

Since a resource can be dangerous for the operator handling the resource, e.g. a flammable reagent, it has a danger classification called DANGER_CLASS. This string may contain a system specific identifier for the handling of hazardous materials.

A resource may have a current quantity, defined by CURRENT QUANTITY.

OWNERSHIP determines the current owner of the resource.

A port may have PHYSICAL_CHARACTERISTICS (dimension, location) as well as user definable properties.

The DESCRIPTION is a user readable text.

A resource, e.g. a rack, can have PORTs. Ports always belong to a resource. The definition does not allow the definition of ports directly in a component, since this would violate the consistency of the model.

A resource can also be located in a port. Therefore the resource has a reference IN_PORT_REF to a port that is containing this resource.

Listing 56: DCD - RESOURCE_TYPE

```
<xsd:complexType name="RESOURCE_TYPE">
    <xsd:sequence>
        <xsd:element name="RESOURCE_ID" type="xsd:ID"/>
        <xsd:element name="RESOURCE_CATEGORY" type="ERESOURCE_CATEGORY"/>
```

```
        <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
        <xsd:element name="DANGER_CLASS" type="xsd:string"/>
        <xsd:element name="OWNERSHIP" type="OWNERSHIP_TYPE"/>
        <xsd:element name="IN_PORT_REF" type="xsd:string" minOccurs="0"/>
        <xsd:element name="ACCESS_PORTS" type="PORT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="CURRENT_QUANTITY" type="VALUE_TYPE" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

A device has additional administration data. This includes name, type, serial number etc.

The PROTOCOL value contains the identifier of the protocol used to connect to the SLM. Currently the values “ASTM LECIS” and “CORBA LECIS” are valid. This list could be extended, if LECIS is adapted to other protocols like SOAP or DCOM.

Listing 57: DCD - ADMINISTRATIVE_TYPE

```
<xsd:complexType name="ADMINISTRATIVE_TYPE">
        <xsd:sequence>
            <xsd:element name="NAME" type="xsd:string"/>
            <xsd:element name="PROTOCOL" type="xsd:string"/>
            <xsd:element name="MODEL_NUMBER" type="xsd:string"/>
            <xsd:element name="SERIAL_NUMBER" type="xsd:string"/>
            <xsd:element name="MANUFACTURER_ID" type="xsd:string"/>
            <xsd:element name="MANUFACTURER_NAME" type="xsd:string"/>
            <xsd:element name="SUPPORT_ADDRESS" type="xsd:string"/>
            <xsd:element name="UPDATE_ADDRESS" type="xsd:string"/>
            <xsd:element name="SOFTWARE_VERSION_NUMBER" type="xsd:string"/>
            <xsd:element name="DCD_VERSION" type="xsd:string"/>
            <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
```

The DIMENSION_TYPE defines the physical dimensions of a component. Height, width, and length are specified in mm.

Listing 58: DCD - DIMENSION_TYPE

```
<xsd:complexType name="DIMENSION_TYPE">
        <xsd:sequence>
            <xsd:element name="HEIGHT" type="xsd:long"/>
            <xsd:element name="WIDTH" type="xsd:long"/>
            <xsd:element name="LENGTH" type="xsd:long"/>
        </xsd:sequence>
    </xsd:complexType>
```

The TRANSLATION_TYPE defines the position relative to the center of the coordinate system.

Listing 59: DCD - TRANSLATION_TYPE

```
<xsd:complexType name="TRANSLATION_TYPE" content="elementOnly">
```

```
        <xsd:sequence>
            <xsd:element name="XTRANSLATION" type="xsd:long" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="YTRANSLATION" type="xsd:long" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="ZTRANSLATION" type="xsd:long" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
```

The ROTATION_TYPE defines the rotation of a component in the coordinate system. X,y, and z rotation are stated in degrees.

Listing 60: DCD - ROTATION_TYPE

```
<xsd:complexType name="ROTATION_TYPE">
        <xsd:sequence>
            <xsd:element name="XROTATION" type="xsd:long" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="YROTATION" type="xsd:long" maxOccurs="1" minOccurs="1"/>
            <xsd:element name="ZROTATION" type="xsd:long" maxOccurs="1" minOccurs="1"/>
        </xsd:sequence>
    </xsd:complexType>
```

The LOCATION_TYPE specifies translation and rotation of a component in the coordinate system, or in other words the exact position in space.

Listing 61: DCD - LOCATION_TYPE

```
<xsd:complexType name="LOCATION_TYPE">
        <xsd:sequence>
            <xsd:element name="TRANSLATION" type="TRANSLATION_TYPE"/>
            <xsd:element name="ROTATION" type="ROTATION_TYPE"/>
        </xsd:sequence>
    </xsd:complexType>
```

The PHYSICAL_CHARACTERISTICS_TYPE defines DIMENSION, LOCATION, and the WEIGHT of a component. The weight is measured in grams.

Listing 62: DCD - PHYSICAL_CHARACTERISTICS_TYPE

```
<xsd:complexType name="PHYSICAL_CHARACTERISTICS_TYPE">
        <xsd:sequence>
            <xsd:element name="DIMENSION" type="DIMENSION_TYPE"/>
            <xsd:element name="LOCATION" type="LOCATION_TYPE" minOccurs="0"/>
            <xsd:element name="WEIGHT" type="xsd:long" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
```

The EXT_MACRO_COMMAND_TYPE is a pre-configured device command that is used in macro commands. It contains a reference to a device command (COMMANDREF) and a list of argument values (ARGUMENTLIST). The argument values are string values, and must be converted into the specified type before their use.

Listing 63: DCD - EXT_MACRO_COMMAND_TYPE

```
<xsd:complexType name="EXT_MACRO_COMMAND_TYPE">
        <xsd:sequence>
            <xsd:element name="COMMANDREF" type="xsd:IDREF"/>
```

```
            <xsd:element name="ARGUMENTLIST" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
```

A EXT_MACRO_TYPE is a predefined list of macro commands. The device manufacturer can define macros for common tasks.

Its main elements are the ID, NAME, DESCRIPTION, PROPERTIES and a list of MACRO_COMMANDS.

Listing 64: DCD - EXT_MACRO_TYPE

```
<xsd:complexType name="EXT_MACRO_TYPE">
    <xsd:sequence>
        <xsd:element name="MACRO_ID" type="xsd:ID"/>
        <xsd:element name="NAME" type="xsd:string"/>
        <xsd:element name="CATEGORY" type="ECOMMAND_CATEGORY"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="MACRO_COMMANDS" type="EXT_MACRO_COMMAND_TYPE" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

The EVENT_TYPE defines a device event.

- EVENT_ID: The unique identifier within the scope of the SLM.
- PRIORITY: A long value that specifies the event priority. It can contain a value from 1 to10. A value of 1 has the highest priority.
- CATEGORY: The event category.
- DESCRIPTION: A user readable description of the event.
- POSSIBLE_EVENT_DATA_VALUES: A device specific event might provide several possible data values for this event. The possible values can be specified in this field. The TSC knows then in advance, which values to exepct.
- EVENT_DATA_ARGUMENTS: An evnt may contain a list of argument values.
- SYSTEM_VARIABLES: Has a System Variable changed, the SLM is sending an event, with the new variable values.
- EVENT_REACTION_COMMANDS: For each event type, a list of commands can be specified, that could be called in reaction to this event, e.g. an error recovery command.
- PROPERTIES: An item-value list of user definable values.

Listing 65: DCD - EVENT_TYPE

```
<xsd:complexType name="EVENT_TYPE">
    <xsd:sequence>
        <xsd:element name="EVENT_ID" type="xsd:string"/>
        <xsd:element name="PRIORITY" type="xsd:long"/>
        <xsd:element name="CATEGORY" type="EEVENT_CATEGORY"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
```

```
        <xsd:element name="POSSIBLE_EVENT_DATA_VALUES" type="xsd:string" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="EVENT_DATA_ARGUMENTS" type=" ARGUMENT_TYPE " minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="SYSTEM_VARIABLES" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="EVENT_REACTION_COMMANDS" type="xsd:IDREF" minOccurs="0"
        maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

The LIMIT_TYPE defines the type and value of an lower or upper limit.

Listing 66: DCD - LIMIT_TYPE

```
<xsd:complexType name="LIMIT_TYPE">
        <xsd:sequence>
            <xsd:element name="RANGE_VALUE_TYPE" type="ENUMBER_TYPE"/>
            <xsd:element name="RANGE_VALUE" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
```

DOWNTIME_TYPE contains information about the scheduled down times for an SLM.

The DOWNTIME_ID is unique within the scope of the SLM.

Listing 67: DCD DOWNTYPE_TYPE

```
<xsd:complexType name="DOWNTIME_TYPE">
        <xsd:sequence>
            <xsd:element name="DOWNTIME_ID" type="xsd:string"/>
            <xsd:element name="CATEGORY" type="EDOWNTIME_CATEGORY"/>
            <xsd:element name="TYPE" type="EDOWNTIME_TYPE"/>
            <xsd:element name="START_TIME" type="xsd:string"/>
            <xsd:element name="DURATION" type="xsd:string"/>
            <xsd:element name="INTERVAL" type="xsd:string"/>
            <xsd:element name="DESCRIPTION" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
```

SYSTEM_VARIABLE_TYPE defines an SLM System Variable.

The VARIABLE_ID is unique within the scope of the SLM.

Listing 68: DCD SYSTEM_VARIABLE_TYPE

```
<xsd:complexType name="SYSTEM_VARIABLE_TYPE">
        <xsd:sequence>
            <xsd:element name="VARIABLE_ID" type="xsd:string"/>
            <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
            <xsd:element name="ACCESS_PRIVILEDGE" type="xsd:string" minOccurs="0"/>
```

```
            <xsd:element name="CATEGORY" type="xsd:string" minOccurs="0"/>
            <xsd:element name="DATA_TYPE" type="EVARIABLE_TYPE"/>
            <xsd:element name="VALUE_RANGE" type="RANGE_TYPE" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
```

SCD Interface Specification 4

4.1 Introduction

This chapter specifies an SCD interface, that may support the interoperability of different system implementations although it is not a requirement of the RFP.

Since the SCD is an integral part of a possible laboratory automation system architecture, a standard interface for the System Capability Dataset would greatly improve the interoperability between different system implementations. Users could choose a TSC implementation without depending on a vendor specific SCD implementation. Therefore the SCD interface definition has been added to this specification.

The XML schema specified in the previous chapter is the exchange format of the System Capability Dataset and Device Capability Dataset.

To use this information it has to be stored in a database management system. Each software vendor could provide a proprietary SCD interface. However, this would bind users to one vendor, because the SCD will be an essential part of the automation system that might be used by many software components.

To allow for more interoperability, this specification provides an SCD interface, which could be implemented by each vendor. Since it publishes only a CORBA interface, it hides the database specific implementations of the SCD. An SCD with this interface could be accessed by many vendor independent software components.

The interfaces in this chapter are not described in detail, because the attributes are mapped directly from the XML specification, where they are described in detail.

4.1.1 Registering the SCD Reference in the CORBA Naming Service

The **ISCDRegistry** (Section 4.1.3, “ISCDRegistry,” on page 4-2) CORBA interface, which is the root-interface of the SCD, has to be registered in the CORBA Naming Service [1].

A system implementing this SCD interface must provide an implementation of the Naming Service. The SCDs ORB must be configured with the Naming Service URL. How this is done is implementation specific.

Each time the SCD server is started, it registers the interface **ISCDRegistry** in the Naming Service with the name “ISCDRegistry.” The naming context, under which the SCD reference is stored, is specified as **automation/LECIS/SCD**.

4.1.2 Exception Handling

This specification maps the XML definition into IDL interfaces with IDL attributes. IDL attributes are easy to implement, understand, and visualize, since the IDL compiler provides the operation declaration for the set/get operations of these attributes. Unfortunately, these operations do not allow the definition of CORBA User Exceptions.

This should not be a problem, since most software components of the automation system will access the SCD in read only mode. Most data is written into the SCD once by the system administrator and does not change during runtime. Information that changes during runtime includes the location of components, the resource locking as well as scheduled downtimes. In these cases the system should provide the means to change this information from only a single point in the system.

If an SCD interface operation is not successful, the operation should return a CORBA::BAD_PARAM system exception, and the database should rollback the last action. In this case the calling client knows that its call had no effect.

4.1.3 ISCDRegistry

A client of the System Capability Dataset first requires a reference to the interface **ISCDRegistry** (Figure 4-1 on page 4-3).

This interface allows for the creation of all known SCD entities that provide their own CORBA interface.

In addition it provides operations to search for specific elements in the SCD registry.

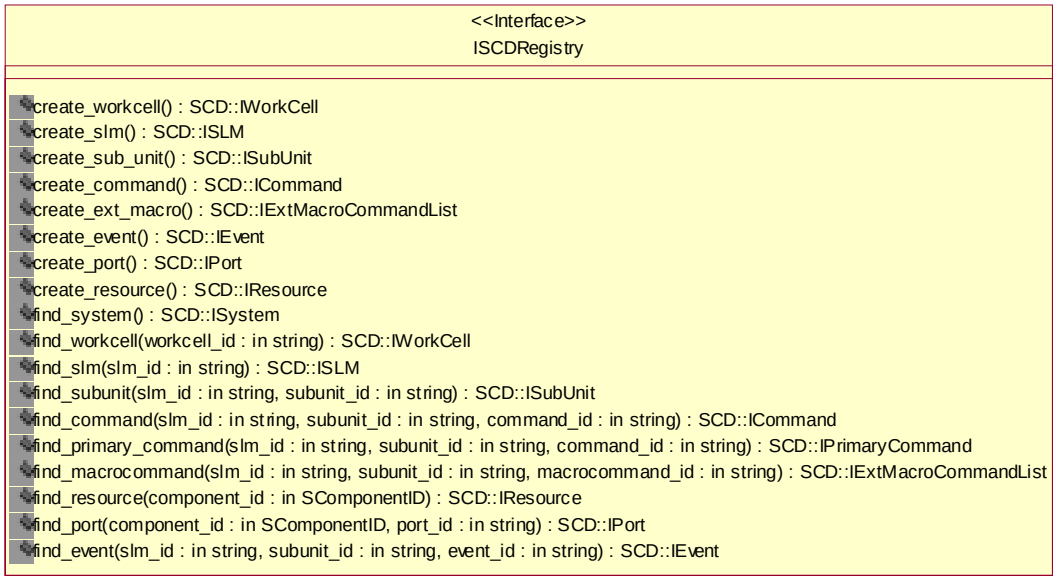

Figure 4-1 ISCDRegistry

4.1.4 ISystem and IWorkcell

ISystem is a singleton interface in the SCD. In one SCD database exists only one System.

The system is mainly a container for the work cells. A work cell can be deleted from the SCD with its delete() operation. This operation deletes the work cell and all its references from the database. After calling the delete() operation, the client must delete its reference to this interface, because the object reference is invalid now.

IWorkCell has references to all SLMs in this work cell. In addition a work cell can be subdivided into sub-cells.

The **parent_id** of an interface references the parent component. This gives the client the possibility to move up in the SCD tree. The system or a supercell can be parents of a work cell.

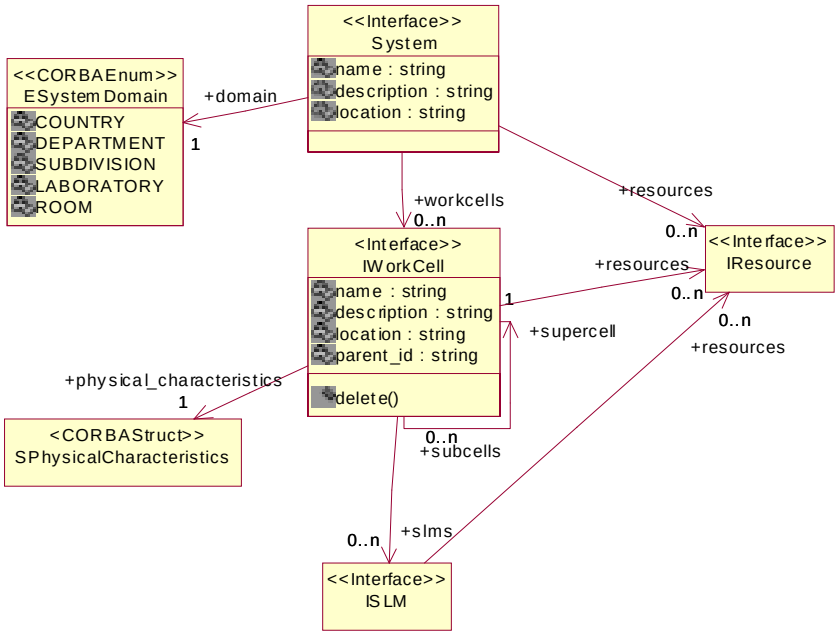

Figure 4-2 ISystem- IWorkcell

4.1.5 ISLM

ISLM represents a Simple Laboratory Module (SLM) in the database. Each SLM usually represents a real life instrument.

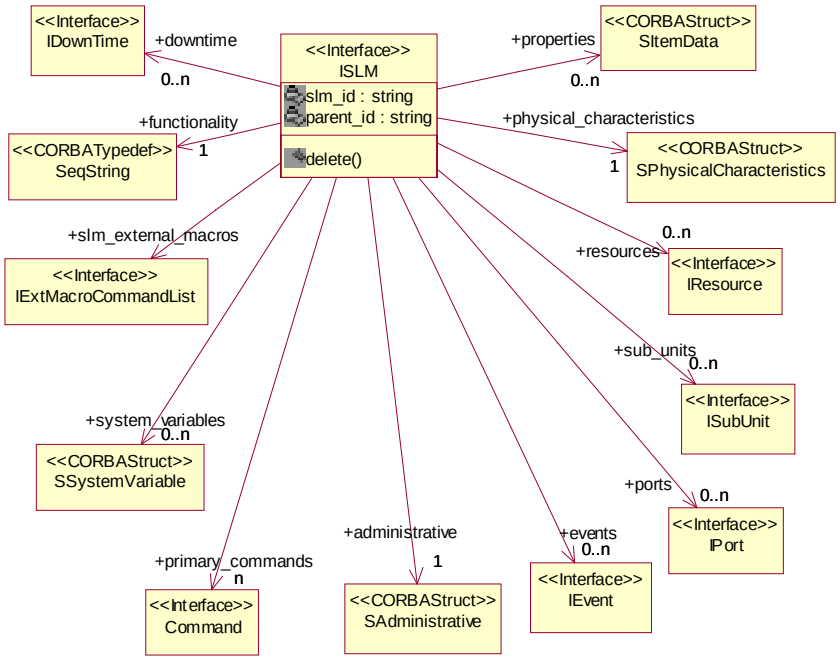

Figure 4-3 ISLM

4.1.6 ISubUnit

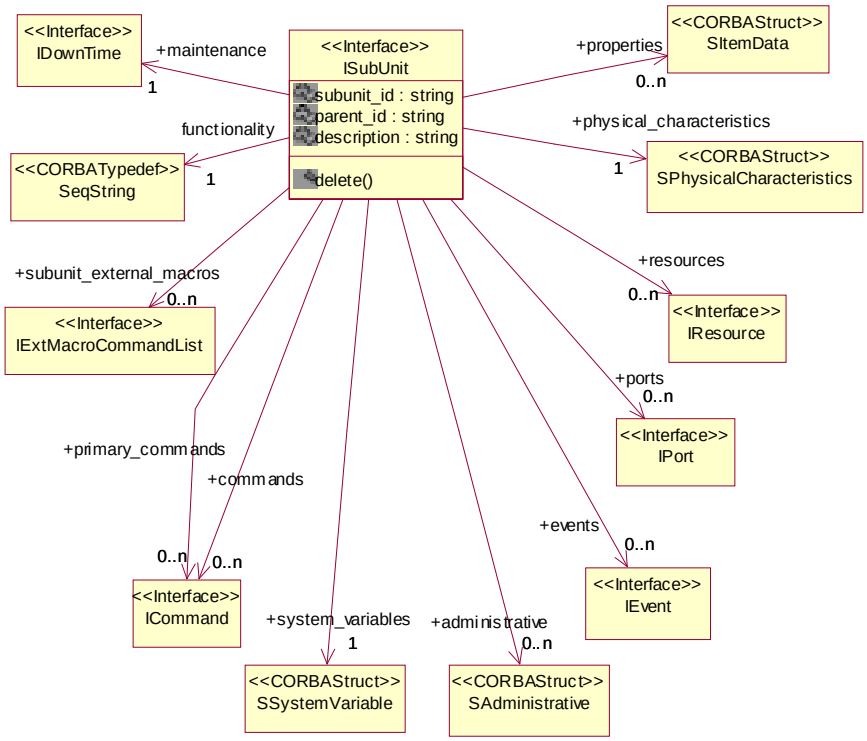

Figure 4-4 ISubUnit

4.1.7 ICommand

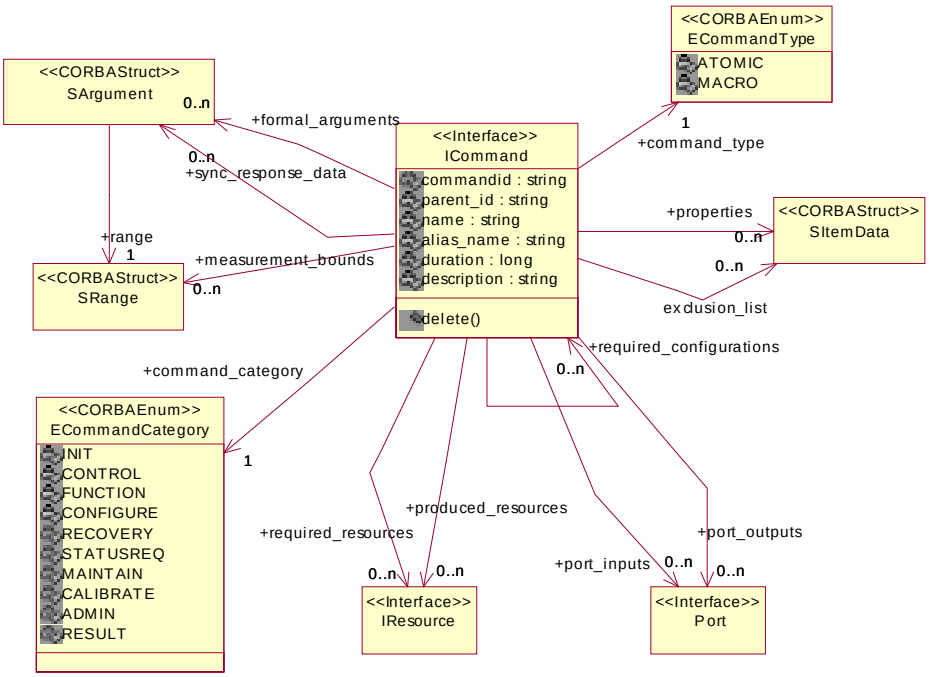

Figure 4-5 ICommand

4.1.8 SArgument

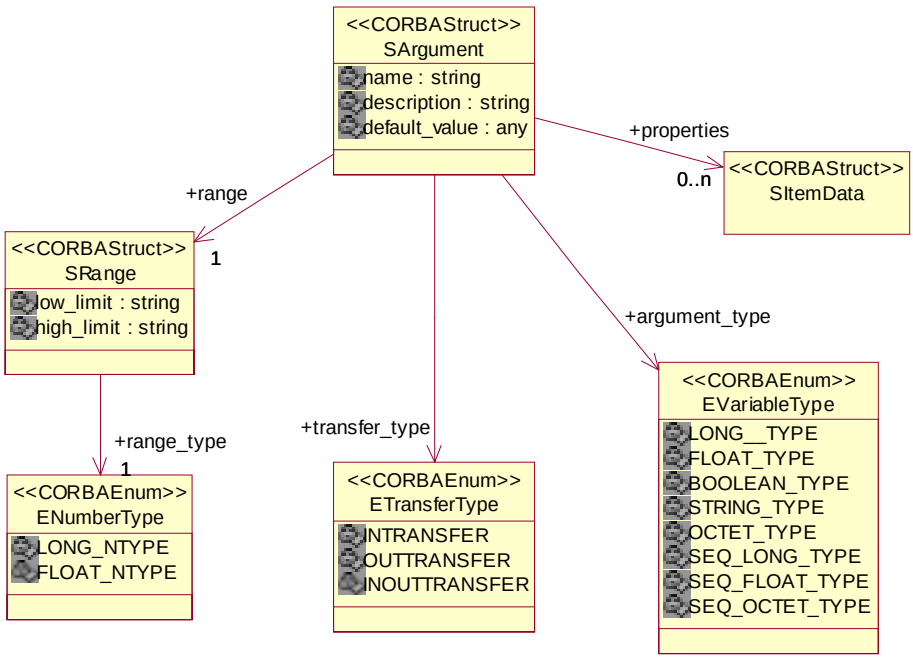

Figure 4-6 SArgument

4.1.9 SAdministrative

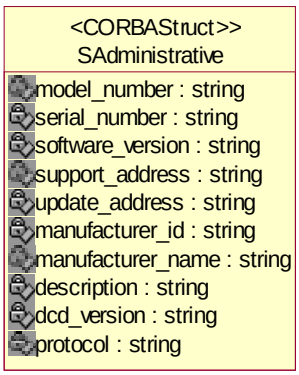

Figure 4-7 SADMINISTRATIVE

4.1.10 IEvent

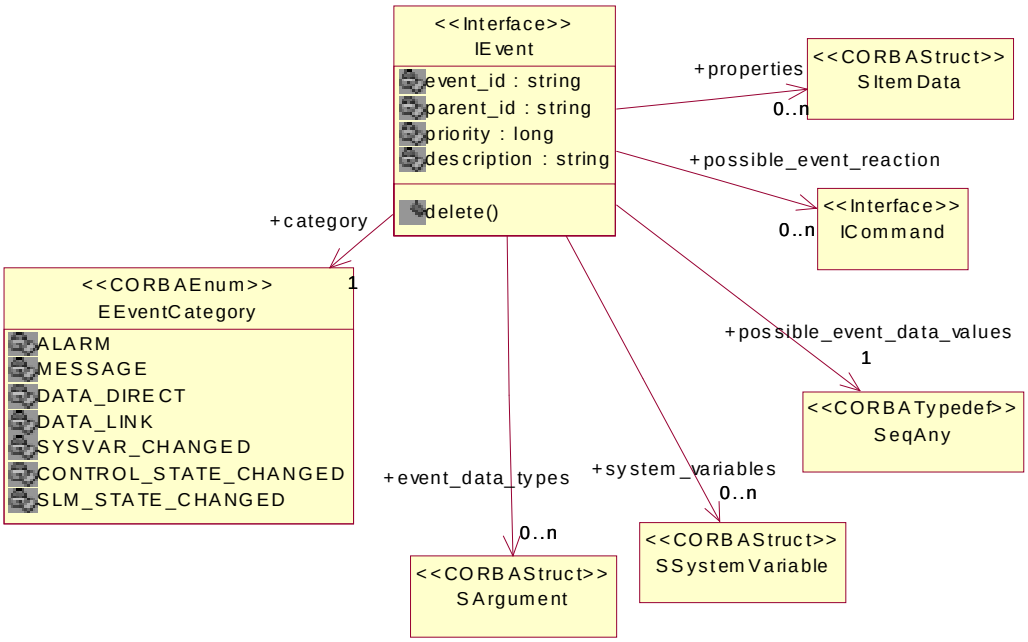

Figure 4-8 IEvent

4.1.11 IExtMacroCommandList

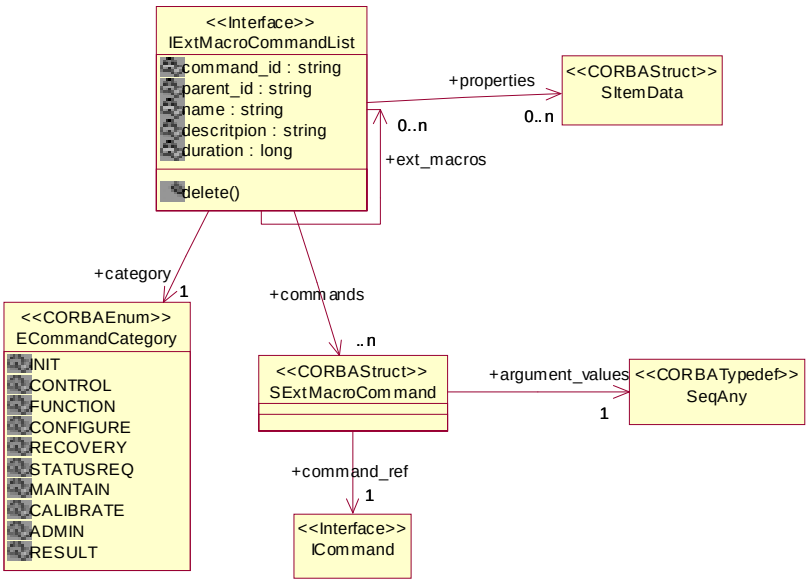

Figure 4-9

4.1.12 IDowntime

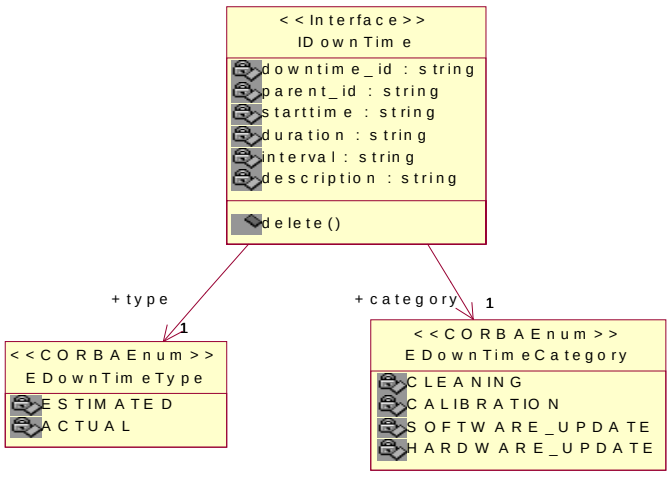

Figure 4-10 IMaintenance

4.1.13 SOwnership

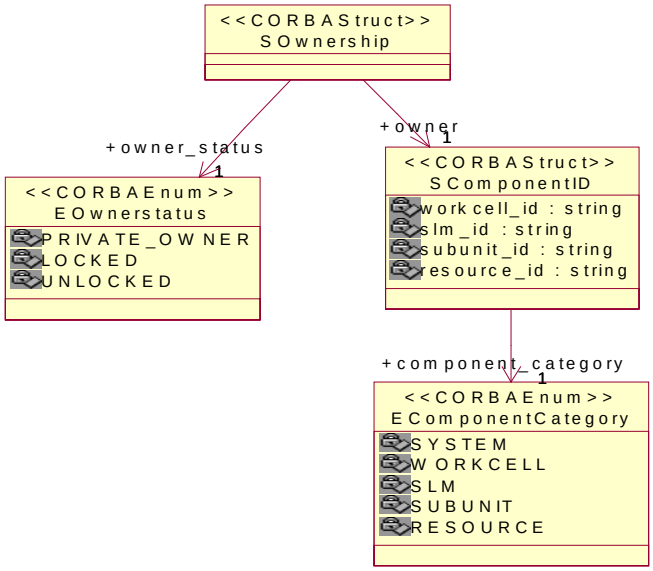

Figure 4-11 SOwnership

4.1.14 SPhysicalCharacteristics

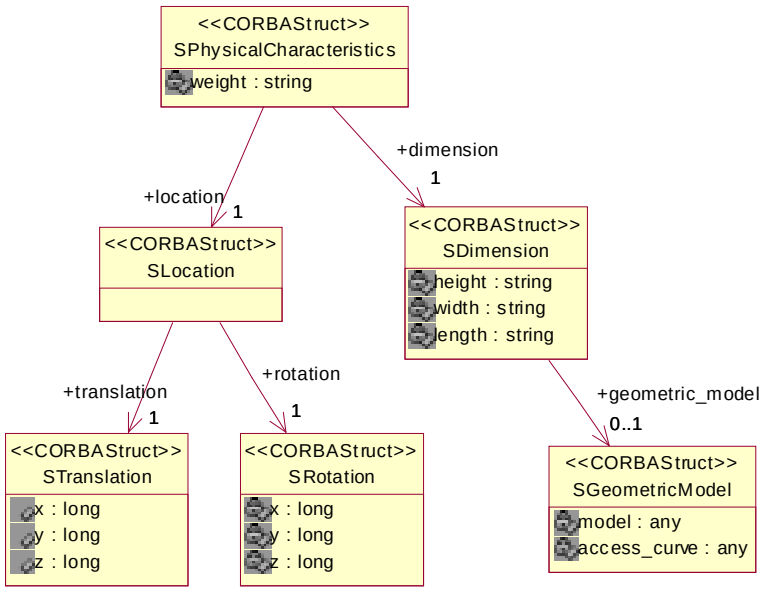

Figure 4-12 SPhysicalCharacteristics

4.1.15 SItemData

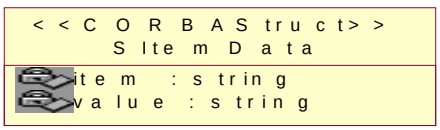

Figure 4-13 SItemData

4.1.16 IPort

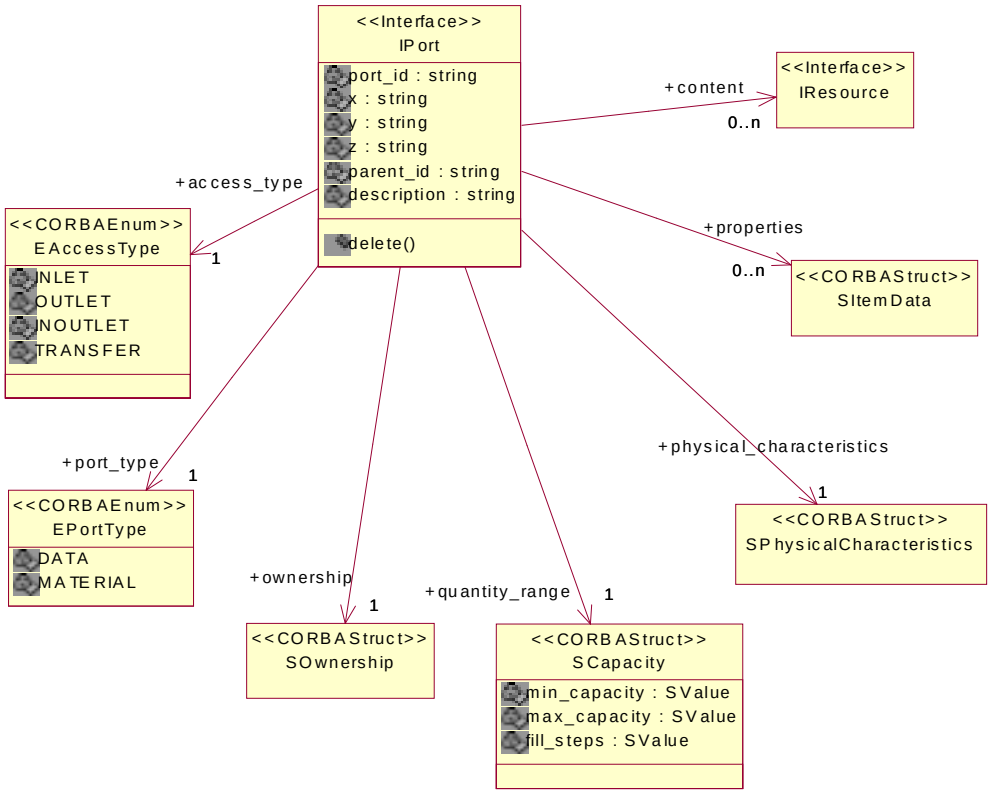

Figure 4-14 IPort

4.1.17 IResource

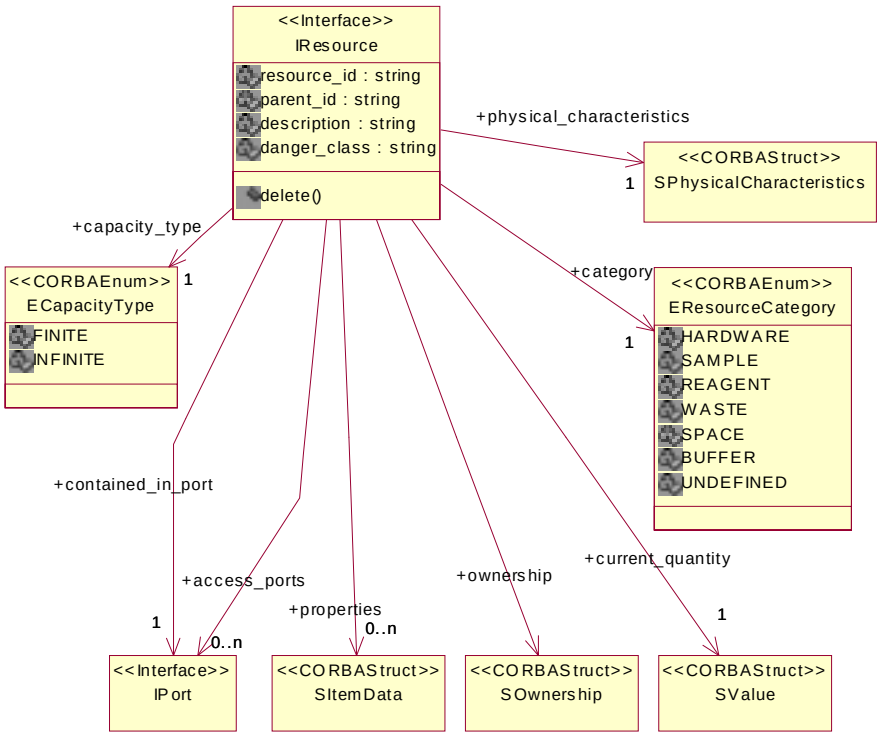

Figure 4-15 IResource

4.1.18 SSystemVariable

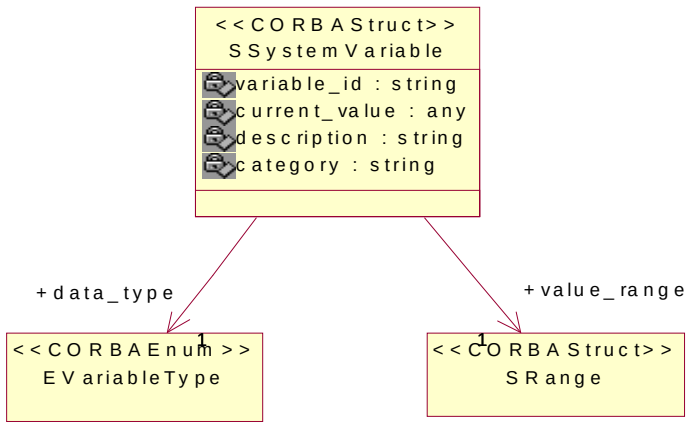

Figure 4-16 SSystemVariable

4.1.19 SValue

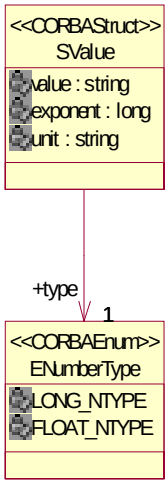

Figure 4-17 SValue

Changes to ASTM LECIS Specification 5

5.1 Interface Design

ASTM LECIS is a string based message protocol for the communication between a device and a system controller. This communication scheme was replaced in this proposal by IDL CORBA interfaces. This has resulted in changes in the device control architecture. It did not seem reasonable to adopt the ASTM LECIS operations directly into IDL interfaces.

5.2 State Models

ASTM LECIS provides one state model per laboratory device. One of its weaknesses is the non-deterministic behavior of the device, when two or more requests are handled concurrently.

Therefore, in LECIS CORBA a laboratory device is split into one Main-Unit and several Sub-Units. Each Sub-Unit has its own state model for deterministic control.

The basic state model of ASTM LECIS has been adopted, with changes made only to improve the overall design and to meet the requirements of the RFP.

5.3 Device Capability Dataset

ASTM LECIS recognized the need for a Device Capability Dataset, but could not provide one in its specification.

This specification provides a XML Schema definition for a SCD/DCD. Only the combination of control interface and device description makes this specification useful in a highly automated environment.

SCD XML - Schema Definition 6

6.1 The Complete SCD XML - Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited by Thorsten Richter (Creon Labcontrol AG) -->
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">


<xsd:element name="SCD">
   <xsd:complexType>
        <xsd:sequence>
             <xsd:element name="SYSTEM" type="SYSTEM_TYPE" maxOccurs="1"/>
        </xsd:sequence>
   </xsd:complexType>
</xsd:element>


<xsd:element name="DCD">
   <xsd:complexType>
        <xsd:sequence>
             <xsd:element name="SLM" type="SLM_TYPE" maxOccurs="unbounded"/>
        </xsd:sequence>
   </xsd:complexType>
</xsd:element>


<xsd:simpleType name="ECOMMAND_CATEGORY">
   <xsd:restriction base="xsd:string">
        <xsd:enumeration value="INIT"/>
        <xsd:enumeration value="CONTROL"/>
        <xsd:enumeration value="FUNCTION"/>
        <xsd:enumeration value="CONFIGURE"/>
        <xsd:enumeration value="RECOVERY"/>
        <xsd:enumeration value="STATUSREQ"/>
        <xsd:enumeration value="MAINTAIN"/>
        <xsd:enumeration value="CALIBRATE"/>
        <xsd:enumeration value="ADMIN"/>
```

```
            <xsd:enumeration value="DATA"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ECOMMAND_TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ATOMIC"/>
            <xsd:enumeration value="MACRO"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ESYSTEM_DOMAIN">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="COUNTRY"/>
            <xsd:enumeration value="DEPARTMENT"/>
            <xsd:enumeration value="SUBDIVISION"/>
            <xsd:enumeration value="LABORATORY"/>
            <xsd:enumeration value="ROOM"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="EEVENT_CATEGORY">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ALARM"/>
            <xsd:enumeration value="MESSAGE"/>
            <xsd:enumeration value="DATA_DIRECT"/>
            <xsd:enumeration value="DATA_LINK"/>
            <xsd:enumeration value="SYSVAR_CHANGED"/>
            <xsd:enumeration value="CONTROL_STATE_CHANGED"/>
            <xsd:enumeration value="SLM_STATE_CHANGED"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ENUMBER_TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="LONG"/>
            <xsd:enumeration value="FLOAT"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="EDOWNTIME_CATEGORY">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="CLEANING"/>
            <xsd:enumeration value="CALIBRATION"/>
            <xsd:enumeration value="SOFTWARE_UPDATE"/>
            <xsd:enumeration value="HARDWARE_UPDATE"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="EDOWNTIME_TYPE">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ESTIMATED"/>
            <xsd:enumeration value="ACTUAL"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ECAPACITY_CATEGORY">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FINITE"/>
            <xsd:enumeration value="INFINITE"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ECOMPONENT_CATEGORY">
```

```
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="SYSTEM"/>
        <xsd:enumeration value="WORKCELL"/>
        <xsd:enumeration value="SLM"/>
        <xsd:enumeration value="SUBUNIT"/>
        <xsd:enumeration value="RESOURCE"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ERESOURCE_CATEGORY">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="HARDWARE"/>
        <xsd:enumeration value="SAMPLE"/>
        <xsd:enumeration value="REAGENT"/>
        <xsd:enumeration value="WASTE"/>
        <xsd:enumeration value="SPACE"/>
        <xsd:enumeration value="BUFFER"/>
        <xsd:enumeration value="UNDEFINED"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="EPORT_TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="DATA"/>
        <xsd:enumeration value="MATERIAL"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="ETRANSFER_TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="INTRANSFER"/>
        <xsd:enumeration value="OUTTRANSFER"/>
        <xsd:enumeration value="INOUTTRANSFER"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="EVARIABLE_TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="LONG"/>
        <xsd:enumeration value="FLOAT"/>
        <xsd:enumeration value="BOOLEAN"/>
        <xsd:enumeration value="STRING"/>
        <xsd:enumeration value="OCTET"/>
        <xsd:enumeration value="SEQ_OCTET"/>
        <xsd:enumeration value="SEQ_FLOAT"/>
        <xsd:enumeration value="SEQ_LONG"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="EACCESS_TYPE">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="INLET"/>
        <xsd:enumeration value="OUTLET"/>
        <xsd:enumeration value="INOUTLET"/>
        <xsd:enumeration value="TRANSFER"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="EOWNER_STATUS">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="PRIVATE"/>
        <xsd:enumeration value="UNLOCKED"/>
```

```
            <xsd:enumeration value="LOCKED"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="SYSTEM_TYPE">
        <xsd:sequence>
            <xsd:element name="NAME" type="xsd:string"/>
            <xsd:element name="LOCATION" type="xsd:string"/>
            <xsd:element name="WORKCELLS" type="WORKCELL_TYPE" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="RESOURCES" type="RESOURCE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="DOMAIN" type="ESYSTEM_DOMAIN"/>
            <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="WORKCELL_TYPE">
        <xsd:sequence>
            <xsd:element name="WORKCELL_ID" type="xsd:ID"/>
            <xsd:element name="LOCATION" type="xsd:string"/>
            <xsd:element name="SLMS" type="SLM_TYPE" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="RESOURCES" type="RESOURCE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="SUBCELLS" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="SUPERCELL" type="xsd:IDREF" minOccurs="0" maxOccurs="1"/>
            <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
            <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="RANGE_TYPE">
        <xsd:sequence>
            <xsd:element name="LOW_LIMIT" type="LIMIT_TYPE" minOccurs="0"/>
            <xsd:element name="HIGH_LIMIT" type="LIMIT_TYPE" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ITEM_VALUE_TYPE">
        <xsd:sequence>
            <xsd:element name="ITEM" type="xsd:string"/>
            <xsd:element name="VALUE" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ARGUMENT_TYPE">
        <xsd:sequence>
            <xsd:element name="NAME" type="xsd:string"/>
            <xsd:element name="ARGUMENT_TYPE" type="EVARIABLE_TYPE"/>
            <xsd:element name="DEFAULT_VALUE" type="xsd:string" minOccurs="0"/>
            <xsd:element name="TRANSFER_TYPE" type="ETRANSFER_TYPE"/>
            <xsd:element name="RANGE" type="RANGE_TYPE" minOccurs="0"/>
            <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
            <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="COMMAND_TYPE">
        <xsd:sequence>
            <xsd:element name="COMMAND_ID" type="xsd:ID"/>
            <xsd:element name="NAME" type="xsd:string"/>
            <xsd:element name="ALIAS_NAME" type="xsd:string" minOccurs="0"/>
            <xsd:element name="DURATION" type="xsd:long"/>
            <xsd:element name="CATEGORY" type="ECOMMAND_CATEGORY"/>
            <xsd:element name="TYPE" type="ECOMMAND_TYPE"/>
```

```
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="FORMAL_ARGUMENTS" type="ARGUMENT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="EXCLUSION_LIST" type="ITEM_VALUE_TYPE" minOccurs="0"  maxOccurs="unbounded"/>
        <xsd:element name="SYNC_RESPONSE_DATA" type="ARGUMENT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="CONFIGURATION_COMMANDS" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="REQUIRED_RESOURCES" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PRODUCED_RESCOURCES" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="OUTPUT_PORTs" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="INPUT_PORTs" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="VALUE_TYPE">
    <xsd:sequence>
        <xsd:element name="VALUE" type="xsd:string"/>
        <xsd:element name="TYPE" type="ENUMBER_TYPE"/>
        <xsd:element name="EXPONENT" type="xsd:string"/>
        <xsd:element name="UNIT" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CAPACITY_TYPE">
    <xsd:sequence>
        <xsd:element name="MAX_CAPACITY" type="VALUE_TYPE"/>
        <xsd:element name="MIN_CAPACITY" type="VALUE_TYPE"/>
        <xsd:element name="FILL_STEPS" type="VALUE_TYPE" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="COMPONENT_ID_TYPE">
    <xsd:sequence>
        <xsd:element name="WORKCELL_ID" type="xsd:string"/>
        <xsd:element name="SLM_ID" type="xsd:string"/>
        <xsd:element name="SUBUNIT_ID" type="xsd:string"/>
        <xsd:element name="RESOURCE_ID" type="xsd:string"/>
        <xsd:element name="COMPONENT_CATEGORY" type="ECOMPONENT_CATEGORY"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="OWNERSHIP_TYPE">
    <xsd:sequence>
        <xsd:element name="COMPONENT_ID" type="COMPONENT_ID_TYPE"/>
        <xsd:element name="OWNER_STATUS" type="EOWNER_STATUS"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PORT_TYPE">
    <xsd:sequence>
        <xsd:element name="PORT_ID" type="xsd:ID"/>
        <xsd:element name="X" type="xsd:string"/>
        <xsd:element name="Y" type="xsd:string"/>
        <xsd:element name="Z" type="xsd:string"/>
        <xsd:element name="CONTENT_RESOURCE" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PORT_TYPE" type="EPORT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="ACCESS_TYPE" type="EACCESS_TYPE"/>
        <xsd:element name="CAPACITY" type="CAPACITY_TYPE"/>
        <xsd:element name="OWNERSHIP" type="OWNERSHIP_TYPE"/>
        <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
```

```
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="RESOURCE_TYPE">
    <xsd:sequence>
        <xsd:element name="RESOURCE_ID" type="xsd:ID"/>
        <xsd:element name="RESOURCE_CATEGORY" type="ERESOURCE_CATEGORY"/>
        <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
        <xsd:element name="DANGER_CLASS" type="xsd:string"/>
        <xsd:element name="OWNERSHIP" type="OWNERSHIP_TYPE"/>
        <xsd:element name="IN_PORT_REF" type="xsd:string" minOccurs="0"/>
        <xsd:element name="ACCESS_PORTS" type="PORT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="CURRENT_QUANTITY" type="VALUE_TYPE" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ADMINISTRATIVE_TYPE">
    <xsd:sequence>
        <xsd:element name="NAME" type="xsd:string"/>
        <xsd:element name="PROTOCOL" type="xsd:string"/>
        <xsd:element name="MODEL_NUMBER" type="xsd:string"/>
        <xsd:element name="SERIAL_NUMBER" type="xsd:string"/>
        <xsd:element name="MANUFACTURER_ID" type="xsd:string"/>
        <xsd:element name="MANUFACTURER_NAME" type="xsd:string"/>
        <xsd:element name="SUPPORT_ADDRESS" type="xsd:string"/>
        <xsd:element name="UPDATE_ADDRESS" type="xsd:string"/>
        <xsd:element name="SOFTWARE_VERSION_NUMBER" type="xsd:string"/>
        <xsd:element name="DCD_VERSION" type="xsd:string"/>
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="DIMENSION_TYPE">
    <xsd:sequence>
        <xsd:element name="HEIGHT" type="xsd:long"/>
        <xsd:element name="WIDTH" type="xsd:long"/>
        <xsd:element name="LENGTH" type="xsd:long"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="TRANSLATION_TYPE">
    <xsd:sequence>
        <xsd:element name="XTRANSLATION" type="xsd:long" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="YTRANSLATION" type="xsd:long" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="ZTRANSLATION" type="xsd:long" maxOccurs="1" minOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ROTATION_TYPE">
    <xsd:sequence>
        <xsd:element name="XROTATION" type="xsd:long" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="YROTATION" type="xsd:long" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="ZROTATION" type="xsd:long" maxOccurs="1" minOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LOCATION_TYPE">
    <xsd:sequence>
        <xsd:element name="TRANSLATION" type="TRANSLATION_TYPE"/>
        <xsd:element name="ROTATION" type="ROTATION_TYPE"/>
```

```
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="PHYSICAL_CHARACTERISTICS_TYPE">
    <xsd:sequence>
        <xsd:element name="DIMENSION" type="DIMENSION_TYPE"/>
        <xsd:element name="LOCATION" type="LOCATION_TYPE" minOccurs="0"/>
        <xsd:element name="WEIGHT" type="xsd:long" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SLM_TYPE">
    <xsd:sequence>
        <xsd:element name="SLM_ID" type="xsd:ID"/>
        <xsd:element name="ADMINISTRATIVE" type="ADMINISTRATIVE_TYPE"/>
        <xsd:element name="FUNCTIONALITY" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
        <xsd:element name="SUBUNITS" type="SUBUNIT_TYPE" maxOccurs="unbounded"/>
        <xsd:element name="RESOURCES" type="RESOURCE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PORTS" type="PORT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PRIMARY_COMMANDS" type="COMMAND_TYPE" maxOccurs="unbounded"/>
        <xsd:element name="EXT_MACROS" type="EXT_MACRO_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="DOWNTIME" type="DOWNTIME_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="SYSTEM_VARIABLES" type="SYSTEM_VARIABLE_TYPE" minOccurs="0" maxOc-
curs="unbounded"/>
        <xsd:element name="EVENTS" type="EVENT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SUBUNIT_TYPE">
    <xsd:sequence>
        <xsd:element name="UNIT_ID" type="xsd:ID"/>
        <xsd:element name="ADMINISTRATIVE" type="ADMINISTRATIVE_TYPE"/>
        <xsd:element name="PHYSICAL_CHARACTERISTICS" type="PHYSICAL_CHARACTERISTICS_TYPE"/>
        <xsd:element name="COMMANDS" type="COMMAND_TYPE" maxOccurs="unbounded"/>
        <xsd:element name="PRIMARY_COMMANDS" type="COMMAND_TYPE" maxOccurs="unbounded"/>
        <xsd:element name="EXT_MACROS" type="EXT_MACRO_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="FUNCTIONALITY" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="RESOURCES" type="RESOURCE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PORTS" type="PORT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="SYSTEM_VARIABLES" type="SYSTEM_VARIABLE_TYPE" minOccurs="0" maxOc-
curs="unbounded"/>
        <xsd:element name="EVENTS" type="EVENT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="DOWNTIME" type="DOWNTIME_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="EXT_MACRO_COMMAND_TYPE">
    <xsd:sequence>
        <xsd:element name="COMMANDREF" type="xsd:IDREF"/>
        <xsd:element name="ARGUMENTLIST" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="EXT_MACRO_TYPE">
    <xsd:sequence>
        <xsd:element name="MACRO_ID" type="xsd:ID"/>
        <xsd:element name="NAME" type="xsd:string"/>
        <xsd:element name="CATEGORY" type="ECOMMAND_CATEGORY"/>
```

```
        <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
        <xsd:element name="MACRO_COMMANDS" type="EXT_MACRO_COMMAND_TYPE" minOccurs="0" maxOc-
curs="unbounded"/>
        <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
<xsd:complexType name="EVENT_TYPE">
    <xsd:sequence>
    <xsd:element name="EVENT_ID" type="xsd:string"/>
    <xsd:element name="PRIORITY" type="xsd:long"/>
    <xsd:element name="CATEGORY" type="EEVENT_CATEGORY"/>
    <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
    <xsd:element name="POSSIBLE_EVENT_DATA_VALUES" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="EVENT_DATA_ARGUMENTS" type="ARGUMENT_TYPE" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="SYSTEM_VARIABLES" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="EVENT_REACTION_COMMANDS" type="xsd:IDREF" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="PROPERTIES" type="ITEM_VALUE_TYPE" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="LIMIT_TYPE">
        <xsd:sequence>
            <xsd:element name="RANGE_VALUE_TYPE" type="ENUMBER_TYPE"/>
            <xsd:element name="RANGE_VALUE" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DOWNTIME_TYPE">
        <xsd:sequence>
            <xsd:element name="DOWNTIME_ID" type="xsd:string"/>
            <xsd:element name="CATEGORY" type="EDOWNTIME_CATEGORY"/>
            <xsd:element name="TYPE" type="EDOWNTIME_TYPE"/>
            <xsd:element name="START_TIME" type="xsd:string"/>
            <xsd:element name="DURATION" type="xsd:string"/>
            <xsd:element name="INTERVAL" type="xsd:string"/>
            <xsd:element name="DESCRIPTION" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SYSTEM_VARIABLE_TYPE">
        <xsd:sequence>
            <xsd:element name="VARIABLE_ID" type="xsd:string"/>
            <xsd:element name="DESCRIPTION" type="xsd:string" minOccurs="0"/>
            <xsd:element name="ACCESS_PRIVILEDGE" type="xsd:string" minOccurs="0"/>
            <xsd:element name="CATEGORY" type="xsd:string" minOccurs="0"/>
            <xsd:element name="DATA_TYPE" type="EVARIABLE_TYPE"/>
            <xsd:element name="VALUE_RANGE" type="RANGE_TYPE" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

SCD-Interface IDL Definition 7

7.1 The Complete SCD-Interface IDL Definition

```
#ifndef __SCD_DEFINED
#define __SCD_DEFINED


module SCD {
        interface ICommand;
        interface IEvent;
        interface IPort;
        interface IResource;
        interface IExtMacroCommandList;
        interface IWorkCell;


        struct SAdministrative {
                string model_number;
                string serial_number;
                string software_version;
                string support_address;
                string manufacturer_id;
                string manufacturer_name;
                string description;
                string update_address;
                string dcd_version;
                string protocol;
        };

        typedef sequence <string> SeqString;

        enum ECommandCategory {
```

```
        INIT,
        CONTROL,
        FUNCTION,
        CONFIGURE,
        RECOVERY,
        STATUSREQ,
        MAINTAIN,
        CALIBRATE,
        ADMIN,
        RESULT
};

enum EVariableType {
        LONG_TYPE,
        FLOAT_TYPE,
        BOOLEAN_TYPE,
        STRING_TYPE,
        OCTET_TYPE,
        SEQ_LONG_TYPE,
        SEQ_FLOAT_TYPE,
        SEQ_OCTET_TYPE
};

enum ETransferType {
        INTRANSFER,
        OUTTRANSFER,
        INOUTTRANSFER
};

struct SItemData {
        string item;
        string value;
};

enum ENumberType {
        LONG_NTYPE,
        FLOAT_NTYPE
};

struct SRange {
        ENumberType range_type;
        string low_limit;
        string high_limit;
};

struct SArgument {
        string name;
        EVariableType argument_type;
        any default_value;
        ETransferType transfer_type;
```

```
        string description;
        sequence <SItemData> properties;
        SRange range;
};

enum EAccessType {
        INLET,
        OUTLET,
        INOUTLET,
        TRANSFER
};

enum EOwnerstatus {
        PRIVATE_OWNER,
        LOCKED,
        UNLOCKED
};

enum EComponentCategory {
        SYSTEM,
        WORKCELL,
        SLM,
        RESOURCE,
        SUBUNIT
};

struct SComponentID {
        string workcell_id;
        string slm_id;
        EComponentCategory component_category;
        string subunit_id;
        string resource_id;
};

struct SOwnership {
        EOwnerstatus owner_status;
        SComponentID owner;
};

struct SValue {
        string value;
        ENumberType type;
        long exponent;
        string unit;
};

enum EResourceCategory {
        HARDWARE,
        SAMPLE,
        REAGENT,
```

```
        WASTE,
        SPACE,
        BUFFER,
        UNDEFINED
};

enum ECapacityType {
        FINITE,
        INFINITE
};

enum EDownTimeCategory {
        CLEANING,
        CALIBRATION,
        SOFTWARE_UPDATE,
        HARDWARE_UPDATE
};

struct STranslation {
        long x;
        long y;
        long z;
};

struct SRotation {
        long x;
        long y;
        long z;
};

enum ESystemDomain {
        COUNTRY,
        DEPARTMENT,
        SUBDIVISION,
        LABORATORY,
        ROOM
};

enum EEventCategory {
        ALARM,
        MESSAGE,
        DATA_DIRECT,
        DATA_LINK,
        SYSVAR_CHANGED,
        CONTROL_STATE_CHANGED,
        SLM_STATE_CHANGED
};

struct SSystemVariable {
        string variable_id;
```

```
        string description;
        EVariableType data_type;
        any current_value;
        string category;
        SRange value_range;
};

struct SCapacity {
        SValue min_capacity;
        SValue max_capacity;
        SValue fill_steps;
};

struct SLocation {
        SRotation rotation;
        STranslation translation;
};

struct SGeometricModel {
        any model;
        any access_curve;
};

struct SDimension {
        string height;
        string width;
        SGeometricModel geometric_model;
        string length;
};

struct SPhysicalCharacteristics {
        string weight;
        SLocation location;
        SDimension dimension;
};


typedef sequence<any> SeqAny;

enum ECommandType {
        ATOMIC,
        MACRO
};

enum EDownTimeType {
        ESTIMATED,
        ACTUAL
};

interface IDownTime {
```

```
        attribute string downtime_id;
        attribute string description;
        attribute string starttime;
        attribute EDownTimeCategory category;
        attribute string duration;
        attribute EDownTimeType type;
        attribute string interval;
        attribute string parent_id;

        boolean delete ();

};

enum EPortType {
        DATA,
        MATERIAL
};

interface ISubUnit {
        typedef sequence <ICommand> commands_def;

        typedef sequence <SAdministrative> administrative_def;

        typedef sequence <IEvent> events_def;

        typedef sequence <IPort> ports_def;

        typedef sequence <IResource> resources_def;

        typedef sequence <SItemData> properties_def;

        typedef sequence <IExtMacroCommandList> subunit_external_macros_def;


        attribute string subunit_id;
        attribute string description;
        attribute IDownTime maintenance;
        attribute string parent_id;
        attribute SeqString functionality;
        attribute commands_def commands;
        attribute administrative_def administrative;
        attribute events_def events;
        attribute ports_def ports;
        attribute resources_def resources;
        attribute SPhysicalCharacteristics physical_characteristics;
        attribute properties_def properties;
        attribute SSystemVariable system_variables;
        attribute subunit_external_macros_def subunit_external_macros;
        attribute commands_def primary_commands;
```

```
        void delete ();

};

interface IResource {
        typedef sequence <SItemData> properties_def;

        typedef sequence <IPort> access_ports_def;

        attribute string resource_id;
        attribute string description;
        attribute string danger_class;
        attribute string parent_id;
        attribute SPhysicalCharacteristics physical_characteristics;
        attribute EResourceCategory category;
        attribute SValue current_quantity;
        attribute SOwnership ownership;
        attribute properties_def properties;
        attribute access_ports_def access_ports;
        attribute IPort contained_in_port;
        attribute ECapacityType capacity_type;

        boolean delete ();

};

interface IPort {
        typedef sequence <SItemData> properties_def;

        typedef sequence <IResource> content_def;

        attribute string port_id;
        attribute string x;
        attribute string y;
        attribute SOwnership ownership;
        attribute string z;
        attribute EAccessType access_type;
        attribute string description;
        attribute string parent_id;
        attribute SPhysicalCharacteristics physical_characteristics;
        attribute properties_def properties;
        attribute SValue capacity;
        attribute content_def content;
        attribute SCapacity quantity_range;
        attribute EPortType port_type;

        boolean delete ();

};
```

```
interface ICommand {
        typedef sequence <SCD::ICommand> required_configurations_def;

        typedef sequence <IResource> required_resources_def;

        typedef sequence <IResource> produced_resources_def;

        typedef sequence <IPort> port_inputs_def;

        typedef sequence <IPort> port_outputs_def;

        typedef sequence <SRange> measurement_bounds_def;

        typedef sequence <SArgument> formal_arguments_def;

        typedef sequence <SArgument> sync_response_data_def;

        typedef sequence <SItemData> properties_def;

        typedef sequence <SItemData> exclusion_list_def;

        attribute string commandid;
        attribute formal_arguments_def formal_arguments;
        attribute string name;
        attribute ECommandCategory command_category;
        attribute string alias_name;
        attribute required_configurations_def required_configurations;
        attribute long duration;
        attribute exclusion_list_def exclusion_list;
        attribute properties_def properties;
        attribute string parent_id;
        attribute string description;
        attribute required_resources_def required_resources;
        attribute produced_resources_def produced_resources;
        attribute port_inputs_def port_inputs;
        attribute port_outputs_def port_outputs;
        attribute measurement_bounds_def measurement_bounds;
        attribute ECommandType command_type;
        attribute sync_response_data_def sync_response_data;

        boolean delete ();

};


interface ISLM {
        typedef sequence <IResource> resources_def;

        typedef sequence <ISubUnit> sub_units_def;
```

```
        typedef sequence <SItemData> properties_def;

        typedef sequence <IDownTime> downtime_def;

        typedef sequence <IPort> ports_def;

        typedef sequence <IEvent> events_def;

        typedef sequence <SSystemVariable> system_variables_def;

        typedef sequence <ICommand> commands_def;

        attribute SPhysicalCharacteristics physical_characteristics;
        attribute string slm_id;
        attribute string parent_id;
        attribute resources_def resources;
        attribute sub_units_def sub_units;
        attribute SAdministrative administrative;
        attribute properties_def properties;
        attribute SeqString functionality;
        attribute downtime_def downtime;
        attribute ports_def ports;
        attribute events_def events;
        attribute system_variables_def system_variables;
        attribute IExtMacroCommandList slm_external_macros;
        attribute commands_def primary_commands;

        boolean delete ();

};


interface IEvent {
        typedef sequence <SItemData> properties_def;

        typedef sequence <SArgument> event_data_types_def;

        typedef sequence <ICommand> possible_event_reaction_def;

        typedef sequence <SSystemVariable> system_variables_def;

        attribute string event_id;
        attribute long priority;
        attribute string description;
        attribute string parent_id;
        attribute EEventCategory category;
        attribute properties_def properties;
        attribute event_data_types_def event_data_types;
        attribute SeqAny possible_event_data_values;
```

```
        attribute possible_event_reaction_def possible_event_reaction;
        attribute system_variables_def system_variables;

        void delete ();

};


struct SExtMacroCommand {
        SeqAny argument_values;
        ICommand command_ref;
};

interface IExtMacroCommandList {
        typedef sequence <SExtMacroCommand> commands_def;

        typedef sequence <SItemData> properties_def;

        typedef sequence <SCD::IExtMacroCommandList> ext_macros_def;

        attribute string command_id;
        attribute ECommandCategory category;
        attribute string name;
        attribute commands_def commands;
        attribute string descritpion;
        attribute SeqAny functionality;
        attribute long duration;
        attribute properties_def properties;
        attribute string parent_id;
        attribute ext_macros_def ext_macros;

        boolean delete ();

};

interface IWorkCell {
        typedef sequence <ISLM> slms_def;

        typedef sequence <IResource> resources_def;

        attribute string name;
        attribute string description;
        attribute string location;
        attribute slms_def slms;
        attribute SCD::IWorkCell supercell;
        attribute resources_def resources;
        attribute SPhysicalCharacteristics physical_characteristics;

        void delete ();
```

```
};

interface ISystem {
        typedef sequence <IWorkCell> workcells_def;

        typedef sequence <IResource> resources_def;

        attribute string name;
        attribute workcells_def workcells;
        attribute string description;
        attribute ESystemDomain domain;
        attribute string location;
        attribute resources_def resources;
};

interface ISCDRegistry {

        SCD::IWorkCell create_workcell ();

        SCD::ISLM create_slm ();

        SCD::ISubUnit create_sub_unit ();

        SCD::ICommand create_command ();

        SCD::IExtMacroCommandList create_ext_macro ();

        SCD::IEvent create_event ();

        SCD::IPort create_port ();

        SCD::IResource create_resource ();

        SCD::ISystem find_system ();

        SCD::IWorkCell find_workcell (
                in string workcell_id
                );

        SCD::ISLM find_slm (
                in string slm_id
                );

        SCD::ISubUnit find_subunit (
                in string slm_id,
                in string subunit_id
                );

        SCD::ICommand find_command (
                in string slm_id,
```

```
                in string subunit_id,
                in string command_id
                );

        SCD::IExtMacroCommandList find_macrocommand (
                in string slm_id,
                in string subunit_id,
                in string macrocommand_id
                );

        SCD::IResource find_resource (
                in SComponentID component_id
                );

        SCD::IPort find_port (
                in SComponentID component_id,
                in string port_id
                );

        SCD::IEvent find_event (
                in string slm_id,
                in string subunit_id,
                in string event_id
                );

    };

};

                                #endif
```

SLM Interface IDL Definition 8

8.1 The Complete SLM Interface IDL Definition

```
#ifndef __SLM_INTERFACE_DEFINED
#define __SLM_INTERFACE_DEFINED


module SLM_INTERFACE {


        enum EMainCtrlState {
                POWERED_UP,
                INITIALIZING,
                NORMAL_OP,
                ERROR,
                CLEARING,
                CLEARED,
                SHUTDOWN,
                DOWN
        };

        enum ESubCtrlState {
                SUB_POWERED_UP,
                SUB_INITIALIZING,
                SUB_SHUTDOWN,
                SUB_DOWN,
                SUB_ERROR,
                SUB_CLEARING,
                SUB_CLEARED,
                SUB_ABORTED,
                SUB_ESTOPPED,
                SUB_IDLE,
```

```
        SUB_PROCESSING,
        SUB_PAUSING,
        SUB_PAUSED,
        SUB_RESUMING
};

enum EResultCode {
        SUCCESS,
        REMOTE_CTRL_REQ_DENIED,
        LOCAL_CTRL_REQ_DENIED,
        FORCE_LOCAL_CTRL_FAILED,
        RELEASE_REMOTE_CTRL_FAILED,
        READ_DCD_FAILED,
        WRITE_DCD_FAILED,
        DCD_NOT_AVAILABLE,
        SUBUNIT_UNKOWN,
        DEVICE_HARDWARE_ERROR,
        COMMUNICATION_ERROR,
        TIMEOUT,
        UNSPECIFIED_ERROR,
        SUB_STATE_INCORRECT,
        MAIN_STATE_INCORRECT,
        PAUSE_REQUEST_DENIED,
        TIME_SYNCHRONIZATION_FAILED,
        UNKNOWN_COMMAND,
        TIME_SYNCHRONIZATION_NOT_AVAILABLE,
        WRONG_ARGUMENT_LIST,
        DATA_ID_UNKNOWN,
        INVALID_DATA,
        ACCESS_DENIED,
        EXECUTING_MACRO,
        EXECUTION_STOPPED
};

typedef sequence <octet> SeqOctet;

enum ELocalRemote {
        LOCAL,
        REMOTE,
        AVAILABLE
};

typedef sequence <string> SeqString;


struct SSysVar {
        string variable_id;
        string description;
        string category;
        any value;
```

```
};


struct SSubState {
        string sub_unit_id;
        ESubCtrlState sub_unit_state;
};

typedef sequence<SSubState> SeqSubStates;


struct SLM_RESULT {
        EResultCode result_code;
        string minor_code;
        EMainCtrlState main_state;
        SLM_INTERFACE::SeqSubStates sub_states;
        ELocalRemote lr_mode;
        string message;
};

typedef sequence<SSysVar> SeqSysVar;

typedef sequence<any> SeqAny;

enum ELocalRemote_ArgType {
        LOCAL_CTRL_REQ,
        REMOTE_CTRL_REQ,
        FORCE_LOCAL_CTRL,
        RELEASE_CONTROL
};

enum ECommandType {
        ATOMIC,
        MACRO
};

enum EVariableType {
        LONG_TYPE,
        FLOAT_TYPE,
        BOOLEAN_TYPE,
        STRING_TYPE,
        OCTET_TYPE,
        SEQ_LONG_TYPE,
        SEQ_FLOAT_TYPE,
        SEQ_OCTET_TYPE
};


enum EEventType {
        ALARM,
```

```
        MESSAGE,
        DATA_DIRECT,
        DATA_LINK,
        SYSVAR_CHANGED,
        CONTROL_STATE_CHANGE,
        DEVICE_STATE_CHANGED
};


enum EDataLinkType {
        FILE,
        DB,
        OPERATION
};



interface ITSC_Callback {

        void slm_event (
                in string slm_id,
                in string unit_id,
                in string event_id,
                in SLM_INTERFACE::EEventType event_type,
                in string interaction_id,
                in string priority,
                in SLM_INTERFACE::SLM_RESULT slm_state,
                in SLM_INTERFACE::SeqAny arguments
                );

};


interface ILECI {

        SLM_RESULT init (
                in string unit_id,
                in SLM_INTERFACE::ITSC_Callback callback_ref,
                in SLM_INTERFACE::SeqAny args
                );

        SLM_RESULT estop ();

        SLM_RESULT abort (
                in string unit_id,
                in SLM_INTERFACE::SeqAny args
                );

        SLM_RESULT clear (
                in string unit_id,
```

```
        in SLM_INTERFACE::SeqAny args
        );

SLM_RESULT pause (
        in string unit_id,
        in SLM_INTERFACE::SeqAny args
        );

SLM_RESULT resume (
        in string unit_id,
        in SLM_INTERFACE::SeqAny args
        );

SLM_RESULT shutdown (
        in string unit_id,
        in SLM_INTERFACE::SeqAny args
        );

SLM_RESULT status ();

SLM_RESULT get_SLM_id (
        out string slm_id
        );

SLM_RESULT get_DCD (
        out string xml_dcd
        );

SLM_RESULT local_remote_req (
        in string unit_id,
        in SLM_INTERFACE::ELocalRemote_ArgType req_type
        );

SLM_RESULT synchronize_time (
        in string time_server,
        in string tsc_timestamp
);

SLM_RESULT get_subunit_ids (
        out SLM_INTERFACE::SeqString subunits
        );

SLM_RESULT set_system_var (
        in string unit_id,
        in SLM_INTERFACE::SSysVar sysvar
        );

SLM_RESULT get_system_var (
        in string unit_id,
        in string sysvar_id,
```

```
                out SLM_INTERFACE::SSysVar sysvar
                );

        SLM_RESULT run_op (
                in string unit_id,
                in string interaction_id,
                in SLM_INTERFACE::ECommandType op_type,
                in string op_name,     //Name of the operation to be called.
                in SLM_INTERFACE::SeqAny args,
                out SLM_INTERFACE::SeqAny return_values
                );

        SLM_RESULT get_result_data (
                in string interaction_id,
                in string data_id,
                out any result_data,
                in SLM_INTERFACE::EVariableType data_type
                );

        SLM_RESULT set_TSC_callback (
                in SLM_INTERFACE::ITSC_Callback callback_ref
                );


    };


};

                                #endif
```

ISO Base Units A

A.1 ISO Base Units

meter
litre
gram
second
ampere
kelvin
mole
candela
radian
steradian
hertz
newton
pascal
joule
watt
coulomb

volt
farad
ohm
siemens
weber
tesla
henry
degree_celsius
lumen
lux
becquerel
gray
sievert

Glossary

Terminology

Entity	**Description**
Task	The task is an individual action performed by a SLM on a laboratory device. It may involve the sending or receiving of commands, status information or data.
Task Sequence Controller (TSC)	The Task Sequence Controller receives a job from the user interface level or an integrated laboratory automation system, and creates a list of tasks to be run on a specific available SLM with the correct capabilities for performing the test(s). It hands these instructions to the SLM and monitors the progress of the tasks
Standard Laboratory Module (SLM)	A SLM is the LECIS representation of a laboratory instrument.
Main-Unit	A SLM Main-Unit contains at least one Sub-Unit. The Main-Unit supervises and manages its Sub-Units.
Sub-Unit	A Sub-Unit provides the SLM's functionality.
Device Capability Dataset (DCD)	The DCD is a descriptive representation of the device’s capabilities and properties. It is defined as an XML-Schema.
System Capability Dataset (SCD)	This is the data describing the capabilities of the system as a whole rather than any one device.It is defined as an XML-Schema.

SCDRegistry	The SCDRegistry stores all information pertaining to the laboratory system (SCD). It contains multiple DCDs, and is available to assist in the selection of the correct laboratory device(s) for an operation. The SCDRegistry holds the command set for each device and is referred to by the SLM controlling the device.
Workcell	The Workcell is a logical grouping of laboratory devices by geography and function, allowing for prioritization in the selection of a group of target instruments based on location as well as availability.
UNC	Short for Universal Naming Convention or Uniform Naming Convention, a PC format for specifying the location of resources on a local-area network. UNC uses the following format: "\\server-name\shared-resource-pathname".
ILECI	SLM interface: Short for Laboratory Equipment Control Interface.
Atomic command	A single command that is executed by a Sub-Unit
Macro command	A sequence of atomic commands from one or multiple Sub-Units.
SLM based macro command	A macro, that has been defined with the device specific software. A LECIS interface does not have explicit control over this macro execution.
External macro command	A macro command that is composed of atomic commands, which are defined in the DCD.
System Variable	A property value of the SLM. The TSC can set/get the property values.
Event	An Event is a response from the SLM to the TSC.
Error-Event	An Error-Event contains information about an error condition of the SLM.
Status-Event	A Status-Event contains information about the control status of the SLM.
Data-Event	A Data-Event contains result data or a link to the result data.