# Modeling Distributed Embedded Applications on an Interface File System

Wilfried Elmenreich, Stefan Pitzek and Martin Schlager
Vienna University of Technology
Institut für Technische Informatik
{wil,pitzek,smartin}@vmars.tuwien.ac.at

May 31, 2005

**Abstract** – *This paper presents a framework for generic modeling of distributed embedded applications. An application is decomposed into services and mapped on a set of distributed nodes, whereas each node hosts one or more services. Each service is described by four interfaces: a real-time input/output, a configuration and planning (CP), and a diagnostic and management (DM) interface. The overall application is described by a cluster configuration description that specifies the interaction of services within and across nodes.*

*The application requirements, the service properties of a node, and the interaction of the services as well as the application mapping are described formally with XML descriptions. The XML format allows a language-neutral and extensible semantic description of interfaces supporting the implementation of context-aware tools for modeling, scheduling, monitoring, simulation, and validation.*

*A central concept of the model is the interface file system (IFS) that acts as a distributed shared memory and transparently implements the interfaces to services from other nodes. In principle, the communication system that updates the data in the IFS data is not bound to a specific implementation as long as it fulfills the given timing requirements. The presented concepts are applied in a case study that uses the time-triggered fieldbus protocol TTP/A for the implementation of a small sensor fusion application.*

## 1 Introduction

An embedded system is a computer system designed to perform a dedicated or narrow range of functions with minimal user intervention. By definition, an embedded system can be built either in a monolithic or a distributed way. In this paper we focus on distributed embedded systems. Distributed embedded systems are motivated by several aspects, i. e., (i) the parts of a system, e. g., sensors and actuators, may be situated spatially far apart from each other (as it is the case in factory and building automation), (ii) the need for parallelism (e. g., grid computing, DSP for special tasks), (iii) the need for fault tolerance (e. g., by introducing redundant nodes and channels), and, (iv) the need for a modular design (e. g., extensions for PCI bus).

Such a distributed embedded system can be treated as a composition of several subsystems or components, whereof each component consists of a hardware and software part. The implementation of these components may differ in used programming language (e. g., Assembler, VHDL, C, C++, Java), timing behavior (in terms of throughput, worst-case behavior, response time), I/O types (digital/analog, determinism, ...), and dependability (reliability, availability, maintainability, ...). Finding an adequate model for such systems is difficult, since on the one hand, the model must be rigid in order to describe critical parameters such as timing and dependability with sufficient precision, on the other hand, due to the many different application requirements, the model should be flexible and extensible.

This paper focuses on the modeling of distributed embedded applications based on a description of components in an XML-based specification language. Each service is mapped onto an Interface File System (IFS) that forms its input and output interface. Additionally, the IFS contains configuration and maintenance data of the subsystem, thus all *state* information of the subsystem is contained in the IFS. All other aspects, such as the syntactic descriptions of the information in the IFS, are mapped into an external XML description of the component. The IFS acts as a common interface that supports a unified mapping of interfaces, while the extensible XML descriptions provide an extensible semantic description for them.

This paper is structured as follows: The next section outlines important requirements for embedded real-time systems. Section 3 depicts

the conceptual model of the framework. Section 4 shows the modeling via the IFS. Section 5 presents the implementation of the framework in the time-triggered fieldbus network TTP/A. Section 6 presents a case study application of the framework for real-time simulation of TTP/A functions. Section 7 lists related approaches and Section 8 concludes the paper.

## 2 Requirements

The following requirements are prevalent in many distributed embedded systems:

**Hard real-time support:** A hard real-time system is a real-time system in which a guarantee can be given that a certain action will always finish before a given deadline. Many embedded systems require hard real-time behavior in order to avoid damage to man or machine.

**Support of low-cost embedded systems:** Although state-of-the-art technology provides powerful 32 bit architectures, the presented modeling approach shall also support applications on networks of small 8-bit microcontrollers such as the ATMEL AVR, Microchip PIC, and the Motorola HC08 families. Still, 8-bit microcontrollers have the greatest share in the volume market for microcontrollers.[1]

**Two-level design approach:** The two-level design approach proposes the separation of the implementation of subsystems and the integration of these subsystem into an overall system. Thus, the implementor of a subsystem focuses on local issues such as interfacing local sensors and actuators, while the system integrator focuses on the interaction of the subsystems.

In order to support a two-level design approach the architecture must support composability [12]. An architecture is composable with respect to a specified property if the system integration does not invalidate this property when it was established at the subsystem level.

**Reuse of existing systems:** Many projects require that legacy code and legacy components must be included into an application. Thus, the presented modeling approach shall support the reuse of existing solutions.

## 3 Conceptual Model

A distributed embedded application will be first described *functionally*, i. e., by a set of interconnected real-time *services*. A service is described by its interfaces, its function, and properties like timing behavior or reliability requirements.

The interfaces of a service are divided into the following categories:

[1]Source: Gartner Dataquest (August 2003)

**Service Providing Linking Interface (SPLIF):** This interface provides the real-time service results to other services (cf. [11]).

**Service Requesting Linking Interface (SRLIF):** A service that requires real-time input information requests these data via the SRLIF (cf. [11]).

**Diagnostic and Management (DM):** This interface is used to set parameters and to retrieve information about intermediate and debugging data, e. g., for the purpose of fault diagnosis. Access of the DM interface does not change the (a priori specified) timing behavior of the service.

**Configuration and Planning (CP):** This interface is used during the integration phase to generate the "glue" between the nearly autonomous services (e. g., communication schedules). The CP interface is not time critical.

**Local interfaces:** The term local interfaces subsumes all kinds of devices, such as sensors, actuators, displays, and input devices, for which the service creates a unified access via the SPLIF or SRLIF services. For example, the service may instrument a physical sensor element by reading the measurement, calibrating the value, and exporting the measurement via its SPLIF.

Figure 1 depicts the interfaces of a service in a block diagram.

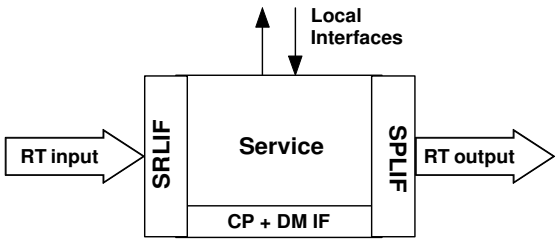


Figure 1: Interfaces of a Service

A particular component may comprise only a subset of the above described interface categories. Typically, a *smart sensor* service will implement a SPLIF, CP, DM, and a local interface to the physical sensor. An *actuator*, in contrast, will implement an SRLIF, CP, DM, and a local interface to the physical actuator.

Data flow over the SPLIF and SRLIF is performed using *ports*. A port is specified by a name, a description, and the structure of the data transmitted over the port (e. g., a 16bit measurement value from a sensor). The port structure consists of the data type of the expected input or, respectively, the produced output.

The functional behavior of a service is implemented by a service *task*. The task of a service consumes the data at its SRLIF and produces an output at its SPLIF after termination. The data at the SRLIF must not be changed during task

execution and the data at the SPLIF must not be read during task execution. The behavior of a task may also depend on configurations via the DM interface, whereas this configuration is not allowed to change during task execution.

An application consists of one or more services that interact with each other via the real-time interfaces SPLIF and SRLIF. One or more services are implemented per *node*, whereas nodes are loosely coupled via a real-time communication system. Services that require local data exchange communicate via a shared memory interface. Services that are required to exchange data between different nodes communicate via the node's communication interface.

The representation of an application by its services deals with the functional and data flow parts of the application. In order to completely represent an actual application, several non-functional properties must be specified. We distinguish the following two classes:

- Service-specific properties that must be set according to requirements of the application (e. g., the update rate of an actuator).
- End-to-end requirements of the application that can only be specified over the distributed application (e. g., end-to-end signal delay in control loops).

The service-specific properties are modeled in the semantic description of the service. The end-to-end requirements are expressed by so-called *dependencies*. We have identified the following kinds of dependencies:

**Connection:** This dependency represents the data flow between ports of services. A connection is directed by having a source and a target part. An input port may have only one connection to an output port, while an output may feed several input ports.

**Causal:** By defining a causal dependency between services, all in-between services (on the directed application graph) must comply to this dependency. In our context causal dependencies always incur timing requirements. An *instant* identifies the timing requirements of participating services. We distinguish the instant before and after execution. The *before* instant of a service is before the service task is executed, i. e., the moments when input data must have arrived. The *after* instant happens after the service task execution, thus a duration of $\mathrm{WCET}_{TASK}$ after the before instant, where $\mathrm{WCET}_{TASK}$ is the worst case execution time of the service task.

**Phase:** The phase dependency specifies non-causal time-related dependencies of instants among services.

All the presented properties are used to model the application requirements. All requirements are expressed in XML descriptions, which support interaction among tools (e. g., for modeling, code generation, and verification). A case study that depicts concrete XML descriptions is shown in Section 6.

# 4 Interface Implementation

An interface must establish a common boundary between services and their users. In order to exchange information across such an interface the engaged communication partners must share a common background of concepts [13].

Basically, the interface to a service will be modeled as a structured shared memory. All interfaces are modeled upon an Interface File System (IFS) [13] that provides a common name space by a uniform addressing scheme. The values that are mapped into the IFS are organized in a static file structure. The address space is organized hierarchically representing the network structure:

**Cluster name:** A cluster comprises a network of fully interconnected nodes. The cluster name is an 8 bit integer value that identifies a particular cluster. Native communication (without routing) among nodes is only possible within the same cluster.

**Node alias:** The node alias or logical name selects a particular node. Node aliases can have values from 0...255, whereas some values have an associated special function, e. g., alias 0 addresses all nodes of a cluster at once, i. e., a broadcast manner [14].

**File name:** The file name is a 6-bit identifier for addressing individual files in the nodes. A subset of files, the system files, have a special meaning in all nodes. Each service of a node is mapped onto a file containing sections for SPLIF, SRLIF, CP, and DM data.

**Record number:** The record number is an 8-bit identifier that addresses the record within the selected file. Each record contains 4 data bytes. Since each file has a statically assigned number of records, a file contains only the necessary number of records.

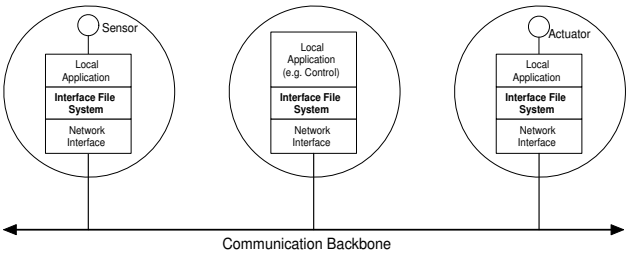


Figure 2: Physical Network Topology

Figure 2 depicts a possible application with a network of three nodes. Each node may host one or multiple services that all use the local IFS as a data source and sink. The communication system will perform a periodical time-triggered communication to copy data from the IFS to the communication system and write received data into the IFS. Thus, the IFS acts as an interface that decouples the local services from the communication subsystem. It is the task of the communication system to keep consistency among the local copies of the data stored in the IFS. A predefined communication schedule defines time, origin, and destination of each communication action.

As depicted in Figure 3, local applications share a common view on the IFS. The application programmer is relieved from considerations concerning low-level communication. Thus, any service perceives the IFS in a logical network structure as depicted in Figure 3.

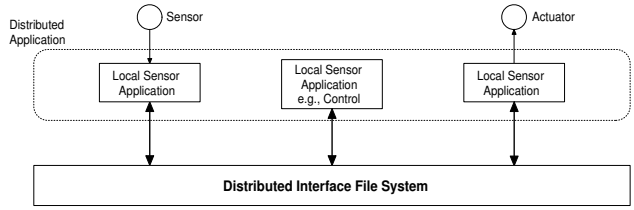


Figure 3: Logical Network Structure

The SPLIF, SRLIF, CP, and DM interfaces of a service are mapped into a file of the IFS on the node that hosts the service. The IFS maps the current state of a all process variables into so-called I/O files. Additionally there may be configuration data, for example communication schedules. In [15] we have described the representation and use of the IFS CP interfaces based on XML-based descriptions. In the following we will describe the structure of the SPLIF and SRLIF, which establish the real-time service.

Each port in the service file specifies a pointer into the local I/O file of the node. The I/O file hosts the actual input values and the output values of the local services. The input values can be provided either by a local or remote service – in the latter case, the communication system will update this value periodically

Besides the ports, the service file contains service configuration data (e. g., parameters for a PID control algorithm) and diagnostic data such as intermediate results or sensor logs.

# 5 Communication System

The communication system that performs the updates of the IFS data is not bound to a specific implementation as long as it provides deterministic timing behavior in order to fulfill the timing requirements.

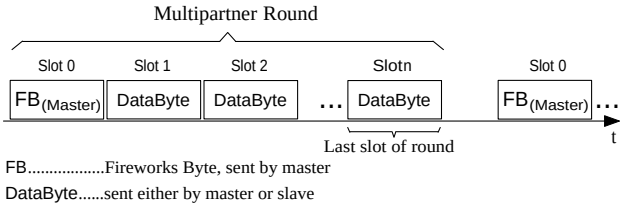


Figure 4: Example for a TTP/A multipartner round

We use the time-triggered fieldbus protocol TTP/A, since it meets the timing requirements and supports low-cost implementations of network nodes. TTP/A is standardized as part of the Object Management Group Smart Transducer Interface Standard [14].

## 5.1 Principles of Operation

In TTP/A, the bus allocation is controlled by a Time Division Multiple Access (TDMA) scheme. Communication is organized into rounds consisting of several TDMA slots. A slot is the unit for transmission of one byte of data. Data bytes are transmitted in a standard UART format. Each communication round is started by the master with a so-called fireworks byte. The fireworks byte defines the type of the round and is a reference signal for clock synchronization. The protocol supports eight different firework bytes encoded in a message of one byte using a redundant bit code supporting error detection. Generally, there are two types of rounds:

**Multipartner round:** This round consists of a configuration dependent number of slots and an assigned sender node for each slot. The configuration of a round is defined in a datastructure called "RODL" (ROund Descriptor List). The RODL defines which node transmits in a certain slot, the operation in each individual slot (read, write, execute), and the receiving nodes of a slot. RODLs must be configured in the slave nodes prior to the execution of the corresponding multipartner round. An example for a multipartner round is depicted in Figure 4.

**Master/slave round:** A master/slave round is a special round with a fixed layout that establishes a connection between the master and a particular slave for accessing data of the node's IFS, e. g., the RODL information. In a master/slave round the master addresses a data record using a hierarchical IFS address and specifies one of the following actions: reading from, writing to, or executing a record.

The master/slave rounds implement the DM and the CP interface to the transducer nodes. The RS interface is provided by periodical multipartner rounds. Master/slave rounds are scheduled

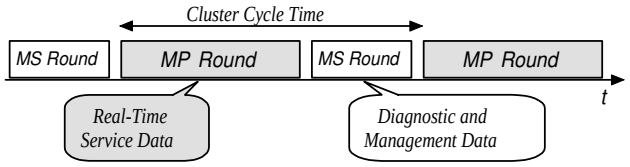


Figure 5: Recommended TTP/A Schedule

periodically between multipartner rounds as depicted in Figure 5 in order to enable maintenance and monitoring activities during system operation without a probe effect [5].

# 6 Case Study

The case study implements a distributed application performing a robust distance measurement. We will show how an application can be modeled according to the presented concepts, providing XML-based descriptions for services and applications and defining an actual low-level mapping of services to the IFS.

## 6.1 Application Specification

The case study hardware comprises three infrared (IR) distance sensors and a display. The application shall perform a reliable distance measurement from the three distance sensors, fuse the measurements and display the result onto the display. All of the three distance sensors have to perform synchronous measurements. According to the precision of these measurements, on each sensor a confidence value (0 for lowest confidence, 12 for highest confidence) is calculated as a result of fusing subsequent measurements from the local sensor. The fusion service performs a confidence-weighted average fusion [2] on the three value/confidence data pairs from the distance measurements.

Figure 6 depicts the service model of the application. Blocks represent the services and lines between blocks show the data flow in the model. The IR sensor service does not receive messages from other network participants, thus it has no service requesting linking interface (SRLIF). The fusion service takes the outputs from the IR sensor services (i. e., distance and confidence values from the IR sensors) as inputs, executes the fusion algorithm, and produces the fused distance and an according confidence value as outputs. Thus, the fusion service contains both, a service requesting linking interface (SRLIF) and a service providing linking interface (SPLIF). Finally, the display service takes the outputs from the fusion service for displaying it. Since the display service does not send messages to other services, it does not implement a SPLIF.

As indicated in Figure 6, each service (IR sensor, fusion, display) is also equipped with a configuration and planning (CP) interface and a diagnostic and maintenance (DM) interface. The CP interface is used at setup time in order to configure the time-triggered communication messages among the nodes. The DM interface can be used for diagnostic and for configuration of local services. Table 1 gives an overview on the services and interfaces of the case study.

| Service | Interface | Description |
|---|---|---|
| IR sensor | SRLIF | – (not required) |
| IR sensor | SPLIF | provide distance and confidence values |
| IR sensor | CP | configure real-time communication pattern |
| IR sensor | DM | calibrate IR sensor (history values, precision) |
| Fusion | SRLIF | receive distance and confidence values |
| Fusion | SPLIF | provide filtered distance and confidence values |
| Fusion | CP | configure real-time communication pattern |
| Fusion | DM | configure fusion algorithm |
| Display | SRLIF | receive filtered distance and confidence values |
| Display | SPLIF | – (not required) |
| Display | CP | configure real-time communication pattern |
| Display | DM | – (not required) |

Table 1: Description of Service Interfaces

Figure 6 also shows the external XML-based service descriptions that provide a straightforward formal representation of the properties of the service. These descriptions take a similar role as classes in object oriented programming, since they act as generic types of services, which are then instantiated in the application.

Note that the respective elements in the XML representation are intentionally kept simple for brevity and easier implementation, but it would be easy to provide semantically richer specifications for parts of the descriptions, e. g., using XML schema for describing the layout of the IFS elements instead of the currently used proprietary data type formats.

For the application in this case study we can identify the following application-specific requirements:

1. The value shown on the display must be updated at least every 0.1s ($d_{update} = 0.1s$).
2. IR measurements must be synchronized with a precision of $\pm 1ms$.
3. The temporal accuracy $d_{acc}$ of the value received by the display service must be 0.05s.

Figure 7 shows fragments from a description of the case-study application in XML. The `service` elements specify the "instantiated" services of the application model. Each `service` element defines an application-wide unique identifier and refers to

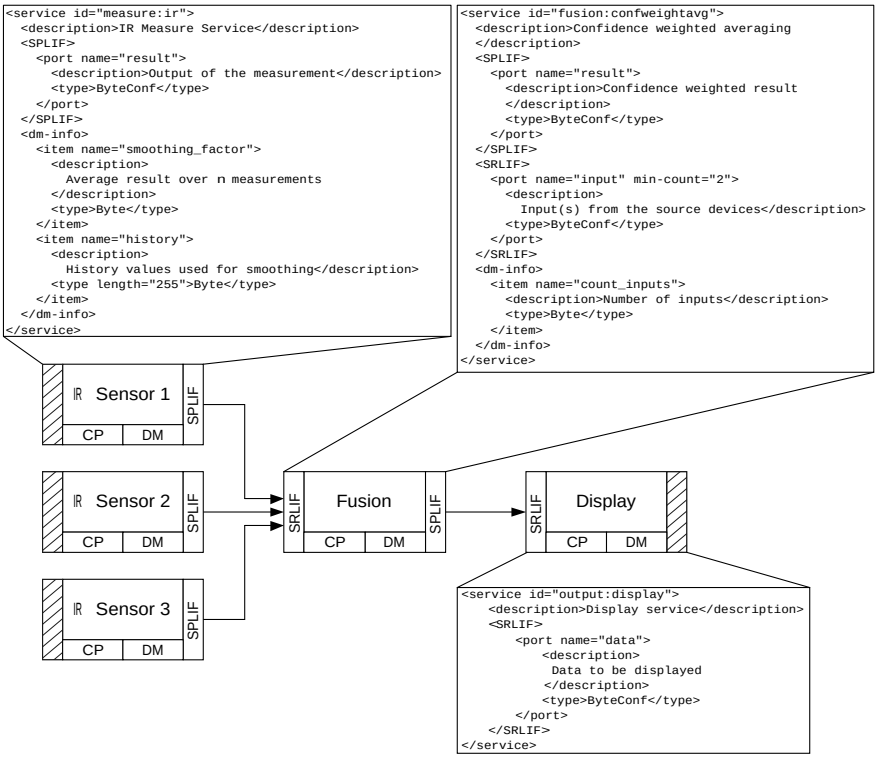


Figure 6: Conceptual model of case study

the location of the associated service description file. Since $d_{update}$ is a property of the display service, it is defined as part of this local service specification. The rest of the application is described using dependency elements.

The `property` elements used in the dependencies specify the relation between the services in detail (e. g., defining the required precision, upper and/or lower bounds for values). For example, the requirement that the three sensor services should be synchronized within 1ms is expressed by specifying that service IR1, IR2, and IR3 must run within a phase of $0 \pm 1ms$.

The third application specific requirement is modeled as a causal dependency between the IR service and the display service, i. e., the time between the start of execution of the IR service and receiving the fused value by the display service must not be greater than 0.05s.

## 6.2 Implementation Model

Figure 8 depicts the mapping of services onto network nodes. The network comprises three autonomous nodes hosting the infrared distance sensors, a node for the display, and a master node that hosts the fusion service. The master node acts as a reference clock for the time-triggered communication within the cluster.

All relevant data of a node has been mapped in the node's interface file system (IFS). The IFS forms the SPLIF, SRLIF, CP, and DM interface for each node. All services, like measurement, triggering a fusion process, and updating the displayed data are mapped as tasks in corresponding IFS files.

## 6.3 Evaluation

The resulting system was implemented on five 8-bit low-cost microcontrollers (Atmel AVR8 family) which are interconnected by an ISO 9141 bus system running at a communication speed of 9600 Bit/s. Figure 9 depicts the hardware used for the case study, which consists of a master node (implemented on an AT90S8515), display node (ATmega128) with display unit, and three IR sensor nodes (ATmega128) with IR sensors (Sharp GP2D12).

The timing requirements defined in section 6.1 are fulfilled as follows:

- The display was updated every 25.73ms. ✓
- The sensor measurements are synchronized within $\pm 0.104ms$. ✓
- The value received by the display service is based on sensor measurements made 13.54ms before the instant of displaying. ✓

```
<application>
  <service id="IR1" spec-location="measureIR.xml"/>
    ...
  <service id="DISPLAY" spec-location="display.xml">
    <property name="d_update"><dur bound="maxInclusive">
        0.1<unit>s</unit></dur></property>
  </service>
  <dependency kind="connection">
    <source service="IR1" port="result"/>
    <target service="FUSION" port="input"/>
  </dependency>
    ...
  <dependency kind="connection">
    <source service="FUSION" port="out"/>
    <target service="DISPLAY" port="data"/>
  </dependency>
  <dependency kind="causal">
    <instant service="IR1" type="before"/>
    <instant service="DISPLAY" type="before"/>
    <property name="d_acc"><dur bound="maxExclusive">
        0.05<Unit>s</Unit></dur></property>
  </dependency>
  <dependency kind="causal">
    ...
  </dependency>
  <dependency kind="phase">
    <instant service="IR1" type="before"/>
    <instant service="IR2" type="before"/>
   <property name="phase">0<Unit>ms</Unit></property>
    <property name="precision">
      <dur bound="maxInclusive">1<Unit>ms</Unit></dur>
      <dur bound="maxInclusive">-1<Unit>ms</Unit></dur>
    </property>
  </dependency>
  <dependency kind="phase">
      ...
  </dependency>
    ...
</application>
```

Figure 7: Representation of the application with XML

### 6.4 Further Aspects of the Case Study

Given a real-time fieldbus system, the above mentioned interface concept eases the seamless integration of simulation aspects. Each functional service is represented as an independent object, that can be mapped to any existing hardware basis. Due to such encapsulation, each node of the distributed system can be emulated through a simulation host with the same functional and temporal behavior.

In experiments, we emulated two out of the three IR sensors through a simulation node [16]. The simulation unit communicates via its SPLIF and SRLIF with the other network participants.

## 7 Related Work

Many approaches for model- or component-based application modeling only specify components by describing their functional interface. In order to support modeling of real-time applications, additional properties such as timing and other non-functional properties (dependability, quality of service [6]) must be taken into account. There exist several approaches that address the timing

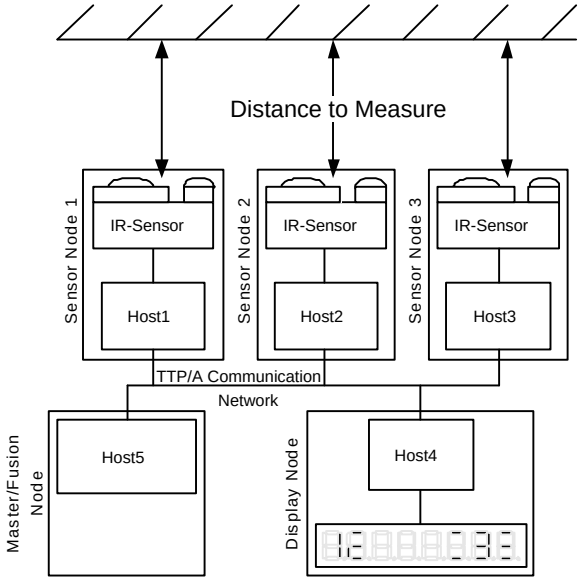


Figure 8: Nodes of the case study application

requirements.

Giotto [9] represents a domain-specific high-level programming language for control applications that exactly specifies the real-time interaction between software components and the real world.

The CIP method (described by Fierz in [4]) decomposes the construction of embedded systems into a functional and a connection part.

The BASEMENT$^{TM}$ [8] architecture presents a high-level application modeling approach based on software circuits, also taking resource requirements and timing into account.

Even though these approaches are targeted to embedded systems they cannot be easily adopted to ultra-low cost systems. An approach that can be compared to the IFS-based modeling approach is the IEEE1451 standard [10] that specifies an application model and a digital low-level interface for smart transducers. In [3] we discuss the main differences in the design decisions between both approaches. For example, IEEE1451 specifies digital communication lines as common interface mechanisms, whereas the CORBA STI interfacing mechanism is based on a shared memory concept.

XML has already been used for describing interfaces of conventional software components, e. g., by Bramley et al.[1], in the Web Services Description Language (WSDL) [17], and, for the description of communication and computation properties, in the Communication-Computation Description Language (CCDL) for ModelJ components [7].

## 8 Conclusion and Outlook

The presented modeling approach builds on generic services that are hosted in nodes which

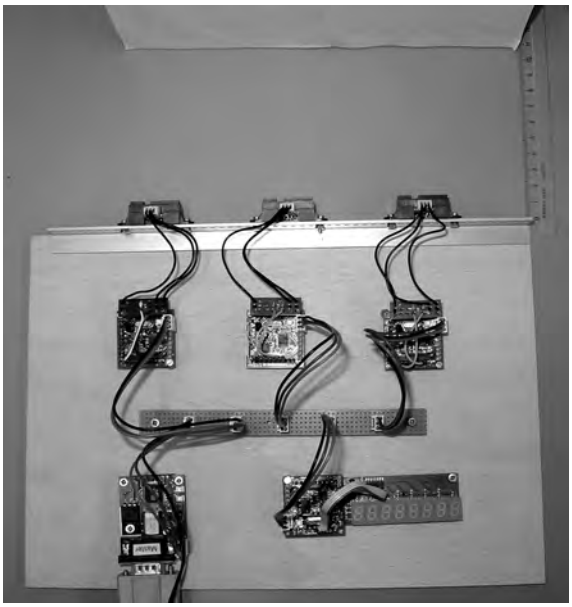

Figure 9: Hardware setup of the case study application

are interconnected by a real-time system. The final application is described by a cluster configuration description that specifies the interaction of services within and across nodes.

The basic concept of the model is the interface file system, which acts as a distributed shared memory and transparently maps the interfaces to services from other nodes. The communication system that performs the updates of the IFS data is, in principle, free to choose as soon as it fulfills the requirements of timeliness and determinism that allow a static analysis of the timing properties of the distributed application. In this paper we have used the time-triggered fieldbus protocol TTP/A for the communication system.

The presented model supports a top-down as well as a bottom-up approach. In a top-down approach, the system will first be described by abstract services, which are then grouped into nodes and implemented as concrete services in a node according to the required service specification. The implementation may also be done automatically by employing an embedded code generation tool.

The bottom-up approach will involve existing implementations of nodes that are used as legacy systems within the application. In this case it is necessary to verify that the services of the nodes comply to the application requirements. Since both, the application requirements and the service properties of a node can be described formally in an XML document, there is an inherent support for automated tools that verify the implementation to its specification.

The presented case study has shown a simple application of the modeling approach. Further steps will include cluster emulation for more complex applications.

In the future we plan to extend the node and service XML descriptions by tags describing power consumption, weight, and dependability properties. Thus, it will be possible to add constraints on these properties for the application specification of an embedded system.

## 9 Acknowledgments

We would like to thank our colleagues Wilfried Steiner, Christian Trödhandl and Ingomar Wenzl for their comments on earlier versions of this work. This work was supported in part by the Hochschuljubiläumsstiftung der Stadt Wien via project CoMa (H-965/2002) and by DOC [DOKTORANDENPROGRAMM DER ÖSTERREICHISCHEN AKADEMIE DER WISSENSCHAFTEN].

## References

[1] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri. A component based services architecture for building distributed applications. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, pages 51–62, August 2000.

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

[3] 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, March 2003.

[4] H. Fierz. The CIP method: component- and model-based construction of embedded systems. In *Proceedings of the 7th European Engineering Conference held jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 375–392. Springer-Verlag, 1999.

[5] J. Gait. A probe effect in concurrent programs. *Software Practice and Experience*, 16(3):225–233, March 1986.

[6] D.K. Hammer and M.R.V. Chaudron. Component-based software engineering for resource-constraint systems: What are the needs? In *Sixth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'01)*, pages 91–96, January 2001.

[7] R. Hamouche, B. Miramond, and B. Djafri. ModelJ: Component-based modeling for embedded systems. In *European Conference on Object Oriented Programming (ECOOP'01)*, June 2001.

[8] H. Hansson, H. Lawson, O. Bridal, C. Eriksson, S. Larsson, H. Lön, and M. Strömberg. BASEMENT: An architecture and methodology for distributed automotive real-time systems. *IEEE Transactions on Computers*, 46(9):1013–1027, September 1997.

[9] T. A. Henzinger, C. M. Kirsch, M. A. A. Sanvido, and W. Pree. From control models to real-time code using Giotto. *IEEE Control Systems Magazine*, 23(1):50–64, 2003.

[10] R. Johnson, K. Lee, J. Wiczer, and S. Woods. A standard smart transducer interface - IEEE 1451. Presentation held at the Sensors Expo, Philadelphia, October 2001.

[11] C. Jones, M.-O. Killijian, H. Kopetz, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and V. Issarny. Final version of the DSoS conceptual model. *DSoS Project (IST-1999-11585) Deliverable CSDA1*, October 2002. Available as Research Report 54/2002 at http://www.vmars.tuwien.ac.at.

[12] H. Kopetz. Composability in the time-triggered architecture. *SAE World Congress 2000, Detroit, Michigan, USA*, March 2000.

[13] H. Kopetz, M. Holzmann, and W. Elmenreich. A universal smart transducer interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, March 2001.

[14] Object Management Group (OMG). *Smart Transducers Interface V1.0*, January 2003. Specification available at http://doc.omg.org/formal/2003-01-01 as document ptc/2002-10-02.

[15] 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*, volume 1, pages 407–414, Lisbon, Portugal, September 2003.

[16] M. Schlager. A simulation architecture for time-triggered transducer networks. In *Proceedings of the First Workshop on Intelligent Solutions for Embedded Systems (WISES'03)*, pages 39–49, Vienna, Austria, June 2003.

[17] World Wide Web Consortium (W3C). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, November 2003. (W3C Working Draft 10 November 2003).