Component @ Work:
Component Technology for Embedded Systems

PECOS
Pervasive Component Systems
EC Project - IST Program
10/2000 - 10/2002

Christian.Stich
peter.o.mueller
christian.zeidler
@de.ABB.com
Goal of PECOS

Build field devices from one common component library.

Component Library
- Profibus Function Blocks
- FF Function Blocks
- Asset Management
- Alarm handling
- Parameter storage
- Display
- PID
- Process application
- ...

Field Devices
- motor controller
- flow meter
- temperature sensor
- pressure sensor
- actuator
- valve
- ...

ABB Corporate Research
© ABB Corporate Research. All rights reserved  Chris Stich-09/2001-2
Industrial automation

- ABB Industrial automation
  - Portfolio: controls, instrumentation, motors, drives power electronics, robotics and flexible automation, marine and turbocharger systems.

- Instrumentation
  - Sensor, Actuator level between process and control
E.G. Fieldbus devices BUI

Profibus DP
- Flow

Profibus PA, FF
- Temperature
- Pressure
- Flow

Electric Actuators

Controllers
- Positioner
Field Device in Process Automation

The transmission of the process data is effected cyclically, while alarms, parameters and diagnostic data also have to be transmitted acyclically if necessary.

Intelligent field devices: Logic could be here, here or here!

Direct communication

Process

Operation
(Process Control, Visualization)

Engineering
(Configuration, Maintenance, Service)

Ethernet

PCS
(Process Control System)

Fieldbus

Segment Coupler

Field devices

Control Algorithm

Control Algorithm

Control Algorithm

Actuator
Transmitter

DP PA
Typical plant for automation

“Beer Brewery”

- 48000-70000 bottles/h
  - 50ms / bottle
- capacity use 90%
  - Requires high reliability
- expansion 70m
  - Fieldbus PROFIBUS DP
- 22 sub controls
  - 600 I/O’s
- 4000 Transmitter/Sensors and Actuators
- Real Time requirement 10..100ms
Field devices today

- monolithic software
- double development
- no reuse at all
- high development and maintenance costs
- know how is mostly hidden in and dependent from developer

Development cost

- HW 50%
- SW 50%
Field devices today

Today:

A lot of common functionality, but no common architecture and implementation.
Typical field device design

- 16 bit MCU
- 256K ROM, 20K RAM
- Instruction execution time 62.5ns@16MHz
- Power consumption 18mW@7MHz
- Implementation language is usually C and assembler
- Static and monolithic software configuration
- 6000 to 30000 ELOC (without fieldbus)
- RTOS
- Low power design
Why low power design?

- Power supply limitation by 2-wire fieldbus (Ex area intrinsic safety -> 10mA/device)
- Heat limitation
- Size limitation
- Costs (chip, supply environment)
Field device software requirements

- limited memory (ROM, RAM, NVOL)
- limited cpu load
- parts with real time execution
  - time constraints - worst case execution time (WCET)
  - simultaneous execution of tasks
- priority execution of tasks (deadline)
- high reliability / robustness
- communication to third party electronic devices
PECOS case study “Electric actuator”

Electric actuators are for the operation of final control elements (valves). During continuous positioning the electronic varies the motor torque until a force balance between actuator and final control element exists.
PECOS case study “Attributes”

- Voltage supply AC 115 V or AC 230 V
- 16 Bit MCU
- Torque dependent switch off
- Robust gearing with high mechanical efficiency
- Microprocessor-controlled power electronics located in actuator or in separate housing. Integrated sensors for position and temperature
- Additional functions, e.g. process controller, maintenance microprocessor, programmable output characteristic
- Fieldbus: Profibus DP
- Local communication RS232 or Infrared
Architecture of a Field Device (1st attempt)

Block Container
- Scheduler

AO Function Block
- SP
- MODE
- READ BACK
- AO Algorithm
- OUT
- OUT_CHAN
- IN_CHAN

Transducer Block
- Pos VALUE
- CTRL_PAR
- FEEDBACK
- Pos Measure
- I/O HW

Profibus PA Mapper

AccessManager

Profibus PA Stack

Local Operation

Persistent Storage

Display, Buttons

PC/Handheld-Interface
Block and Block Container

Block
- most natural component
- analog output
  - automation function
- transducer block
  - parameter
    - position
    - state
  - functions
    - measurement
    - control

Container
- provides run-time environment
- scheduling
- parameter transfer between blocks
- interfacing to bus stack
  - Profibus
  - FF
- access policy
Stack Adaptation to Block Container

- mapping of fieldbus specific addressing schemas to Block and Parameter schema internally used

- translation of cyclic and acyclic fieldbus services into read/write accesses to Block Parameters managed by the Block Container

- handling of device management (directory objects)

- handling of link objects, i.e. configuration of the inter-Function Block communication at the Block Container level
Block Types

Passive:
- cycle time or timestamps within a cycle

Active:
- task priority
- cycle time or timestamps within a cycle
- Developer must make sure that the execute method synchronizes access to signal bus with its thread (create local copies of signals used from the thread). In the execute method these local signals are updated with the global signals
Typical software components

- **Fieldbus**
  - Stack, Function blocks, Transducer blocks

- **Persistence**
  - FLASH, EEPROM

- **Local user interface**
  - Local bus system (RS232, IrDa, Bluetooth)
  - LCD/Key Pad

- **Diagnosis (AOFT)**

- **I/O** (0..10V, 0..20mA)

- **Algorithm** (FFT, Approximations, etc.)
## Components and Characteristics

<table>
<thead>
<tr>
<th>Component</th>
<th>Reusability</th>
<th>Variation Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Container</td>
<td>Through all field devices</td>
<td>different Scheduler, Access Manager and Fieldbus Mapper possible for adapting to field bus</td>
</tr>
<tr>
<td>Block (concept and interface)</td>
<td>Through all field devices</td>
<td>parameters and their configuration, sub-blocks</td>
</tr>
<tr>
<td>Function Blocks</td>
<td>Through all field devices</td>
<td>none, most of them are predefined by fieldbus specs</td>
</tr>
<tr>
<td>Transducer Block</td>
<td>only within one device family (profile)</td>
<td>Implementation of transducer block algorithm composed of sub-blocks. Same transducer block algorithm composed of sub-blocks can be packed into FF and PA Transducer Blocks having different interface in terms of parameters.</td>
</tr>
<tr>
<td>Local Operation</td>
<td>through all field devices</td>
<td>different implementations (e.g. display/buttons, infrared, bluetooth)</td>
</tr>
<tr>
<td>Persistent Storage</td>
<td>through all field devices</td>
<td>different implementations possible for EEPROM or Flash-PROM</td>
</tr>
<tr>
<td>Fieldbus Stack and Fieldbus Mapper</td>
<td>through all devices of one bus type</td>
<td>none beside stack configuration (will be a third part component)</td>
</tr>
</tbody>
</table>
Signalbus Architecture

Object-Manager

Access Methods
- checking permission

PA-Mapper

AO

Channel

Transducer Block 1

Transducer Block 2

out_channel
in_channel
out
sp
readback
tb1_positioning_value
tb1_reaback_value
tb2_positioning_value
tb2_reaback_value

Signal Bus

FbSignal consists of:
- access property: rw, r
- location in mem: RAM, ROM, NV
- Datatype
- index
- default value

- FbSignals are all available on the signal bus.
- Object Manager owns the signals. Implementation of a block has private members that references a signal on the signal bus.
- Blocks like AO... can access all signals
- PAWrapper or Local Operator Interface (others) are restricted by the access property (r or rw)
Component technology for embedded systems.

Component Model
- how to specify components and architectures including resource constraints (memory consumption, real-time execution)

Component-based Architecture for field devices
- specification of architecture and reusable components for field devices

Component Repository
- storage and retrieval of components during analysis, design, and implementation
- supports the reuse of components

Composition environment
- build applications and components from other components
- check composition rules

Run-time environment
- C++/C-based for low-end devices
- Java based run-time
Component Model

- addresses non-functional properties and constraints such as worst-case execution time and memory consumption

- allows to specify efficient functional interfaces (e.g. procedural interfaces)

- allows to specify architectural styles that describe components connections and containment relations

- allows for code generation and controlled component adaptation when architectural styles are applied to components (source language or generative components)
Component-based Architecture

- a framework for field devices that is expressed as standard interfaces, components, and architectural styles

- is based on field bus architecture

- express compile-time optimization abilities, which could be applied during target code preparation
Repository

- storage and retrieval of components during analysis, design, implementation, and composition

- stores components and architectural styles according to the component model including interface descriptions, non-functional properties, implementation (potentially for different micro controllers), support scripts for composition environment, test cases

- supports component versioning
Composition Environment

- supports composition techniques (visual or script based)

- checks composition rules attached to architectural styles in order to verify that a component configurations meets their constraints

- performs component adaptation and code generation for the application

- supports definition of composition rules, which in an subsequent step could be compiled to architectural styles description
Run-time Environment

- provides an efficient implementation model for components
- addressing the constraints for field devices: low available memory, implementation possibly necessary in C or optimized C++
- supports the approach to compile a component-based design into a optimized firmware for the embedded device, thus having no run-time environment beside the RTOS
- allows for a hardware and RTOS independent implementation of components (e.g. by an RTOS abstraction layer)
Links

- http://www.pecos-project.org
- http://www.abb.com
- http://www.ilogix.com/fs_prod.htm
- Profibus homepage, www.pno.org