Exploiting Self-Descriptions for Checking Interoperations between Embedded Components

Joerg Kaiser, Sebastian Zug, Michael Schulze,
Otto-von-Guericke-University Magdeburg, Germany
Hubert Piontek, University of Ulm, Germany
kaiser@ivs.cs.uni-magdeburg.de

1 Introduction

Today's embedded systems like cars, robots and industrial plants are more and more built from a large number of independent networked devices provided by third party suppliers and are configured to a complete system after the software of the individual components has been designed and implemented. The components are assumed to have a their own computational core, some periphery like sensors or actuators, and a network interface. They cooperate with other components via messages. To an extreme, a complete vehicle as a car or a robot can be viewed as a component communicating and co-operating with other mobile systems or an instrumented smart environment sharing information and services. There are multiple dimensions and system levels of interoperability like syntactic, semantic and temporal, which have to be considered when performing the final configuration towards an operational system or even support dynamic interactions.

Failures frequently occur as a result of system integration, configuring a system from independent hardware devices or software components. It is usually assumed that in most cases they are detected during integration testing and then removed from the system. However, this firstly may be a tedious task and some faults will remain in the system causing disastrous system failures. Spectacular, well-known examples are the loss of Ariane-5 [Lio1996], partly because of interpreting diagnostic information as inertial data and the Metric-English units' mismatch destroying the Mars Climate Orbiter [Nas1999]. Secondly, a dynamic system like a team of robots performing a joint action cannot be checked by integration testing at all because interaction occurs dynamically at run-time. One key issue for these systems is a precise description of the devices that covers the above mentioned interoperability dimensions. These descriptions should be kept favourably with the devices itself or be at least electronically accessible when needed to evaluate compatibility between components. The benefits range from discovering the abilities of components e.g. in a team robot application to checking compatibility of devices that are evaluated when a relation between them is established. The paper mainly addresses the aspects of checking compatibility of interactions using such self-descriptions. This may be at configuration time or even at run-time. Detecting a mismatch is a contribution to preventing serious failures resulting from incompatible data structures, misinterpretation of data or wrong temporal assumptions.

The rest of the paper is structured as follows: we first give a brief overview of the system model realized by the COSMIC middleware. Then we introduce CODES, a scheme for describing embedded devices. Chapter 4 sketches related work in this area and finally, we provide a summary and an outlook on future work.

2 System model

Compatibility checking is intimately related to the system model and the model of communication. Our model is based on two basic concepts, autonomous objects[1] that are active entities performing all computations and events that constitute typed communication objects disseminating data. The notion of autonomous objects is inspired by smart sensor and actuator components. This can be hardware components incorporating an individual computational component, some periphery and a network interface or also concurrently executed software components. From a conceptual point of view they

[1] Autonomous Objects in COSMIC are referred as Sentient Objects in other publications [VCC2002], [CKV2007] because these objects also can adapt their behaviour according to the operational context. Because this sentience is not in the scope of the paper, we just emphasize the autonomy aspect.

can be related to the concept of actors [Agh1986], however with a specific semantic and interaction model. Autonomous objects consume and produce events that are the only visible entities that are exchanged between objects and that are occurring at the object interface. No further access to internal state variables is supported. The term "event" is used to denote a typed communication object and expresses the publish/subscribe-based model of communication where nodes are notified when data is available. Events do not refer to any synchrony model as e.g. in real-time systems where they are associated with asynchrony nor to a specific low level computational model as e.g. in TinyOS [HSW2000]. Events are disseminated through event channels. Event channels are typed unidirectional communication channels that transport events of the respective type and provide latency properties and guarantees. Autonomous Objects, events and event channels are supported by the COSMIC middleware, which is particularly designed for tiny embedded systems. A detailed description of the COSMIC middleware is beyond the scope of this paper and is available in [KBM2005]. The important property of the publish/subscribe is that the communication is based on the content of a message rather than on an address. This allows the dynamic use of information because a component has to specify what information it needs to perform a computation rather than of who provides this information. Because any interaction between components is performed via events we describe the interface of an autonomous object in terms of these events. Events have: 1.) A subject identifying the event, 2.) attributes related to the context in which the event was generated like a location and a time, 3.) quality attributes describing temporal validity, periodicity and optional constraints on the physical parameters like ranges and precision and 4.) a contents carrying the respective payload. Because of the application area in mind, we precisely define the kind and dimension of the physical units of the contents, which goes beyond the type checking usually available when invoking an object. The event channel description comprises the synchrony attributes for the dissemination. Parameters to specify a channel include, the subject that is disseminated, the synchrony class, information about the period of dissemination, and whether it is an outgoing or incoming event channel. The parameters of the event and the event channel are contained in the device description are the main information used for compatibility checking.

3 Describing components with CODES

Device descriptions for embedded devices are well known in industrial automation and part of many field-bus standards. However, in many cases they are only available as text documents (see chapter 4). If they are used for on-line compatibility checking they have to be in a form that can be processed automatically. CODES [KaP2006] uses XML to provide this possibility because the wide spectrum of tools that are available to process XML documents. We call a device description an "Electronic Data Sheet (EDS)" following the terms of the IEEE 1451.2 [Lee2000] standard. The EDS contains many parts describing general information about a device like supplier, software and hardware versions and so on. We will concentrate on the event and event channel descriptions here because they are the most important elements for compatibility between devices. Fig. 1a gives an example for an event description. The most important fields for compatibility checking are:

- The name of the event, which should have a meaning to the consumer. There is a textual representation of the subject "oil_temperature" which is mapped to a UID that during operation is used to identify an event. It should be noted that this binding to a name always allows to unambiguously identify an event from a structural and a semantic aspect throughout its lifetime.
- The data type of the contents, which in the case of Fig. 1a is a 16 Bit unsigned integer. For data types involving multiple bytes, coding information as the byte order is specified additionally.
- The physical dimension of the contents, that is represented in the SI (Système international d'unitès, SI) notation [TyT2008] and is most important if this event should be used dynamically by other devices. In the example the temperature will be disseminated in degrees Celsius. Because the SI system specifies temperature in Kelvin, an additional offset has to be added.

As mentioned above, the event description also comprises a number of attributes. An example of a time stamp is given in Fig. 1b. As the contents itself, the physical attributes are specified according to the SI system. Attributes may be related to the context in which an event is generated or to non-functional and temporal issues like whether an event is sporadic or periodic and what is the period.

Additionally, a validity interval is specified for real-time events after which the event expires and the system notifies an application about an unsuccessful transmission.

```
<Event>
  <Subject> oil_temperature</Subject>
  <SubjectUID>0x1234567890ABCDEF</SubjectUID>
    <Description>Samples oil temperature, output in
      degrees celsius. </Description>
    <DataStructure>
        <Field>
            <Name>oiltemp</Name>
            <Description>oil temperature </Description>
            <DataType>u_int_16</DataType>
            <ByteOrder>BigEndian</ByteOrder>
            <Dimension>
            <SIUnit><Kelvins>1.0</Kelvins></SIUnit>
                <Magnitude>0.0</Magnitude>
                <Offset>273.15</Offset>
                <Scaling>1.0</Scaling>

            </Dimension>
            <Attributes> ... </Attributes>
        </Field>
    </DataStructure>
    ...
</Event>
```

```
<Attributes>
        .....
        .....
        <Field>
            <Name>time</Name>
            <Description>absolute time in NTP
              format</Description>
            <DataType>u_int_32 </DataType>
            <ByteOrder>BigEndian</ByteOrder>
            <Dimension>
                <SIUnit><seconds>1.0
                          </seconds></SIUnit>
                <Scaling>1</Scaling>
            </Dimension>
        </Field>
        ....
        ....
</Attributes>
```

Fig. 1a: CODES event specification

Fig. 1b: CODES event attribute specification

In the same way as events, the properties of event channels are described. The main information is the subject and the synchrony class of the channel. The entire description of a device in an XML document sums up to a typical size of up to 50kbyte for sensor or actuator devices. This EDS for a device is used during multiple stages of system life to check compatibility of interaction and detect possible faults:

- During development of components, XML schemata are used to check the completeness of a specification. Furthermore, as a means to prevent implementation errors, a tool for generating code from the descriptions has been realized. At the moment, this mainly ensures that the correct data structures are defined on the consumer side and appropriate conversions be made.
- During the configuration phase, the descriptions of events are evaluated and it has to be ensured that the incoming event specification of a subscriber matches the respective outgoing specification of a publisher. In a system built from a large number of devices, it is often hard to determine the properties of all individual components that may be dependent on manufacturer, version and other characteristics. This is particularly true when devices have been added or replaced after the initial configuration. If the description is stored with the device itself, the actual building blocks can always be identified easily.
- During run-time, every new subscription can be checked. This is exploited for dynamic interactions e.g. in applications where robots dynamically interact with each other or with an instrumented smart environment. In these scenarios a vehicle may want to use environment events from a remote sensor. The descriptions allow discovering and using events dynamically and checking the structural and content properties. This requires the respective computational resources to evaluate the EDS on-line and needs additional time.

Because of the memory and performance restrictions in small devices, it is not possible to store the EDS in plain text format or evaluate the XML-descriptions with a smart device. We therefore realised a solution in which storage and the evaluation are separated. The descriptions are stored in a compressed form. E.g. for the 38,6 kbyte description of an acceleration sensor, the size of the gzip representation is 2.8 kbyte. This is an acceptable size to store in the flash memory even of a tiny device. On start-up or during maintenance these descriptions are uploaded to a dedicated more powerful node in the system. This also seems to be a reasonable assumption because even if the reactive level of a car or a robot can be seen as a network of tiny nodes, there are some more powerful nodes responsible for higher-level functions. These nodes collect and process the descriptions perform the checks whenever a node newly subscribes to some event provided in the system.

4 Related Work

Much effort already has been expended on describing properties and services in systems configured from multiple components. This ranges from the field of service-oriented systems like WSDL [CCM2001], Jini [Wal2000] and UPnP [WeJ2003]. They exploit descriptions for discovering services dynamically. The reason why these systems are not fully appropriate for our purposes is their complexity and their lack of describing physical units and the quality of dissemination. Addressing this issue, device profiles have been introduced in industrial automation long ago. Among them are CANopen [CiA1996-2005] and IEEE 1451 [Lee2000], as well as TTP/A [KHE2000], [EHK2001] and LIN [ABD2003] in the automotive area. However, most of the device profiles in industrial automation are expressed in a very special and proprietary way. Secondly, they reflect the specific properties of the underlying network and therefore, these standards do not support interactions between different communication networks. Finally, in most cases, the descriptions are not available in a form, which automatically can be used and manipulated by a machine. Table 1 summarizes the properties of standards related to this work.

	CANopen	IEEE 1451	LIN	TTP/A	CODES
Descr.	text	binary	text	XML	XML
Config.Mngmt.	vendor spec.	vendor spec.	yes	yes	yes
Comp. check.	no	no	no	no	yes
Real-time	limited	limited	limited	HRT	yes
Resrc. Usage	low	very low	very low	very low	low

Table 1: Properties of device descriptions

The work closest to CODES is the IEEE 1451.2 standard and the smart transducer interface of the TTP/A system. The EDS of the 1451 standard inspired particularly our way to represent physical units and general information. The main differences to IEEE 1451.2 are the use of XML to describe the devices, the representation of temporal aspects and our integration with the COSMIC middleware. COSMIC provides the possibility to run across multiple heterogeneous networks and enables cross-network communication. TTP/A is focussed on minimal resource consumption and keeps the references to the descriptions in the devices only. Because it is based on a static Time Triggered Architecture there is no need for any dynamic check. It is possible to access a smart device from a CORBA system via the smart transducer interface [OMG2003] where these descriptions may be exploited.

5 Conclusion

The intention of CODES device description is to enable the correct interoperation of autonomous objects in a heterogeneous networked system. In a scenario, in which we need to integrate heterogeneous devices from multiple sources with no complete knowledge of the internal code, precise descriptions are inevitable to ensure compatibility. These descriptions have to be an inherent part of the device and inseparably related to it. The descriptions must have a sufficient level of detail and completeness that allows compatibility checking. CODES addresses these needs in the design, integration and operation of the system by a suite of tools and services. During device specification, the CODESCreator assists the correct creation of a description by respective templates and a graphical user interface. The generated electronic data sheet (EDS) is validated against the respective XML schema. In this way, the Creator ensures that all aspects of the device description are covered. A further problem is that a correct specification not always results in correct code. This aspect is treated by CODES during the implementation phase. Part of the code to use a certain event of a component is generated automatically. Here we exploit XML style sheets and an XSLT processor for the respective transformations. It ensures the compatibility of data types and data structures by omitting coding errors resulting from low-level program details. While these tools enforce compatibility between producing and consuming components during the design phase, the COSMICMonitor and the LogPlayer further exploit the generated EDSs when testing and maintaining the system. The Monitor allows observing specific events. Rather than inferring information and dissemination details from the application code, the observable features, like specific events, periods and other quality properties can be derived from the descriptions of events and event channels. Finally, CODES provides a service to discover devices and events in a dynamically. An XML style sheet is used to express a request and according to this, the XLST processor extracts the respective information from the EDS. This allows a very fine-grained search for sensor or actuator services exploiting the entire expressivity of the device description. A complete and detailed description of CODES is available in [Pio2007].

6 References

[Agh1986] Gul Agha: "Actors: A model of Concurrent Computation", MIT Press, 1986

[ABD2003] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc., Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN specification v2.0, 2003.

[CCM2001] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, March 2001.

[CiA1996] CiA Draft Standard 201–207 Version 1.1. CAN Application Layer for industrial applications, February 1996.

[CiA2002] CiA Draft Standard 301. CANopen Application Layer and Communication Profile, February 2002.

[CiA2004] CiA Draft Recommendation 303–2. CANopen Representation of SI units and prefixes, December 2004.

[CiA2005] CiA Draft Standard 306. Electronic data sheet specification for CANopen, January 2005.

[CKV2007] António Casimiro, Jörg Kaiser, Paulo Verissimo, "Generic-Events Architecture: Integrating Real-World Aspects in Event-Based Systems", Lecture Notes in Computer Science (Architecting Dependable Systems IV), Volume 4615/2007, Springer 2007, pp. 287-315

[EHK2001] W. Elmenreich, W. Haidinger, and H. Kopetz. "Interface Design for Smart Transducers", In IEEE Instrumentation and Measurement Technology Conference, volume 3, pages 1642–1647, Budapest, Hungary, May 2001.

[HSW2000] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, "System Architecture Directions for Networked Sensors", ACM SIGPLAN Notices, Volume 35 , Issue 11, November 2000

[KBM2005] J. Kaiser, C. Brudna, C. Mitidieri, "COSMIC: A real-time event-based middleware for the CAN-bus", Journal of Systems and Software, Volume 77, Issue 1 (July 2005), Special issue: Parallel and distributed real-time systems, pp. 27 - 36

[KaP2006] J. Kaiser and H. Piontek, "Codes: Supporting the development process in a publish/subscribe system", in Fourth Workshop on Intelligent Solutions in Embedded Systems (WISES 2006), Vienna, Austria, June 2006.

[KHE2000] H. Kopetz , M. Holzmann, W. Elmenreich, "A Universal Smart Transducer Interface: TTP/A", Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, St. Malo France, 2000

[Lee2000] K. Lee, "IEEE 1451: A standard in support of smart transducer networking", Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference, 2000. IMTC 2000.

[Lio1996] J. L. LIONS (Chairman of the board), "ARIANE-5, Flight 501 Failure" Report by the Inquiry Board, July 1996

[Nas1999] Mishap Investigation Board "Mars Climate Orbiter", Phase I Report, November 10, 1999

[OMG2003] Object Management Group, "Smart Transducers Interface Specification", Version 1.0, January 2003, available from: http://www.omg.org/cgi-bin/doc?formal/03-01-01

[Pio2007] H. Piontek, "Self–description mechanisms for embedded components in cooperative systems", PhD Thesis, University of Ulm, 2007

[TyT2008] B. N. Taylor, A. Thompson (Eds.), "The International System of Units (SI)", NIST Special Publication 330, 2008 Edition, National Institute of Standards and Technology Gaithersburg, MD 20899, March 2008

[VCC2002] P.Verissimo, V.Cahil, A.Casimiro, K.Cheverst, A.Friday and J.Kaiser, "Cortex: Towards supporting autonomous and cooperating sentient entities", In Proceedings of European Wireless 2002, Florence, Italy, Feb 2002

[Wal2000] J.Waldo, "The Jini Specifications", 2nd edition. AddisonWesley, 2000

[WeJ2003] J. Weast M. Jeronimo, "UPnP Design by Example", Intel Press, 2003.