Self-describing devices in COSMIC

Jörg Kaiser
Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2
39106 Magdeburg
kaiser@ivs.cs.uni-magdeburg.de

Hubert Piontek
Universität Ulm
Albert Einstein Allee 11
89069 Ulm
piontek@informatik.uni-ulm.de

Abstract

This paper describes an approach on the way towards dynamically configuring an application by discovering and using smart devices in a sensor/actuator network. The key to this are self-describing smart devices. Self description means that all information necessary to use the respective device are stored within the device itself. As an appropriate description language XML has emerged during the last years. XML provides considerable flexibility and ease of post-processing using XSLT stylesheets. However, because of stringent space constraints, there is the need to have a more compact representation. We describe what information is stored, what it can be used for, and how we make this information accessible during run-time.

1. Introduction

In an environment where sensor/actuator networks are dynamically configured, it is desirable to query which devices with what functions are available. Consider a mobile robot moving through a smart space of a factory or some office environment instrumented with intelligent sensors. The robot is able to communicate physically with these components and may wish to know what kind of environment information it can obtain from the sensor infrastructure and automatically adapt to this. Reflective mechanisms in the software aim at exploiting such information. Another, more short-term example is a robot, a car or another complex machine which has a rich spectrum of sensors and intelligent components. During maintenance or when a new component or a new function is added, it may be desirable for a programmer or service staff to know the actual configuration state. We think that for truly autonomous devices and dynamic environments, the information about the components must be stored in the components themselves and it must be possible to dynamically retrieve this information (a view which is shared also by [3-5]). There are three categories of information which are of interest: 1. general information about the component like what it is, its name, its manufacturer and its hardware and software versions. 2. information about the physical connections and the temporal properties of the supported communication channels and 3. information about the semantics of data which is provided by the component. This is probably the most difficult part because sensors may provide data about the quality of air in particles/$m^{3,}$ acceleration, proximity, radiation or just temperature. Thus we identified a number of problems, as:

- How to obtain an appropriate level of detail for a useful description.
- How to store this information in a rather constraint memory of a smart device.
- How to make this information available for dynamic discovery.
- How to allow fine grained queries.

The paper presents this ongoing activity in the context of our COSMIC middleware [8] for intelligent devices. Beside the standard information for the devices, the respective COSMIC communication objects (events) and communication channels (event channels) have to be described in their temporal and functional aspects. Therefore, we briefly introduce COSMIC before we show the details of the device description and the mechanisms behind the fine grained search and discovery service.

2. COSMIC

COSMIC (COoperating SMart devICes) is a middleware that provides communication facilities for autonomous smart sensors and actuators [8]. The communication model is based on a publish/subscribe scheme. Particularly, COSMIC integrates multiple networks under a common communication model and considers quality of service requirements. COSMIC provides an application interface which allows expressing temporal and reliability attributes on a high, application related level. In our robot demo scenarios, smart devices are connected using a CAN bus (figure 1). A CAN bus is interconnected via a gateway to wireless LAN. The gateway is realized on a component which has more processing power and memory than the smart sensor and actuator devices. Therefore, the gateway is also the place where the information about the devices on the CAN-Bus is collected and prepared for external discovery requests.

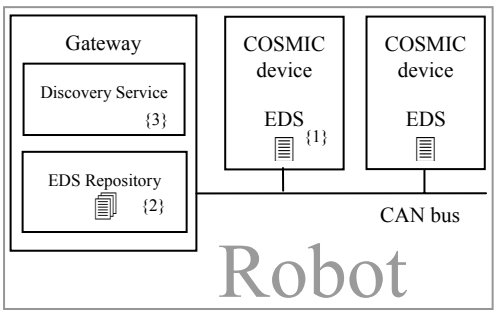

Figure 1. COSMIC architecture

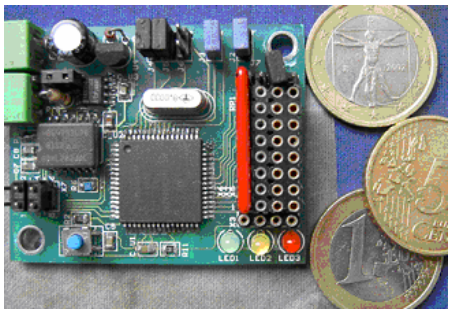

Figure 2. Smart device: acceleration sensor

3. Electronic Data Sheets in COSMIC

In COSMIC, we use XML to describe our components in an Electronic Data Sheet (EDS). The description contains common (and static) information like manufacturer, UID and version of the component. The next part specifies what events the component consumes and/or produces. For each event, we specify at least its name and UID, its temporal attributes (expiration time), and its structure. The structure is broken down into data fields containing a single value. Naturally, we specify the type of this value (e. g. integer or floating point), as well as its physical meaning (e. g. kilograms). COSMIC always uses big endian encoding on the wire, so there is no need to specify the encoding of multi-byte values in the EDS. The EDS is completely stored within each COSMIC component. As COSMIC mainly deals with very small systems, storing and communicating plain XML would consume too many resources. Therefore, we store the data sheet in WAP binary format – the full data sheet of the example in the following section consumes about 2k of non-volatile memory. Upon device configuration, the data sheets ({1} in figure 1) are uploaded to the gateway node. Clearly, the data sheet does not fit into a single CAN message, which can hold up to 64 bits of payload. The data sheets are uploaded using our fragmentation protocol [9] for the CAN bus that enables us to transmit large amounts of data over the CAN bus. On the gateway node, all EDS are converted back to XML and collected ({2} in figure 1). The gateway node runs a device discovery service ({3} in figure 1) that we describe in section 5.

4. Example Data Sheet: Acceleration Sensor

The following data sheet has been shortened for readability. It describes our acceleration sensor (see figure 2) that features a 2 axis acceleration sensor. Additional sensors can be stacked to the base unit which runs our COSMIC middleware.

```
<DeviceInformation
xsi:noNamespaceSchemaLocation="sensor.xsd">
  <static_device_info>
    <!-- General Device information tags -->
    <NodeUID>0x1234567812345678LL</NodeUID>
    <operational_connection>CAN 2.0b
        </operational_connection>
    <event>
      <EventName>Acceleration 2-axis
          </EventName>
      <Subject>acceleration</Subject>
      <SubjectUID>0x000000000000000BLL
          </SubjectUID>
      <data_structure>
        <Name>acceleration x-axis</Name>
        <DataType>unsignedByte</DataType>
        <pe>
          <SIUnit><!-- m/s^2 -->
            <Enumeration>128</Enumeration>
            <Radians>128</Radians>
            <Steradians>128</Steradians>
            <Meters>130</Meters>
            <Kilograms>128</Kilograms>
            <Seconds>124</Seconds>
            <Amperes>128</Amperes>
            <Kelvins>128</Kelvins>
            <Moles>128</Moles>
            <Candelas>128</Candelas>
          </SIUnit>
          <Magnitude>-3</Magnitude>
        </pe>
      </data_structure>
    </event>
  </static_device_info>
  <dynamic_device_info>
    <event_to_channel>
      <SubjectUID>0x000000000000000BLL
          </SubjectUID>
      <eChannelType>HRT</eChannelType>
      <Direction>outgoing</Direction>
    </event_to_channel>
  </dynamic_device_info>
</DeviceInformation>
```

The data sheet is structured into three parts: general information, event definitions, and event-to-channel mapping.

4.1. Part 1: general information

The first part contains information like the device's name, its manufacturer, and similar entries that are mainly useful for documentation purposes. The <NodeUID> is the unique identifier of the component. This unique ID is vital to the communication layer on the CAN bus.

4.2. Part 2: event definition

The <event> tags specify what kind of events the system produces and/or consumes. Each entry has a unique <SubjectUID> that uniquely identifies the event's subject. Each event is mapped to one message on the communication network. The structure of the payload of this message is specified within the <event> tag. In the above example, the acceleration event consists of two data fields – one for each axis of the sensor. Each field contains an unsignedByte and its unit is m/s². The encoding is described in detail in [10], and is similar to the encoding in IEEE 1451.

4.3. Part 3: event to channel mapping

Finally, <event_to_channel> tags map events to a channel class (hard, soft or non real-time), and specify whether these events are consumed or produced.

5. Device Discovery Service

The device discovery service runs on the gateway node and can be accessed using SOAP [11] in an RPC-like fashion. SOAP is a widely used standard that is also based on XML and therefore neatly integrates into our scheme. Queries to the device discovery service are formulated as XSLT stylesheets [12] which are sent to the device discovery service within the SOAP body. The device discovery service uses these stylesheets to call an external XSLT processor, which processes the collected electronic data sheets. This technique enables a client to specify arbitrarily complex queries. Queries can be used to e. g. search for certain types of components in the system, or to generate a list of all available components.

5.1. Discovering the gateway node

For simplicity, we decided to use a simple UDP broadcast mechanism to search for the gateway in the local network. Each gateway answers this request with its name, its IP address, and its supported transport protocols (e. g. HTTP). Queries to the device discovery service can then be transmitted using one of the supported transport protocols.

5.2. Example: search query

The following XSLT stylesheet can be used to search a COSMIC EDS for the value of a specific tag. The tag to be searched for is given as a parameter (param1) to the XSLT processor.

```
<xsl:stylesheet xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <xsl:param name="param1"
    select="'default value'"/>
  <xsl:template match="DeviceInformation">
   <xsl:for-each select="static_device_info/*">
      <xsl:if test="name()=$param1">
        <xsl:value-of select="$param1"/>:
        <xsl:value-of select="text()"/>
      </xsl:if>
   </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

This stylesheet starts with a standardized header. <xsl:param> declares a parameter that can be set from the outside. XSLT stylesheets can be thought of as a combination of pattern matching and programming. The XSLT processor heavily uses templates that are matched to the XML data to be processed. <xsl:template match ="DeviceInformation"> tells the XSLT processor to apply this template on all <DeviceInformation> tags in the XML input – in our case this is the EDS. <xsl:for-each> loops through all child tags of any <static_device_info> tag within the current <DeviceInformation> tag. Finally, if (<xsl:if>) the name of the current tag matches the value of param1, the name of the tag and its contents are output. Specifying the parameter param1 as *DeviceName* and processing the above example data sheet with this stylesheet will produce:

```
DeviceName:
Acceleration Sensor
```

5.3. More transformations

To store the data sheets within our devices, we need its WAP binary representation. To generate this, we use the same mechanism as for search queries, i. e. we use a XSLT stylesheet to generate the WAP binary file. Currently we expand the list of actual and potential uses. In section 7 we give a short list of current and near-future activities.

6. Existing standards

Dynamic configuration and spontaneous usage of smart components firstly needs awareness that a devices is available and secondly, a description of the information and services provided. Awareness is an issue which is handled by discovery and dynamic registration. An adequate description specifies the format and contents of the delivered data. There are a couple of approaches and standards which already address these issues. We briefly review some popular representatives and indicate where our scheme adds new properties.

6.1. UPnP

UPnP [6] specifies a discovery service that must be implemented by all devices. Whenever a device is connected to a network, it announces its availability to the network using UDP multicast. It must also listen on the multicast port for any search request, and answer them. Device descriptions are provided as both XML documents, and as HTML documents in parallel. For our purposes, the discovery mechanism is too coarse grained. Beyond discovery services and device descriptions, UPnP specifies device control.

6.2. Jini

The Jini [7] middleware framework provides an infrastructure for providing services in a network. To discover services, clients access a lookup service. The lookup service can either be contacted by knowing its IP address or by searching for it using a multicast. Queries to the lookup service are in terms of Java interfaces. Services also search for a lookup service on startup. Once found, they register a proxy object. Such a proxy object is returned to the queries of clients. The use of Jini implies the use of Java on all participants.

6.3. CANopen

CANopen [1-2] is a CAN-based higher layer protocol that is widely used in factory automation. For certain classes of devices, device profiles are specified. These device profiles define the device's functionality and are similar to our electronic data sheets. CANopen provides flexible means to specify data encoding, e. g. integer values can be specified using arbitrary bit lengths. Bit and byte ordering on the bus are specified. Temporal properties are taken into account. CANopen is a collection of standards that not only cope with application layer communication, but also with unit representation, and low-level configuration details, such as bit-timings. Electronic Data Sheets for CANopen devices are plain text files. They are partitioned into sections, which contain a flat list of attribute/value pairs. Moving to XML is planned for the future. CANopen currently does not offer any discovery services.

6.4. IEEE 1451.x

IEEE 1451 [3-5] aims towards standardizing "smart transducers". The standards describe a network independent model for smart transducers, and their interface specification. They include the specification of a Transducer Electronic Data Sheet (TEDS). These data sheets are partitioned into five parts. The layout is rather strict and inflexible. It describes a static structure of information that must be present, and provides the possibility to extend the standard entries. One of the main goals is to have self-describing plug and play components on the network. Unlike CANopen and our COSMIC, IEEE 1451 breaks down components attached to the network into smaller parts that interact via standardized hard- and software interfaces. IEEE 1451.3 defines multi-drop smart sensor systems which connect several transducer bus interface modules (TBIMs) to a single network capable application processor. It is important to note that in COSMIC, the granularity is a smart device. We would equip all TBIMs with an NCAP. Yet, a hierarchical structure is possible in COSMIC via the WAN-of-CAN structure. The IEEE 1451 standards do not concern themselves with real-time guarantees. Time correlation is mentioned in [5], but not elaborated on.

7. Future Work

We will further exploit our EDS to generate human readable user documentation, e. g. in HTML or PDF (using XSLT Formatting Objects) format. To achieve a more stringent development process, we are working on generating application code skeletons from the data sheets. This helps the application developer to keep the code consistent with the specification. We also want to thank Holger Mönnich for his implementation work.

References

[1] CAN in Automation e. V. *CANOpen Application Layer and Communication Profile*. CAN in Automation, Erlangen, Germany, 2002.

[2] CAN in Automation e. V. *CANOpen Device Profile for Generic I/O Modules*. CAN in Automation, Erlangen, Germany, 2002.

[3] IEEE P1451 Draft Standard for a Smart Transducer Interface for Sensors and Actuators, http://ieee1451.nist.gov/

[4] K. Lee, A Synopsis of the IEEE P1451 – Standards for Smart Transducer Communication. National Institute of Standards and Technology, Maryland.

[5] K. Kee. IEEE 1451. A Standard in Support of Smart Transducer Networking. In *IEEE Instrumentation and Measurement Technology Conference*, Baltimore, 2000.

[6] Microsoft Corporation *Understanding Universal Plug and Play*. Microsoft Corporation, Redmond, USA, 2000.

[7] J. Waldo. *The Jini Specifications*. 2nd edition. Addison Wesley, 2000.

[8] J. Kaiser, C. Mitidieri, C. Bruna, C.E. Pereira. *COSMIC: A middleware for event-based interaction on CAN*. ETFA, Emerging Technologies and Factory Automation, Lisbon, Portugal, September 2003.

[9] A. Bernauer. A fragmentation protocol for the CAN bus. Lab Report, University of Ulm, 2004.

[10] H. Mönnich. *Kompakte Repräsentation der Eigenschaften von intelligenten Hard- & Softwarekomponenten und die Realisierung eines geeigneten Discovery Service*. Diplomarbeit, Universität Ulm, August 2004.

[11] N. Mitra (editor) *SOAP Version 1.2 Part 0: Primer*. http://www.w3.org/TR/soap12-part0/

[12] D. Tidwell. *XSLT. Mastering XML Transformations*. O'Reilly & Associates, 2001. ISBN 0-596-00053-7