



SIIA

Standard for the Implementation of Interoperable Applications

**Version 1.0
2.5.2000**

LNO AK II

LNO-Sekretariat
c/o TEMA
Junkerstr. 77
D-52064 Aachen
Deutschland
Tel.: +49-241-88-970-0
Fax.: +49-241-88 970-42
info@tema.de



SIIA

Standard for the Implementation of Interoperable Applications

Version 1.0

2.5.2000

Thomas Rauscher

rauscher@ict.tuwien.ac.at

The members of the Working Group AKII are:

| | |
|----------------------|--|
| Noel Bonné | noel.bonne@philips.com |
| Dietmar Dietrich | dietrich@ict.tuwien.ac.at |
| Thomas Frank | th.f@nodus-online.de |
| Bruno Hilkens | Bruno.Hilkens@jci.com |
| Ulrich Howorka | Ulrich.Howorka@Germany.Honeywell.de |
| Klaus Kabitzsch | kabitzsch@iis.inf.tu-dresden.de |
| Jan Kirk | j.kirk@elka.de |
| Markus Kissel | markus.kissel@de.sibt.com |
| Stefan Klimmer | stefan.klimmer@honeywell.com |
| Nils Meinert | nils.meinert@de.sibt.com |
| Thomas Rauscher | rauscher@ict.tuwien.ac.at |
| Horst Sachs | Horst.Sachs@germany.honeywell.com |
| Wilhelm Schluckebier | Wilhelm.Schluckebier@jci.com |
| Hermann Schwagmann | hermann.schwagmann@mail.weidmueller.de |
| Martin Trunk | entwicklung@reko-electronic.de |



Mail: j.kirk@elka.de

Web: <http://www.elka.de>

ICT Institut für
Computertechnik

Mail: rauscher@ict.tuwien.ac.at

Web: <http://www.ict.tuwien.ac.at>

MOELLER 

Mail: hermann.schwagmann@weidmueller.de

Web: <http://www.moeller.net>



PHILIPS

Mail: noel.bonne@philips.com

Web: <http://www.philips.com>

SIEMENS
Landis & Staefa Division

Mail: markus.kissel@de.sibt.com

Web: www.sibt.de

Honeywell

Mail: Horst.Sachs@germany.honeywell.com

Web: <http://www.honeywell.com>

**JOHNSON
CONTROLS**

Mail: Wilhelm.Schluckebier@jci.com

Web: <http://www.jci.com>

Nodus

ENERGIE- UND AUTOMATISIERUNGSTECHNIK

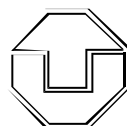
Mail: th.f@nodus-online.de

Web: <http://www.nodus-online.de>

REKO®
electronic

Mail: entwicklung@reko-elektronic.de

Web: www.reko-elektronic.de



TECHNISCHE
UNIVERSITÄT
DRESDEN

Fakultät Informatik

Mail: kabitzsch@iis.inf.tu-dresden.de

Web: <http://iis141.inf.tu-dresden.de/tis>

Foreword

In the early nineties, the US based company Echelon provided the 'world-wide system-development community' with a complete new set of ideas, accompanied with a comprehensive set of tools. It was the time that this world was ready for the next step down from the Mainframe - PC - PLC ladder to the Local Operating Network (LON). This technology made it possible to distribute the intelligence right into the hearts of sensors and actors, meanwhile abstracting the application from communication, and also allowing communication over lots of different media. The concept of network variables was a breakthrough, and context typing them (SniViTs) opened the opportunity to interoperability - promising devices from different manufacturers and applications to work together.

It was a few years later, and a lot of discussions between several manufacturers in a London airport hotel, that the concept of objects for LonWorks was born, a second milestone in the LonMark history. An object is being a software representation of real a world instance, an object with information sinks and sources, methods and configuration properties. Initially there were 3 main object types (for people out of the automation business not surprisingly) a sensor, an actuator and a controller.

All the above led to the first LonMark standard, to be consolidated at the 'object feast' held in Silicon Valley. The meeting was a next milestone in the history of LonWorks. A consensus was reached on the next step: being the definition of 'Functional Profiles'. A functional profile is in fact an extension of an object, allowing it to be application oriented and allows the possibility to embed in the same profile a multitude of the 3 basic objects (remember sensor, actuator and controller). It was not an easy step - I remember the participants receiving after the meeting a small boxing glove.

Today, again a few years later, a lot of functional profiles are defined and available, already quite a lot of projects are realized with LonMark products, but even more with products that are not LonMark-ed. Problems primarily due to the lack of tools and interoperable devices-, applications. During the past LonMark history already several attempts were made to try to specify what is known as the integrated room control. Until today no unified solution is defined. It is here that the document you are holding could be a guideline on how to solve the definition for interoperable application problems.

It were the members of the German Lon Users that saw this lack of specification as a treat for the future of LonMark. Within the organization a taskforce was founded, being a co-operation of major players in the building automation industry and a few universities. The taskforce goal was to get the horizontal integration (inter industry) up and running. Also this taskforce, identical to earlier LonMark attempts had a very difficult start. It was only when an academic approach was taken to analyze the LonMark progress until today that clarity was reached, and the next step in the so desired horizontal integration could be set.

Although the document is focused on the building industry, the members of the task force are convinced that the fundamentals used are a general benefit to the LonMark society. It is from this point of view that we challenge all LonMark friends to comment on the document as to make it more complete. We also ask the owners of the LonMark standard to see the benefits this document can bring to the society, and may we hope, adopt large portions and structure into the LonMark specification.

The LNO taskforce will not see this document as an end of the proceedings, but as a start to cross new frontiers, uncover new applications, for the benefit of our customers.

Meantime I wish lots of interesting reading.

25.2.2000

Noël Bonné

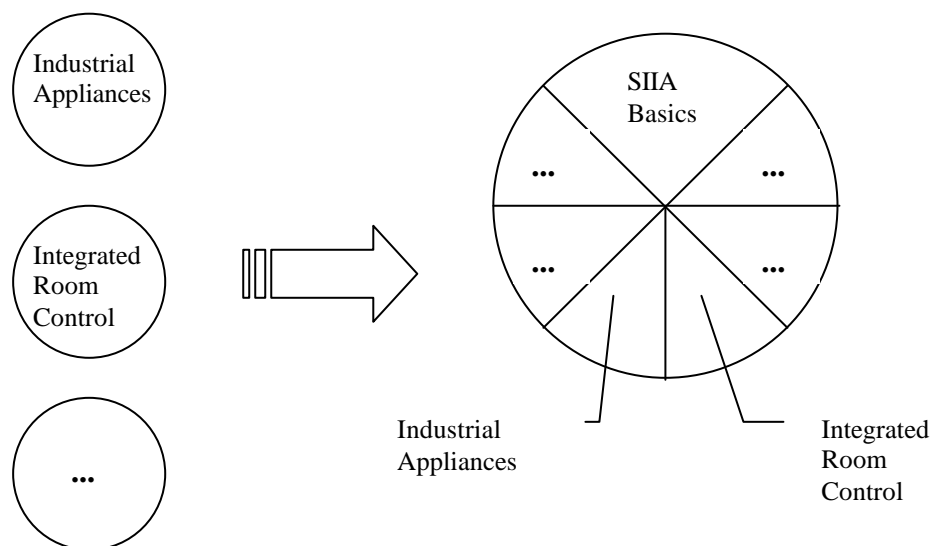
Preface

In 1994, the LonMark Association was founded to provide a common discussion and development forum. The mission of the LonMark Association is the easy integration of LonWorks based control systems, even if the devices come from different vendors. Over 200 member companies engage in this common goal nowadays.

When LonWorks appeared in the automation market, many of the currently installed and planned systems were fiction. Today, applications with more than ten thousand nodes are not fiction anymore.

The original LonMark interoperability guidelines have proven their practicability. Anyhow, there are reports from system integrators that the integration of large systems with current tools is not as easy as it is supposed to be. On the one hand, there is an ongoing tools discussion dealing with how network management systems should look and feel. On the other hand, there is little work done on how control networks and their application can be described although this a rather important prerequisite for discussing how these systems are presented by tools.

The authors feel, that the evolution of the interoperability guidelines is an important factor for efficient network design and management. The interoperability guidelines concentrate on single device interfaces, not on network applications. Since profiles are often derived from existing devices, their definition emphasizes device properties and not possible roles in a distributed application. The interoperability guidelines lack the notion of network and application structures. Due to this gap, device designers and network integrators have a different views on profiles. In other words, the design steps from a profile down to an application follow a top-down design from a defined interface definition to a programming language. The design steps for a network application lack a common application specification method.



SIIA theory and practice

Current challenges are not only large systems, but also systems containing vertical and horizontal integration. Horizontal integration means to integrate devices from the same industry area to a working system. Vertical integration copes with different levels of the control hierarchy, for example in building automation systems. Both, interoperability between

devices from the same industry and devices from different industries must be achieved. Future systems must avoid having different solutions from different industries for the same problem. Several aspects of a single device must be maintained consistently in a model to fulfil these requirements.

This document provides a first attempt to define a object model in which profiles are not only distinctly defined, but are also put into relationship to other profiles. As a result, more abstract profiles correlating to different system aspects can be defined and used for application development. However, compatibility with existing profiles and methods is one key aspect of this work.

This paper is accompanied by a Microsoft Access database which contains the raw profile data gathered during our work. Parallel to this work, some other papers coping with practical interoperability aspects are written and will be published by the LNO.

Vienna, April 2000

Thomas Rauscher

Table of Contents

| | | |
|----------|---|-----------|
| 1 | <i>Introduction</i> | 13 |
| 1.1 | Interoperability | 13 |
| 1.2 | Current Situation | 14 |
| 1.3 | Outline | 14 |
| 2 | <i>LonMark</i> | 15 |
| 2.1 | Overview | 15 |
| 2.2 | Network Variables | 16 |
| 2.3 | SNVTs and SCPTs | 16 |
| 2.3.1 | SNVTs | 16 |
| 2.3.2 | SCPTs | 17 |
| 2.4 | LonMark Objects | 18 |
| 2.5 | LonMark Functional Profiles | 21 |
| 2.6 | Profile Implementation | 21 |
| 2.6.1 | Device Design | 21 |
| 2.6.2 | Object Design | 22 |
| 2.6.3 | Network Variable Implementation | 22 |
| 2.6.4 | Configuration Property Implementation | 22 |
| 2.6.5 | The Implementation Behind the Interface | 22 |
| 2.7 | Profile Usage | 23 |
| 3 | <i>Profile Models</i> | 25 |
| 3.1 | Motivation | 25 |
| 3.1.1 | Audience | 25 |
| 3.1.2 | Objectives | 25 |
| 3.1.3 | Intra-Industry Interoperability | 26 |
| 3.1.4 | Inter-Industry Interoperability | 27 |
| 3.1.5 | Network Management | 27 |
| 3.1.6 | Compatibility | 28 |
| 3.2 | General Services | 28 |
| 3.2.1 | Service Definition | 28 |
| 3.2.2 | Service Aggregation | 28 |
| 3.2.3 | Service Implementation | 28 |
| 3.2.4 | Example | 29 |
| 3.2.5 | Integrated Room Control Example | 29 |
| 3.3 | Network Variable Models | 30 |
| 3.3.1 | Introduction | 30 |
| 3.3.2 | Shared Memory Model | 30 |
| 3.3.3 | Event Model | 31 |
| 3.3.4 | Functional Model | 32 |
| 3.4 | Network Variable Type Classification | 32 |
| 3.4.1 | Hierarchy | 32 |
| 3.4.2 | Abstract Types | 33 |
| 3.4.3 | Arithmetic Types | 34 |
| 3.4.4 | Enumeration Types | 34 |
| 3.4.5 | Structural Types | 35 |
| 3.4.6 | Context Definition | 35 |
| 3.4.7 | Other Classification Schemes | 35 |
| 3.5 | Major Device Classification | 36 |

| | |
|---|-----------|
| 3.6 Applications and LonMark Objects | 39 |
| 3.7 Structured Hierarchy | 40 |
| 3.7.1 Overall Design | 40 |
| 3.7.2 Inheritance | 41 |
| 3.7.3 Root Class Layer | 44 |
| 3.7.4 Signal Direction Layer | 44 |
| 3.7.5 Signal Type Layer | 46 |
| 3.7.6 Industry-Independent Profile Layer | 48 |
| 3.7.7 Profile Layer | 48 |
| 3.7.8 The Manufacturer-Dependent Layer | 49 |
| 4 Analysis Results | 51 |
| 4.1 Existing Profiles | 51 |
| 4.1.1 Released Profiles | 51 |
| 4.1.2 HVAC | 51 |
| 4.1.3 Generic Input/Output | 51 |
| 4.1.4 Energy Management | 52 |
| 4.1.5 Lighting | 53 |
| 4.1.6 Motor Control | 53 |
| 4.1.7 Sensors | 53 |
| 4.1.8 Refrigeration | 53 |
| 4.1.9 Fire & Smoke Detection | 53 |
| 4.1.10 Industrial | 54 |
| 4.2 Profile hierarchy | 54 |
| 4.3 Natural Hierarchy | 55 |
| 4.3.1 Introduction | 55 |
| 4.3.2 Sensor Devices | 55 |
| 4.3.3 Fire Alarm Devices | 56 |
| 4.3.4 HVAC Devices | 57 |
| 4.4 Profile Database | 57 |
| 5 Current Problems | 59 |
| 5.1 Overview | 59 |
| 5.2 Missing Aggregates | 59 |
| 5.3 Fat vs. Thin Interfaces | 59 |
| 5.4 Missing Profiles | 61 |
| 5.5 SNVT Ambiguities | 61 |
| 5.5.1 Problem | 61 |
| 5.5.2 Digital Data | 61 |
| 5.5.3 Discrete Data | 61 |
| 5.5.4 Analog Data | 62 |
| 5.6 Bundled Bindings | 62 |
| 5.7 Higher-Level Designs | 63 |
| 5.8 Specification Errors | 64 |
| 6 Conclusion | 65 |
| <i>Literature</i> | 67 |

1 Introduction

1.1 Interoperability

Interoperability is an often-heard term nowadays. There are interoperable products, devices, tools, systems and profiles. For this work we need to separate between the marketing aspect and the development aspect of the term interoperability. Definitions for interoperability can be found in several LonWorks related publications.

[LonM731, p.1-3] defines interoperability as

A condition that ensures that multiple nodes (from the same or different manufacturers) can be integrated into a single network without requiring custom node or tool development.

[LonT97, p. 266] defines interoperability conforming to IEC¹ as

Interoperability is the capability of two or more devices, which may be produced by different manufacturers, to participate in one or more distributed applications. The devices must have the same view on the application parts of the other devices. The variable and application specifications have to ensure that exchanging a device by a similar one does not require modifications for retaining the functionality of the distributed application. Exchanging a device may change the dynamic behavior of the distributed application.

These definitions have some remarkable statements and implications:

- Interoperability is a property of two or more devices that are utilized in a distributed application without having to customize them. Interoperability is not a property of a single device, so the term “interoperable device” does not exist at all. When two devices (A,B) are interoperable, they can be non-interoperable to a third one (C). If the term “interoperable device” is used in this work, it refers to a device that can take part in a distributed application.
- Interoperability has to be regarded with respect to an application class which defines the interpretation of variables and shared data. For example, each realized building automation application belongs to the class of building automation applications. An application class typically has more than one application, so it is interesting whether the device can be used in an interoperable way within the whole application class.
- Since interoperability is a property of a set of devices, it is a property of the implementation of these devices. Automated tools can check whether the device interfaces are done correct, which is called a conformance test. Automated tools cannot check for interoperability, since different implementations can be conformant, but still cannot work together. Guidelines can simplify building interoperable devices by defining how things should be done, but they cannot assure that things are done in that way. Further, guidelines are generally not complete, so there are freedoms that might lead to non-interoperability.

¹ IEC TC65 SC65C WG7

1.2 Current Situation

The number of large and heterogeneous applications is increasing continuously [MT2499] and LonWorks is expected to be one of the key players in this market. Many manufacturers participate in the market of LonWorks based nodes. This leads to a huge amount of different node types and many opportunities to solve automation problems with LonWorks.

The increasing number of manufacturers makes it easier to disagree on how to implement a particular service. Since different interpretations do not show up until two devices are integrated in a single distributed application, integrators face problems which are hard or even impossible to solve after the purchase of the devices. This work tries to initiate new activities for making the design and utilization of new functional profiles simpler.

Although interoperability is a property of a set of devices, some basic necessary conditions for interoperable devices can be specified. The LonMark interoperability guidelines are an attempt to capture these conditions. This work takes a closer look on the interoperability guidelines and presents new approaches that describe the specification of functional profiles more precisely.

The need for interoperability forces different manufacturers to agree on a common interpretation on how to describe systems and devices. Customers benefit by being able to choose from a large amount of different devices. Since interoperability poses some constraints on the implementation of a device, manufacturers lose an opportunity to diversify their products from those of their competitors. Interoperability guidelines should specify how manufacturers can make extensions to their devices without losing interoperability.

Integrators take individual devices and connect them to distributed applications. Installation efficiency and costs are correlated to the way how devices are specified. The decomposition of the application specification to the required nodes and the market research can be simplified when device data, such as interfaces and special features, are documented in a machine-readable way. This work shows how a relational database can be used to make the communication between integrators and manufacturers simpler.

Currently, there is an intensive discussion on network management tools. It would be advantageous to coordinate the further developments in profile technology and tool development. The quality of network management depends on the amount of information available about the devices.

1.3 Outline

The organization of the chapters is as follows:

Chapter 2 summarizes the key features of the LonMark Application Layer Interoperability Guidelines. Readers familiar with the guidelines can skip this chapter.

Chapter 3 presents several extensions to LonMark, further specification methods and a connection to the object-oriented paradigm. The core of this chapter copes with type and profile hierarchies which provide a structured extension to the classical flat LonMark profile model.

Chapter 4 takes a closer look on currently existing profiles and applies the theoretical basics of chapter 3 on already defined profiles.

Chapter 5 provides a list of problems discovered throughout this work. Each problem is described, the core problem and known workarounds are discussed.

Chapter 6 summarizes the results of this work and gives an outlook on our future work.

2 LonMark

2.1 Overview

This section gives a short overview over the concepts of the LonMark Application Layer Interoperability Guidelines. The purpose of this overview is to describe the LonMark elements that are discussed in this paper. For a more detailed description, see [LonM731], p.2-1ss.

The LonMark Application Layer Interoperability Guidelines provide a framework for specifying the data and configuration interface of LonWorks nodes. The LonMark Layer 1-6 Guidelines [LonM1630] are not in the scope of this work. They define channel and protocol parameters for LonMark compliant nodes.

The Application Layer Guidelines provide

- Syntactic Elements
- Semantic Elements
- Structural Elements
- Documentation Elements

These elements are realized by SNVTs, SCPTs, LonMark Objects and Functional Profiles.

SNVTs² are type definitions specifying the representation of various physical and abstract data elements. SNVTs can represent physical data, like temperatures, lengths, etc. Each physical SNVT has a certain defined range and resolution. SNVTs can be enumerations or structured data representing more complex or specialized data, too. SNVTs are listed in [SNVT]. If there is no suiting SNVT for an application, a new SNVT can be requested from LonMark. The SNVT proposal is reviewed, and if accepted, gets a SNVT index.

SCPTs³ are similar to SNVTs, but they concentrate on application configuration, not on data exchange. SCPTs are listed in [SCPT]. The representation of SCPTs is often compatible with those of SNVTs. SCPTs are written during the configuration process. They can be realized as network variables or as a file which is downloaded at once using the LonMark file transfer.

A LonMark object is a collection of network variables and configuration properties building the interface of a particular device or subdevice. The parts of a LonMark object can be either mandatory or optional. Mandatory parts must exist in each implementation while optional parts can be added if necessary. LonMark objects can serve several purposes: The node object is responsible for controlling other objects. All other objects can be either sensors, actuators or controllers. Sensors have hardware inputs and output network variables. They import physical conditions into the network image. Actuators have hardware outputs and input network variables. They export the network image to the real world. Controllers are processing devices that process data taken from input network variables and offer processed data using output variables. The interaction of a control network, its sensors, actuators and controllers is shown in Fig. 2.1.

² Standard Network Variable Type

³ Standard Configuration Property Type

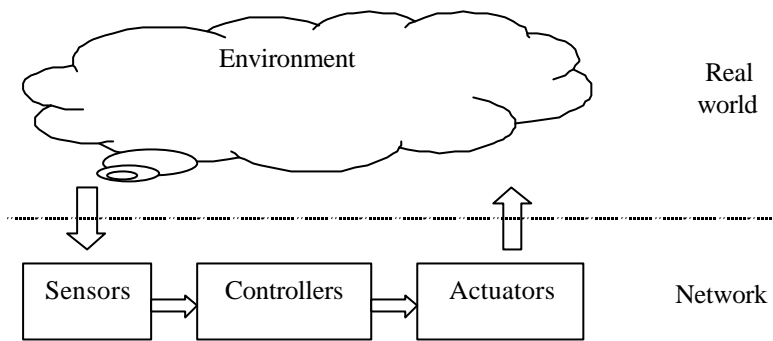


Fig. 2.1: Control networks and their environment

A functional profile specifies a LonMark object for a special device, e.g. a light switch. It defines which network variables are contained in the object and which purpose and behavior they have. Functional profiles are proposed and reviewed by LonMark members. After a review period, a proposed LonMark profile is published for public use.

2.2 Network Variables

Network variables are the fundamental communication concept for LonWorks and LonMark compliant nodes. Network variables are special variables of the Neuron-C programming language. They act like normal global variables, but the firmware watches write operations to network variables and sends new values to listening nodes in the network.

Network variables can be either output or input network variables. Output network variables propagate their updates to bound input network variables. The updates are sent automatically by the firmware, the programmer does not need to be aware of the network. Input network variables trigger a special event in the receiving node upon which the application can react. Besides the automatic transfer mechanism, the values of an output network variable can be explicitly polled.

Each network variable can have a self-documentation string which describes the network variable in more detail. Self-documentation makes a node more independent on written documentation.

Network variables can have an arbitrary Neuron-C type, including enumerations, structures and unions. Their maximum length is 31 bytes. Interoperable devices must agree on the data structure. LonMark compliant nodes do this by using standardized types, called SNVTs and SCPTs.

2.3 SNVTs and SCPTs

2.3.1 SNVTs

For an interoperable communication, it is necessary to agree upon a common data representation. The ISO/OSI reference model implements this functionality in the presentation layer (layer 6). Well known presentation layer specifications are ASN.1⁴ [ASN87] or XDR⁵ [XDR95].

⁴ Abstract Syntax Notation One

⁵ External Data Representation

The LonTalk protocol does not use a certain data representation language, but allows a network variable to store an 8-bit SNVT identifier. The definition and semantics are defined in another document, the SNVT Master List [SNVT] that is published by the LonMark Association (<http://www.lonmark.org/products/guides.htm>).

SNVTs can be classified into three major categories:

1. Arithmetic SNVTs represent scalar values (with or without physical units). Each of them has a range, a resolution and an SI unit⁶. They are represented by an integer or a float variable. Range and resolution are correlated by the variable size. Most physical data types have several types coping with different ranges and resolutions. Some types have a value indicating out-of-range or error conditions. Arithmetic SNVTs can be used by a wide range of devices, since they are meaningful to many industry areas.
2. Enumeration SNVTs are used for indicating device states. They use an 8-bit enumeration type for their representation. The meaning of each value is part of the SNVT definition. Most enumeration SNVTs are used for a particular device (e.g. HVAC or node object). Another common usage for enumeration types are command variables. Every enumeration value corresponds to a certain command that is executed when the network variable is updated. The use of enumeration-type SNVTs is mostly limited to a certain industry area.
3. Structured SNVTs are defined by Neuron-C structure definitions. Like enumeration SNVTs they are dedicated to a particular device type in most cases. They are used whenever several data items must be transmitted with a single update message.

Fig. 2.2 shows a class diagram which represents the different SNVT types.

This classification shows that SNVTs are specified at very different semantic levels. Some definitions are rather simple and can be used by a large number of devices, while others are very specialized to a certain industry area or even to a certain device type.

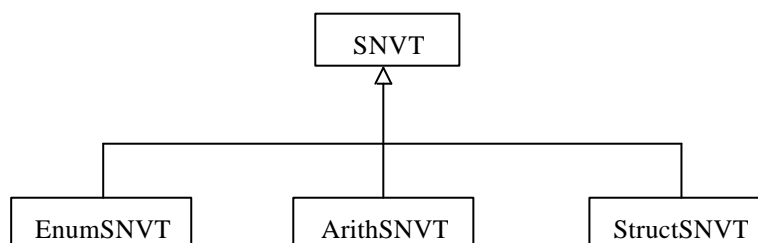


Fig. 2.2: Abstract SNVT classes

The 8-bit SNVT identifier will be a problem when many new SNVTs are requested. Today, around 140 SNVTs are registered. Since each enumeration and each structure requires a new index, there may be the need to extend the Neuron-Chip structures to a larger SNVT index type.

Network management tools use the SNVT index to check whether two network variables can be bound. If and only if they match, the binding will yield a communication relationship guaranteeing that the communication partners agree on the data interpretation.

2.3.2 SCPTs

Most of what has been said for SNVTs is also valid for SCPTs. The main difference between SNVTs and SCPTs is their semantics. SNVTs are mainly used to exchange data in an

⁶ Système Internationale

installed system. SCPTs are updated very rarely, mainly during the installation and calibration process.

SCPTs mostly cope with calibration, parameterization, limits or fundamental device operation modes. They are stored in non-volatile memory (e.g. EEPROM or Flash memory), so they need not be set after a power-down.

Compared to SNVTs, SCPTs are more specific, because they deal with a particular device type. Typically, a certain SCPT is used by devices of the same industry area. The SCPT index range is partitioned according to the industry areas.

Configuration properties can be implemented in two ways:

1. A configuration property can be implemented using an input network variable which is stored in non-volatile memory. The variable can be set like any other network variable. This approach is the simpler one, but uses up a network variable. There can be at most 62 network variables on a Neuron-hosted node so network variables can be a scarce resource in complex applications.
2. A configuration property can be implemented as a configuration parameter. Configuration parameters are organized as a file. This file can be downloaded using the LonMark File Transfer. This approach is more complicated to implement since the LonMark File Transfer must be implemented on the node. However, it allows a more compact configuration procedure and spares network variables.

2.4 LonMark Objects

The LonMark Application Layer Interoperability Guidelines present a system of objects which can be used to build the data interface of a LonWorks node.

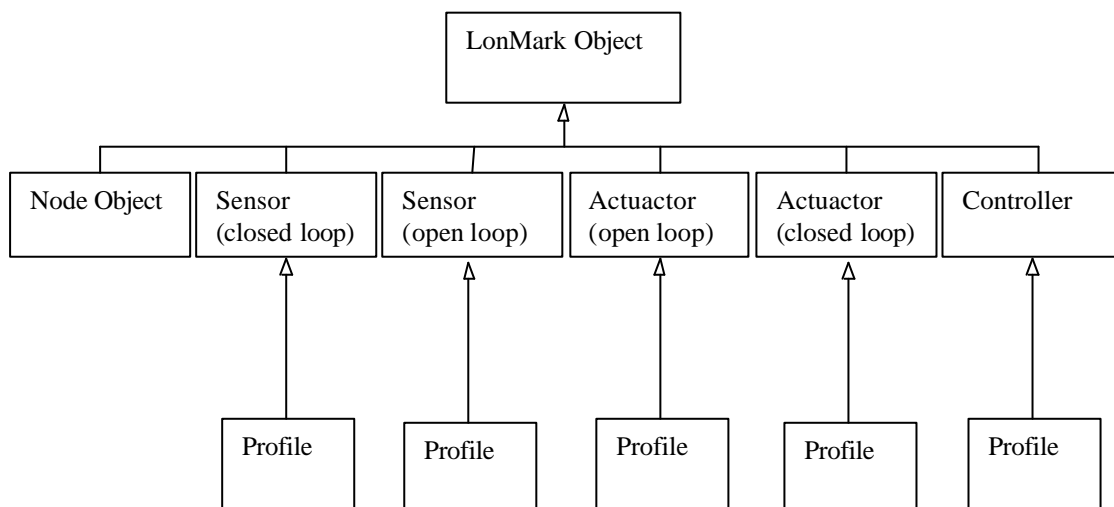


Fig. 2.3: The LonMark object hierarchy⁷

The structure of this framework is shown in Fig. 2.3. The model consists of three layers:

- 1) The root class represents the concept of a “LonMark Object”. Every LonMark profile and every basic LonMark object “is a” LonMark object.
- 2) The 6 basic object types are used for the device interface design:

⁷ The empty-headed arrows describe an inheritance relationship, where the arrow points to the parent class.

- a) There can be at most one Node Object in a node. It has the responsibility to control the state of the other objects. Further it can implement the interface for the LonMark File Transfer Protocol.
 - b) Open-loop and closed-loop sensors are used for specifying the sensor interfaces for representing environmental conditions to other network nodes.
 - c) Open-loop and closed-loop actuators are used to specify actuator interfaces for interacting with the environment in response to network activity.
 - d) Controller objects represent the interface of controllers which are devices with both, input and output network variables. Typically, sensors are connected to the input side of a controller, while actuators are connected to the output side of a controller object.
- 3) Profiles are specializations of the basic LonMark objects. They specify the network interface of a particular device, like a motor, a switch or a fan.

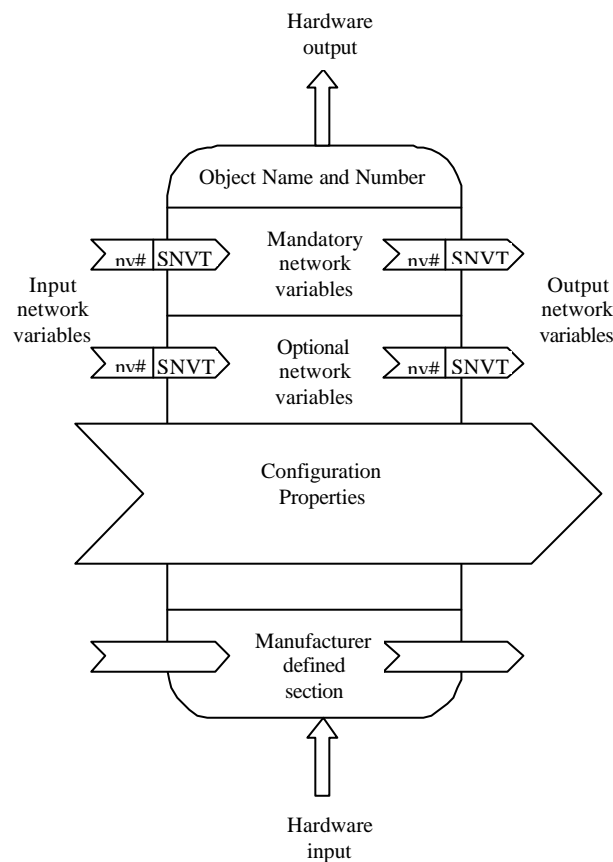


Fig. 2.4: The graphical representation of a LonMark object

The LonMark Interoperability Guidelines divide the interface of an object into several parts:

1. The mandatory network variables section defines network variables which have to be implemented for a particular object.
2. The optional network variable section defines optional network variables which can be implemented by devices supporting these extended services.
3. It is allowed that a device implements additional non-standard network variables. These have to be documented in the manufacturer-defined section.
4. The mandatory configuration property section defines configuration properties that must be implemented.

5. The optional configuration property section defines configuration properties which can be implemented if the device can handle them.
6. The hardware section defines which hardware inputs and outputs must exist for a particular object.

Fig. 2.4 shows the graphical representation of a LonMark object.

A LonMark-compliant application is composed of LonMark objects. There can be either a single LonMark object or an arbitrary number of LonMark objects including a mandatory node object.

The complete interface is built from the union of the LonMark object interfaces. The interface composition is described by the self-documentation string of the device and its network variables.

There are no limits in the number of the LonMark objects, as long as there are sufficient free network variables to implement the interface of the LonMark object.⁸

Composite objects have two major applications:

1. The sub-objects are independent. In this case, the processor and memory subsystem can be shared to reduce costs. For example, a controller can be implemented several times on a node, so only one node must be installed, even if multiple controllers are required.
2. The sub-objects work together to provide a more complex service. In this case, the sub-objects cannot be used distinctly. This may be the case when a device integrates multiple sensors or actuators.

Fig. 2.5 shows an exemplary weather sensor whose interface is composed of its sub-sensors. Since LonMark doesn't define encapsulation methods, each interface can be used distinctly or together with others. Each network variable is publicly accessible. The interface of a device or sub-device can be used by more than one other device. This is important for realizing inter-industry communication, since it allows the use of a single device in different systems.

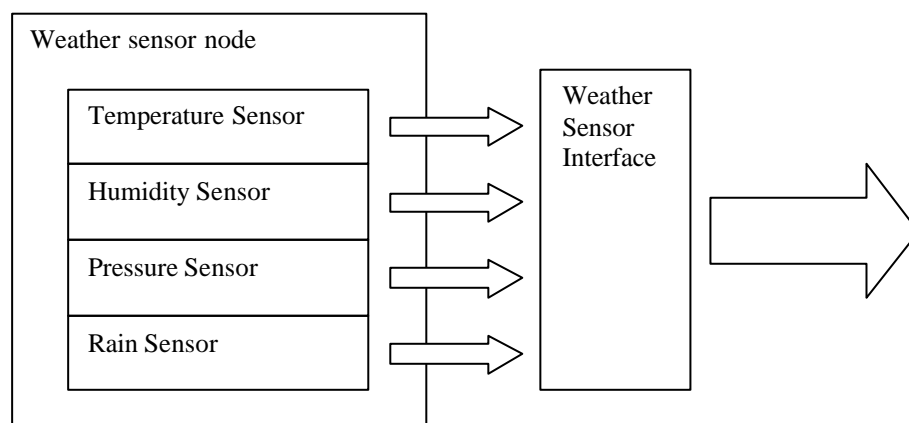


Fig. 2.5: Application composition by LonMark objects

Using object composition, new devices can be built upon existing ones without having to define new profiles. The node self documentation string allows to document the object composition of a particular application.

The LonMark object composition is not recursive, so it is not possible to make composites of composites.⁹

⁸ Neuron-Chips are limited to 62 network variables, while host-based nodes can contain up to 4096 network variables.

2.5 LonMark Functional Profiles

Functional profiles are more specific than the basic LonMark objects. The main goal of a functional profile is to define the network interface for a particular device type. Ideally, it should be manufacturer-independent. On the one hand it should be precise enough to avoid misinterpretations, on the other hand it should allow different conformant implementations.

A functional profile specification consists of several sections:

1. The overview section describes the purpose of the profile.
2. An exemplary usage pattern is presented. This example shows which other objects can be used with this functional profile.
3. The graphical representation gives a coarse overview about the object.
4. Two further sections describe the mandatory input and output network variables. The typical range, error values, default values, transmission conditions, service types and update rates are specified.
5. The next two sections specify the same properties for optional input and output network variables.
6. In the next two sections, mandatory and optional configuration properties are described. Especially their interpretation for this particular profile is defined.

The functional profile specification addresses integrators and device designers. Integrators use the profile specification to build systems out of devices that implement LonMark objects. Device designers use profiles to give their devices a standard interface. If both groups agree on the interpretation of the functional profile, the devices can be integrated into a system. If there are misunderstandings, there may be subtle errors that can be hard to track.

2.6 Profile Implementation

2.6.1 Device Design

Before a device is programmed, the required objects have to be determined. If there are two or more objects, a node object must be implemented, else it is optional.

The functionality of the device must be examined and analyzed:

- Is it an actuator, a sensor or a controller?
- Can the functionality be split into independent parts?
- In which situations, beside the primary planned one, is this device useful?
- Are parts of the device already specified by a functional profile?
- Can the complexity of the device be implemented with a 3120, a 3150 or a host-based node?

If an existing functional profile provides a part of the object functionality, it can be used to build this part of the object interface. If there is no suiting functional profile, the functionality can be implemented using basic LonMark objects or a new functional profile can be proposed to LonMark.

Both approaches have their strengths and weaknesses. Defining a new functional profile means to define an industry-wide standard for similar devices. Implementing a basic LonMark object however is faster, but lacks the industry-wide agreement on how such devices should be implemented. This can lead to non-interoperability if two manufacturers work in parallel.

⁹ At least, there is no self-documentation feature for this purpose.

These considerations lead to a list of functional profiles and basic LonMark objects as well as their mappings to the hardware. These objects must be implemented in the further steps.

If the device is rather complex, a decision between network variable based configuration properties and LonMark FTP must be made. If the FTP method is chosen, the node object is mandatory.

2.6.2 Object Design

If a partial function can be implemented by a functional profile and some parts are not covered by the mandatory network variables and configuration properties, the optional elements of the profiles must be checked. If they can be used to implement the not-covered functionality, they must be used. If the optional elements do not suit, either proprietary network variables can be used, or the necessary network variables can be requested to be added to the optional section of the functional profile.

Once the decision, which network variables and configuration properties are necessary, is made, the network variables can be implemented.

2.6.3 Network Variable Implementation

Each network variable must be declared like it is described in the functional profile. The name should be the same as in the profile, just for clarity. If two profiles use the same name for a network variable, the naming can be changed to avoid the name clash.

For each variable, some dynamic attributes (priority, polled, sync) must be considered. The self documentation string must be added to each network variable to reflect the object structure of the device.

Each output network variable whose completion state is important for the program logic must have completion event handlers. Either the program must check for the completion event, or it must check for both, the success and the fail event.

For each input network variable, a proper update event handler must be implemented.

2.6.4 Configuration Property Implementation

If the configuration properties are implemented using network variables, the same procedure as for input network variables can be used, except that the variables must have the eeprom class for persistent storage.

If the FTP method is used, the FTP network variables and the LonMark FTP must be implemented. The node must support the proper file types. This procedure is documented in [LonM731, p. 4-5ss] in detail.

2.6.5 The Implementation Behind the Interface

The program behind the interface can be implemented in an arbitrary fashion. Typical designs follow the event-triggered model of Neuron-C.

An application typically has following elements:

1. Internal state variables: These variables are either global or static variables.
2. External visible variables: These are network variables and configuration properties.
3. Externally triggered events: Events coming from the network, the hardware or the firmware.
4. Internally triggered events: Events coming from timers or from logical conditions.
5. Functions.

Events and functions read and write internal and external state variables. To exchange data between events, global variables must be used. They should be avoided in general because they can be read and written from any event and function which makes software maintenance more difficult.

2.7 Profile Usage

Integrators basically face the same problem as device designers, just from a different point of view. They have to decompose the requirements for a whole system into the basic functionalities. At the end they have to find devices that implement the required partial functionalities.

Functional profiles are a rendezvous-point. Functional profiles define often used functionalities within an application area. Devices implementing functional profiles provide this functionality in a ready-to-use fashion. So the lookup-process for integrators looks like:

1. Is my required functionality defined by a functional profile?
2. If so, are there devices that implement this functional profile?
3. If there is no suiting functional profile, are there devices implementing my required functionality directly, e.g. with basic LonMark objects?
4. Do the found devices cover my required functionality completely?
5. If so, integrate them and check them for interoperability.
6. If there is something missing, initiate its development.

After the devices are selected, they can be integrated using a network management tool. How they are used to form the application is depended on the selection of the devices. This is the reason why there should be a closer integration of the tool discussion and the profile design, as mentioned in the introduction.

3 Profile Models

3.1 Motivation

3.1.1 Audience

This chapter mainly addresses profile designers. It presents new views on LonMark objects and opportunities to integrate inter-industrial considerations in an early design phase. The new features are introduced by regarding network variables, configuration properties, LonMark objects and functional profiles as a set of related classes. The classes can be ordered into distinct layers coping with different aspects of interoperability.

The class hierarchy can help integrators to plan their devices and systems also. It provides new abstraction levels making devices more exchangeable and designs more robust to future changes.

Readers should have a basic knowledge of the object-oriented terminology.

Today, LonMark profiles are specified in a two-level hierarchy. Basic LonMark objects define interface-building blocks, such as sensors, actuators and controllers. Profiles are LonMark object specifications that define a particular interface for a useful device, such as a lamp, a heating device, a control panel or a fire initiator.

LonWorks is intended to be used by a wide range of industries. LonMark profiles are typically proposed by task groups associated with a certain industry area. It is possible that different task groups propose a profile for the same device type. These profiles can overlap because they describe interfaces to similar devices.

A structured profile system allows to avoid parallel developments because a common subset can be defined in advance. Afterwards, it is easier to define a full-featured profile. Without structure, parallel developments are very hard to detect and the observed uncontrolled growth of profiles and data types will get unmanageable.

Vertical integration relies on abstracting core functionality from device-specific details. Without defined multi-hierarchy levels, there is no place where to integrate a device into a higher-level control system. As more and more nodes are integrated, the reuse of existing devices, even those from other industries, becomes crucial. Only a defined operation subset between industries can provide a base for inter-industrial interoperability.

At last, a more structured approach encourages designers to think of other users of the system. Sophisticated stand-alone solutions may be a monument to the inventor, but are not valuable for others.

3.1.2 Objectives

Adding new features to a design primarily adds complexity. A design can benefit from the additional complexity, if it results in increased flexibility and additional design freedoms.

The overall goal of this work is to improve inter-industrial interoperability so that devices build by different industries can cooperate more easily. This enforces multiple use of existing sensors and actuators, as well as additional data exchange between formerly independent systems. In this case, the increased design complexity can lead to simpler and cheaper systems.

The benefits may not be reached by introducing incompatibilities to the LonMark Interoperability Guidelines. The additional system must be an extension to LonMark that can be used whenever its usage seems to be beneficial. Further, existing and additional profiles

must be compatible to the class hierarchy, regardless whether they have been designed with the class hierarchy or not.

The last objective of this work is to create an awareness that helps to develop future LonMark extensions.

3.1.3 Intra-Industry Interoperability

For a particular industry no changes to the devices are required, since each device still implements the necessary LonMark profiles. Fig. 3.1 shows an exemplary design using only intra-industry communication.

The specifications for intra-industry interoperability are simpler than those for inter-industry interoperability. Within a certain industry, the naming and data representation is more uniform. The network interface can match the device interface formerly build of analog and digital control lines.

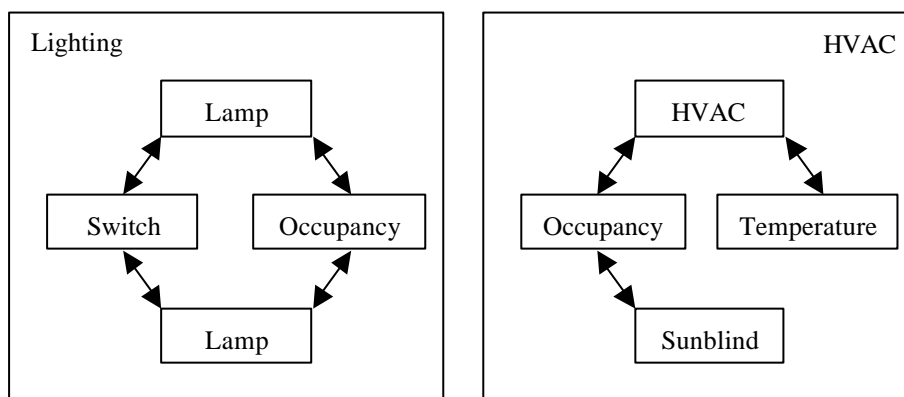


Fig. 3.1: Intra-industry communication

For interoperable devices, a large set of agreements must be met. These include:

- Data representation (resolution, range, error conditions, ...)
- Data semantics (physical units, meaning, ...)
- Data context (location, responsibilities, ...)
- Communication semantics (absolute vs. relative, trigger conditions, ...)
- Dynamic behavior (timing, heartbeats)

The LonMark Interoperability Guidelines have formal methods for data representation, data semantics and some context description. The other parts are informally described, so they are missing for some profiles.

Even worse, some profiles miss specifications which are essential for the targeted application area. For example, integrators face problems integrating constant light applications with sensors and actuators from different manufacturers. A constant light application requires a reasonably fast and smooth sensor update to avoid flickering effects. The light sensor profile, which is intended for light control applications, does specify only optional configuration properties for timing behavior (minimal and maximal send times, send on delta feature). Sensors from different manufacturers lacking these configuration properties provide a different behavior and their use can result in unsatisfactory control behavior. This example shows that a profile must be defined looking at the whole application class for which it is intended. Any non-defined agreement for that application area can lead to non-interoperability someday.

3.1.4 Inter-Industry Interoperability

For inter-industry interoperability, devices from different industries must interpret data in the same way. This is more complicated, since industries can have different naming and data conventions, as well as different timing constraints¹⁰.

An important decision is the set of data items that have to be shared between different industries. As a rule of thumb, high-level data (such as room temperatures, occupancy states, etc.) is useful in different industries more likely than low-level data (such as a process-internal device state).

Even if there is an agreement on the representation of a data item, there will be different opinions on implementation details. E.g., a small electric motor in a copy machine and a diesel motor in an emergency power supply have common process data like torque and angular velocity. Their other features, such as power supply constraints and safety issues are extremely different.

A way to capture these differences in a model is to place the devices into a hierarchy, where higher-level devices are more general, while lower-level devices are more specific to a certain industry.

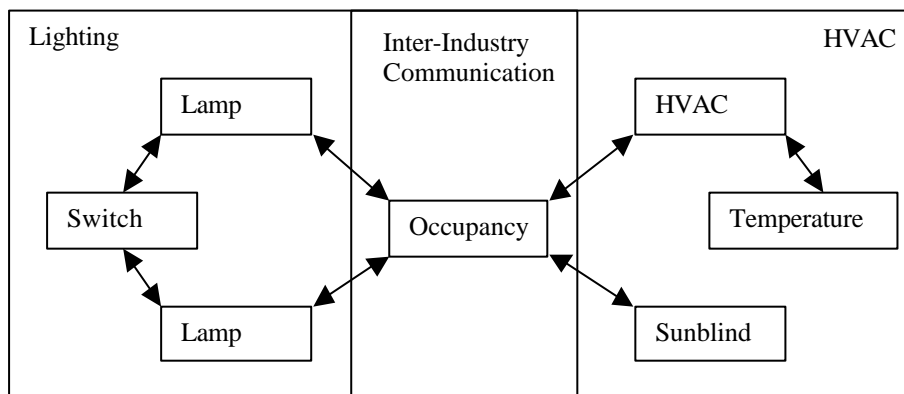


Fig. 3.2: Inter-industry communication

Another topic of inter-industry interoperability handles the reuse of already installed devices. If the abstraction hierarchy is used, nodes from different industries still can use the common operation subset. Fig. 3.2 shows an exemplary installation which benefits from exploiting inter-industry communication opportunities. To enable this device reuse, the HVAC and lighting industry must agree on a common functionality for the occupancy control.

3.1.5 Network Management

Network management covers the process of integrating individual devices to a functional system. The efficiency of network management procedures depends on the amount of information available about the individual devices. If a network management tool can handle higher-level concepts, such as industry-independent profiles or generic profile classes, network management can be done in a top-down manner. At the beginning, the coarse functionality is defined which can be refined in further steps. This also simplifies vertical integration because data exchange between vertically related systems mostly concerns high-level data.

¹⁰ For example, heating devices have long time constants (minutes to hours), while lighting has short time constants (sub-second) because of ergonomic constraints. The same yields for industrial processes that can have time constants from milliseconds (control loops) to days (some chemical processes).

3.1.6 Compatibility

The LonMark Interoperability Guidelines are widely accepted in the LonWorks world. Many existing devices are already implemented using LonMark profiles, so new ideas should fit seamlessly.

Existing released profiles must be used as they are to provide interoperability, but future profiles should be created with respect to inter-industry applications. A profile should be checked for applicability in other application areas and the core idea of a profile should be defined in an industry-independent profile allowing other industries to specialize the general profile.

3.2 General Services

3.2.1 Service Definition

Devices or groups of devices can provide services to their users. These are tasks which run autonomously or are invoked by the user. A service is expressed in terms of the user and not in terms of the implementation.¹¹ Therefore it is important to specify who the user is. Some services might be “consumed” by the user of the system, while others are used by other subsystems.

There can be parts of a service which need to be used to operate a device, while others may be optional and control additional partial services.

3.2.2 Service Aggregation

A device can provide multiple services for different users at once. The union of all services is the external interface of a device.

A device can be built out of sub-devices. The interface of the containing device can also expose the interfaces of the contained interfaces. If the interfaces of the sub-devices are not important for a user of the containing device, they also can remain internal interfaces that are not visible to the user. For example, the interface of a heating device can exclude internal process parameters from the public interface, especially, if all internal states can be handled without external influence. The interface of the device would be obfuscated if too much internal data is exposed to the environment.

Since the interface of a newly created device can be used by several other subsystems, it must be expressed at the appropriate abstraction level for providing inter-industry interoperability.

For larger systems, device interfaces should be kept reasonably small. Having an interface which can do everything and provides hundreds of partially dependent services do not simplify the life of a user.

3.2.3 Service Implementation

If a service is an abstraction hiding implementation details from the user, there must be means to map the language of the user to the language of the implementers.

For LonWorks systems, services are typically implemented by network variables. Thus, an implementation of a service is done by sensors and actuators and their network variables.

¹¹ For example, a service might be “keep room temperature constant”. While the user is interested in a comfortable room, the implementers must define “what is constant”, which means can be used to get it working, etc.

The next service level is provided by functional profiles which provide higher-level services by putting some lower-level services together into a single interface. These services describe the capabilities of a component.¹²

Several components can make up an application by combining sensors, controllers and actuators. These are services that can be typically used by end-users or maintenance personnel.

3.2.4 Example

Consider a simple illumination problem. A worker has to do some work in a room. To do so properly, there must be a convenient illumination. Two conditions can make his work uncomfortable – it can be too dark and it can be too bright.¹³

The service for the worker can be expressed like: “I must have enough light to see, but it should not be too bright to avoid getting dazzled”.

A specialist for ergonomics would express this like: “The illumination of the working space must be between x Lux and y Lux for z % of people. Some people have different preferences, so they must be able to override the automatic control”.

A LonWorks engineer can implement the service by following means: An illumination sensor measures the illumination of the room. Lamp actuators can make the room brighter and a sunblind can prevent the sun from dazzling the worker. A control knob must allow overriding the default settings.

An application programmer would explain the task in terms of network variables: An illumination sensor has an output SNVT_lux that is updated whenever the illumination changes significantly or a heartbeat timer expires. The illumination knob has an output switch SNVT which indicates whether the illumination is overridden and if so, which light level should be reached. The lamps and sunblinds could have an input percent level SNVT that indicated how bright or how “open” the device should be. An illumination controller has the corresponding input and output SNVTs and calculates a suitable lamp/sunblind setting out of the sensor and user input.

A translation that maps the languages of the users and the implementers exists for each pair of views. This is illustrated in Fig 3-1.

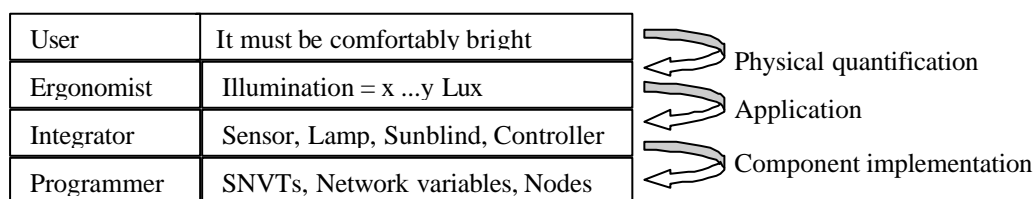


Fig 3-1: The description of the illumination service in terms of different users

3.2.5 Integrated Room Control Example

This subsection describes the services from which users of an integrated room application can benefit:

¹² For example, an intelligent sensor might use several sensors for making a better sensor by temperature compensation and linearization. The service of the sensor is higher-level than the services provided by its parts, since the sensor output and a temperature at the sensor location might be rather useless for everything else but the sensor.

¹³This example is very simplified. A real illumination controller also should consider the occupancy state, time constants to avoid a too fast control loop, etc.

The integrated room provides a convenient climate. Inhabitants find a proper setting but can override the automatic control if they have different preferences.

Lights and sunblinds regulate the illumination to a proper value. As above, the user can select a convenient light level and the controller tries to maintain a constant light level.

Since most services are only useful for occupied rooms, occupancy detectors and schedules can simultaneously optimize comfort and energy consumption. Override settings allow deviations from the automatic behavior.

The integrated room is a unit of energy management. It can monitor its devices and can start energy saving procedures on request.

For fire alarms, an integrated room can act as a fire alarm initiator and it can react to fire alarms by setting its devices to a proper emergency safety setting.

An integrated room also can be a basic block for a building management system. It can provide interfaces exporting filtered data that is useful for higher-level management systems.

Finally, an integrated room can be the unit of access control. It can act as a high-level sensor for intrusion detection systems.

3.3 Network Variable Models

3.3.1 Introduction

This section introduces some approaches to describe network variables. Since each view emphasizes a different aspect, they are all useful to describe the properties of network variable based communication system.

The shared memory model copes with the data sharing property of network variables, the signal model captures the communication aspects, while the functional model explains how programs access the network variables. An event model connects the network-oriented and node-oriented views.

3.3.2 Shared Memory Model

Two or more bound network variables can be regarded as distributed memory with multiple writers and multiple readers. The network variable selector is an “address” within the distributed memory. The nodes that can share the network variable data are defined by the address and domain table configuration of the sending and receiving nodes. Since a network variable update is non-instantaneous, the knowledge of sending and receiving nodes is incoherent until all receivers have processed the network variable update. Multiple writers are not synchronized. Fig. 3.3 shows a simple network. The selector of each network variable selector is an index into the shared memory. Arbitrary senders and receivers can connect to the same memory location. The network variable with the selector 7 shows the multiple writers problem. A writer does not know that its value has been overwritten by an other writer, so sender 1 and sender 2 have different local values for the network variable. The synchronization of multiple writers must be done at a higher software layer. LonMark defines closed-loop sensors and actuators for this purpose.

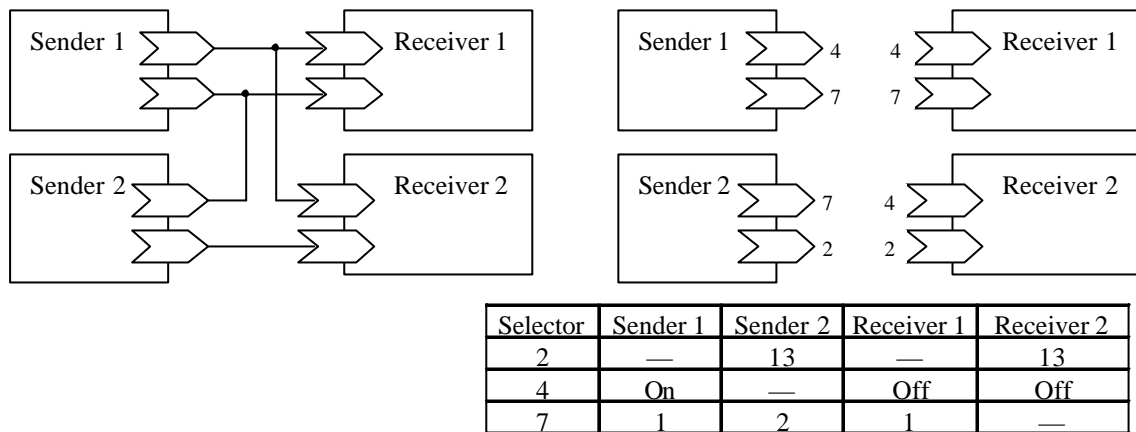


Fig. 3.3: Network variables as shared memory

3.3.3 Event Model

An output network variable can be interpreted as an event object which sends a message to bound receiver objects. Since each node has packet queues, the event notifications are also queued and processed by the receiver upon their arrivals. The receiver object is basically a server waiting for the arrival of an event, processes data transmitted with the event and performs appropriate actions.

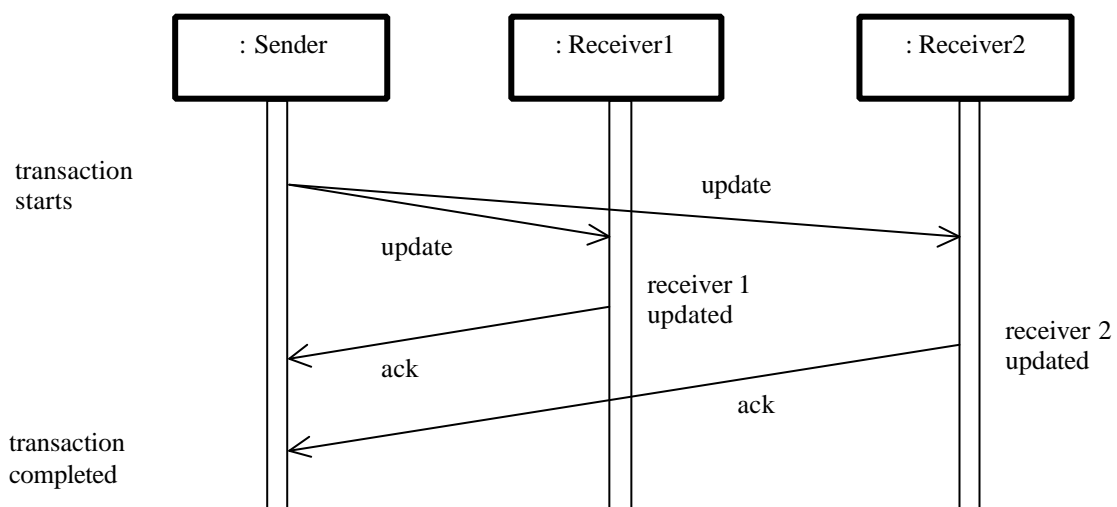


Fig. 3.4: Network variables as signal objects

The communication is asynchronous, since the sending node can do something different after sending the event. However, the sending node can get notified whether the message has been received (acknowledged service). In this case, the sending object also is an event listener. It can receive either a success or fail event indicating whether the update was successful. Fig. 3.4 shows an exemplary sequence diagram involving one sender and two receivers.

3.3.4 Functional Model

At the first glance, LonMark objects are data-oriented interfaces. Since a network variable cannot be directly accessed via memory in general, a network variable must provide methods for reading and writing stored data.

For output network variables, this is a simple activity. The local network variable memory is updated to the new value and a signal containing the new value is sent to the network. The network variable data can be read from the local memory. An output network variable can listen to two signals. It can react on network variable update success or failure.

For input network variables, the value transmitted with the signal is copied to the local network variable memory and the corresponding signal handler is invoked. The data of the network variable can be read from the local memory then.

3.4 Network Variable Type Classification

3.4.1 Hierarchy

Standard network variables types are the fundamental concept for the LonMark Interoperability Guidelines. Although each type is independent of each other, there are certain similarities between them. By working out these similarities, some interoperability topics can be directly deduced from the SNVT definitions.

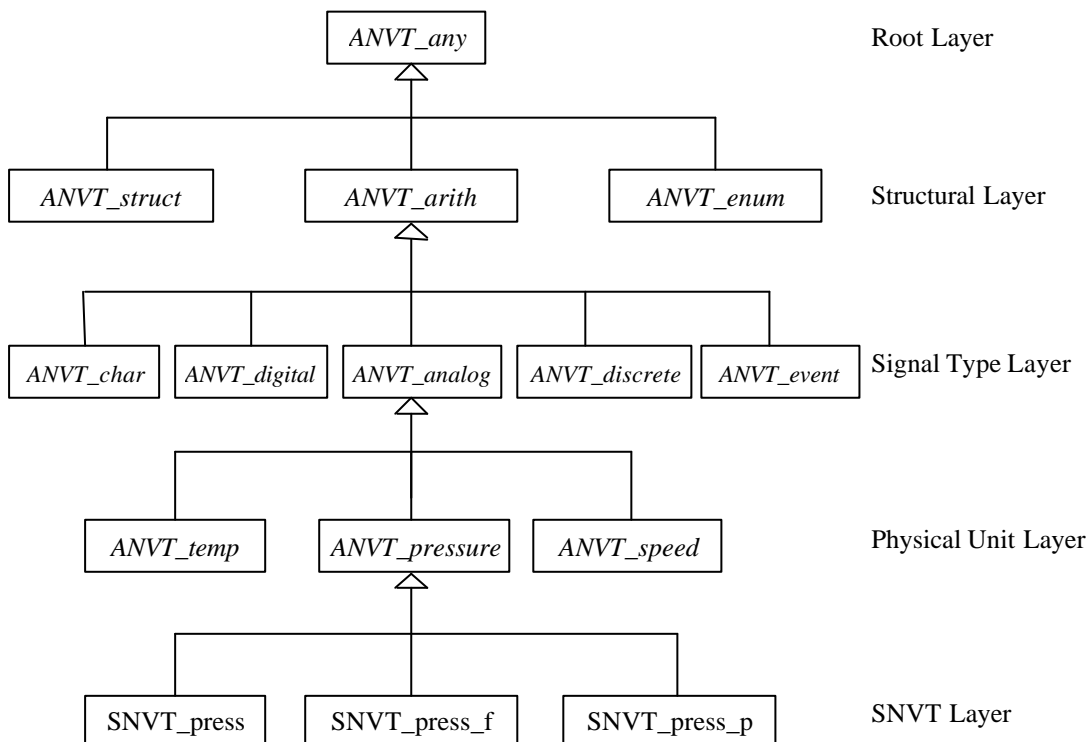


Fig. 3.5: A network variable type hierarchy

When network variable types are interpreted as classes, they can be interpreted as the leaves of an abstract data type hierarchy. That means that any non-leaf class cannot be used in a concrete device, but is used for structuring the concrete leaf classes. The abstract classes are organized into layers which cope with different layers of interoperability.

Each hierarchy level introduces a new semantic concept. An exemplary hierarchy for physical SNVTs is shown in Fig. 3.5. It shows only an exemplary unit to avoid bloating the class diagram. Abstract types are shown in an italic font. They cannot be used in a device, but they can be used to specify more abstract designs. Since an implementation requires refining the design to concrete types, this step cannot be omitted accidentally.

The layers introduce following concepts:

1. The root of the hierarchy is the parent class for all network variables.
2. The structural layer makes a distinction between the structure of the types. Arithmetic types are simple types which can be used in arithmetic expressions. Enumeration types cope with states and commands. Structured types are complex types containing more than one data item.
3. Arithmetic types are further structured into character, digital, analog, discrete and events in the signal type layer.
4. The physical unit layer introduces the concept of physical units and is used to group SNVTs with the same physical unit.
5. The SVNT layer contains the concrete SNVT types which can be used in a program.

This hierarchy is particularly useful for specifying more general profiles. A generic analog sensor can be specified using the “analog type”. Basic procedures like linearization or closed-loop control algorithms can be specified without taking the special properties of more specialized types into account. When utilized in a certain device, they can be replaced by a derived concrete type.

3.4.2 Abstract Types

The new abstract types are called Abstract Network Variable Types (ANVTs). Since abstract types cannot be instantiated in a program, they get an impossible SNVT identifier, for example a negative number.

A network variable type is defined by its name and its parent type. A textual description provides the intent of this type. The basic ANVTs are shown in Table 3.1, some example SNVT/ANVT mappings are shown in Table 3.2. The complete table is stored in the prototype database.

| SNVT Name | Parent Name | Description |
|---------------|-------------|--|
| ANVT_any | - | Describes any type, parent of the type hierarchy |
| ANVT_struct | ANVT_any | Describes structure/union types |
| ANVT_enum | ANVT_any | Describes enumeration types |
| ANVT_arith | ANVT_any | Describes scalar types |
| ANVT_event | ANVT_arith | Event type carrying a scalar event information |
| ANVT_char | ANVT_arith | Represents a single character with an arbitrary encoding |
| ANVT_digital | ANVT_arith | Represents a binary (on/off) signal |
| ANVT_discrete | ANVT_arith | Represents a n-way signal (one out of n levels). |
| ANVT_analog | ANVT_arith | Represents an analog signal with an arbitrary unit |

Table 3.1: Basic Abstract Network Variable Types

| SNVT Name | Parent Name | Description |
|------------------|-------------|---|
| ANVT_temp | ANVT_analog | Represents a temperature |
| SNVT_temp | ANVT_temp | Temperature type for industrial processes |
| SNVT_temp_f | ANVT_temp | Float temperature type |
| SNVT_temp_p | ANVT_temp | Temperature type for HVAC |
| SNVT_obj_request | ANVT_struct | Object request variable for Node Object. |
| SNVT_defr_mode | ANVT_enum | Defrost Object mode enumeration |

Table 3.2: Examples for ANVT/SNVT specializations

3.4.3 Arithmetic Types

The semantics of arithmetic types can be defined in several layers. Numeric types can be separated into several subtypes that are used to specify the basic signal type. Analog types can be further distinguished by their physical unit. The last layer specifies the physical representation and a mapping between the physical value and the binary representation.

The basic arithmetic types are:

1. Binary signals: This signal type represents a single on/off signal. These are represented by a SNVT_switch whose value part is ignored or by a SNVT_lev_disc.
2. Discrete signals: The values are taken from a set of predefined settings. Discrete level SNVTs are in this category.
3. Analog signals: A continuous signal whose granularity is defined by the resolution of its network variable. Most sensors deliver an analog signal.
4. Event type signals: These are signals which indicate the change of a state. Buttons are typical examples. Since LonWorks is an event-triggered system, all network variables updates are events indicating a state change. In contrast, the event signal type does not emphasize the state change, but a noteworthy occurrence in time.
5. Characters: Single characters, such as output from a keyboard.

Analog signals have an additional sublayer which defines the SI unit. At this stage, resolution and range are not defined.

Table 3.3 shows the units currently covered by SNVTs.

| | | | |
|---------------------|------------------|---------------|-------------|
| angle | electric voltage | length | temperature |
| angular velocity | energy | mass | time |
| area | flow | dimensionless | volume |
| density | frequency | power | |
| electric current | grammage | pressure | |
| electric resistance | illumination | speed | |

Table 3.3: Arithmetic SNVT units

3.4.4 Enumeration Types

All enumeration types are basically compatible, since they have the same data size. Each enumeration has a different interpretation of its values, so it is not possible to put them into a simple hierarchy. Most enumeration types are dedicated to a particular device or device group, so common value interpretations are rare. Thus, they are considered as a group of distinct types.

3.4.5 Structural Types

Structural types are basically well suited for inheritance. However, the structures used by the actual SNVTs don't have many properties in common, so the resulting hierarchy would contain an empty base class and many leaf classes. Therefore, the structure types are not modeled at this point and are considered as a group of distinct types.

Some structured signal types are:

1. Structured data: Higher-level structures composed of simpler data types. Most profiles whose primary network variables have a structure type can be considered to exchange this signal type. It is the most general signal type.
2. Binary files: Files consisting of several binary data records having the same structure.
3. Discrete files: Files consisting of several discrete data records having the same structure.
4. Analog files: Files consisting of several analog data records.
5. Character files: Typically text files, for example text for a LCD.

3.4.6 Context Definition

When a variable is actually used, it is assigned a responsibility within the distributed application. Some responsibilities appear repeatedly, so it makes sense to make common definitions for them.

The network variable type hierarchy is open ended, so it is possible to make further specializations to define additional semantics. Each derived network variable type must have a stricter semantics than its parent type, so only conjunctive extensions are allowed.

For example, HVAC applications are concerned with measuring temperatures, say an indoor temperature, an outdoor temperature, fluid temperatures and so on. Noting the responsibility of a variable explicitly allows to simplify the network variable binding procedure, since only variables with the same context should be bound. This would reject errors like binding an outdoor temperature sensor to the indoor temperature input of an HVAC controller.

The context of a variable can be modeled as a second type identifier, independent of the SNVT, which indicates how a variable is used. Context definitions would be useful for the implementation of whole profile families (multiple sensors, controllers and actuators), since each profile can pick a network variable and its intended usage from a predefined pool of context definitions. This would avoid misinterpretations within the profile family.

Using a network variable within a certain context is a dynamic process, so the context of a variable may change. This allows to utilize a sensor or actuator in different contexts. A variable can be described by set of contexts in which it can be used. For example, a temperature sensor for building automation uses a SNVT_temp_p variable for transmitting temperature data. They can be used as indoor or as outdoor sensors. Since each of these uses applies a different meaning to the measurement value, it would be advantageous to record the meaning of the device.

3.4.7 Other Classification Schemes

The notion of abstract (generic) network variables is not a new invention. IEC1131-3 defines a variable hierarchy for PLCs in a very similar manner. The purpose of this hierarchy is to provide a framework for specifying generic algorithms that can be applied to different data types. The object constraint language (OCL) of the unified modeling language (UML) also uses a very similar type hierarchy [OMG99].

3.5 Major Device Classification

For describing a control system, data sources and data sinks must be identified. In a networked control system, communication relationships must be identified, too. Each device can have a combination of data sources, data sinks and communication relationships.

Table 3.4 shows a classification of the different data transfer paths between the environment and the network. A device can be inert to the environment or can be a data source¹⁴ (importing data from the environment), a data sink¹⁵ (exporting data to the environment) or both. A device can be not networked, a transmitter (having output network variables), a receiver (having input network variables) or both.

| | No network | Transmitter | Receiver | Transmitter & Receiver |
|-------------------------|------------|--------------------------|------------------------|---|
| Inert | ✘ | ✘ | ✘ | Controller |
| Data source | ✘ | Data source | ✘ | Controller & Data source |
| Data sink | ✘ | ✘ | Data sink | Controller & Data sink |
| Data source & Data sink | Standalone | Standalone & Data source | Standalone & Data sink | Standalone & Controller & Data source & Data sink |

Table 3.4: Data transfer between the environment and the network (✘=meaningless)

The most fundamental device is the standalone device. It acts as a single centralized controller that gathers data from data sources and calculates the output for its data sinks. Since we discuss decentralized systems, the standalone device is not very interesting in terms of network design. It is interesting for the system design, since the network disappears when all its parts are observed together. From the viewpoint of a customer or a user, the network is an internal implementation detail because the network is completely within the system borders. Fig. 3.6 shows an exemplary network and how the network disappears within the system borders when looking at the basic functionality.

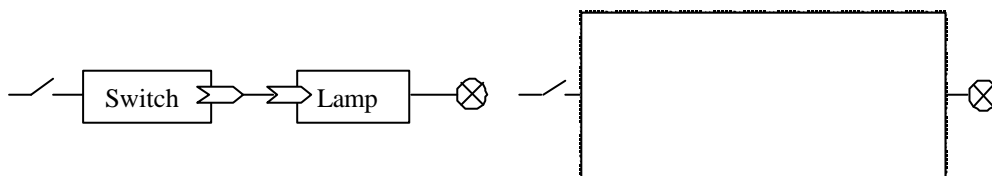


Fig. 3.6: The network is not visible at a system-level view.

In a networked environment, a data source is a device which imports data from the physical environment and presents it to the network. Sensors are typical data sources, but random generators, clocks and gateways also can be regarded as data sources. A device which reads data from the network and actively interacts with the physical environment is a data sink. Abstract data sinks like storage devices or gateways are also allowed. The most common data sinks are actuators.

A device that does not export or import data from/to the environment, but receives and sends processed data from/to the network is called a controller. There are mixed forms where a device implements a combination of the standalone, data source, data sink and controller devices.

¹⁴ This also includes abstract data sources, such as constants, data generators, storage retrievals or gateways.

¹⁵ This also includes abstract data sinks, such as null devices (destroying information), storage devices or gateways.

The smallest network device set consists of data sources and data sinks only. These are sufficient to perform all control tasks. Data sources get data from the environment and send processed data to the data sinks which in turn interact with the environment again. The disadvantage of this architecture is the mixing of importing, exporting and processing data. A data source performs importing and maybe a processing task, a data sink performs an exporting and maybe a processing task.

| | Import | Process | Export |
|----------------------|--|---|--|
| Open-Loop Sensor | Imports data from the environment Does preprocessing (linearization, ...) | none | none |
| Closed-Loop Sensor | Import data from the environment Preprocessing (linearization, ...) | Synchronizes with other nodes in control loop | none |
| Open-Loop Actuator | none | none | Exports data to the environment Does preprocessing (actuator control,) |
| Closed-Loop Actuator | none | Synchronizes with other nodes in control loop | Export data to the environment Does preprocessing (actuator control,) |
| Controller | none | Does real processing (control algorithms) | none |

Table 3.5: Mapping between LonMark objects and basic control functionalities

To solve this design weakness, the set of required device types can be extended by the controller type which does all processing. Now data sources just do the importing task, controllers do the processing task and data sinks do the exporting task. The control algorithm is now concentrated in the controller devices which can be simply exchanged whenever another system behavior is desired. The processing part of a distributed control system can be moved between nodes which is not true for the importing and exporting parts. This allows to freely choose on which node the controller is placed. Multiple controllers can be implemented on a node to share the same hardware.

The combined devices (e.g. controller & data source) are not required for this model, since they mix the responsibilities of their parts. However, most current LonMark objects are of such a type. For example all HVAC devices include sensor, actuator and controller functionalities. Table 3.5 shows the mapping between LonMark objects and the import, processing and export tasks.

Fig. 3.7 shows some possible network and node designs for a small lighting problem. In Fig. 3.7(a), the problem is shown. A staircase connecting three floors shall be lighted by an intelligent lighting system. Each floor has a motion detector. Since one can go either upstairs or downstairs, the light in the adjacent floors is also activated. Thus, the lighting follows the inhabitant when moving around. When a certain period of time passed without any motion, the light is turned off. For safety reasons, all lamps can be manually overridden in case of an alarm (fire, intrusion).

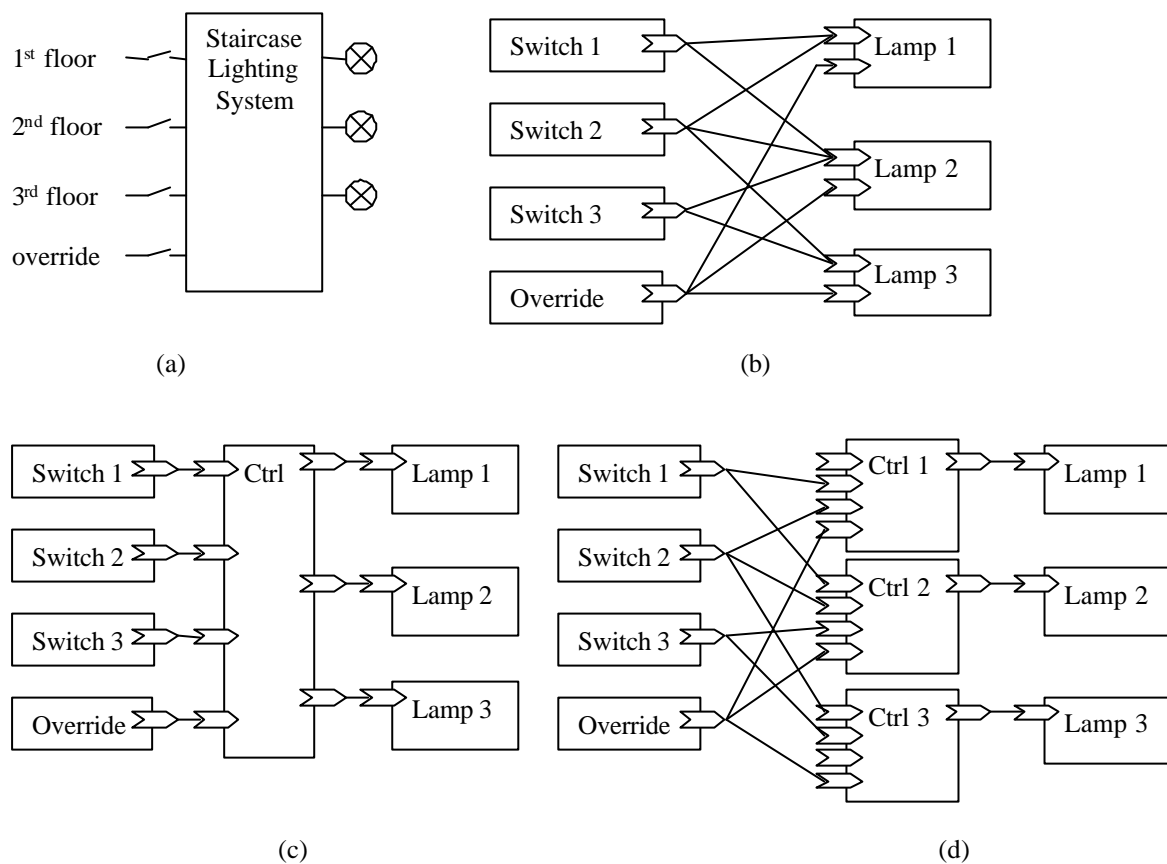


Fig. 3.7: Network and node designs for a lighting problem

An approach that doesn't require a controller is shown in Fig. 3.7 (b). Every switch is connected to the lamps of the same and of the adjacent floors. Another network variable is required for the override feature. The timing behavior must be implemented in the lamp nodes. Thus, the lamp device must contain processing parts that might be unnecessary in other applications.

The next approach utilizes a centralized controller shown in Fig. 3.7 (c). The controller is fitted to the application and minimizes the complexity of network variable bindings. Because of the application-centric design, this approach is not very scalable. The algorithm for the override feature is now implemented by the controller. This simplifies the lamp nodes which do not need the extra network variable anymore.

The final approach presents a decentralized controller. Each lamp gets an own controller which does all processing concerning its lamp. As with the centralized controller, the lamp is now a simple device. The decentralized controllers have the same network complexity at their input sides as the approach without a controller. The advantage over the centralized controller is the scalability.

These examples show that simple data sinks and data source can be used in more environments than complex ones. As soon as the processing part is removed, they are simple enough to be connected to different controllers. As long as the processing parts are implemented in the lamp, they waste the resources of the lamp node as soon as a dedicated controller is used.

The presented approaches are extremes. Since the processing parts can be moved to any of the actuators, controllers or sensors, many combinations of these designs can be implemented. A

summary of the advantages and disadvantages of the different approaches is presented in Table 3.6.

| Approach | Advantages | Disadvantages |
|------------------------|---|--|
| No controllers | <ul style="list-style-type: none"> • Scales well due to the distributed algorithm and the binding flexibility of LonWorks. • Shortest network delay. • No controller object required. | <ul style="list-style-type: none"> • Sensors and actuators must implement processing, thus they are application specific. • Controller algorithm is hidden in the sensors and actuators. • Is not modular and cannot be used in a wide range of applications. |
| Centralized Controller | <ul style="list-style-type: none"> • Minimizes binding effort. • Suited for a predefined set of applications (performance). • Makes sensors and actuators simpler. | <ul style="list-style-type: none"> • Does not scale well. • Is not modular and cannot be used in a wide range of applications. • One extra object for the controller • Additional network delays. |
| Distributed Controller | <ul style="list-style-type: none"> • Scales well due to the distributed algorithm and the binding flexibility of LonWorks. • Network variable bindings show relationships directly. • Makes sensors and actuators simpler. | <ul style="list-style-type: none"> • High resource usage (one object for each controller) • Additional network delays. |

Table 3.6: Advantages and disadvantages of different network and node designs

3.6 Applications and LonMark Objects

An application can be composed of several LonMark Objects. Fig. 3.8 shows an UML class diagram containing all important parts.

An application interface is composed of one or more network variables and zero or more configuration properties. The network variables and configuration properties are grouped by LonMark objects.

Applications can be either simple applications, containing one LonMark object, or complex applications containing a Node Object and an arbitrary number of other LonMark objects.

Each LonMark object that is not the Node object, is either a closed loop actuator, a closed loop sensor, a controller, an open loop actuator or an open loop sensor.

The diagram is complex because of the following reasons:

- The application is a composite that is responsible for its parts. The network variables and configuration properties are owned by the application and are created and destroyed with the application. LonMark objects are just aggregates that order the network variables of a node to higher-level objects. Therefore network variables and configuration properties are aggregated twice, once by the node and once by a LonMark object.
- The distinct node object requires a special class to model the multiplicity of simple and complex applications. This is necessary, since LonMark objects do not contain a control interface, but are controlled via the node object.

- Configuration properties are owned by the node, but can be connected to either the node, an object or a single network variable. This is expressed by the triple association with the xor-constraint between the configuration property class and the node, LonMark object and network variable classes.

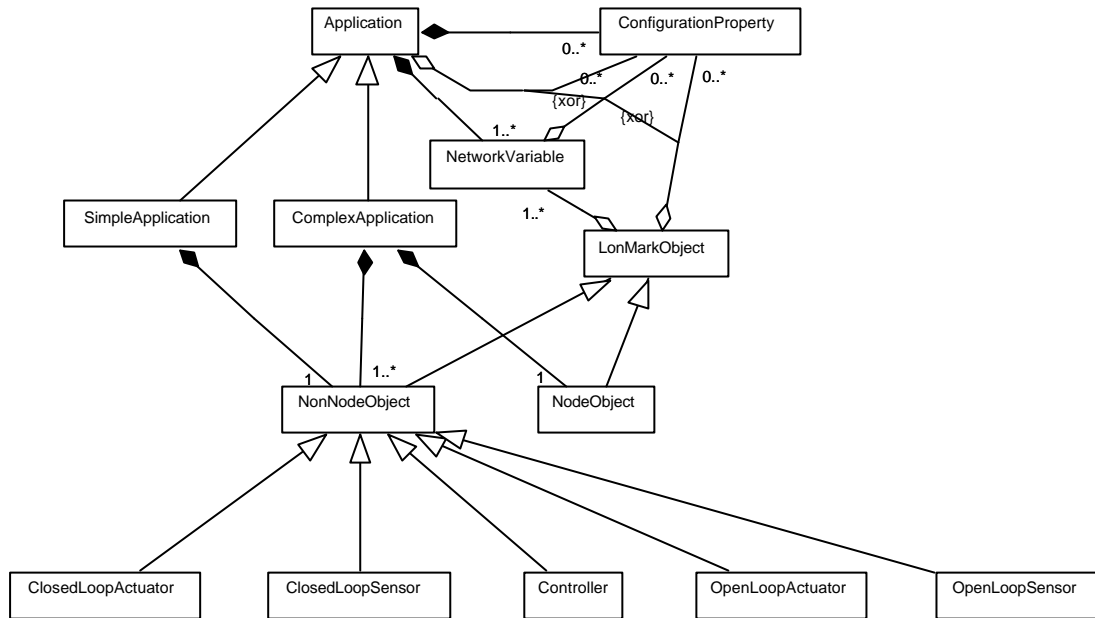


Fig. 3.8: Application composition with LonMark Objects

Future designs should avoid that much special cases as they make the system more difficult to understand and pose unnecessary constraints to implementations.

This diagram is also the base for the prototype database accompanying this paper. The classes, compositions and aggregations are represented by tables, known objects as records of these tables.

3.7 Structured Hierarchy

3.7.1 Overall Design

This hierarchy design [LNO1699] introduces new layers coping with the primary data flow properties. The mapping of the new layers to the LonMark Interoperability Guidelines is shown in Fig. 3.9.

The first additional layer introduces the concept of basic signal types, like analog, digital or stream signals. The abstract types presented in section 3.4 are used for this purpose.

The second additional layer introduces the concept of industry-independent device classes. These classes provide LonMark object specifications for a particular set of devices, such as pumps or temperature sensors. Industry-dependent details are neglected and their specification is left to the profile layer.

The manufacturer dependent layer can be used to add proprietary network variables to a profile. This enables companies to augment their products by unique features.

The resulting class hierarchy is shown in Fig. 3.10 for a temperature sensor in detail.

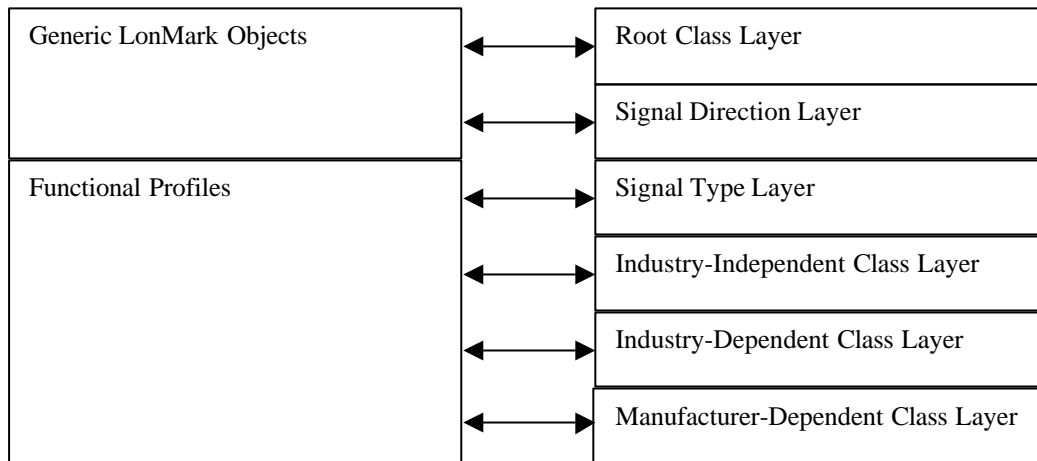


Fig. 3.9: Comparison between LonMark and the extended class hierarchy structure

3.7.2 Inheritance

Extending a profile means to specify the relationship of network variables and configuration properties, as well as the relationship between parent profiles and child profiles. Because the network variable indices of existing profiles are uncorrelated, the indices in parent and child profiles can be different.

The association between the index within the parent profile and the child profile is made by a translation table. This translation table is like the network variable and configuration property tables used by functional profiles, but also lists the index of a network variable within the parent profile. The field is left empty, when the network variable does not exist in the parent profile.¹⁶

If a new profile is derived from a generic profile, new SNVTs must have a higher index than the largest index of the generic profile. This prevents overwriting an already existing network variable.¹⁷ If an independently defined profile is inserted into the class library, it is likely that the network variable indices do not fit.

For output network variables, the table looks like Table 3.7:

| SensorWithRawOutput extends GenericAnalogSensor | | | | | | | |
|---|------------|-----------|------------|--------------------|------------|--------------|-------------------|
| NV# | NV# parent | Name | Heart-Beat | SNVT Type SNVT_ | SNVT Index | Config class | Description |
| 1 (M) | 1 (M) | nvoAnalog | No | lev_percent | 81 | RAM | Analog output |
| 2 (O) | - | nvoRaw | No | lev_percent | 81 | RAM | Sensor raw output |

Table 3.7: Specification table for output network variables.

¹⁶ This relates to an adapter pattern in OO-Programming where two differently designed classes are made compatible by an adapter class which “transforms” the interface of one class to a useful form of the second class.

¹⁷ This is one common way to build method tables for classes. This method is typically used by compilers for statically-typed languages, such as C++.

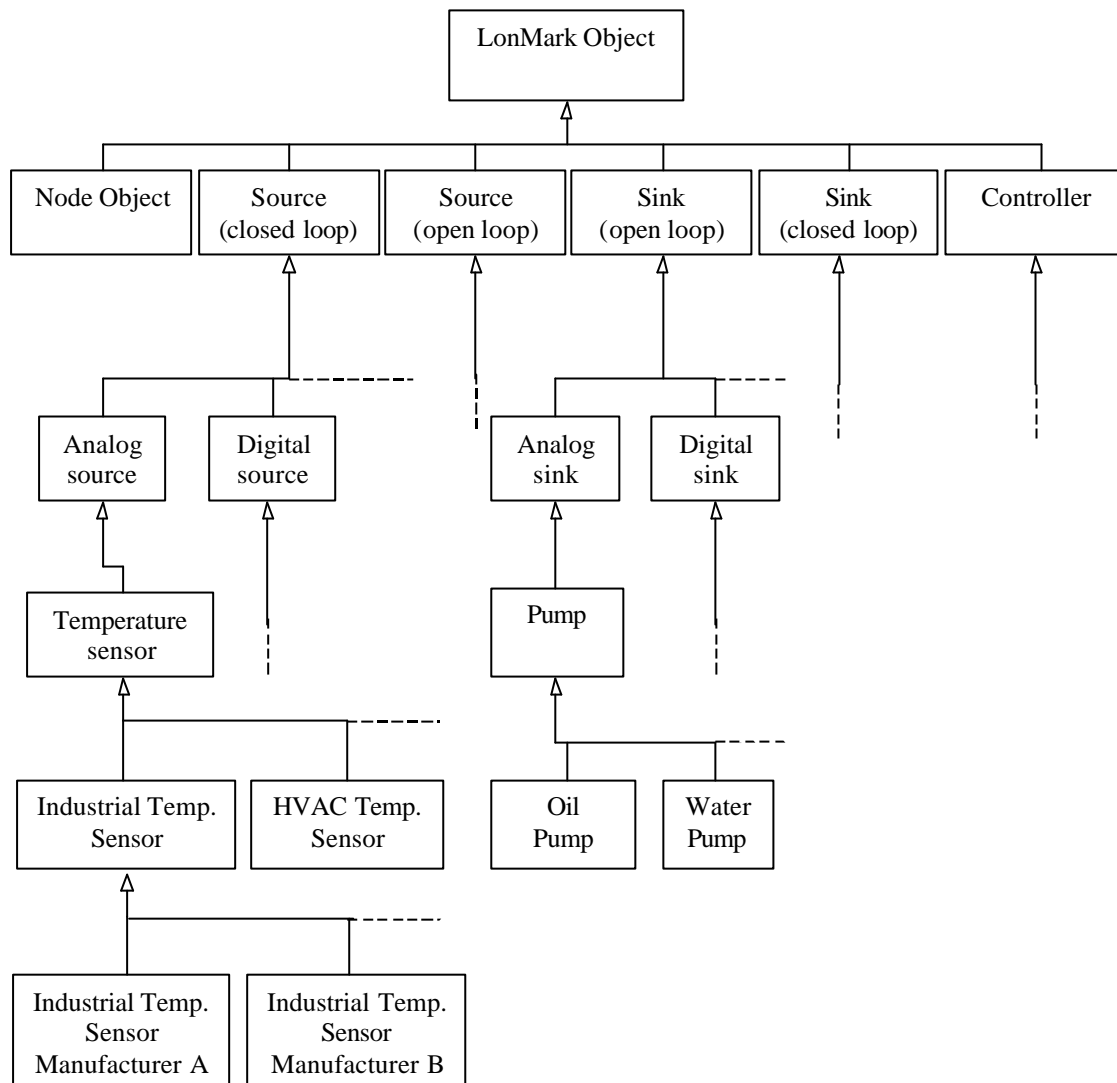


Fig. 3.10: The Augmented Class Hierarchy

In this example, the generic analog sensor is extended to a new profile.

All network variables and their key properties are listed.

- The NV# property is the network variable index within the object. An (M) indicates a mandatory network variable, while an (O) indicates an optional network variable.
- The NV# parent property indicates the network variable index within the parent object and whether it is optional or mandatory. The field is empty, if the network variable has been added by the derived object.
- The Name is the Neuron-C variable name.
- The Heartbeat Property indicates whether the network variable is updated periodically, even if there are no new values.
- The SNVT Type and SNVT Index fields define the data type of the network variable. They are redundant, but it's convenient to have both at hand.
- The Config Class field specifies whether the variable is volatile (stored in RAM) or non-volatile (stored in EEPROM).
- The Description field shortly defines the meaning and usage of the network variable.

If a network variable is inherited, the NV index within the parent object is printed, else it is omitted. The specification in the derived object must be not weaker than the specification in the parent object. This means that a derived class might define a mandatory network variable that is only optional in the parent class. The opposite is not true, if a network variable is mandatory, it has to be mandatory for all child classes. This is based on the semantics of aggregation specialization. Mandatory items have a “1” multiplicity, while optional items have a [0..1] multiplicity. The objects of the derived class must subset the objects of the base class, and so the multiplicity range can only remain or shrink, but not grow.

The table for an input network variable looks very similar, as shown in Table 3.8.

| AnalogActuatorWithOffset extends GenericAnalogActuator | | | | | | | |
|--|------------|-----------|------------|-----------------|------------|--------------|--------------|
| NV# | NV# parent | Name | Heart-Beat | SNVT Type SNVT_ | SNVT Index | Config class | Description |
| 1 (M) | 1 (M) | nviAnalog | No | lev_percent | 81 | RAM | Analog input |
| 2 (O) | - | nviOffset | No | lev_percent | 81 | RAM | Offset value |

Table 3.8: Specification table for input network variables

This example creates a new profile by extending the generic analog input profile.

For SCPTs, a table like Table 3.9 can be used:

| ExtendedProfile extends GenericProfile | | | | | | |
|--|------------|-------------|-----------------|------------|--------------|------------------------|
| CP# | CP# parent | Name | SNVT Type SNVT_ | SCPT Index | Config class | Description |
| 1 (M) | 1 (M) | nciSndHrtBt | time_sec | 107 | NVM | Send Heartbeat Time |
| 2 (O) | - | nciLocation | str_asc | 36 | NVM | Additional Location ID |

Table 3.9: Specification table for configuration properties

It is organized like a SNVT inheritance table, but contains important properties of the SCPTs and their relationships to the parent class.

How can these extended network variable tables be used to take advantage of the class hierarchy? If the designer uses a specialized profile, all network variables that are required by the design are present in the profile. If the designer has used the generic profile, only the variables of the general profile have to be considered. The additional variables, which are defined by a child profile, can be omitted. When integrating the system, any child class can be used instead of the parent class.

Note, that the designer has to decide whether the design should be more abstract or more concrete. Both decisions have their advantages and disadvantages:

1. Using more generic classes allows to exchange concrete profiles. The bad thing is that only the common subset of these profiles can be used for system specification.
2. Using the concrete profiles allows to use their special extensions, but makes it impossible to exchange the device by another profile with the same parent.

If an independently defined profile has totally incompatible SNVTs, it cannot be used as a subtype of the generic profile. This means the profiles cannot be used in an interoperable manner, since the data types are different. This problem arises because there are different integer and floating point network variable types for the same physical units.¹⁸

If a design uses a generic profile, any derived profile of that generic profile can be used in its place. For each network variable in the parent profile, the corresponding network variable in the child profile must be searched in the network variable or configuration property table of the child class. This way, the network variables of the specialized profile are used in the right

¹⁸ For example, integer kilograms, integer grams, floating-point kilograms,

place. Usually, the additional properties of the specialized profile are not used, since only the generic properties are needed.

3.7.3 Root Class Layer

The root class is the parent of all LonMark objects. It defines all properties and methods common to all LonMark objects.

- Each object has an integer number which identifies itself within a node. This corresponds to the order in which LonMark objects are listed in the self-documentation string.
- Each network variable has an index which is unique within the object. Since this is true for every functional profile, this fact is defined in the root object. In other words, the root class defines the possibility of having network variables and configuration properties and provides methods to access them. Further a network variable has an index which is unique within the node (network variable table order). Each object can have network variables taken from following sets:
 - Mandatory input variables
 - Mandatory output variables
 - Optional input variables
 - Optional output variables
 - Manufacturer defined input variables
 - Manufacturer defined output variables
- Each object can contain configuration properties
When the configuration properties are implemented as a downloadable file, the node object must implement the LonMark FTP protocol.
- Each object can interface to its physical environment by
 - Hardware output objects
 - Hardware input objects

If an object has network variables or configuration properties, it automatically provides access to them (public visibility).

3.7.4 Signal Direction Layer

The hierarchy layer copes with the primary data flow of LonMark objects. The classes are derived from the basic LonMark objects found in [LonM731]. There are 6 basic classes:

1. The node class
2. Open loop sensor classes
3. Closed loop sensor classes
4. Open loop actuator classes
5. Closed loop actuator classes
6. Controller classes

The key property of each sensor class is one or several output network variables, while each actuator class must have one or more input network variables. Controllers must have both, input and output network variables.

The node class is equal to the node object predefined and cannot be extended.¹⁹

¹⁹ Unfortunately, the LonMark nomenclature doesn't distinguish between classes and objects. We do so, since it helps making a difference between the specification and an implementation (instance) of a particular profile.

Table 3.10 to Table 3.14 show the signal direction profiles and the definition of their minimum network variables. Table 3.15 and Table 3.16 show the elements of the node object.

OpenLoopSource

| NV# | NV# parent | Name | Heart-Beat | SNVT Type SNVT_ | SNVT Index | Config class | Description |
|-------|------------|----------|------------|--------------------|------------|--------------|-------------|
| 1 (M) | - | nvoValue | | ANVT_any | - | RAM | |

Table 3.10: Open Loop Sensor

ClosedLoopSource

| NV# | NV# parent | Name | Heart-Beat | SNVT Type SNVT_ | SNVT Index | Config class | Description |
|-------|------------|------------|------------|--------------------|------------|--------------|-----------------------|
| 1 (M) | - | nvoValue | | ANVT_any | - | RAM | |
| 2 (M) | - | nviValueFb | | ANVT_any | - | RAM | Same type as nvoValue |

Table 3.11: Closed Loop Sensor

OpenLoopSink

| NV# | NV# parent | Name | Heart-Beat | SNVT Type SNVT_ | SNVT Index | Config class | Description |
|-------|------------|----------|------------|--------------------|------------|--------------|-------------|
| 1 (M) | - | nviValue | | ANVT_any | - | RAM | |

Table 3.12: Open Loop Actuator

ClosedLoopSink

| NV# | NV# parent | Name | Heart-Beat | SNVT Type SNVT_ | SNVT Index | Config class | Description |
|-------|------------|------------|------------|--------------------|------------|--------------|-----------------------|
| 1 (M) | - | nviValue | | ANVT_any | - | RAM | |
| 2 (M) | - | nvoValueFb | | ANVT_any | - | RAM | Same type as nviValue |

Table 3.13: Closed Loop Actuator

Controller

| NV# | NV# parent | Name | Heart-Beat | SNVT Type SNVT_ | SNVT Index | Config class | Description |
|-------|------------|------------|------------|--------------------|------------|--------------|-------------------------------|
| 1 (M) | - | nvoOutput1 | | ANVT_any | - | RAM | First output network variable |
| 2 (M) | - | nviInput1 | | ANVT_any | - | RAM | First input network variable |

Table 3.14: Controller

NodeObject

| NV# | NV# parent | Name | Heart-Beat | SNVT Type SNVT_ | SNVT Index | Config class | Description |
|-------|------------|------------------|------------|--------------------|------------|--------------|-------------|
| 1 (M) | - | nviRequest | | obj_request | | RAM | |
| 2 (M) | - | nvoStatus | | obj_status | | RAM | |
| 3 (O) | - | nviTimeSet | | time_stamp | | RAM | |
| 4 (O) | - | nvoAlarm | | alarm | | RAM | |
| 5 (O) | - | nviFileReq | | file_req | | RAM | |
| 6 (O) | - | nvoFileStat | | file_status | | RAM | |
| 7 (O) | - | nviFilePos | | file_pos | | RAM | |
| 8 (O) | - | nvoFileDirectory | | address | | RAM | |

Table 3.15: Node Object (NVs)

NodeObject

| CP# | CP# parent | Name | SNVT Type SNVT_ | SCPT Index | Config class | Description |
|-------|------------|----------------|--------------------|------------|--------------|-------------|
| 1 (O) | - | nciNetConfig | config_src | 25 | NVM | |
| 2 (O) | - | nciMaxStsSendT | elapsed_tm | 22 | NVM | |

Table 3.16: Node Object (CPs)

3.7.5 Signal Type Layer

Sensor and actor classes can be further classified by the basic signal type which is produced or consumed.

Sensors or actuators having the same signal type typically have comparable operations. Each analog sensor can be read, has typical send times, delta values and so on. It doesn't matter whether a temperature or a pressure is measured, the operations are basically the same.

Control algorithms also typically apply to a basic signal type and not on a certain physical unit. All control algorithms typically use normalized equations that can be applied to any physical unit.

For actual profiles, following generic profiles are necessary at the signal type layer. The network variables are redefined to more concrete, but still abstract, types. The interfaces of these classes are shown in Table 3.17 to Table 3.24.

AnalogOpenLoopSource extends OpenLoopSource

| NV# | NV# parent | Name | Heart-Beat | SNVT Type | SNVT Index | Config class | Description |
|-------|------------|----------|------------|-------------|------------|--------------|-------------|
| 1 (M) | 1 (M) | nvoValue | | ANVT_analog | - | RAM | |

Table 3.17: Analog Open Loop Source Interface

AnalogClosedLoopSource extends ClosedLoopSource

| NV# | NV# parent | Name | Heart-Beat | SNVT Type | SNVT Index | Config class | Description |
|-------|------------|------------|------------|-------------|------------|--------------|-----------------------|
| 1 (M) | 1 (M) | nvoValue | | ANVT_analog | - | RAM | |
| 2 (M) | 2 (M) | nviValueFb | | ANVT_analog | - | RAM | Same type as nvoValue |

Table 3.18: Analog Closed Loop Source Interface

DigitalOpenLoopSource extends OpenLoopSource

| NV# | NV# parent | Name | Heart-Beat | SNVT Type | SNVT Index | Config class | Description |
|-------|------------|----------|------------|--------------|------------|--------------|-------------|
| 1 (M) | 1 (M) | nvoValue | | ANVT_digital | - | RAM | |

Table 3.19: Digital Open Loop Source Interface

DigitalClosedLoopSource extends ClosedLoopSource

| NV# | NV# parent | Name | Heart-Beat | SNVT Type | SNVT Index | Config class | Description |
|-------|------------|------------|------------|--------------|------------|--------------|-----------------------|
| 1 (M) | 1 (M) | nvoValue | | ANVT_digital | - | RAM | |
| 2 (M) | 2 (M) | nviValueFb | | ANVT_digital | - | RAM | Same type as nvoValue |

Table 3.20: Digital Closed Loop Source Interface

AnalogOpenLoopSink extends OpenLoopSink

| NV# | NV# parent | Name | Heart-Beat | SNVT Type | SNVT Index | Config class | Description |
|-------|------------|----------|------------|-------------|------------|--------------|-------------|
| 1 (M) | - | nviValue | | ANVT_analog | - | RAM | |

Table 3.21: Analog Open Loop Sink Interface

AnalogClosedLoopSink extends ClosedLoopSink

| NV# | NV# parent | Name | Heart-Beat | SNVT Type | SNVT Index | Config class | Description |
|-------|------------|------------|------------|-------------|------------|--------------|-----------------------|
| 1 (M) | - | nviValue | | ANVT_analog | - | RAM | |
| 2 (M) | - | nvoValueFb | | ANVT_analog | - | RAM | Same type as nviValue |

Table 3.22: Analog Closed Loop Sink Interface

DigitalOpenLoopSink extends OpenLoopSink

| NV# | NV# parent | Name | Heart-Beat | SNVT Type | SNVT Index | Config class | Description |
|-------|------------|----------|------------|--------------|------------|--------------|-------------|
| 1 (M) | - | nviValue | | ANVT_digital | - | RAM | |

Table 3.23: Digital Open Loop Sink Interface

DigitalClosedLoopSink extends ClosedLoopSink

| NV# | NV# parent | Name | Heart-Beat | SNVT Type | SNVT Index | Config class | Description |
|-------|------------|------------|------------|--------------|------------|--------------|-----------------------|
| 1 (M) | - | nviValue | | ANVT_digital | - | RAM | |
| 2 (M) | - | nvoValueFb | | ANVT_digital | - | RAM | Same type as nviValue |

Table 3.24: Digital Closed Loop Sink Interface

3.7.6 Industry-Independent Profile Layer

Generic profiles define particular device classes, such as motors, lamps, heating devices and so on, but do not introduce any industry-specific (such as parameter ranges or device attributes that are only useful for just one industry).

For example, a motor can be realized in many ways. Its primary task is to convert energy into rotational energy. There are many motor types – diesel motors, electrical motors, pneumatic motors. Each one has different properties, different speed ranges, different power output ranges and so on. Each one also has some properties useless for other motor types. A diesel consumption property is useless for an electrical motor, while electric currents and voltages consumption is useless for a diesel motor.

Working out differences and similarities allows to see a device from two directions. Once, the abstract function of a device is important. For example, a controller needs to set an engine to 3000 rpm. It doesn't need to know whether this is an electric or diesel driven device. The network manager just needs to connect the network variables and gets a functional system.

Another time, the exact implementation might be important. For example, an energy management tool needs to know which motor type is used, since it make a difference to get the diesel consumption in liters/h or to get the electric power consumption in kW.

This perfectly fits to the notation of inheritance and methods in the object oriented paradigm. An object can inherit a method from its parent class and override it. When a reference to an object is used to execute the method, the exact object type is automatically determined and the right method implementation is executed. For LonWorks nodes, this is implemented by network variables. Each device can implement the reaction on a network variable operation distinctly. The sender of an update must not care which object is actually updated. The instantiation of a LonMark object is not a software issue. It is done by integrating a node implementing that object into the system. The dynamic method binding provided by object-oriented languages is implemented by network variable bindings and hardware integration in LonWorks. For some cases, the exact object type must be known. Especially when creating an object, one has to know what should be constructed. This also applies to LonMark objects. When an object is integrated into a network, its exact type must be known.

Since the relationship between the two viewpoints is defined by the class hierarchy, it is easy to switch between the viewpoints without losing the model consistency.

3.7.7 Profile Layer

Industry-specific profiles refine generic profiles to satisfy the requirements of a specific industry. These classes are equivalent to existing and future functional profiles.

In this layer, industry-specific properties are added to the generic profiles. These can be additional SNVTs and SCPTs which extend the functionality of the generic profiles. These extensions shall not modify the already defined basic functionality.

In the discussed example, a diesel motor might use additional SNVTs and SCPTs to report the current diesel consumption, fuel level, oil temperature and so on.

A specialized profile can be used wherever a generic profile is expected. A specialized profile has to contain all properties of the generic profile, since it only can add new properties, but cannot take away already defined properties.

The opposite is not true. If a specialized profile is used, a generic device cannot be used since it does not provide all properties of the specialized profile. For example, if the designer decided to use the electric motor and also uses its special properties, a diesel motor cannot be used in this case.

3.7.8 The Manufacturer-Dependent Layer

A manufacturer can add proprietary network variables or configuration properties to an existing profile. They can provide additional features or extended configuration possibilities. However, they cannot change the interface defined by the other layers. The inheritance relationship guarantees interchangeability when the unique features are not required. If they are required, other parts of the system have to support that extended features, too.²⁰ The proprietary section of the LonMark profiles can be encapsulated in the manufacturer-dependent layer.

An inheritance table must be defined to correlate the manufacturer-dependent profile with the more generic profile.

²⁰ These may be profiles or devices from a product family of the manufacturer.

4 Analysis Results

4.1 Existing Profiles

4.1.1 Released Profiles

Released profiles are grouped into industry areas:

- HVAC profiles deal with heating, ventilation and air-conditioning devices.
- Input/Output profiles provide simple dimensionless basic sensor and actuator types which can be used for implementing devices using a percentile input or output.
- Sensor devices provide interfaces for different sensor types, such as temperature, pressure or occupancy sensors.
- Energy Management profiles define the interfaces for devices used for monitoring and optimizing the energy consumption of an unit or a complete facility.
- Lighting profiles deal with devices for artificial light, sensor devices and HMI devices for controlling the actual illumination.
- Motor Control functional profiles provide interfaces for motor actuators.
- Sensor functional profiles contain interfaces for various sensor types. They can be used in conjunction with other industry areas.
- Refrigeration profiles define interfaces for refrigeration devices and their components.
- Fire&Smoke detection profiles are used to implement fire alarm systems. These profiles include fire sensors (called initiators) and alarm devices (called indicators).
- Industrial profiles include interfaces for devices used in the area of industrial automation.

Table 4.1 lists all currently defined functional profiles (January 2000). They are coarsely classified by their industry. Each industry area gets an own block of 1000 numbers.

4.1.2 HVAC

The HVAC category contains profiles for different types of heating, ventilation and air-conditioning devices. These include VAV controllers²¹, fan coil units, roof top units, chillers, heat pumps, thermostats, chilled ceiling controllers, unit ventilator controllers and damper actuators. A room comfort controller template profile provides a very rich interface for a very general HVAC device.

4.1.3 Generic Input/Output

The Analog Input profile provides a SNVT_lev_percent output network variable. It can be used to implement sensors that provide a percentile output. The Analog Output profile can be used to implement a simple actuator that receives a percentile input.

These generic profiles are very simple and can be used in more complex devices which require multiple inputs. They are dimensionless, so on the one hand, they can be used for any normalized value, but on the other hand, they require additional documentation to describe which information is exchanged.

²¹ Variable Air Volume Controllers

| Category | Profile | Number |
|-------------------------------------|-----------------------------------|-----------------------------|
| Sensors | Light sensor | 1010 |
| | Temperature sensor | 1040 |
| | Frost sensor | 1042 |
| | Relative humidity sensor | 1050 |
| | Rain sensor | 1051 |
| | Occupancy sensor | 1060 |
| | CO ₂ sensor | 1070 |
| Utility | Utility data logger | 2110 |
| Lighting | Lamp actuator | 3040 |
| | Constant light controller | 3050 |
| | Occupancy controller | 3071 |
| | Switch | 3200 |
| | Scene panel | 3250 |
| | Scene controller | 3251 |
| | Partition wall controller | 3252 |
| | Real time keeper | 3300 |
| | Real time based scheduler | 3301 |
| | Lighting panel controller | 3401 |
| Motor | Variable Speed Motor Drive | 6010 |
| HVAC | VAV Controller | 8010 |
| | Fan Coil Unit | 8020 |
| | Roof Top Unit | 8030 |
| | Chiller | 8040 |
| | Heat Pump | 8051 |
| | Thermostat | 8060 |
| | Chilled Ceiling Controller | 8070 |
| | Unit Ventilator Controller | 8080 |
| | Space Comfort Command Module | 8090 |
| | Damper Controller | 8110 |
| | Space Comfort Controller | 8500 |
| | Refrigeration | Refrigerator defrost object |
| Refrigerator evaporator ctrl object | | 10011 |
| Refrigerator thermostat object | | 10012 |
| Fire | Smoke intelligent fire initiator | 11002 |
| | Smoke conventional fire initiator | 11003 |
| | Thermal fire initiator | 11004 |
| | Pull station fire initiator | 11005 |
| | Audible fire indicator | 11006 |
| | Visible fire indicator | 11007 |
| | Universal fire initiator | 11010 |
| | Universal fire indicator | 11011 |
| Industrial | Generator set | 13110 |
| | Automatic transfer switch | 13120 |

Table 4.1: Currently published profiles

4.1.4 Energy Management

The Utility Data Logger Register is a device which can log arbitrary data. A single logger can register multiple data streams. It provides some statistical functions to extract logged data. Its

primary purpose is to do energy/power logging, but since its data type can be drawn from a set of predefined units, it can be used in other industry areas too.

4.1.5 Lighting

This group consists of several sensors, actuators and controllers

The switch sensor can be used as a binary or continuous switch, such as a dimmer knob. The lamp actuator defines the interface for a lamp or other equivalent lighting devices.

The Real-Time Keeper and the Real-Time Based Scheduler can be used to implement time-controlled lighting schemes.

A group of profiles is used for entering and using lighting scenes. These are the lighting panel controller, the scene panel, the scene controller and the partition wall controller.

Two controller types are used to provide a constant light level and occupancy-dependent lighting: The occupancy controller and the constant light controller.

4.1.6 Motor Control

The Variable Speed Motor Drive Profile specifies the interface of a generic motor. Its speed can be set from 0% to 100% using a SNVT_switch. The actual meaning of the percentile value is determined by speed SCPTs. The profile also provides an output indicating the actual speed of the motor.

4.1.7 Sensors

Sensor profiles are designed straight forward. They measure a physical entity and report them to the network. Some provide optional network variables with different ranges and resolutions.

The Light Sensor profile defines an illumination sensor. It simply outputs the light level by a Lux SNVT.

The Pressure Sensor profile provides a network interface for reporting pressure values to the network. It has a mandatory pressure network variable and optional pressure network variables for different pressure ranges.

The Temperature Sensor profile has a temperature output network variable and two optional network variables for different temperature ranges.

The Relative Humidity Sensor profile has a percent level output network variable to report the measured values.

The Occupancy Sensor indicates the occupancy state of a room.

Rain and Frost Sensors use a SNVT_switch to indicate rain and frost conditions.

4.1.8 Refrigeration

There are three refrigeration profiles dealing with devices of refrigerators, especially display case refrigeration devices. These are profiles for thermostat, evaporator and defrost controller objects.

4.1.9 Fire & Smoke Detection

The fire detection profiles are divided into two groups:

The fire initiator devices are binary sensors indicating fire conditions. There are universal, smoke and thermal initiators.

Fire indicator devices are binary actuators for visible and audible fire alarms.

4.1.10 Industrial

The generator set profile is an interface for generator sets. Its allows to control and monitor generators.

The automatic transfer switch profile is used to control transfer switches that connect an electric load with one of two power sources.

4.2 Profile hierarchy

The reviewed profiles can be inserted into the following class hierarchy (Table 4.2) (shown as a textual inheritance tree).

LonMark object

```

----- NodeObject
----- OpenLoopSource
----- ----- AnalogOpenLoopSource
----- ----- ----- Light Sensor
----- ----- ----- Temperature Sensor
----- ----- ----- Relative Humidity Sensor
----- ----- ----- CO2 Sensor
----- ----- DigitalOpenLoopSource
----- ----- ----- Fire Initiator
----- ----- ----- ----- Smoke (intelligent) fire initiator
----- ----- ----- ----- Smoke (conventional) fire initiator
----- ----- ----- ----- Thermal fire initiator
----- ----- ----- ----- Pull station fire initiator
----- ----- ----- ----- Universal fire initiator
----- ----- ----- Rain Sensor
----- ----- ----- Frost Sensor
----- ----- ----- Occupancy Sensor
----- ----- ----- Automatic Transfer Switch
----- ----- ----- Real Time Keeper
----- ----- ----- Real Time Based Scheduler
----- ClosedLoopSource
----- ----- AnalogClosedLoopSource
----- ----- ----- Switch
----- OpenLoopSink
----- ----- DigitalOpenLoopSink
----- ----- ----- Fire Indicator
----- ----- ----- ----- Fire Audible Indicator
----- ----- ----- ----- Fire Visible Indicator
----- ----- ----- ----- Universal Fire Indicator
----- ----- AnalogOpenLoopSink
----- ----- ----- Generator Set
----- ClosedLoopSink
----- ----- AnalogClosedLoopSink

```

```

----- Lamp
----- Variable Speed Motor Drive
----- Controller
----- Utility Data Logger Register
----- Constant Light Controller
----- Occupancy Controller
----- Scene Controller
----- Scene Panel
----- Partition Wall Controller
----- Lighting Panel Controller
----- Chiller
----- Space Comfort Control Command Module
----- Damper Actuator
----- Defrost Controller
----- Evaporator Controller
----- Thermostat Controller
----- HVAC Controller
----- VAV Device
----- Fan Coil Unit
----- Roof Top Unit
----- Heat Pump
----- Thermostat
----- Chilled Ceiling Controller
----- Unit Ventilator Controller
----- Space Comfort Controller

```

Table 4.2: Inheritance tree for LonMark profiles

4.3 Natural Hierarchy

4.3.1 Introduction

Since there are many similar objects (see fire and HVAC profiles), their similarities can be shown by extracting common network variables into a fictive base class. This hierarchy shows how which profiles can be derived from a common base class.

4.3.2 Sensor Devices

The sensor classes don't have common concrete network variables, since all of them have different SNVT types for their primary output. This is because of their different physical units. The occupancy sensor is isolated, since its primary sensor data type is an enumeration.

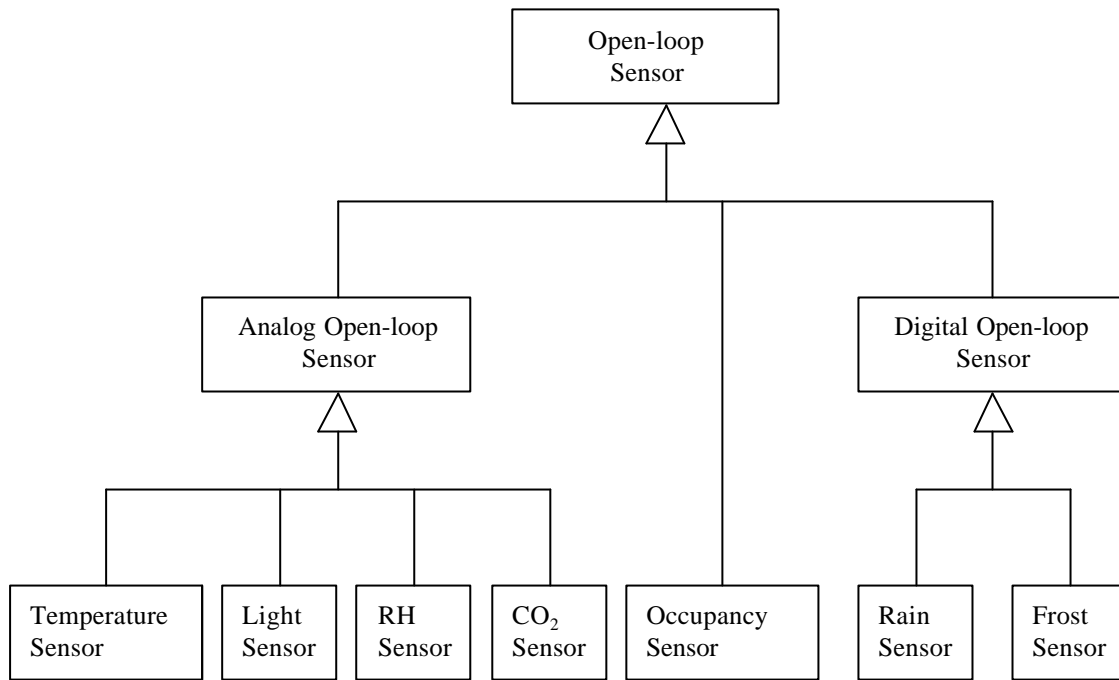


Fig. 4.1: Sensor class hierarchy

4.3.3 Fire Alarm Devices

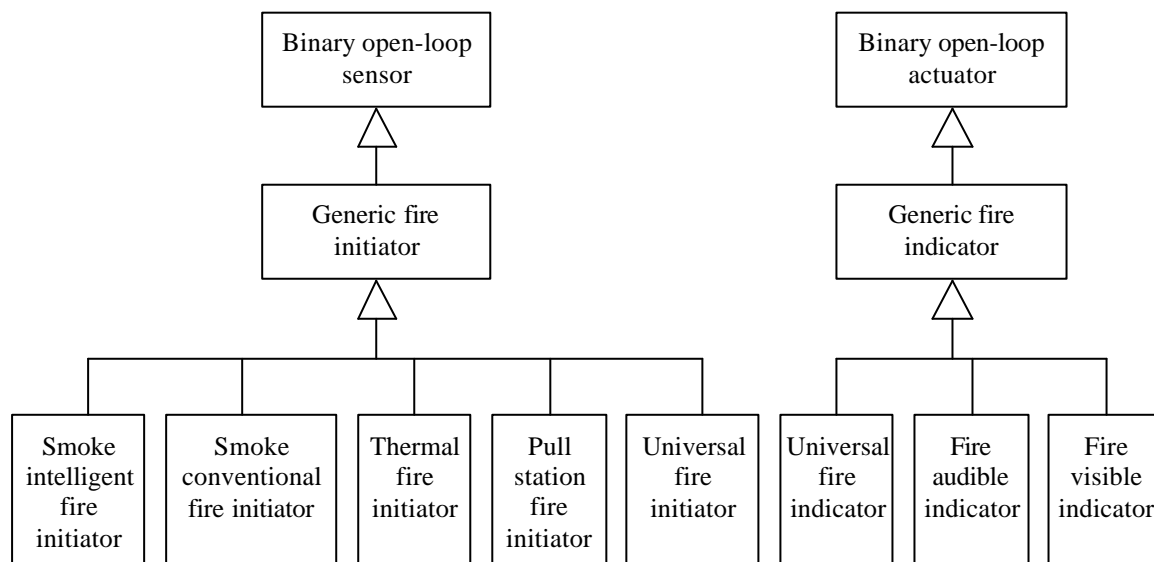


Fig. 4.2: Fire alarm hierarchy

All fire indicators and fire initiator profiles have a small and very similar interface, so they are well suited for generalization. In fact, the universal initiators and indicators seem to have the purpose of a base class.

All initiators and indicators are exchangeable, since they provide large common interface parts.

The Switch Information Sources and Switch Information Sinks are basically digital information sinks and sources that rely on the switch SNVT which integrates a digital switch

and an analog value (for example used for light dimmers).

4.3.4 HVAC Devices

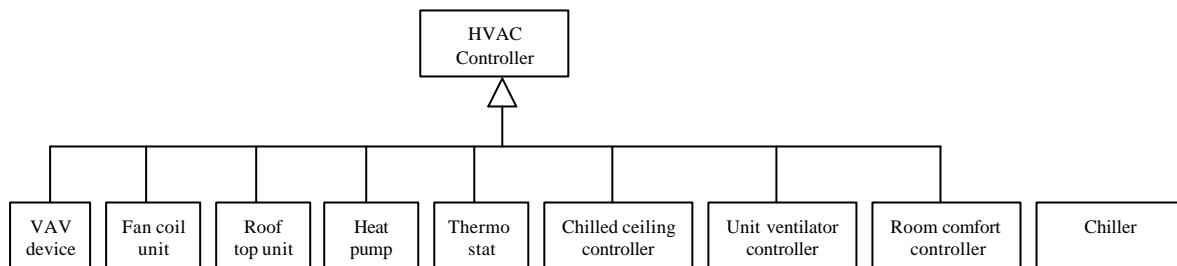


Fig. 4.3: HVAC profile hierarchy

A generic HVAC controller class can be defined for nearly every HVAC profile. The common subset includes the most important network variables, such as temperature and setpoints, as well as occupancy. Unfortunately, even some important network variables like the setpoint input are optional in some profiles, so they need to be optional in the generic HVAC controller.

The Chiller has a different interface that doesn't resemble the interfaces of any other HVAC controllers, so it is not derived from the generic HVAC controller class.

4.4 Profile Database

For this analysis, the profile data had to be available in a machine-readable way. Since profiles are only available as documents in Adobe's Acrobat format, the structural information had to be translated into a database.

The database is implemented in Microsoft Access, since this database is available for most members of the working group. It is a prototype database which is used to evaluate the consequences of storing product and profile information in a relational database.

The database implements the model shown in Fig. 2.5.

The table "SNVTs" contains all SNVT types. "SNVT_arith" contains additional information for all SNVT types. "SNVT_enum" contains all enumeration values for enumeration-type SNVTs. SNVT_struct contains additional information for structure-type SNVTs.

SCPTs are listed in the table "SCPTs".

Profiles are listed in the table "Profiles". The association between profiles, network variables and SCPTS is made in the tables "Profiles_Nvs" and "Profiles_Scpts". The profile hierarchy is captured in "Profiles_Hierarchy".

The advantage of having this data stored in a relational database is the possibility to make SQL queries. The relations can be sorted and filtered to retrieve the required information.

A real-life database would also contain a product database where every network variable and configuration property is connected to the profile tables. This would enable a true market research for integrating a particular application since not only marketing but also technical information can be queried.

5 Current Problems

5.1 Overview

This chapter presents some problems of the LonMark Interoperability Guidelines. Each section describes a problem, its implications and its background. When possible, workarounds are presented.

5.2 Missing Aggregates

Sensors can produce complex data which cannot be represented by a single SNVT. For example, a position reader indicating 2 linear positions of an effector. LonMark profiles and SNVTs cannot express new data-types based on simpler ones, so this sensor cannot be built without workarounds. A single measurement contains 2 network variable updates. They must be received by bound receivers in a way that they can reconstruct the original measurement. If there is a many-to-one relationship, the single network variable updates from the senders must be correlated in the right way, else illegal measurements occur at the receiver.

A communication example is shown in Fig. 5.1 (a). Two data sources (S_1 and S_2) send their measurements to a receiver (R).

A first solution could rely on the sequence of network variable updates. The first part is sent first, then the second one. This works, as long as data transfer is not lossy, which cannot be expected on real networks. The loss of two adjacent packets would lead to an illegal measurement. The communication sequence for the example is shown in Fig. 5.1 (b).

Another possible solution is to use three network variables. Two are used for data exchange, while the third, say a SNVT_switch, is used as a trigger. When all data items of an aggregate are transferred, the trigger variable is updated to indicate the validity of the new data. An exemplary sequence is shown in Fig. 5.1 (c). In this example, sender 1 fails to update one of the aggregate parts, so it does not send the synchronization signal. This approach works, but requires some extra work. The data network variables have to be acknowledged. Only if all updates are successful, the trigger variable can be sent. Using this approach, the synchronization problem is solved, but the communication relationship is not expressed very transparently.

Both approaches lack the possibility to express many-to-one relationships, since two or more senders would interfere and produce illegal measurements. A communication sequence that leads to an invalid measurement is shown in Fig. 5.1 (d).

5.3 Fat vs. Thin Interfaces

A functional profile captures the interface of a device. The amount of functionality which is considered belonging to the device determines how large the interface of that device is. Small interfaces – such with a small number of network variables and configuration properties – and fat interfaces – such with a large number of interface elements – are two extreme designs that both have their problems.

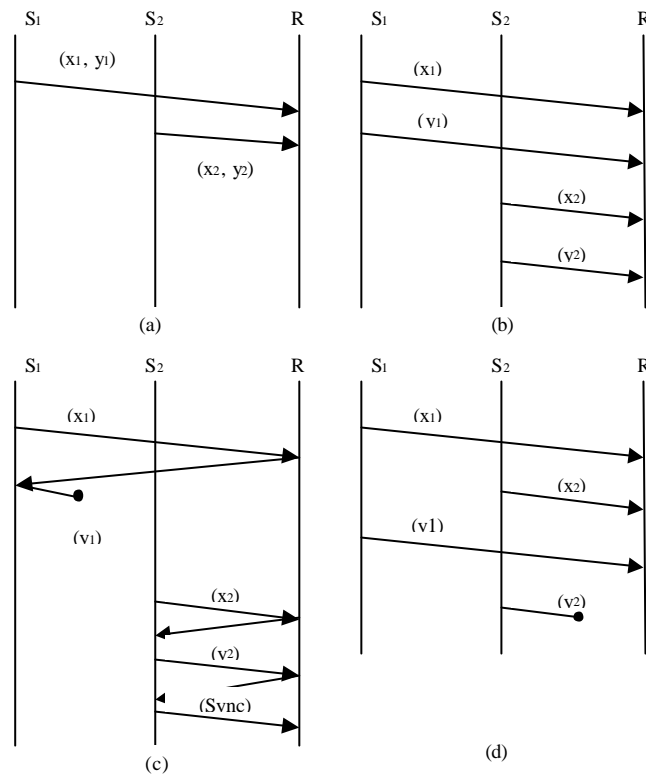


Fig. 5.1: The aggregate problem. (a) A solution with aggregates. (b) An order-based solution. (c) A sync-variable based solution. (d) An illegal measurement due to concurrency.

A profile should contain at least all features that enable it to be useful. It can contain additional features that make it more convenient to use. The additional convenience is gained at the cost of simplicity. Another reason to add interface elements are performance considerations. A single application controlling all network variables directly is more efficient than a couple of applications connected by turnaround bindings. This leads to fat interfaces that implement functionality which is not necessary for each possible use of the device.

Fat interfaces make it difficult to spot the relevant parts of a device. Many network variables are specified as optional, so they may be present. Each optional network variable doubles the number of different interfaces of a profile. E.g., if there are 10 optional network variables, there are $2^{10}=1024$ possibilities to implement that profile. Unfortunately, many optional network variables depend on each other, since they implement a subdevice. There is no way to specify these dependencies using standard LonMark methods.

For these cases it is advantageous to split the profile into its independent parts. For example, the Unit Ventilator Profile can contain a ventilation controller, two temperature sensors (indoor/outdoor), two humidity sensors (indoor/outdoor), a CO₂ sensor and an occupancy sensor. By splitting the profile into its parts, it is possible to use them for different independent control circuits. This allows to use those parts in different contexts, different devices and different industries. The formerly internal dependencies are now externally visible as turnaround bindings. The profile gets more simple to implement. The whole device is better described, since its composition is exposed.

Furthermore, the increased modularity allows a finer control for network installation. Another advantage of making simpler profiles is the increased possibility to create inter-industrial applications. A simple device is less specialized to a certain industry than a complex one is.

The disadvantage of this approach is the increased amount of network management procedures, since the number of LonMark objects increase.

Furthermore, a turnaround binding, although more efficient than a network binding, involves more time overhead than a direct implementation.

A typical example is energy management. Some profiles allow to read the power consumption of a device. If there were a simple profile for an energy meter, this additional LonMark object could be added to nodes and an energy monitor could gather the power consumption of all devices.

5.4 Missing Profiles

If all devices in a system are implemented by functional profiles, every output variable must be bindable to an input network variable. This is possible if each output network variable of a functional profile has a counterpart input network variable in another functional profile.

If this condition is not met – and it is not met by the currently existing functional profiles – parts of a system must be implemented by using non-standard LonMark objects.

When defining a profile, its counterparts (sensors, actuators and controllers) must also be considered. E.g., a HVAC controller should be considered in conjunction with its sensors and actuators.

Note, that the “missing profile” problem is correlated with the fat interface problem. If the building blocks of a system are more granular, the probability for fitting together is larger, since there are less possibilities to make small profiles than to make large ones.

5.5 SNVT Ambiguities

5.5.1 Problem

Some common used data can be represented by more than one SNVT. This is a big interoperability problem, since devices using different SNVTs (for the same data) cannot work together.

5.5.2 Digital Data

There is no special SNVT for generic digital data. Instead, several SNVTs can be used to code on/off-conditions.

The SNVT_lev_disc represents discrete levels and provides enumeration values for “On” (value \in {ST_LOW, ST_MED, ST_HIGH, ST_ON}) and “Off” (value = ST_OFF). The SNVT_switch represents a percentile value plus a digital data²².

Both representations are incompatible and both are used by different profiles. The possibility to choose different data representations for the same data item leads to non-interoperable devices.

This ambiguity appeared historically. The original SNVT set had the SNVT_lev_disc type which was not sufficient for lighting switches (dimmers) and so a more versatile type (SNVT_switch) has been introduced which is used by newer profiles.

5.5.3 Discrete Data

Discrete data items face the same problem as digital data items. Again, SNVT_lev_disc and SNVT_switch can be used to represent these data items. Additionally SNVT_lev_cont is also suitable.

²² This models a complex dimmer which has a brightness setting and additionally can be switched on and off.

The SNVT_lev_disc has five steps (OFF, LOW, MED, HIGH, ON) and thus can represent at most 5-state data.

The SNVT_switch type allows to specify the analog value from 0 to 100 % with a 0.5% resolution. Thus it can represent up to 201-state data. Formulas for calculating a proper percentile value for a certain n-state data is presented in [LM8080].

The SNVT_lev_cont type has a similar range as the SNVT_switch, but lacks the on/off-Feature.

Again, being able to choose a representation for a certain data type (n-way discrete data) is a big opportunity to introduce incompatibilities.

5.5.4 Analog Data

Analog data offers the most opportunities for misinterpretation. Since they map a real number (and a physical unit) to a finite (range and resolution limited) binary data representation, there is a tradeoff between range and resolution.

Most integer SNVTs use a 16-bit data type, so most physical types have 3 to 5 different variations to cover a wider range.

The most clean solution would use the floating-point variations of the SNVTs. They scale well to different ranges and a single type (IEEE 854 single precision) seems to be sufficient for most automation tasks.

Unfortunately, floating point variables are not very well supported by the LonWorks technology. The Neuron-C language does not support floating point arithmetic directly but requires the programmer to use a floating point library that implements floating point operators as functions. Floating point operations are slow compared to integer operations, so they cannot be used for fast control systems.

Another problem appears with the 3120 Neuron-Chip. If a program uses the floating point library, there is little space for the application program in the 3120. Thus, floating point types cannot be used for LonMark profiles if the devices should be implemented on a 3120. If an application requires a different range or resolution, it must request a new SNVT index. This has the disadvantage being incompatible to all other SNVTs.

5.6 Bundled Bindings

Often, a set of network variables is bound in the same way again and again, e.g. the network variables for occupancy control. LonWorks devices are typically bound network variable by network variable. For large systems, this can be a long and daunting task. To solve this problem, a new element would be needed. Network variables that are bound together from one to another object in the same way should have a modeling element that allows bundled bindings.

For example, a TV set and a VCR can be connect via 4 cables: a video cable, 2 audio cables, and one cable for switching the TV set between TV and VCR mode. This scheme is flexible, since one can modify the “binding”, so that the audio cables are connected to the HIFI amplifier instead of the TV set, when a concert is transmitted. However, in most cases, all 4 cables will be connected in the same way from the VCR to the TV set. Therefore, a SCART cable simplifies the “binding” procedure. Instead of “binding” four data sources to four data sinks, one has to connect one bundled data source to one bundled data sink.

The same could apply to LonWorks nodes. For commonly used bindings, a SCART-like bundle binding can be used, while the flexibility of individual bindings for special solutions still exists.

5.7 Higher-Level Designs

A problem can be simplified by finding patterns which appear repeatedly again and again. For example, a building typically has several unique rooms (entry halls, special rooms), but there are also rooms which are similar (office rooms).

If such a pattern can be found, a prototype of this room can be defined which is then implemented for each appearance of the pattern.²³ Unfortunately, LonMark does not define a model where such relationships can be implemented by appropriate means. Therefore network management tools must maintain this meta-information²⁴ separately.

Encapsulation is useful for making higher-level design. This concept is well-known to object-oriented designs and helps reducing dependencies between different entities. The idea behind encapsulation is to hide internal information and make only parts of it accessible through a well-defined interface. This also works for larger systems such as a single room, where not all information has to be exported to other rooms.

The designer has to consider the implications of this decision:

- Making the externally visible interface too restrictive can make the integration of higher-level services more difficult. If, for instance, a building consists of non-communicating rooms, services such as energy-management or building management cannot be easily implemented since the required information is not exported.
- If the externally visible interface is too “fat”, useless information is available to other systems. This does not bother them, but can make the system hard to maintain.

To make this more practical: A designer might choose to build a separate room control for each major room in the building. Each room gets an integrated room controller, a light switch, a sunblind, a heating and cooling device and so on. Additionally, every room is equipped with sensors, a light sensor, an occupancy sensor, an indoor-temperature sensor and an outdoor-temperature sensor and so on.

Some of the sensors provide information which can be very important to other systems. The occupancy sensor can be also used for handling access control and fire emergency situations. Other sensors like light sensors might be of little interest for other systems, since the integrated room controller already can handle a low-light situation.

By using encapsulation and defining proper interfaces, the overall system design becomes more simple. Several information sources and targets are not visible at a system level view, since they are completely inside of a component and do not need to be considered.

Fig. 5.2 shows an exemplary room controller and a small subset of attached sensors and actuators. LonWorks does not have a concept for hiding information, so all network variables are seen outside the room. The designer can define a restricted interface nevertheless. There is just no way to enforce design decisions in the implementation, so explicit documentation is required to specify the restrictions. The second part of the figure shows the same situation with a restricted interface. Only network variables intended to be used by other systems are left in the interface.

²³ Ideally, such a pattern should be put into a library where it can be maintained centrally. Current network management tools just allow a copy-paste operation where the relationship between the library and the pattern disappears.

²⁴ Information about the system itself.

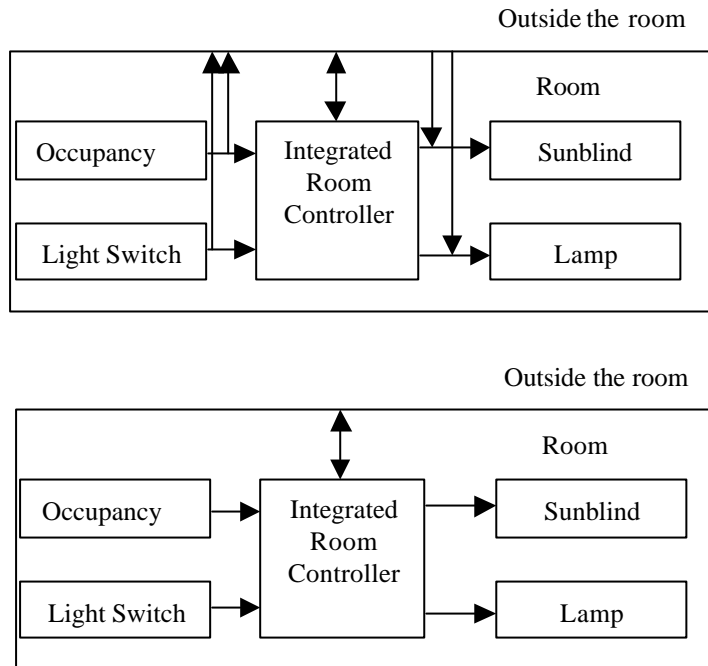


Fig. 5.2: An example room controller without and with applied encapsulation)

5.8 Specification Errors

Some profiles contain specification errors. Some are only typos but some are real severe errors.

For example, the switch should be a closed-loop sensor having mandatory output and feedback input network variables, as defined in the Interoperability Guidelines. The switch has an optional feedback input variable which is a weaker assertion than a mandatory network variable. Thus, a switch implementation could lack the feedback variable and be incompatible with a design that relied on the LonMark specifications.

6 Conclusion

A flat profile model, just listing interfaces, is not an appropriate means for describing large distributed systems. The need for integrating devices from different manufacturers, industries and even different control hierarchy levels requires a more structured approach. Top-down design is a prerequisite to avoid objectless engineering.

The step from an object-based to an object-oriented system model allows to specify relationships between devices, profiles and basic data types. This is the key feature for enabling well-known design methods such as abstraction, refinement and design reuse.

Two related hierarchies allow to describe devices at different abstraction levels. This simplifies vertical integration since unimportant details can be omitted for higher level considerations. The hierarchies defines distinct specification levels, each introducing new interoperability issues. Each layer models a certain interoperability aspect and all layers are correlated by inheritance relationships.

An analysis applies the theory to already defined profiles. Some profiles are well suited for integration in a structured profile model, while others lack compatibility or no suiting abstract parent profile can be found. The lack of a suiting framework lead to a uncontrolled growth of profiles. Future profiles can benefit from the theory since they can take inter-industry interoperability considerations into account. Vertical integration can be easily applied if proper generic profiles are defined.

Problems of the current guidelines, their reasons and known workarounds have been analyzed and discussed. Some applications, especially those with complex data flows, cannot be solely implemented with standard network variables.

This is the first step for making the specification of LonWorks based control systems and devices more coherent. The existing LonMark model concentrates on devices and does not cover system structures. The next step will be the definition of design patterns for selected applications. These patterns can be applied for often required automation tasks. Please look for further SIIA publications of the LNO in this area which will go deeper into detail and also bring practical hints and applications.

The goal of each interoperability system must be a theory, a model and tools to describe and handle control systems and their devices coherently. We hope that this work initiates further proceedings in the development of such a system. We greatly appreciate comments, additions and support. Please contact rauscher@ict.tuwien.ac.at for more information.

Literature

- [ASN87] ISO: Information processing systems - Open Systems Interconnection. Specification of Abstract Syntax Notation One (ASN.1). International Standard 8824. December 1987.
- [LonM731] LonMark: Application Layer Guidelines. Version 3.1. USA: LonMark Interoperability Association. 1996.
- [LonM1630] LonMark: Layers 1-6 Interoperability Guidelines. Version 3.0. USA: LonMark Interoperability Association. 1994.
- [LonT97] D. Dietrich, D. Loy, H. Schweinzer (Hrsg.): LON-Technologie. Verteilte Systeme in der Anwendung. Hüthig-Verlag. Heidelberg. 1997.
- [LM8080] LonMark: LonMark Function Profile: Unit Ventilator Controller. USA: LonMark Interoperability Association. 1998.
- [LNO1699] LNO: LNO Brief. Ausgabe 16/99. Germany. TEMA. 1999.
- [MT2499] Markt&Technik. Unklare Verhältnisse. 24/99.
- [SCPT] Echelon: The SCPT Master List. USA: Echelon Corporation January 1995.
- [SNVT] Echelon: The SNVT Master List and Programmer's Guide. USA: Echelon Corporation January 1995.
- [OMG99] Object Management Group: OMG Unified Modeling Language Specification (draft) V1.3. June 1999.
- [XDR95] Srinivasan, R: XDR. External Data Representation Standard. August 1995. RFC 1832.