Plug-and-Play: Bridging the Semantic Gap Between Application and Transducers

Stefan Pitzek and Wilfried Elmenreich
Vienna University of Technology
Institute of Computer Engineering
Treitlstraße 1-3/182-1, Vienna, Austria
{pitzek,wil}@vmars.tuwien.ac.at

Research Report 64/2005

Abstract – *Plug-and-play is an important mechanism for achieving component integration and improving interoperability in smart transducer systems. While recent research in smart transducer interfaces has achieved a syntactically well-defined interface to arbitrary sensors and actuators, current plug-and-play approaches are limited by the semantic gap between the control application and the generic smart transducers.*

In this paper we propose a solution to this problem by establishing a smart interface system that consumes the sensor data from the smart transducers and provides a syntactically and semantically standardized interface to the control application. The smart interface system will be configured using data from the transducer meta-description, a system meta-description and the requirements from generic sensor data processing.

As case study for such an interface system we present a generic high-level application component in form of the generic certainty grid, a sensor fusion algorithm for obstacle detection, and an approach for automatically configuring this application component and its associated smart transducer nodes in a plug-and-play like manner.

1 Introduction

Plug-and-play describes a feature for the automatic integration of a newly connected device into a system without user intervention. While this feature has been in use for some time in personal computers within an office environment, it is quite difficult to achieve fully automatic configuration behavior for automation systems, since, without user intervention, there is usually not enough information, what to do with the data from a new sensor or how an actuator should be instrumented.

Therefore, in the automation domain the term *plug-and-participate* better describes these abilities of recognizing and integrating new transducers, i. e., sensors and actuators. For example, after connecting a new sensor to a network, it can be automatically detected, given a local name and assigned to a communication slot; but a human system integrator is still necessary to decide on the further processing and usage of the sensor data.

Plug-and-participate thus only covers the basic-level issues of connecting and recognizing a new smart transducer, and is usually realized by providing some self-identification information on the component to be integrated and by using commonly shared templates, profiles, data types and message formats.

Trying to realize true plug-and-play exceeds the above described basic services and leads to a series of new problems, such as connecting different sensors to the correct inputs of an application component and a dependence of some configuration-relevant properties on the target system.

Consider for example, a steer-by-wire application that uses two identical actuators for electromechanical steering (one for each front tire). While plug-and-participate allows recognizing and addressing these nodes, this is not sufficient for successfully setting up the application, because each actuator requires at least information on its location, i. e., either at the right or left front tire.

Our approach introduces an interface system between a high-level application component and the transducers. This interface system abstracts from the connected sensors and exports the sensor information

in a generic format, which is independent of the actual sensor configuration.

In this paper we address this problem by means of a common conceptual domain model that allows the integration of information from the intended application, the smart transducers, and the target system, so that all required parts can be configured with minimal human intervention.

We demonstrate this approach with an application that creates a map of the environment of a mobile robot. The map is implemented by a so-called two-dimensional certainty grid, which maps the environment to a set of grid cells. Each grid cell is assigned a probability that the area corresponding to the grid cell is covered by an object. The certainty grid can be updated by different sensors, like infrared, ultrasonic, LIDAR, etc. Each of these sensors can be connected and integrated into the application in a truly plug-and-play manner, that is, without user intervention.

The remaining part of this paper is structured as follows:

Section 2 presents related work on plug-and-play, common data models, and generic applications in the robotics domain. Section 3 describes the proposed system architecture. Section 4 presents a case study application for examining the configuration aspects in detail. In Section 5 we discuss the proposed approach and conclude the paper.

2 Related work

Several existing approaches from the automation and fieldbus domain deal with interoperability by defining shared models of the system.

The OPC foundation published a set of standards for interoperability based on COM technology for accessing control data using common models [15]. Venzke, Weyer and Turau [17] work with high level conceptual models based on web services technologies for automation systems, but focus on vertical integration. The European Installation Bus (EIB) [2] also uses a high-level interworking approach. Unlike the target applications for our approach, both vertical integration and home automation are usually not considered time-critical.

Plug-and-play capabilities are becoming increasingly important in the domain of smart transducer networks, as becomes evident with the recently accepted IEEE 1451.4 standard [9]. IEEE 1451.4 specifies a very compact template-based representation format for transducers, their parameters and operation values, which can be used for self-identification and low-level plug-and-play operation.

In the field of robotics, plug-and-play features for sensors and actuators are desired[1], but only a few implementations go further than a plug-and-participate approach. A prerequisite for plug-and-play features is a modular design of robotic components. Such an approach is followed by the G^{en}oM [12] modules for distributed robot architectures. Another approach for software reuse is given by the CLARAty architecture [13] and an application of CLARAty by a generic framework for robotic navigation [18]. In this work, Ursom, Simmons and Nesnas describe a framework of generic software components operating along sense-think-act lines. In contrast to our approach, the components are regarded as software components, instead of hardware/software units as it is the case with smart transducers.

3 Conceptual Model for Smart Transducer Systems

A smart transducer system consists of multiple nodes that communicate via a shared bus. A smart transducer application is built from components that interact using a shared interface system. For configuration we require specifications for the application, the target system, and the interfaces.

3.1 Application Model and Specification

Considering the overall application of a smart transducer system we divide the application into three levels (see Figure 1). For this purpose we use a modified version of the three-level sensor fusion model proposed in [3].

At the top-most level, the **control level** a control application defines the functional and temporal requirements and behavior of the overall application. At this level, the application developer should only have to operate with semantically rich, abstract data structures and not have to deal with low-level system specifics (e. g., hardware dependencies, low-level data encoding).

In [7] we proposed an XML-format for modelling real-time smart transducer applications as networks of interacting application components (services), using a data-flow-graph notation. Addition-

[1] *Eventually a modular plug-and-play system will be developed to control a variety of drive and actuating systems whether servo, stepper or pneumatic in nature. [1]*

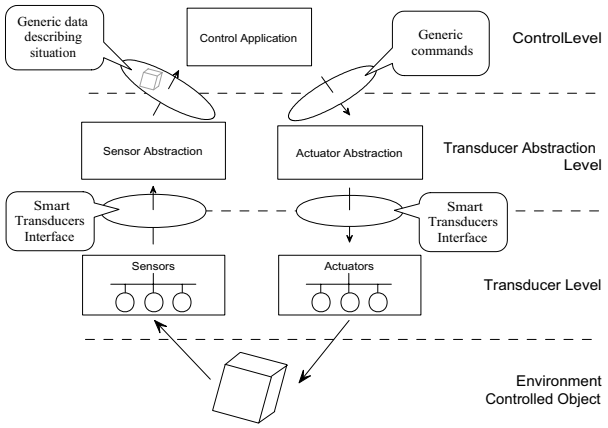

Figure 1: Application Model

ally, the specification format also allows defining further global constraints, such as end-to-end timing behavior and dependencies (phase, causal) between different components in the system.

The intermediate level, the **transducer abstraction level**, contains the hardware and software that acts as the glue between the control application and the transducer nodes in the system. In this paper we focus on the sensor abstraction part, which has to mediate between newly connected sensors and the control application.

Finally, the **transducer level** refers to the actual sensors and actuators that interface the environment. An important aspect at this level is the interfacing approach to the nodes, which allows hiding specifics of the underlying sensor/actuator hardware. As examples for such generic smart transducer interfaces see for example the IEEE 1451.2 standard [**?**] or the CORBA Smart Transducer Interfaces Standard [14] (see [6] for a comparison of both approaches).

This separation reduces configuration complexity by reducing the interaction between high and low-level system parts. In the remainder of this work the term application will refer to the control application.

3.2 Target System Specification

Since the application specification should be independent of the underlying hardware (as far as this is reasonably possible), we require a specification of the actual physical target system.

This specification deals with properties of the target system, such as network specific properties (physical layer, communication protocol), the list of connected transducer and processing nodes in the network, and information on the physical layout of the target system (e. g., physical dimensions, possible locations for transducers).

In general the specification is used, together with the application specification, for integrating the different components and setting up and configuring the final system. This includes among other things, recognizing the connected network nodes, setting up the communication (e. g., by creating the communication schedules in case of an underlying time-triggered communication protocol), and connecting the available transducer nodes to the high-level application components.

In [16] we proposed formats for representing a smart transducer node and cluster; but these approaches only partially cover these requirements.

3.3 Interface Model and Specification

All interactions in the system occur via a common, well-defined interface system. For the area of distributed real-time systems Kopetz and Suri propose the separation of the interfaces into the following interface classes [10]:

- *The Service Providing Linking Interface (SPLIF)* offers the service of the component to other components.
- *The Service Requesting Linking Interface (SRLIF)* is used for requesting services from other components, whereas a *user may not be aware that a component requests the services from other components* [10].
- *The Configuration Planning (CP) Interface* allows setting up the 'glue' (e. g., configuring components, setting up the communication) between the interacting components so that the system can provide the intended service.
- *The Diagnostic and Management (DM) Interface* allows *selective access to the (operational) internals of the component for monitoring and diagnostic purposes* [10].

For the specification of the Linking Interfaces, Kopetz and Suri distinguish the *operational* and *meta-level* service specification of an interface. The former deals with syntactical and temporal properties of the data transmitted over the interface. The latter shall give a *deeper meaning* to the data chunks transmitted over the interface by relating these chunks to a shared conceptual interface model [10]. The authors mainly focus on the gap between the informational and the user's level, but this conceptual service

model, if sufficiently formalized, can also be used for supporting our intended configuration operations. In the case study in Section 4 we will shortly outline a simple shared conceptual model that we use for this purpose.

4 Plug-and-play configuration of a Certainty Grid

A *certainty grid* is a multidimensional (typically 2D or 3D) representation of a robot's environment. The observed space is subdivided into cells, where each cell stores information about the corresponding environment and an estimated probability for the correctness of this information. Typically, a cell state can be "free", if the place appears to be void, or "occupied" if an object has been detected for that cell. Cells not reached by sensors reflect an "uncertain" state. The cell state and the probabilistic estimate of its correctness can be mapped into a single number reflecting the confidence of a cell to be free. Basically, it is assumed, that the application using the certainty grid has no a priori knowledge of the geometry of its environment and the objects in this environment are mostly static.

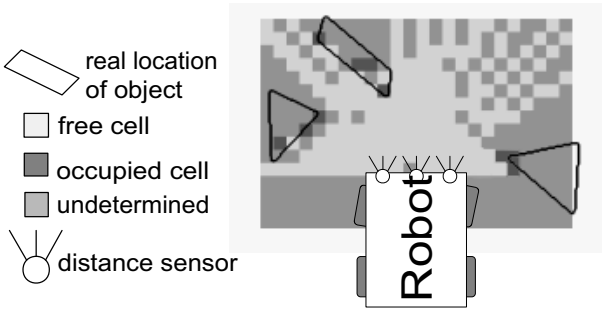

Figure 2: Example for a certainty grid

4.1 Case Study Setup

In the case study we use a small sensor fusion application for demonstrating our approach. Figure 2 gives an overview on the example application. A set of distance sensors delivers measurements about the distance to objects situated in the viewing direction of the sensors to the generic certainty grid component. By using the information about the current position and orientation in world coordinates, the generic certainty grid algorithm updates the map of the environment (in form of a regular grid) and presents this information to the navigation component. The navigation component runs a path planning algorithm and instruments the steering and motor actuators, which will move the robot according to the intended path.

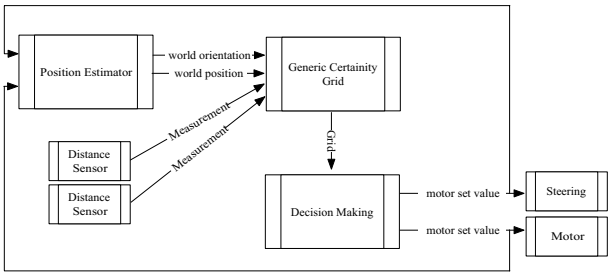

Figure 3: Components of the case study application

In this work we focus on the plug-and-play configuration of the generic certainty grid component. In order to configure the example system we require the following specifications in a formal computer-processable way:

Specification of the application. The application specification consists of a set of (high-level) application components, parametrization information for these components and a specification of timing requirements between components (end-to-end timing, phase relations). For the given example only the timing requirements and the description of the Generic Certainty Grid (GCG) are important. While the description of application components and their interconnection as described in Figure 3 is platform-independent, the mapping of components to the actual nodes must also be created. In our case this mapping is specified by the user as part of the application description.

Specification of the target system. The target system specification consists of a description of network properties, the available nodes in the network, and other information that is relevant for configuration. For the case study we need to specify the following information:

- Dimension of the target system (the robot car): We treat the robot's dimension as a plain rectangle with coordinate (0,0) as reference point located in the center.
- Possible locations (Transducer Slots) for smart transducers: each transducer slot has an identifier (a 4 bit value that is encoded in the sockets),

a two-dimensional (metric) 'coordinate' in relation to the reference point, and an orientation (as an angle). In this application, we assume that the orientation is fixed for each transducer slot.

Specification of the transducers. For successful configuration we also require a specification of the transducers, distance sensors in case of the GCG, involved in the configuration process. Section 4.3 presents our approach for representing a generic sensor.

Specification of the Generic Certainty Grid. Section 4.4 presents the specification of the GCG in detail.

Shared interface system. For accessing the nodes in the network and setting up the communication between the components we require a shared interface system. We use the CORBA Smart Transducer Interface [14] for this purpose, since it follows the interface approach outlined in section 3.3, i. e., it supports an interface for hard real-time data exchange, as well as a non-time critical configuration interface. In particular, we use the TTP/A protocol [11], which is the reference implementation of the CORBA STI, as the underlying communication protocol. In TTP/A, configuration data is guaranteed not to interfere with the real-time traffic, since it uses a fixed amount of reserved bandwidth. All interface data are mapped to a common interface mechanism, the Interface File System (IFS), which is similar to a shared memory that is organized as a simple flat file system.

4.2 Properties, Components, and Shared Concepts

In order to integrate information from the different specifications we require a common representation format and system model, interfacing mechanisms to the different system entities (hardware and software components), and configuration tools.

We opted for using an ontology format as common representation format. According to Gruber an ontology is a "*a specification of a conceptualization*" [8]. In particular, the following properties of ontologies are advantageous for our purpose:

- Support for creating networks of interconnected concepts
- Uniform access for all represented information
- Generic support for reasoning and querying
- Can be used as meta-data for existing formats (which are accessible from the meta-data by reference)
- Concepts in the model are templates for instances, which can be used to represent actual systems and system parts during configuration

Since the full domain models are far beyond the scope of this paper we will only present the most important shared concepts and how they are organized:

Data items represent the data chunks that are used (transmitted, configured) in the system. In our case we distinguish several sub-concepts. **Measurements** and **Set Values** are data items with an additional *transducer data type* property that specifies the kind of measurement or set value (e. g., distance, speed). The possible values for this property are represented in the shared domain model as instances. **Configuration parameters** allow controlling the behavior of an entity in the system and are treated as type-value pairs, whereas the value usually will have a simple data type (number, string).

Properties represent configuration-relevant attributes of an entity in the system. They represent concepts from the domain model (e.g., location) and convey information by their presence and their interconnection with other concepts in the domain model. Additionally, they can also have values of arbitrary type, either a simple data type or any other concept.

Most entities in the model (nodes, target system, application components) can have properties. Example properties in the context of our case study are **Location** or **Measurement Range**. Strictly speaking, configuration parameters, measurements and set values should also be considered as properties.

Nodes represent the physical transducer and processing nodes in the network. Main property for configuration is a list of hosted application components. **Smart Transducer Nodes** also have sensor/actor specific properties (e.g., scattering angle for distance sensors, active/passive). During configuration we distinguish between generic nodes that only represent properties that are independent of a particular target system (static information on a node, similar to the information provided in data sheets),

and nodes that are part of a target system and also store the nodes' current configuration state.

Application components represent the main functional building blocks of an application. They consume/process/provide application data, and shield the application from 'lower level specifics' of the system. As a unit of functionality an application component is specified by its interface, i. e., by the data it receives (e. g. measurements), the data it provides (e. g. set values), and configuration parameters that allow modifying its behavior (e. g. number of connected nodes). In addition, the component specification can also define restrictions on the data. For example, a component can require that a *distance measurement* must fall into a *range* from $0.5 - 1m$.

What is important for this approach is that the configuration framework does not deal with most of the interpretation of the actual data chunks that are moved between the different parts of the network, but defers the interpretation to the parties that produce/consume the data. Since the semantic information annotates the existing data formats as meta-information, the respective tools need not interpret the meta-information and the configuration framework need not interpret the actual content of the data formats.

Data that should be processed by the configuration framework must adhere to the data model of the domain model, which should be at a high level of abstraction (e.g., data items represented as numbers using SI units).

4.3 Specification of a Generic Distance Sensor

A distance sensor as used in our example is a component that produces a measurement on the distance to the next object in line of sight. The width of the measurement beam is defined by the scattering angle, e. g., a LIDAR laser sensor will have a very small scattering angle, while an ultrasonic sensor has a very broad sensing field. The sensor's measurement is also assigned a confidence value, which indicates the reliability of the measurement. The confidence value may be static for a given sensor type or be transmitted as a real-time value indicating the reliability of the actual measurement. In this case, confidence values are estimated by self diagnosis of the smart sensor. Sensors delivering different types of measurements (image, temperature, etc.) are not suitable to be automatically configured and included in the case-study application.

Basically, we treat a sensor like any other application component, i. e., by specifying its real-time and configuration data; but unlike high-level application components it will usually not directly be interfaced by the application developer. Smart transducers are often treated as software/hardware components with pre-defined functionality. To fit our application model, where we treat an application component as independent of the physical node it is mapped to, we distinguish between the software component responsible for communicating the measurement (*measurement component*) and the smart transducer as a physical device. In this case, the mapping from application component to node is already defined in the transducer description.

This example measurement component has no configuration parameters or further properties; but the underlying smart transducer has properties that are required for configuration, which are the *scattering angle*, *measurement range*, and *location*.

4.4 Generic Certainty Grid

The output of the generic certainty grid (GCG) is an abstract data structure representing a certainty grid that can be passed on to other components for high-level decision processes like path-finding, etc.

Figure 4 depicts the GCG component in terms of the interface model outlined in Section 3.3. The DM interface is not depicted in the figure, since it is not relevant for this work.

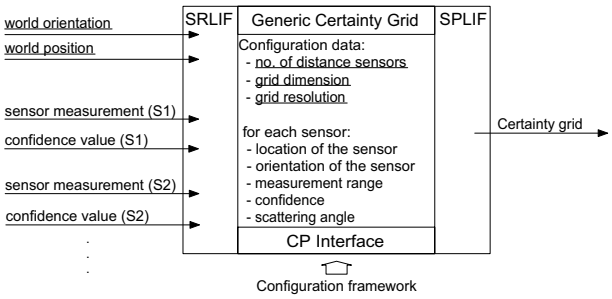

Figure 4: Interfaces of the Generic Certainty Grid Component

The underlined items displayed in the figure are specified by the user as part of the control application. The others are only available to the configuration framework.

Table 1 gives an overview on the configuration parameters. 'Obtained from' denotes the source of the

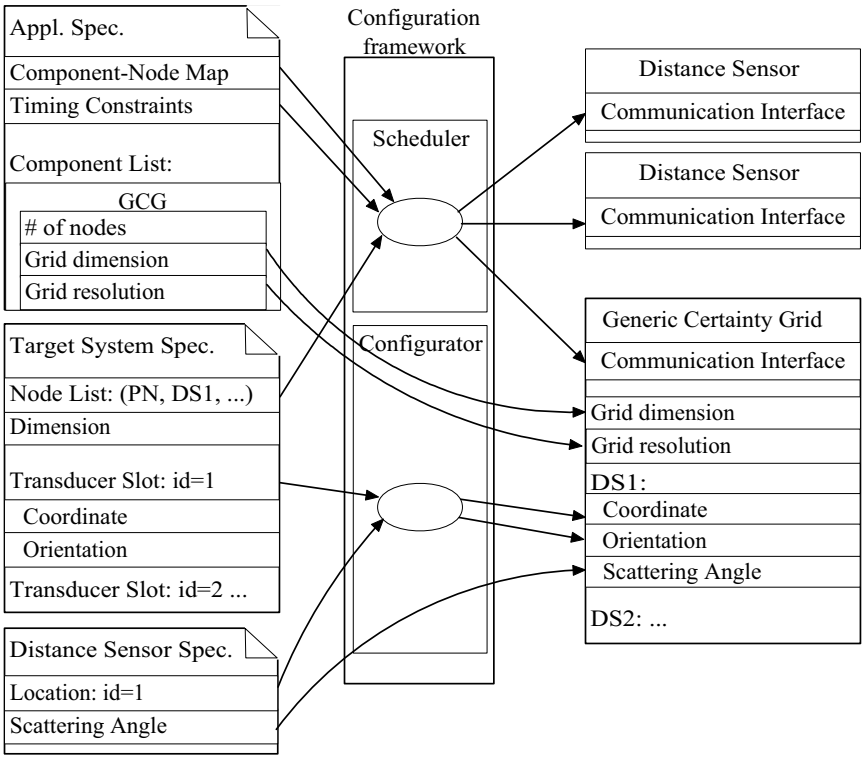

Figure 5: Data flow in the configuration framework

description of the configuration information (for obtaining the structural/conceptual meta-information required for interpreting the data), 'established by' denotes the entity that is responsible for creating the actual configuration value (either the *user* or the *configuration environment*).

From the configuration parameters, the user must explicitly specify the number of connected sensors, since the configuration environment requires this information for reserving memory space in the interface system on the node hosting the GCG (in the IFS) for the data coming from the sensors.

The location must be obtained from the connected sensor (by the configuration framework; if the sensor cannot obtain its location from the environment, the user must configure the sensor accordingly). The scattering angle is a property of the connected sensor (and available from the sensor description).

Table 2 shows the real-time data produced/consumed by the GCG. 'Connected to' specifies producer/consumer of the currently described data in relation to the overall application. The interaction with other application components is specified by the user and, thus, available in the application description.

The configuration tool must set up the communication connection between the connected sensors and the application component by using the data format that is 'specified in' the sensor description.

4.5 Configuring the Generic Certainty Grid

We assume that there is a valid application description and a physically correctly configured system, i.e., the required transducers are placed at suitable locations of the target system and connected to the network, and all nodes use the same protocol and identical communication parameters.

The basic idea behind the configuration process is to first identify the parameters required for the configuration of an entity in the system (application component, node), and then obtaining the required configuration values from the available specifications (either directly or after processing by the configuration framework). To enable this approach all configuration-related data must be available, and 'connected' to the common system model, so that it is accessible to the configuration framework.

Figure 5 depicts the system entities involved during

Configuration data		
Data	*obtained from (during configuration)*	*established by*
Dimension of grid in m	Application description	User
Resolution of grid in cells	Application description	User
# of connected sensors	Application description	User
For each connected sensor		
Location of sensor	Sensor/System description	Config system
Orientation of sensor	System description	Config system
Scattering angle	Sensor description	Config system

Table 1: Configuration data of the GCG

Real-time data		
Data	*connected to (during run-time)*	*specified in*
Car orientation	Position estimator component	Application description
Car position	Position estimator component	Application description
From each connected distance sensor		
Distance measurement	Distance sensor	Sensor description
Measurement reliability	Distance sensor	Sensor description

Table 2: Real-time data of the GCG

configuration.

Configuration proceeds as follows: connect and detect sensors (plug-and-participate), configure new sensors, configure the GCG, set up real-time communication. First, the network is scanned for the available transducer nodes. This is done using the baptize algorithm in TTP/A [4]. After this step we can address and access all connected transducers via a non-real-time configuration interface. In this particular example application the distance sensors require no further configuration by the configuration tool.

The location property of a sensor is undefined before the sensor is mounted and connected to the robot. When the smart transducer is actually placed in the system, the location information becomes available. This can be either done by setting dip switches on the sensors or by using encoded sockets. Each socket in the system is then encoded with an identifier that indicates the physical location on the robot. When a sensor is plugged in, it reads the socket identifier and provides this information to the configuration tool. The configuration tool can then derive the actual location information and orientation, for configuring the respective configuration parameters of the GCG, from the system description and the socket identifier. The last configuration parameter is the scattering angle, which is available from the sensor description.

The maximum number of possible connected distance sensors is limited by the number of physical sockets and available bandwidth in the communication schedule.

Note the difference between parameters that can only be interpreted in context of a particular target system (e. g., location identifier) and generic parameters that are required for the function of a component (e. g., metric coordinates, scattering angle). The latter should be represented using common (preferably standardized) formats.

After the GCG and the connected sensors have been successfully configured we can setup the real-time communication, with the target that the correct transducer delivers its information to the respective correct input of the high-level application component. Since we know which sensor outputs are associated with the respective inputs of the GCG we can reserve memory space for the input data on the node hosting the GCG and adopt the communication schedule by feeding the resulting list of components and the user specified timing constraints to a scheduling tool [5], which creates the scheduling tables for all nodes connected to the network.

All these configuration parameters can then be downloaded to the cluster via the CP interface.

5 Discussion and Conclusion

In this paper we outlined an approach for bridging the semantic gap between a control application and transducers by applying a smart interface system that preprocesses the sensor data and provides it to the control application in a standardized form independent from the employed transducer types. This system allows the connection of new sensors of arbitrary type. If the sensor's meta description matches the requirements of the interface system, the sensor can be integrated into the application without changing the software.

By treating all the information from the different sources in the same framework, we avoid inconsistencies and conversion problems (or at least remove them from the central configuration problems) that may arise by directly combining information from heterogenous sources.

Currently we perform the major part of the configuration, i. e., interpretation of the meta-data, creating schedules, off-line on a host PC. This has the advantage of reducing the overhead on the smart transducer nodes to a minimum, which is important, since our current implementations mainly focus on low-cost 8bit micro-controllers. Since there is hardly any meta-data that must be stored in the nodes' memories, the amount of configuration data that must be transmitted to the cluster is very small; in the order of a few hundred bytes including the communication schedules.

Integrating the plug-and-play functionality into the embedded system is possible, although an increased overhead onto the network nodes can be expected.

Although the approach is limited by the number of physical sockets and the available communication bandwidth, it is a significant improvement from the trivial approach of pre-reserving fixed communication slots for each particular possible new sensor, since our generic approach allows the connection of arbitrary distance sensors, even if a particular sensor type has not been expected at design time of the system.

In the presented case study we focused on sensor properties that are relatively simple to process. In the future we intend to include more complex relations between sensors and between transducers and the environment into the common model. For example, if an application uses multiple active sensors of the same type, this can lead to interferences. In this case the configuration tool must recognize this interference and schedule the measurements sequentially.

6 Acknowledgments

This work was supported in part by the FIT-IT project "SDSTI" under contract No. 808693 and by the European IST project DECOS under contract No. IST-511764. FIT-IT is funded by the Austrian ministry for transport, innovation and technology (BM-VIT). We would like to thank our colleagues at the Institute of Computer Engineering for their valuable comments on earlier versions of this paper.

References

[1] G. Bright and J. Potgieter. PC-based mechatronic robotic plug and play system for part assembly operations. In *Proceedings of the IEEE International Symposium on Industrial Electronics*, pages 426–429, Pretoria, South Africa, July 1998.

[2] EIBA Association. *EIBA Handbook Series Release 3.0*, Mar. 1999.

[3] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[4] W. Elmenreich, W. Haidinger, P. Peti, and L. Schneider. New node integration for master-slave fieldbus networks. In *Proceedings of the 20th IASTED International Conference on Applied Informatics (AI 2002)*, pages 173–178, Feb. 2002.

[5] W. Elmenreich, C. Paukovits, and S. Pitzek. Scheduling tool for reconfigurable time-triggered embedded sensor networks. Research Report 29/2005, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2005.

[6] W. Elmenreich and S. Pitzek. Smart transducers – principles, communications, and configuration. In *Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems*, volume 2, pages 510–515, Assuit – Luxor, Egypt, Mar. 2003.

[7] W. Elmenreich, S. Pitzek, and M. Schlager. Modeling distributed embedded applications on an interface file system. In *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 175–182, May 2004.

[8] T. Gruber. A translation approach to portable ontologies. *Knowledge Aquisition*, 5(2):199–220, June 1993.

[9] Institute of Electrical and Electronics Engineers, Inc. *IEEE Std 1451.4-2004, IEEE Standard for A Smart Transducer Interface for Sensors and Actuators - Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, Dec. 2004.

[10] H. Kopetz and N. Suri. Compositional design of rt systems: A conceptual basis for specification of

linking interfaces. *6th IEEE International Symposium on Object-Oriented Real-Time Computing (ISORC03), May 14 - 16, 2003, Hokkaido, Japan*, May. 2003.

[11] H. Kopetz et al. Specification of the TTP/A protocol. Technical report, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, Sept. 2002. Version 2.00.

[12] A. Mallet, S. Fleury, and H. Bruyninckx. Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 22922297, Lausanne, Switzerland, Oct. 2002.

[13] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W. S. Kim. CLARAty: An architecture for reusable robotic software. In *Proceedings of SPIE Aerosense Conference*, Orlando, Florida, 2003.

[14] Object Management Group (OMG). *Smart Transducers Interface V1.0*, Jan. 2003. Specification available at http://doc.omg.org/formal/2003-01-01 as document formal/2003-01-01.

[15] OPC Task Force. *OPC Overview — Version 1.0*, Oct. 1998.

[16] S. Pitzek and W. Elmenreich. Configuration and management of a real-time smart transducer network. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*, Lisbon, Portugal, Sept. 2003.

[17] V. Turau, M. Venzke, and C. Weyer. Vertical integration of TTP/A fielbus systems using web services. In *Proc. of the First Int. Conf. on Informatics in Control, Automation and Robotics (ICINCO)*, Aug. 2004.

[18] C. Urmson, R. Simmons, and I. Nesnas. A generic framework for robotic navigation. In *Proceedings of the IEEE Aerospace Conference*, volume 5, pages 2463–2470, Montana, Mar. 2003.