

Abstract

This document provides a discussion of the strengths and limitations of the LECIS standard as discovered during, and restricted to a narrow window of the Userspace Corporation implementation issues for its instrument routers. Userspace has used the protocol aspects of LECIS specifically for the communication grammar between its router and the controller. We wanted to provide an easy instrument function interface within a process perspective. This meant moving away from a CORBA implementation to a small-footprint RPC based, embedded implementation that was XML ready. The middleware Userspace chose was web- and network-centric. Integration to various middle-ware technologies (BEA Weblogic, IBM WebSphere, Silverstream, etc) and data warehouses was an important consideration.

The following were important considerations during the design phase:

- Multiple instrument (and manufacturer) connectivity
- TCP implementation and network management
- Security
- device management (with human intervention)
- status and error notification and management
- programming language independence
- process templates (in an SOP context)
- real-time messaging and data management
- scalability and XML compatibility
- API portability

Userspace gathered requirements for its instrument integration platform during extensive discussion with scientists and technicians who use various classes of laboratory instruments in genomics and proteomics research. For the initial implementation of the platform, API portability and a small-footprint real-time implementation was critical.

Userspace chose to use a customized version of the LECIS protocol over other potential standards: AUTO3P, IML, etc. Userspace modified the LECIS protocol to use a much smaller LECIS message exchange, leaving the acknowledgement and function parameters (Remote Procedure Call -- RPC based) to the TCP stack.

To gather a critical mass for LECIS usage and adoption, we would recommend that the supporters of the standard further develop the implementation layer, since the greater acceptance of the standard will be driven in part by the device driver community within the instrument manufacturers. However, the needs of end-users of instrument functionality and device driver writers are very different. The former is very focused on custom solution development, cross-platform integration, and exerting low-level control over their instruments. The latter is concerned about usability, vertically integrating their suites of instruments, and of attracting and protecting market share. Yet in addition, instrument platform development is often very discontinuous and the result of technology acquisition. These are at times mutually exclusive needs. However, we hope that going forward, both become more concerned with integration and its costs. We think that future versions of LECIS can be used to bridge and satisfy these disparate concerns.

Introduction

LECIS is the Laboratory Equipment Control Interface Specification. It is a device neutral protocol for instrument control and data acquisition. The documents available for LECIS are:

- 1) The LECIS specification: protocol document which specifies the message format, state transitions etc. The protocol also implements the generic functionality needed by any instrument like device locking/ unlocking and device acquisition/release.
- 2) The Device Capability Dataset (DCD) specification
 - Physical details like the Manufacturer, Serial number, Model number and resources needed.
 - Logical details like data ports to access the firmware or LECIS wrapper controlling the instrument.
 - Commands available in an instrument.
 - Parameters for commands.
 - Expected responses for commands.
 - Error messages for the commands.
 - The notification events that the instrument provided.
- 3) The LECIS Implementer's Guide.

The three legs of the stool: manufacturer (device driver writer), systems integrator and end-user

LECIS versus TCP/IP

Comparing LECIS to TCP/IP is instructive. TCP/IP is also a protocol standard that does not talk about implementation at all. This has resulted in many differing APIs to TCP. We have at least 2 different implementations for Unix. The Berkeley sockets interface and the X/Open Transport Interface (XTI). Microsoft too has its WinSock network interface. Each provides a different programming API. What this means is that while all these different TCP/IP implementations can talk to each other, writing a portable program which will work on all of them is not be easy.

However, the LECIS Implementer's Guide recommends TCP. LECIS does it's own acknowledgements (in fact every interaction – a message -- needs an acknowledgement) and retransmissions, eschewing similar capabilities provided by some of the underlying transports.

Portability

Although LECIS does specify what the protocol messages look like, the meaning of the transported data and when the messages are appropriate, it does not dictate an implementation (*It's not a good idea for any protocol to specify an implementation*). This will lead people to implement the standard, but provide a different programming interface each time around (which is problematic for portability).

Message dialog

The other issue with LECIS stems from the amount of protocol messages one has to exchange to get a small command executed. 14 messages (in our implementation) for one command seems excessive. LECIS seems to have been designed with an unreliable transport mechanism (*UDP is an example*) in mind. A standard LECIS implementation can have as much as 56 messages for one command.

Limitations of CORBA

The official implementation is based on CORBA (they probably wanted to be language independent.) CORBA based implementations create several issues:

- There aren't many CORBA and IDL resources available (programmers, implementers, maintainers)
- CORBA is an enterprise-wide implementation, restricting LECIS only to larger companies
- Technology is moving to Web-Services (although WSDL is closer to IDL)
- Acceptance of CORBA by the instrumentation industry (and their device driver staff)
- Ease of implementation and interfacing tools (moving away from a low-level LECIS implementation to a high level instrument functionality implementation)
- CORBA is fat (from an object baggage perspective), slow and expensive to implement
- Small installed base

What does 'LECIS compliant' mean?

Userspace has built a 'stripped down' version. Other players are customizing their own versions as well. This is good in that there appears to be gradual movement towards recognition of a common solution to the problem and general agreement around the form of the solution.

A product claiming 'LECIS compatibility' may not guarantee that a developer will be able to produce portable code for all implementations.

Look at POSIX

The "holy grail" can only be achieved if there is an implementation standard. What is needed is an API standard like POSIX for which programs can be written. Note that while there is a POSIX standard for an OS API and utilities, there is as yet no standard for networking. Most operating systems are POSIX compliant.

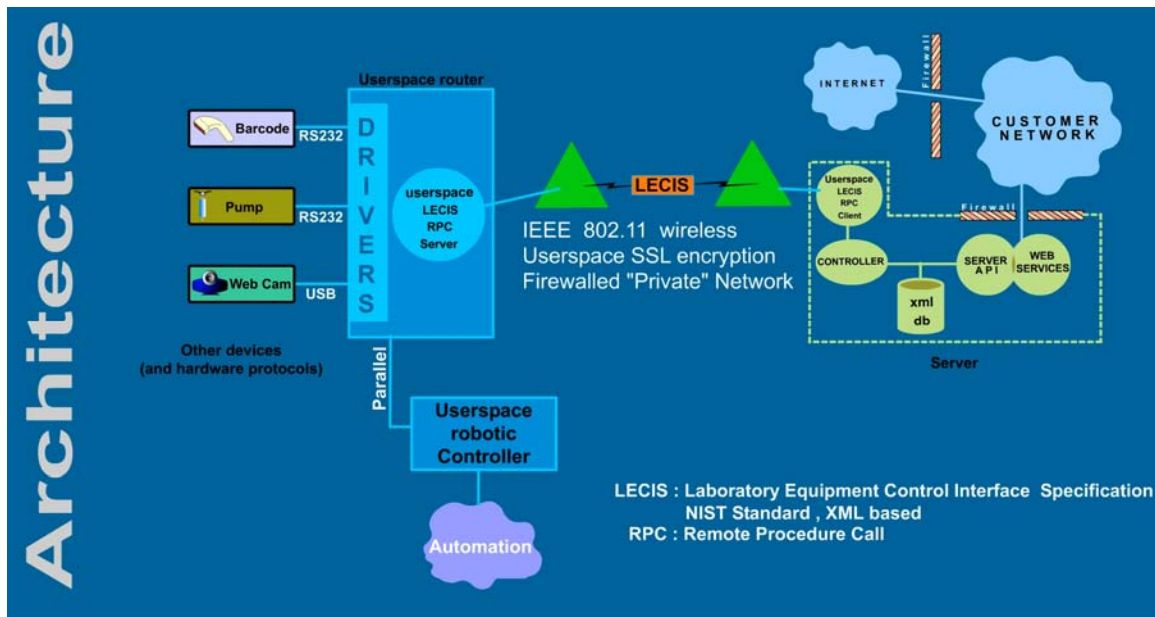
For more information on POSIX visit:

<http://standards.ieee.org/regauth/posix/>

For more information on LECIS visit:

<http://www.lecis.org/documents/LabAutomation2002/PfizerLECISPoster.pdf>

The Userspace implementation of LEICS



Commonality with LEICS

LEICS has functionality that has been adopted in the Userspace LEICS-based protocol.

- Permit calls to an ICP (Instrument Control Program) when another is pending. Currently calls to the same ICP are serialized and the ICP will wait till the call is completed before running the next one. This is part of the Userspace ICL (Instrument Control Layer).
- Provide asynchronous notification for unexpected events
- Use the message architecture minimizing the number of ACK (acknowledgement) messages per handshake

Security, Acknowledgements and an Inference engine

Userspace has made this simpler by making assumptions, some of it influenced by the requirements for the product:

- Use of TCP as the underlying transport. This is mandated by SSL (Secure Sockets Layer). SSL does not work over an unreliable transport like UDP.
- Use of TCP also means that we don't have to worry about acknowledgements for everything.
- The ICP is smarter than in the LEICS model where the smarts reside in the Task Sequence Controller (TSC). Protocol messages (RPC calls in our case) can be made much simpler, consisting of just a sequence number, the RPC calls and parameters. Data processing and parallel-tasking can be added to the 'smarts'.

Clients and servers: distribution of tasks

One difference between LEICS and the Userspace XML-RPC (Remote Procedure Call) protocol is that in LEICS, the instrument control program is a client which connects to a controller (TSC — Task Sequence Controller) which is the server, while we have the Controller which is a client connect to the programs which run on the Userspace routers and request services. With the Userspace architecture, multiple controllers can connect to the Userspace routers. This is not possible in the LEICS design. In fact the protocol forbids such a connection from being made.

Each control program or SLM (Standard Laboratory Module) can connect to only 1 TSC. This is a problem for LEICS SLMs who have lost their TSC. We don't know how LEICS handles this disconnect, but if the TSC is shifted from the IP configured into the SLM, the SLMs will have to be physically reconfigured somehow.

Network problems: TCP socket disconnect

Contrast this to a server running on the Userspace routers, where a TCP disconnect due to the controller dying will just cause the server to close that socket, open another and wait for a fresh connection.

Another relates to the fact that the SLMs being clients need to have networking code built into them to connect to the TSC. In our case, the networking code is all in the multiplexer (part of the Userspace ICL layer) and the ICPs talk to the multiplexer and not to the controller.

Error Handling

Another LECIS issue is the actual handling of device errors (say from the device firmware or state).

In the Userspace implementation of LECIS, error handling is RPC and Process algorithm based. RPC reports the error in a standard way and the Process algorithm acts upon it as defined by the user. Task sequencing is done by the Process algorithm (since the process can include multiple instruments and the human interface).

The current version of the Process algorithm is an XML based schema with conditional inputs. A graphical paradigm with data grammar has been built.

An example Userspace implementation

```
# A sample LECIS-RPC process file for a LM520 barcode scanner
```

```
#
```

```
# create LECIS RPC types
```

```
#prop (process protocol based on process algorithm)
```

```
lm520 = rpc("prop://controller1/mrna.userspace.com/lm520")
```

```
#check if scanner is present
```

```
if not lm520.check()
```

```
    print lm520.errcode, lm520.errstr
```

```
    error_handling()
```

```
# read scanner 5 times
```

```
while (count != 5)
```

```
if not lm520.read()
```

```
    print lm520.errcode, lm520.errstr
```

```
    error_handling()
```

```
else
```

```
    print lm520.rc
```

Conclusion

After mentioning the issues with the implementation of LECIS, we must mention that LECIS has areas that are indispensable and will need adoption in any instrument protocol.

Most instruments in a laboratory are part of a process. It would be to the advantage of LECIS adoption to define the process aspects of instrument functionality. The Human interface is most often the bottleneck, and the decision point within a process.

What remains to be seen is the critical mass within the large instrument manufacturers to provide support and tools for the LECIS standard.