The Facility Control Markup Language FCML

François Bry*, Bernhard Lorenz†, Hans Jürgen Ohlbach*, Martin Roeder†, and Marc Weinberger†
*Institute for Informatics, University of Munich, Oettingenstr. 67, 80538 München, Germany
http://www.pms.ifi.lmu.de, Email: {bry|ohlbach}@ifi.lmu.de
†Skytec AG, Keltenring 11, 82041 Oberhaching, Germany
http://www.skytecag.com, Email: {bernhard.lorenz|mroeder|mweinberger}@skytecag.com

***Abstract*—In the domain of Supervisory Control and Data Acquisition (SCADA), significant challenges reside in the variety of proprietary interfaces and protocols which accompany the components and devices provided by different manufacturers. Centralized supervision and control is hampered by incompatibility issues and additional costs occur because a number of different systems have to be installed and maintained. The Facility Control Markup Language (FCML) is designed to provide standardized and uniform access to different devices which usually only adhere to proprietary interfaces and protocols. This way, operators of SCADA systems can easily integrate and access additional devices and manufacturers can offer their products to a greater number of customers due to increased interoperability. As an application of the Extensible Markup Language (XML), FCML is designed as an open and extensible standard which can easily be adapted and extended to include requirements of future applications and devices. FCML is not platform dependent, uses open software components and standards, and reduces the resource requirements of Remote Terminal Units (RTUs). One vision subjacent to FCML is that web-based SCADA systems are very promising for a wide range of SCADA applications, for reasons of modularity and ease of deployment.**

I. Introduction

This section starts with a description of the fundamentals of Supervisory Control and Data Acquisition (SCADA) systems. Then, basic concepts and the system design of the Facility Control Markup Language (FCML) and their relation to SCADA systems are introduced. The subsequent sections contain an overview of the role of FCML in SCADA systems, parts of the specification, and some examples, followed by the conclusion and a discussion of possible extensions and future application scenarios. A comprehensive introduction to SCADA technology is beyond the scope of this article, see, for example, Daneels and Salter [1] instead.

A. SCADA Systems

To operate complex automated systems, such as manufacturing assembly lines, a number of different components have to interact in an organized way. So called Remote Terminal Units (RTUs), consisting of a programmable logic converter, are attached to a device such as a sensor, actuator, electric motor, valve, switch, or a more complex combination of a number of devices in form of, for example, a conveyor belt. RTUs communicate with one another, usually controlled and supervised via a Human Machine Interface (HMI). If more than a few components have to exchange data, this cannot be achieved by direct connections between all RTUs (not counting the HMI, a complete network connecting 10 RTUs would already require 45 direct connections, and each RTU would require 9 I/O ports). One solution is a common communication bus, the fieldbus, usually in form of a star, bus, or ring topology, which connects all necessary components and standardizes communication. Due to the very different requirements of their individual application, a number of fieldbus systems exist today. In the IEC 61158 specification [2], eight protocol sets have been defined, e.g. PROFIBUS [3], Interbus, and FOUNDATION Fieldbus H1 [4]. There exist many more, each designed for individual use in a specific domain, such as aviation, automotive, process control, and automation. LON [5], LCN, and BACnet, for example, can be found in the domain of building automation. This is only one dimension of heterogeneity. Similar effects are introduced by the use of different hardware and software platforms, programming environments, network infrastructures, and different standards in hardware and software of interfaces.

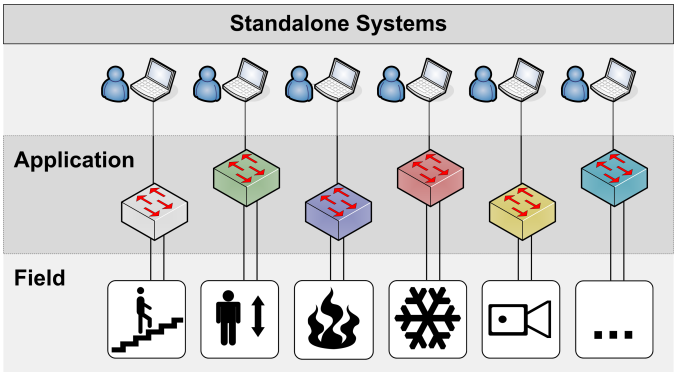

Fig. 1. Standalone Systems

Due to the number of different components, interfaces, fieldbusses, and manufacturers, most management and control systems come in form of incompatible, proprietary standalone systems, as shown in figure 1, which have to be operated in parallel. A setup like this generates increased capital (CAPEX) and operational expenditures (OPEX) due to parallel installation and maintenance cost for independent systems. Furthermore, data exchange between systems is often cumbersome or not feasible at all. This problem remains, even if, in some cases, the necessary proprietary applications can be installed on a smaller number of clients. A key issue in this scenario is that almost all communication between the different components is realized using non-standard proprietary

interfaces and/or protocols. Another factor can be the impelled commitment to specific platforms or systems.

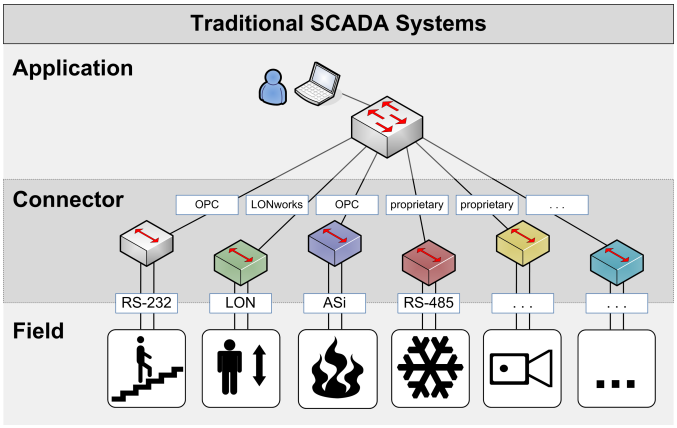

Fig. 2. Traditional SCADA Systems

Traditional SCADA systems try to alleviate these issues by providing a unified HMI which features a number of proprietary hardware and software interfaces. Figure 2 illustrates a traditional SCADA setup. Here, a centralized HMI provides access to a number of so-called connectors. Between the application level and the connector level, many different hardware and software protocols (e.g. OPC, LON, LONworks, other proprietary protocols) have to be implemented. Although this reduces the number of systems that have to be installed and maintained, the problems of different identifiers, units of measurement, and different formats for data and commands remain. FCML was designed to standardize communication at this level. As shown in figure 3, communication between the application level (HMI) and the connector level is entirely achieved using FCML and standardized network infrastructure. As FCML is platform independent, there is no need to commit to a certain hardware or software system, manufacturer, or vendor. Devices which implement FCML do not require a dedicated connector and can be directly attached to the network.

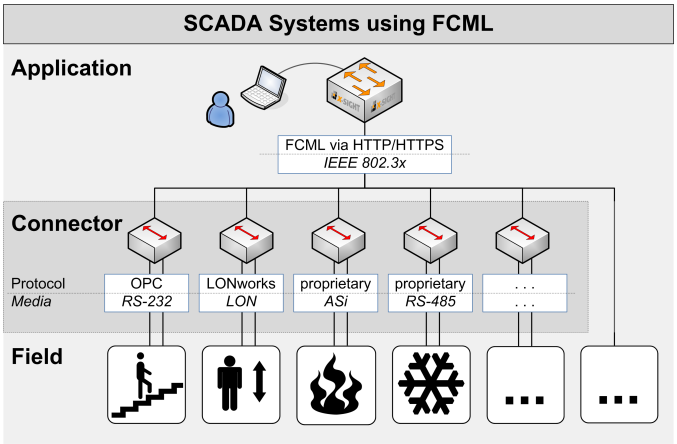

Fig. 3. A SCADA System using FCML

B. OPC and DCOM

A similar approach to achieve standardized access at this level is OPC, formerly an acronym derived from OLE for Process Control but now an independent term (referring to Openness, Productivity, and Connectivity [6]; OLE stands for Object Linking and Embedding [7]). While the main goal behind OPC is similar to FCML, there exist certain differences. OPC is not platform independent (see next paragraph) and it is not an open standard, it has to be licensed from the OPC Foundation.

OPC is based on the Distributed Components Object Model (DCOM), a proprietary standard from Microsoft which facilitates communication between software components distributed across several networked computers. While DCOM furthered the dissemination of OPC, it also introduced some major drawbacks:

- frequent configuration issues
- no configurable timeouts
- only available for Windows
- no inherent security features
- black box issue (closed source)

DCOM and its main competitor CORBA (for a comparison, see, for example, Thompson and Watkins [8], Plasil and Stal [9], or Watkins [10]) were originally devised as models for code and service reuse over the internet. This, however, did not materialize due to persistent difficulties in making these technologies work through firewalls and on unknown or insecure machines. Instead, HTTP and generic browser software have taken over.

As any other binary protocol, OPC DA has some intrinsic disadvantages in comparison with a format based on the Extensible Markup Language (XML) [11]:

- not human readable
- no document-inherent structure
- data visualization involves much more than, for example, an XSL-T style sheet
- often proprietary (license fees)
- often platform dependent
- long term support often uncertain

OPC XML DA, an XML version, is not widely available, due to an upcoming major revision of OPC which is expected to lead to an improved unified standard, the OPC Unified Architecture (OPC UA) [12]–[14]. Parts of this specification are still in “draft” or “release candidate” state.

C. FCML

FCML [15] is an application of XML used to describe the properties of, and control, various types of devices, such as elevators, escalators, fire dampers, or motion sensors.

Among many others, these types of devices can be found, for example, in shopping malls, airports, train stations, or similar facilities. Supervision and control is ideally achieved through a centralized system. In the domain of SCADA systems, this is still a problematic issue, due to the fact, that

the different devices are often provided by different manufacturers. Using proprietary software, interfaces, and protocols is often the only possibility to control and supervise a certain device, and integration in existing SCADA infrastructure is cumbersome and results in a number of problems:

- no standardized interface / protocol
- dependence on individual manufacturers
- increased complexity due to a number of different interfaces and formats
- data from different sources cannot be normalized and processed efficiently (different identifiers and/or units for the same data)

Using FCML, these issues are solved by mapping proprietary interfaces on open, uniformly structured XML-based interfaces:

- unified (remote) commands
- unified parameter identifiers
- unified values, ranges, and units

Devices of different manufacturers can be attached to an FCML-based centralized management system as long as they provide an FCML interface. Components which cannot provide such an interface (either due to proprietary standards or due to lack of computing resources) can be attached by using an intermediary device (e.g. PC, embedded device).

Thus, facility operators not only benefit from better integration of different devices of a number of manufacturers, but also from the availability and comparability of statistical data documenting the operation of each individual component. Operators can also flexibly decide on new devices because they are independent of proprietary technology. Manufacturers benefit from increased compatibility of their products which facilitates easy integration in existing infrastructures.

II. FCML in Detail

The FCML specification defines the basic structure of the different document types which are described below. Device specific extensions are defined in separate schema definitions. Thus, a device specific extension of the FCML specification requires only an additional (device specific) schema definition, but no changes to the core specification. Independent from its type and/or manufacturer, a broad range of devices can, therefore, be integrated into an existing SCADA installation.

A. Connecting via FCML

If (existing) devices which do not implement FCML have to be connected to a SCADA system, an FCML interface becomes necessary. Here, device specific data and commands have to be transformed from proprietary formats into FCML, and vice versa. Thus, this component operates two I/O interfaces:

- the FCML interface to the HMI (possibly connected via intermediary components) is based on IEEE802.x and HTTP/HTTPS
- the interface to the device controller (e.g. RS-232, USB, a fieldbus) implements the proprietary protocol

B. FCML Interface

FCML compliant devices have to support the methods "GET" and "POST" as defined by the HTTP [16], [17] protocol, they also have to provide standardized access in form of the five URLs shown in tables I and II. Secure communication via Transport Layer Security (TLS) [18], [19] is optional and recommended.

1) Data Interface: In response to a "GET" request, an FCML compliant device has to provide the corresponding document, as listed in table I. The contents are defined by the device specific FCML schema. There exist three different return document types as shown in table I.

TABLE I
Query Types

Type	URL	return document
GET	http://host:port/fcml/diag	diagnose data
GET	http://host:port/fcml/master	master data
GET	http://host:port/fcml/config	config data

All data documents share the following frame, while the contents depend on the type of document and on the device specific schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<fcml:data
 xmlns:fcml="http://schema.fcmlgroup.org/fcml/core/1.1">
 ...
</fcml:data>
```

a) Diagnose Data: Generic diagnose data takes the following form:

```
<fcmlDev:diag
 xmlns:fcmlDev="http://schema.fcmlgroup.org/fcml/device/1.0"
 deviceId="Vendor.Location.DeviceType.Id"
 deviceState="available"
 documentDate="2007-09-05T14:30:03 GMT[offset]"
 sourceData="valid"
 sourceURI="http://hostname:port/fcml/diag ">
 <!--
  list of device specific diagnose elements
 -->
</fcmlDev:diag>
```

The element `fcmlDev:diag` is, strictly speaking, part of the device specific FCML extension, but is nonetheless derived from an element from the core specification to ensure a consistent series of properties. The namespace prefix for this element is arbitrary, examples could be `fcmlAsi` for ASi controllers, `fcmlEle` for elevators, or `fcmlEsc` for escalators. The namespace URI follows the pattern `http://schema.fcmlgroup.org/fcml/device/version`, whereas `device` should indicate the device type and `version` should hold the version of the schema which must be used to validate the data. The property `deviceState` indicates whether the device is available. Devices with native FCML support default to "available" with this property. Whenever contents of the document cannot be validated against a given schema, `sourceData` is set to "invalid". This property can also be used for semantic checks on the data. In-depth information regarding any of the

presented schemas, examples, or elements can be found in the specification [15].

The structure of device specific data is defined by the device specific schema. Currently, there exist schemas for devices such as fire dampers, elevators, or escalators. Due to their size, both schemas and examples cannot be shown here, but are available on the web [20]. The following is a short excerpt from a diagnose document regarding an escalator:

```
...
 <general>
  <operatingState>alternation</operatingState>
  <direction>none</direction>
  <availabilityState>available</availabilityState>
  <powerSupply />
  <emergencyStop automaticRestartSystem="true">
     inactive</emergencyStop>
  <miscErrors status="ok" />
 </general>
 ...
 <temperature>
  <temperatureIndicatorList>
   <temperatureIndicator temperatureIndicatorId="top">
    <temperature unit="C">15</temperature>
   </temperatureIndicator>
  </temperatureIndicatorList>
  <temperatureWarningList>
   <temperatureWarning temperatureWarningId="cabinet"/>
  </temperatureWarningList>
 </temperature>
 ...
 <statistics>
  <operatingTime unit="s">432432</operatingTime>
  <failureTime unit="s">2311</failureTime>
  <failures>11</failures>
  <emergencyStops>13</emergencyStops>
  <ridesUp>432</ridesUp>
  <ridesDown>435</ridesDown>
 </statistics>
...
```

While schemas for practically any device type can be integrated into FCML, another option is the use of the generic FCML-IO to define specific (digital or analog) I/O ports, which, as a collection, represent the state of a device. FCML-IO is a generic alternative to a device specific schema, which facilitates access to I/O controllers. The idea is that in the SCADA domain, any device can be sufficiently described by the collection of sensors, actuators, etc. Attached to an I/O controller, these components comprise the technical operating state of the device in the sense that the collective data represent the device in its entirety and commands sent to the controller allow for full operational control. This way, very complex devices or groups of devices (irrespective of type, manufacturer, interfaces, etc.) can be integrated, addressed, and controlled in a very structured and elegant way.

b) Master Data: Master data comprise all static data associated with a device. They are typically only changed when the device itself is changed, i.e. replaced or structurally modified in an essential way. These data are not polled on a regular basis, since they almost never change.

```
<fcmlDev:master
  xmlns:fcmlDev="http://schema.fcmlgroup.org/fcml/dev/1.0"
  deviceId="Vendor.Location.DeviceType.Id"
  deviceState="available"
  documentDate="2005-10-05T14:30:03 GMT[offset]"
  sourceURI="http://hostname:port/fcml/master">
...
 <fcml:deviceId>ESC-VEND-456</fcml:deviceId>
 <fcml:serialNumber>243569854</fcml:serialNumber>
 <fcml:vendorId>VEND-EU</fcml:vendorId>
 <fcml:constructionYear>2002</fcml:constructionYear>
 <fcml:description>an escalator</fcml:description>
 <fcml:location>
  <fcmltypes:description>west wing, north entrance
  </fcmltypes:description>
 </fcml:location>
 <modelNumber>1138</modelNumber>
 <comissioningDate>2003-03-10T14:12:34.0Z
 </comissioningDate>
 <mountingType>Indoor</mountingType>
 <rise unit="m">13</rise>
 <inclinationAngle unit="">15.50</inclinationAngle>
 <alternation>true</alternation>
...
</fcmlDev:master>
```

c) Config Data: Properties relevant for the configuration of a device are contained in the config data document. In contrast to master data, these change (or can be changed) more or less often, sometimes on a regular basis. The thresholds for environmental properties of an escalator fall into this category, for example to prevent freeze-up during winter time. The following excerpt shows some examples.

```
<fcmlDev:config
  xmlns:fcmlDev="http://schema.fcmlgroup.org/fcml/dev/1.0"
  deviceId="Vendor.Location.DeviceType.Id"
  deviceState="available"
  documentDate="2005-10-05T14:30:03 GMT[offset]"
  sourceURI="http://hostname:port/fcml/config">
 ...
 <heating>
  <temperatureTrigger>
   <upperBound unit="C">10</upperBound>
   <lowerBound unit="C">5</lowerBound>
  </temperatureTrigger>
 </heating>
 ...
 <drive>
  <revolutionRate><min>200</min><max>1500</max>
  </revolutionRate>
  <deltaWyeTime unit="s">5</deltaWyeTime>
  <runtimeVF unit="s">300</runtimeVF>
  <passengersDeltaMode>20</passengersDeltaMode>
  <passengersWyeMode>15</passengersWyeMode>
  <brake>
   <lag unit="s">2</lag>
   <stoppingDistance>
    <min unit="m">0.14</min>
    <max unit="m">0.28</max>
   </stoppingDistance>
  </brake>
 </drive>
 ...
</fcmlDev:config>
```

2) Command Interface: Properties of devices, for example those of a micro controller, can be manipulated using this interface. Furthermore, control functions can be accessed, such as initiating a self test or a reset. There exist two different return document types as shown in table II.

TABLE II
COMMAND TYPES

HTTP Type	URL	content	return document
POST	http://hostname:port/fcml/command	FCML command	response
GET	http://hostname:port/fcml/result?commandId=<xxx>	–	result

In detail, the command interface facilitates two different

types of commands:

- device specific commands
- generic commands (arbitrary arguments)
  - fcmlcom:setProperties
  - fcmlcom:executeCommand

Device specific commands require a definition in the device specific command schema, such as `fcmlcomAsi:setDigitalOutputModule` in the ASi specific FCML extension [20]. `commandId` is a unique identifier set by the requesting entity. Due to the processing of commands, such an id is necessary, for example to associate a number of requests with their respective response and result documents. Further information on command signalling is discussed below. In the following example, the operating state of an escalator is set to upwards, intermittent operation.

```
<fcmlcom:commandRequest
 xmlns="http://schema.fcmlgroup.org/fcmlcom/esc/1.0"
 ...
 documentDate="2001-12-17T09:30:47.0Z"
 version="1.1">
 <setOperatingState commandId="3625243">
  <operatingState>intermittentUp</operatingState>
 </setOperatingState>
</fcmlcom:commandRequest>
```

Generic commands facilitate control of devices without the necessity of a device specific command schema. This way, no specific definition at the level of FCML is necessary, which allows for easy integration of devices for which no device specific schema is available. The `setProperties` command is used to set the values of named properties of devices:

```
<fcmlcom:setProperties commandId="67862134">
 <fcmlcom:property name="myProperty">value 1
 </fcmlcom:property>
 <fcmlcom:property name="myOtherProperty">value 2
 </fcmlcom:property>
</fcmlcom:setProperties>
```

Generic commands executed using `executeCommand` work in a similar way. Provided that the device supports a certain command (in the following example named ”commandName”), the command name and a number of arguments (i.e. name/value pairs) are sent.

```
<fcmlcom:executeCommand
 commandName="commandName" commandId="67862135">
 <fcmlcom:argument name="argument name 1">value 1
 </fcmlcom:argument>
 <fcmlcom:argument name="argument name 2">value 2
 </fcmlcom:argument>
</fcmlcom:executeCommand>
```

Each command document can contain a number of commands intended for any single device. The simultaneous transmission of commands (e.g. bulk updates of properties) to more than one target device is not supported. Although a sequence of command documents sent to different devices over a short period of time renders very similar functionality.

a) Command Signalling: Command signalling can be divided into three different phases (see also figure 4):

- request phase (request document)
- response phase (response document)
- result phase (result document)

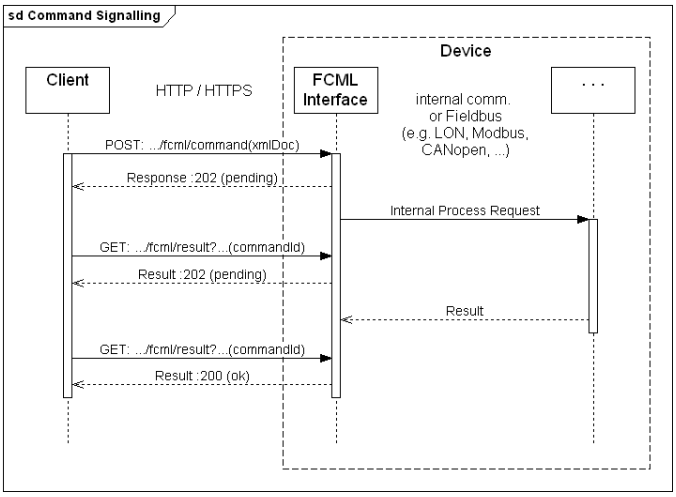

Fig. 4. Command Signalling

In the **request phase**, the command interface receives a command document via HTTP-POST from a requesting entity. Along with the command and the required parameters, a `commandId`, generated by the client, is contained in the document. This `commandId` serves to assign later communication to the correct data.

The contents of the command are processed and a response document is generated in the **response phase**. In case the command requires the device to execute further (time consuming) processing, the response document is marked “pending” (status code 202) to indicate that the final result is not contained in the response document and can be retrieved at a later point in time. Actually, the indication is different, as, by default, any response document not marked “ok” (status code 200) means that the final result must be retrieved later.

When previously requested data becomes available, i.e. processing has completed, the corresponding result document is kept available in the **result phase**. If once again the data is requested, which can be identified with a matching `commandId`, the result document, marked “ok” (status code 200), is sent back. FCML compliant devices must be able to store a configurable number of result documents for later retrieval. Requests for data associated with an illegal or invalid `commandId` must be replied with the status code 404 (“not found”).

C. FCML Schemas

All FCML document types are specified using XML Schema [21]. **Core schemas** define the frame for all FCML documents, such as data documents, command documents, and basic types and units. Extensions to the core schemas for the purpose of supporting device specific functionality can be defined as **device specific schemas**. Corresponding data and command documents provide for FCML compliant mapping of properties.

Unfortunately, an in-depth discussion of schemas is beyond the scope of this article. The same applies to even rudimen-

tary placement of excerpts of individual schemas. Examples, schema files, and other information can be found on the web [20].

III. Conclusion

In this article, we introduced the domain of SCADA systems and the underlying technologies for the purpose of motivating the advantages of an open XML-based standard for the communication between HMI and RTUs. Traditionally, available components stem from proprietary developments which, from the operator's point of view, introduce a number of different problems. These include, but are not limited to, dependency on certain manufacturers, dependency on certain platforms, closed sources, uncertain long term support, financial issues, technical issues.

FCML, as an open standard for communication with RTUs in the SCADA domain, is a key component in a system architecture which not only alleviates many of the problems found in todays SCADA applications, but also introduces the use of up to date web-based technologies. The main advantages include complete independence from specific hard- and software vendors, the use of open standards which are supported by a broad community, and great flexibility in installation, maintenance, and expansion of SCADA installations. FCML can be seen as a specialized web service language for SCADA applications. The need for specialized languages results from both very specific requirements that cannot be sufficiently and accurately fulfilled by generic web service formalisms, and from the intranet (or closed world) deployment context of most, if not all, SCADA applications.

IV. Future Developments and Applications

The field of web-based SCADA systems is one of the youngest fields in the SCADA domain and, therefore, a lot of research efforts are underway. In the near future, these systems will experience an convergence with other technology to further improve service quality and to provide new features. For a holistic realization of the vision of web-based SCADA systems further research is necessary. Reactivity on the web [22]–[25], for example, requires a high-level, rule-based language for expressing complex reactions and workflows, and push- instead of pull-model processing. Additionally, an abstract data model for families of tools, events, errors, etc. is needed.

Video streams captured by CCTV cameras, a fast growing sector, will be integrated into SCADA systems. Semantic annotation and automatic processing will provide important sources of input, as will data from sensor networks, RFID tags, and other technologies. This stream of information will enable operators to make more informed decisions in everyday operation, but, more importantly, in crisis situations and emergency management. Intelligent rule-based systems will be providing a stream of relevant information which can be handled by the authorities. This will be achieved by reducing the flow of information, for example by filtering and prioritizing data, but also by intelligent combination of the semantics, causality, and other correlation of data (possibly from different sources).

References

[1] A. Daneels and W. Salter, "What is SCADA?" in *Proceedings of the 7th International Conference on Accelerator and Large Experimental Physics Control Systems*, 1999.

[2] *Digital Data Communications for Measurement and Control – Fieldbus for use in Industrial Control Systems*, International Electrotechnical Commission (IEC) Std. IEC 61 158, 1999.

[3] U. Jecht, W. Stripf, and P. Wenzel, "PROFIBUS - Open Solutions for the World of Automation," in *The Industrial Information Technology Handbook*, R. Zurawski, Ed. CRC Press, 2005, pp. 1–23.

[4] S. Cavalieri, "Foundation fieldbus: History and features," in *The Industrial Information Technology Handbook*, R. Zurawski, Ed. CRC Press, 2005, pp. 1–16.

[5] D. Loy, "Lonworks/eia-709 networks eia 709 protocol (lon talk)," in *The Industrial Information Technology Handbook*, R. Zurawski, Ed. CRC Press, 2005, pp. 1–6.

[6] J. Lange and F. Iwanitz, "OPC - Openness, Productivity, and Connectivity," in *The Industrial Information Technology Handbook*, R. Zurawski, Ed. CRC Press, 2005, pp. 1–30.

[7] The Microsoft Corp., "COM: Component Object Model Technologies," http://www.microsoft.com/com/default.mspx, accessed September 2007.

[8] D. Thompson and D. Watkins, "Comparisons between CORBA and DCOM: Architectures for Distributed Computing," in *TOOLS (24)*. IEEE Computer Society, 1997, pp. 278–283.

[9] F. Plasil and M. Stal, "An architectural view of distributed objects and components in CORBA, Java RMI and COM/DCOM," *Software - Concepts and Tools*, vol. 19, no. 1, pp. 14–28, 1998.

[10] D. Watkins, "CORBA and DCOM: Architectures for Distributed Computing," in *TOOLS (29)*. IEEE Computer Society, 1999, p. 401.

[11] World Wide Web Consortium (W3C), "The Extensible Markup Language (XML)," http://www.w3.org/XML/, 1996–2003, accessed September 2007.

[12] *OPC Unified Architecture, Draft, Parts 7 and 9*, OPC Foundation Std. OPC UA, March–July 2006.

[13] *OPC Unified Architecture, Release Candidate Specification, Parts 6 and 10–11*, OPC Foundation Std. OPC UA, June–November 2006.

[14] *OPC Unified Architecture, Specification, Parts 1–5 and 8*, OPC Foundation Std. OPC UA, July–September 2006.

[15] FCML-Group, "The Facility Control Markup Language (FCML) Specification Version 1.1," http://www.fcml-group.org/fileadmin/Downloads/FCML/Anlagen/Core/Version_1.1/FCML-v1.1.pdf, 2006–2007, accessed September 2007.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Jun. 1999, updated by RFC 2817 [18]. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[17] World Wide Web Consortium (W3C), "The Hypertext Transfer Protocol (HTTP)," http://www.w3.org/Protocols/, 1996–2003, accessed September 2007.

[18] R. Khare and S. Lawrence, "Upgrading to TLS Within HTTP/1.1," RFC 2817 (Proposed Standard), May 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2817.txt

[19] E. Rescorla, "HTTP Over TLS," RFC 2818 (Informational), May 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2818.txt

[20] FCML-Group, "FCML-Group Website, Download Section: Specifications, Schemas, Examples," http://www.fcml-group.org/index.php?id=6, 2006–2007, accessed September 2007.

[21] World Wide Web Consortium (W3C) – The XML Schema Working Group, "XML Schema," http://www.w3.org/XML/Schema, 2001–2007, accessed September 2007.

[22] B. Berstel, P. Bonnard, F. Bry, M. Eckert, and P.-L. Pătrânjan, "Reactive rules on the web," in *Reasoning Web, Int. Summer School*, ser. LNCS. Springer, 2007.

[23] F. Bry, M. Eckert, and P.-L. Pătrânjan, "Reactivity on the Web: Paradigms and applications of the language XChange," *J. of Web Engineering*, vol. 5, no. 1, pp. 3–24, 2006.

[24] F. Bry and M. Eckert, "Twelve theses on reactive rules for the Web," in *Proc. Int. Workshop Reactivity on the Web*. Springer, 2006.

[25] ——, "Rule-based composite event queries: The language xchange^{eq} and its semantics," in *Proc. Int. Conf. on Web Reasoning and Rule Systems*. Springer, 2007.

[26] R. Zurawski, Ed., *The Industrial Information Technology Handbook*. CRC Press, 2005.