UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ARQUITETURA REUTILIZÁVEL DE HARDWARE E SOFTWARE PARA SUPERVISÃO E CONTROLE REMOTOS DE SISTEMAS DE AUTOMAÇÃO INDUSTRIAL

VICTOR ENRIQUE LAURIA VALENZUELA

Manaus
2013

UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

VICTOR ENRIQUE LAURIA VALENZUELA

ARQUITETURA REUTILIZÁVEL DE HARDWARE E SOFTWARE PARA SUPERVISÃO E CONTROLE REMOTOS DE SISTEMAS DE AUTOMAÇÃO INDUSTRIAL

Dissertação apresentada para o curso de Mestrado em Engenharia Elétrica do Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal do Amazonas

Orientador: Prof. Dr. –Ing. Vicente Ferreira de Lucena Júnior
Co-Orientador: Prof. Dr. –Ing. Nasser Jazdi

Manaus
2013

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Valenzuela, Victor Enrique Lauria
V161a Arquitetura de Hardware e Software para Supervisão e Controle Remotos de Sistemas de Automação Industrial / Victor Enrique Lauria Valenzuela. 2013
123 f.: il. color; 30 cm.

Orientador: Vicente Ferreira de Lucena Júnior
Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Amazonas.

1. protocolos de automação industrial. 2. tecnologias web. 3. supervisão remota. 4. automação industrial. I. Lucena Júnior, Vicente Ferreira de II. Universidade Federal do Amazonas III. Título

VICTOR ENRIQUE LAURIA VALENZUELA

ARQUITETURA REUTILIZÁVEL DE HARDWARE E SOFTWARE PARA SUPERVISÃO E CONTROLE REMOTOS DE SISTEMAS DE AUTOMAÇÃO INDUSTRIAL

Dissertação de Mestrado em Engenharia Elétrica apresentada para a obtenção do título de M.Sc através do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas.

Banca examinadora

Prof. Dr. –Ing. Vicente Ferreira de Lucena Júnior
Universidade Federal do Amazonas – UFAM

Prof. Dr. Lucas Carvalho Cordeiro
Universidade Federal do Amazonas – UFAM

Prof. Dr. Raimundo da Silva Barreto
Universidade Federal do Amazonas – UFAM

Resumo

Com o aumento das demandas de mercado a nível mundial, em resposta ao crescimento econômico, os sistemas de automação industrial tornaram-se maiores e mais complexos a fim de atendê-las, o que também aumentou o número de falhas e a necessidade de manutenção. No entanto, as soluções em automação muitas vezes envolvem configurações proprietárias dos fabricantes de equipamentos industriais, prevenindo sistemas de automação de serem reutilizados em diferentes aplicações industriais sem grandes esforços de desenvolvimento. Então, sistemas reutilizáveis em um grande número de aplicações industriais diferentes são desejados por muitas empresas, visto que se podem reduzir os custos com o desenvolvimento, manutenção e treinamento, ainda mais se também permitem coleta remota de informação sobre seu estado de funcionamento.

O presente trabalho apresenta uma arquitetura reutilizável, composta de *hardware* e de *software*, para monitorar e controlar os sistemas de automação. Esta, por sua vez, não se limita a uma única aplicação, podendo ser ligada a outras, desde que uma porta de acesso ao protocolo de automação esteja disponível. Isto é alcançado através da utilização de tecnologias padronizadas nas interfaces entre os componentes da arquitetura, bem como definindo as estruturas para o software em execução nos mesmos. Um protótipo foi desenvolvido com base na arquitetura proposta e seus casos de aplicação são apresentados em uma máquina industrial de café, ativada por voz e com protocolo CANOpen, bem como em uma rede de sensores e atuadores sem fio, com dispositivos ZigBee. Finalmente, o processo de implantação é descrito em um caso de aplicação generalizado.

Palavras-chave:
Protocolos de automação industrial, tecnologias web, supervisão remota, automação industrial.

Abstract

As market demands increased worldwide in response to economic growth, industrial automation systems became larger and more complex in order to meet these demands. This also increased the number of failures and maintenance necessity. However, solutions in automation often involve proprietary setups from manufacturers of industrial devices, preventing automation systems to be reused in different industrial applications without large efforts of development. Thus, systems that can be applied to a large number of industrial applications are desired by many companies, since they can reduce costs with deployment, maintenance and training, and even more if these systems also enable remote collection of information from their operation status.

This work proposes a reusable hardware and software architecture to monitor and control industrial automation systems, which is not limited to a single industrial application; instead, it can be connected to any other one, as long as a communication port is available to access the process automation protocol. This is achieved by utilizing standardized technologies in the interfaces between components of the architecture, as well as by defining the structures of the software running on the hardware devices. A prototype was developed based on the proposed architecture and its application cases are later shown in a voice-activated coffee dispenser with a CANOpen protocol and in a wireless network of sensors and actuators with ZigBee-enabled devices. Finally, the deployment procedures of the prototype are described in a generalized application case.

Keywords:

Process automation protocols, web technologies, remote supervision, automation systems.

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

ABS	Anti-lock Braking System
ADC	Analog-Digital Converter
ANSI	American National Standards Institute
API	Application Programming Interface
ARM	Advanced RISC Machine
CAN	Controller Area Network
CANOpen	Application-layer protocol for CAN network
CIM	Computer-Integrated Manufacturing
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
DAC	Digital-Analog Converter
EMC	Electromagnetic
ESD	Electrostatic Discharge
FCS	Flight Control System
FTP	File Transfer Protocol
GPIO	General Purpose Input and Output
GPS	Global Positioning System
GUI	Graphic User Interface
HMI	Human-Machine-Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAS	Institut für Automatisierungs- und Softwaretechnik
IC	Integrated Circuit
ID	Identifier
IEEE	Institute for Electrical and Electronics Engineering
I²C	Inter-Integrated Circuit

IPC	Industrial Personal Computer
ISA	International Society of Automation
JSON	JavaScript Object Notation
LED	Light Emitting Diode
LAN	Local Area Network
M2M	Machine to machine
MES	Manufacturing Execution Systems
MESA	Manufacturing Enterprise Solutions Association
MMU	Memory Management Unit
OEM	Original Equipment Manufacturer
OPC	Object Linking and Embedding for Process Control
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
PHY	Physical layer
PLC	Programmable Logic Controller
RAM	Random Access Memory
REST	Representational State Transfer
REPAC	Ready, Execute, Process, Analyze, and Coordinate
RISC	Reduced Instruction Set Computer
RF	Radio Frequency
ROM	Read-Only Memory
RPC	Remote Procedure Call
RTC	Real-Time Clock
SCADA	Supervisory Control and Data Acquisition
SCOR	Supply-Chain Operations Reference
SDK	Software Development Kit
SMTP	Simple Mail Transfer Protocol
SPI	Serial Peripheral Interface
SoC	System-on-a-Chip
SODIMM	Small Outline Dual In-line Memory Module
SOAP	Simple Object Access Protocol
UART	Universal Asynchronous Receiver/Transmitter

URI	Uniform Resource Identifier
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSD	Web Service Description
WSDL	Web Service Description Language
XML	Extended Markup Language
XML Infoset	Extended Markup Language Information Set

Lista de Termos

CAN	It is a vehicle bus standard developed by Bosch. It is a physical layer protocol that is widely used on the automation world.
CANOpen	It is a communication protocol based on the CAN Application Layer. It defines communication profiles and mechanisms for exchanging data between CANOpen devices.
Embedded Systems	They are less powerful computers designed to specific solutions, which involve mainly controlling techniques. These systems are found in innumerous devices.
Institute for Electrical and Electronics Engineering	It is an association dedicated to promoting technology advancement through publications, standardization, and education in the fields of Electrical and Electronics Engineering and Computer Science.
Industrial Automation System	A system that automates a technical process. For example, a plastic injection machine or an automatic insertion machine.
Microcontroller	It is a small IC that has a CPU, RAM memory, Programming memory and GPIOs, but has very limited processing when compared to a normal PC.
OEM	It is designated to a company that manufactures and sells products or components that are retailed under the purchasing company's brand. This is common in the automotive industry.
Operating Systems	They are a set of software tools that run on a computer, manage hardware resources and provide services for programs to run on the PC.

Real-time	It is said of hardware and software systems that are defined to operate using time constraints. These systems define that a response from an execution of a task will always be available at after the same time.
RESTlet	It is an open-source REST framework for the Java programming language.
Time-critical	Said of systems that have operations that must be controlled using real-time practices. For example an ABS brakes or nuclear power plants systems.
Wi-Fi Alliance	It is a trade association responsible for promoting the Wi-Fi technology and for certifying Wi-Fi products.
World Wide Web	It is a system of interconnected hypertext documents that accessed via Internet and visualized on a Web Browser.
World Wide Web Consortium	It is an international organization responsible for providing standards for the World Wide Web.
ZigBee	It is a specification for communication protocols to create WPAN of small, low-power radios. It is based on the IEEE 802.15 standard to provide mesh networking.

Sumário

Resumo estendido

Este capítulo tem como objetivo apresentar os aspectos importantes deste trabalho de forma resumida e em língua portuguesa, visto que este trabalho foi originalmente escrito em língua inglesa para ser apresentado no Instituto de Automação Industrial e Engenharia de Software (IAS), localizado na cidade de Stuttgart, Alemanha. No seu formato original, este trabalho possui sete capítulos, os quais descrevem o problema, a motivação, os fundamentos teóricos necessários, os trabalhos relacionados, o conceito proposto, a realização do protótipo e a avaliação de resultados obtidos. Este conteúdo será explicado nesta seção de forma sintetizada, logo, o capítulo equivalente no texto original deverá ser consultado para maiores detalhes. Esta seção é dividida em sete seções, as quais são as seguintes: introdução, fundamentos teóricos, trabalhos relacionados, conceito da arquitetura, solução proposta, casos de aplicação e conclusão.

Introdução

Controle e Automação é um campo da engenharia responsável por pesquisar métodos e mecanismos capazes de tornar automáticos e eficientes os processos técnicos industriais por meio da redução da interferência humana, ou capazes de aumentar a precisão de ferramentas usadas nesses processos. Os sistemas que automatizam processos técnicos, chamados sistemas de automação, são compostos basicamente de dispositivos eletrônicos e mecânicos, os quais são capazes de realizar tarefas de forma mais precisa e mais rápida que humanos (IAS, 2010a).

Em resposta às demandas de mercado, as plantas industriais e processos técnicos tornaram-se maiores e mais complexos, o que consolidou a presença dos sistemas de automação na indústria moderna. Entretanto, esse crescimento criou demanda de gerenciamento de informação em diferentes níveis das fábricas com o objetivo de melhorar a eficiência operacional da planta industrial. O conceito de *Computer Integrated Manufacturing* (CIM) dos anos 70 foi a primeira tentativa de alcançar o objetivo de padronização do gerenciamento de todos os departamentos das fábricas, desde a camada administrativa até as linhas de produção (SAUTER, 2011, p. 36). Posteriormente surgiram novos modelos, tais como *Supply-Chain Operations Reference* (SCOR) desenvolvido pelo Supply-Chain Council, o *Manufacturing Execution Systems* desenvolvido por *Manufacturing Enterprise Solutions Assostiation*

International (MESA International) e o *Ready, Execute, Process, Analyze and Coordinate* (REPAC) da AMR Research, para definir aplicações de manufatura a partir de uma perspectiva funcional (UNGER, 2001, p. 46).

Esses modelos são usados pelo padrão ANSI/ISA-95 para definir uma arquitetura para automação fabril de cinco níveis, mostrada no exemplo da Figura 0.1. Cada nível representa parte do chão de fábrica: no nível 1, encontra-se o processo técnico; no nível 2, encontram-se os equipamentos de controle de processo; no nível 3, a supervisão de processo fornece mecanismos para monitorar parâmetros do processo; no nível 4, encontra-se a supervisão de produto, a qual monitora a produção a partir da perspectiva do produto final; e no nível 5 encontra-se o gerenciamento da planta como um todo.

A estrutura piramidal apresentada na Figura 0.1 foi empregada em várias plantas industriais, o que implicou aumento da complexidade do ambiente fabril. Consequentemente, sistemas de automação mais complexos começaram a surgir, requisitando mais supervisão da operação das máquinas e causando aumento no número de dispositivos para controle. Isso causou também aumento no número de falhas e requisitos de manutenção para estes sistemas. Assim, engenheiros e técnicos devem estar preparados constantemente para reagir a situações planejadas e não-planejadas, de forma a manter esses sistemas em funcionamento. Entretanto, tais reações requerem mecanismos que garantam acesso rápido à informação confiável sobre o sistema de automação, assim como disponibilizar intervenção rápida no processo técnico caso necessário.

A construção desses sistemas de automação complexos, e também de suas ferramentas de supervisão, geralmente envolve o uso de soluções proprietárias pré-definidas de fabricantes de dispositivos industriais. Essas soluções operam apenas com dispositivos pré-determinados de Interface Homem-Máquina (HMI), sensores, atuadores, Controladores Lógicos Programáveis (PLC) e protocolos de comunicação. Mesmo que essa abordagem garanta controle robusto e confiável de processos técnicos, não é possível transferir o sistema de automação e mecanismos de supervisão para outras aplicações industriais. Esse procedimento requer nova especificação completa do projeto para habilitar a supervisão em outros casos de aplicação, visto que pouca, ou nenhuma, padronização é aplicada no desenvolvimento de sistemas de automação industrial. Tais características definem os sistemas de automação e ferramentas de supervisão como soluções industriais altamente especializadas e dependentes do fabricante, o que implica na redução da flexibilidade, da modularidade e da compatibilidade entre dispositivos de automação e níveis da planta fabril.

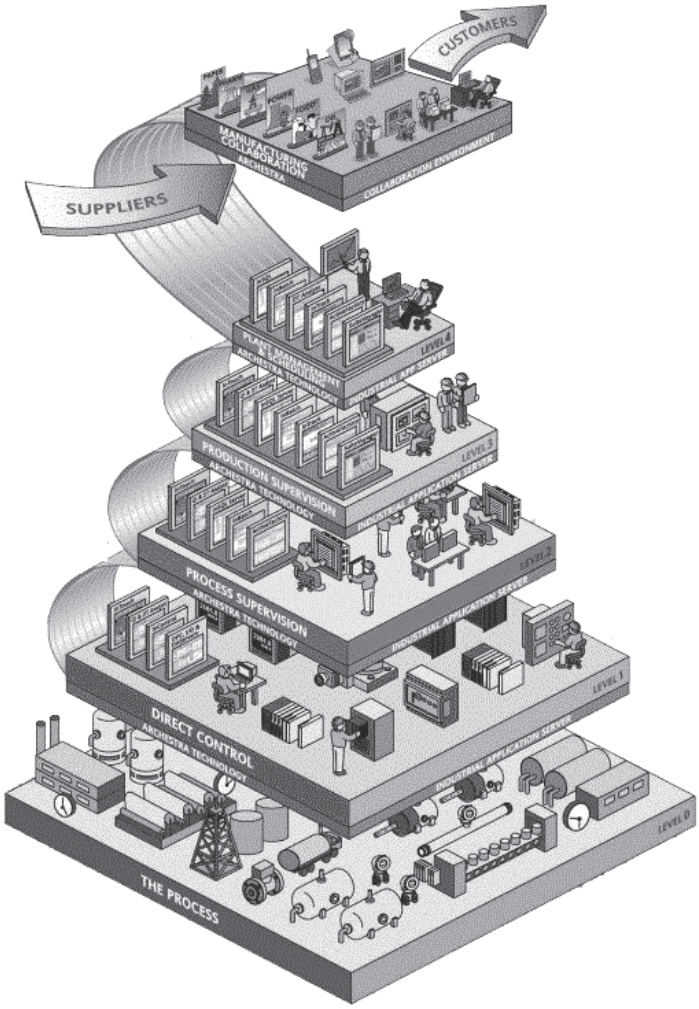


Figura 0.1: Os níveis de automação fabril do padrão ANSI/ISA-95, usando a tecnologia ArchestrA (WONDERWARE).

Atualmente, as redes industriais compatíveis com ANSI/ISA-95 possuem a infraestrutura apresentada na Figura 0.1, porém as melhorias citadas anteriormente não são alcançadas com essa organização estrutural atual. Por isso, uma abordagem para aumentar flexibilidade, modularidade e compatibilidade deve adicionar a combinação de *hardware* e *software* à rede industrial, como mostra a Figura 0.2, com o intuito de isolar das redes de supervisão os dispositivos de controle de processo atuais. Esse isolamento traria benefícios para a supervisão e controle, tais como o uso de dispositivos computacionais comuns em vez de dispositivos industriais especializados, além de permitir a inclusão de novas funcionalidades aos sistemas de automação industrial.

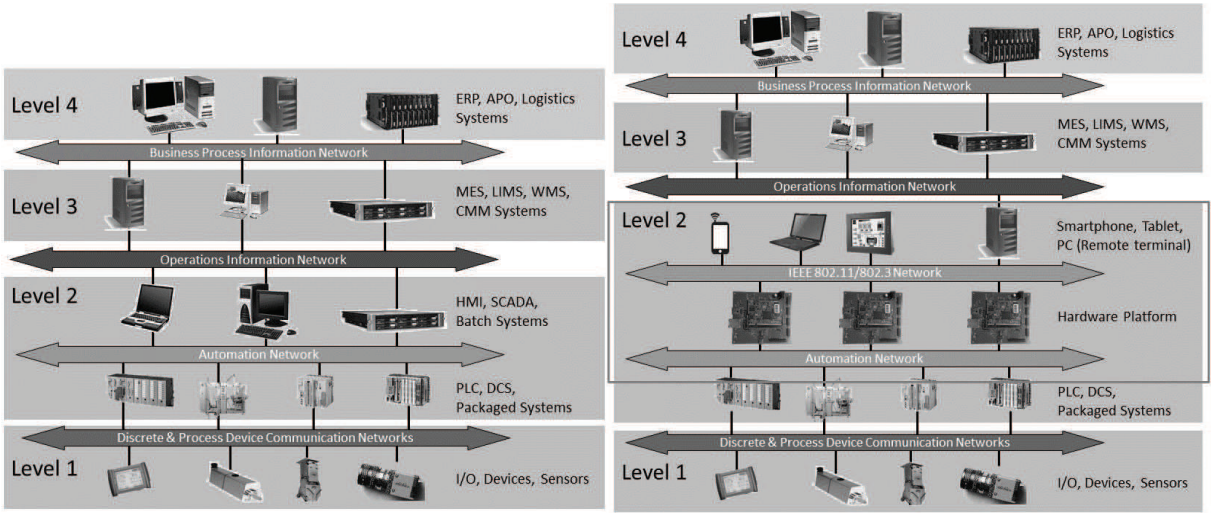


Figura 0.2: Infraestrutura atual (esquerda) e proposta (direita) para redes industriais (BR&L CONSULTING).

Motivação

Usar sistemas de automação flexíveis, modulares e compatíveis na indústria, assim como suas ferramentas de supervisão, permite que fábricas sejam fáceis de operar, projetar, montar e adaptar a novos produtos, usando linhas de produção variáveis (LASTRA, 2008). Sem flexibilidade em fábricas, o modelo hierárquico apresentado na Figura 0.1 torna-se estático e possui problemas para adaptação às mudanças de pedidos de clientes, para a fixação de novos dispositivos de campo e para a reação a casos de exceção nos níveis 0 e 1 (BRATUKHIN, 2010; STARKE, 2013). Também há consenso entre diversos especialistas da indústria sobre o fato de que componentes de campo, tais como sensores, atuadores, dispositivos eletrônicos, células e mesmo linhas de produção inteiras, deveriam ser *Plug & Play*, ou seja, fáceis de configurar (CUCINOTTA *et al*, 2009).

Em um esforço para fornecer acesso remoto aos sistemas de automação mencionados acima, uma arquitetura deve fornecer a base para integrar os níveis 1 e 2, como mostra a Figura 0.2, aumentando a modularidade, a flexibilidade e a compatibilidade das ferramentas de supervisão. Como um resultado, essa integração pode ser usada para elaborar funcionalidades de alto nível, como reconfiguração de software em tempo de execução, diagnóstico remoto, monitoramento remoto e interfaces homem-máquina avançadas.

As estruturas de *hardware* e *software* usadas para fornecer a integração reduzem os esforços de desenvolvimento para conectar ferramentas de supervisão aos sistemas de automação, além de permitir que a conexão de monitoramento e controle seja remota. A estrutura de *hardware* permite acesso aos barramentos de comunicação de vários protocolos de comunicação industrial, o que permite aos técnicos e engenheiros flexibilidade de usar a mesma

plataforma de *hardware* como *gateway* para os dados de sistemas de automação. Em quanto isso, a estrutura de *software* reduz o processo de integração de pilhas de protocolo que não fazem parte dela, mesmo as pilhas de protocolo que sejam proprietárias. Essa mesma estrutura também permite que múltiplos dispositivos conectem-se como terminais de acesso remoto.

Juntas as duas abordagens garantem acesso remoto transparente às informações de processo, ou ao controle de sistemas de automação, por meio do uso de tecnologias padronizadas que sejam independentes de plataforma. Isso oferece flexibilidade a técnicos e engenheiros de usarem quaisquer linguagens de programação e plataformas de desenvolvimento no terminal de acesso remoto. Desta forma, a arquitetura reduz-se a uma plataforma de *hardware* que pode ser conectada a qualquer barramento de comunicação suportado, em conjunto com uma estrutura de *software* modular que permite qualquer dispositivo com capacidade computacional realizar a tarefa de terminal de acesso remoto, como mostra a Figura 0.2.

Um benefício desta arquitetura é o uso de qualquer dispositivo capaz de conectar-se a redes WLAN ou LAN como terminal de acesso remoto, sejam estes PC Industriais (IPC), *Smartphones*, *Tablets* ou *Single Board Computers* (SBC). Além disso, o *software* de supervisão pode usar qualquer tipo de API ou biblioteca para gerar e processar informações relacionadas a sistemas de automação industrial. Neste caso podem ser utilizados lógica fuzzy, algoritmos de redes neurais, processamento de voz ou simples lógica de programação.

Objetivos

Como objetivo primário, este trabalho propõe uma arquitetura para supervisão e controle de sistemas de automação industrial, a qual utiliza tecnologias padronizadas de *hardware* e *software* para aumentar a flexibilidade, a modularidade e a compatibilidade com o intuito de reduzir esforços de integração com diferentes aplicações industriais.

O objetivo acima é alcançado ao completar-se os cinco objetivos específicos determinados por uma análise de solução:

- Analisar diferentes arquiteturas para determinar o melhor arranjo de *hardware* e *software* para supervisão e controle.
- Elaborar uma interface universal para permitir a supervisão e o controle remotos de sistemas de automação industrial. Essa interface deverá permitir acesso transparente para múltiplos sistemas sem priorizar um em detrimento de outro.
- Elaborar módulos de comunicação como interfaces com sistemas de automação industrial, baseados no arranjo escolhido. Esta interface deverá usar uma conexão

comum a todos os módulos para permitir permutabilidade entre eles, e assim, permitir acesso a diferentes protocolos de automação de processo.

- Desenvolver uma plataforma de hardware para comunicar com sistemas de automação industrial, que possua a interface universal e os módulos de comunicação. Esta plataforma deverá reduzir os esforços para permitir supervisão e controle remotos de múltiplos sistemas de automação industrial.
- Experimentar múltiplos casos de uso de controle e monitoramento de sistemas de automação industrial.

Organização

Esta dissertação é organizada em oito capítulos, um em língua portuguesa e os demais em língua inglesa, os quais são apresentados como segue: no capítulo 1 é apresentado um resumo do trabalho em língua portuguesa, contendo apenas os aspectos mais importantes de cada um dos capítulos subsequentes; encontra-se no capítulo 2 a introdução em língua inglesa, onde descreve-se o problema, a motivação e os objetivos; o capítulo 3 apresenta os conceitos de *web services* que são usados em comunicações máquina-a-máquina (*machine-to-machine* ou M2M), mais especificamente SOAP e REST. Também são apresentados neste capítulo os conceitos de alguns dispositivos e tecnologias usados em automação industrial, tais como PLC e *Supervisory Control and Data Acquisition* (SCADA); no capítulo 4, realiza-se a análise dos trabalhos relacionados, utilizando como parâmetros de comparação a flexibilidade, a modularidade e a compatibilidade dos métodos propostos para supervisão e controle de sistemas de automação. Estas informações são, então, comparadas com a solução proposta por este trabalho; o conceito da arquitetura é explicado no capítulo 5 em relação às características que cada componente e interface deverá possuir, o que poderá envolver métodos de comunicação, formatos de dados, estrutura de *software*, todos padronizados, a fim de solucionar o problema.

A realização do conceito de arquitetura é então explicada no capítulo 6 por meio da descrição das tecnologias e dos princípios de funcionamento tanto de *hardware*, quanto de *software*, integrados no protótipo desenvolvido para solucionar o problema; descreve-se no capítulo os experimentos realizados com o protótipo em casos reais. Em cada caso discute-se as estruturas de *software* específicas para a aplicação em questão, assim como os módulos de comunicação utilizados para permitir a comunicação com o sistema de automação industrial. Também é descrito o processo de integração do protótipo a novas aplicações industriais. No fim do capítulo os resultados são discutidos através de uma comparação entre os casos de uso em respeito às modificações necessárias para cada um; por fim, no capítulo 8, este trabalho é

concluído com a apresentação de aspectos importantes e obstáculos encontrados durante o desenvolvimento. Os trabalhos futuros também são apresentados, os quais não descrevem apenas melhorias técnicas para o uso comercial do protótipo, mas também a evolução da arquitetura, como a integração dos níveis 1 e 2 do padrão ISA-95 com serviço de computação em nuvem para gerenciamento flexível de fábricas.

Fundamentação teórica

A fim de compreender os conceitos e as soluções discutidas neste trabalho, é necessário entender como funcionam algumas tecnologias, bem como compreender alguns componentes usados em sistemas de automação industrial. A primeira tecnologia a ser descrita é o *web service*, o qual é um método de comunicação que permite dois dispositivos distintos trocarem informações através de uma rede. Em seguida, descreve-se um ramo dos dispositivos computacionais, chamados de sistemas embarcados, e seu uso como componentes de sistemas de automação industrial. Este tópico aborda principalmente dispositivos utilizados para automatizar os processos técnicos e os métodos de comunicação entre eles.

Web service

A padronização da Web, agregados no chamado Padrão da Internet, permitiu definir interfaces de comunicação por meio de protocolos com sintaxe, semântica e restrições específicas, a fim de coordenar o sequenciamento na troca de mensagens. Isto permitiu esta forma de comunicação ser independente de plataformas e linguagens de programação, garantindo-lhe a classificação como *framework* (WORLD WIDE WEB CONSORTIUM, 2004a; WORLD WIDE WEB CONSORTIUM, 2004b).

Qualquer *software* que implemente essas interfaces possui funções de *web service*, logo, permite a comunicação entre dispositivos por meio de uma rede, mais especificamente, uma rede IP. As mensagens trocadas através desta interface devem ser processáveis pelos dispositivos computacionais e transportadas por meio do protocolo de aplicação HTTP, podendo usar serialização XML, ou outro padrão de formatação. A comunicação opera com uma estrutura cliente-servidor, onde o cliente é responsável por enviar e receber mensagens através dos métodos fornecidos pelas funções de *web service*, enquanto que o servidor fornece os métodos de acesso, além de ser responsável por processar e retornar as informações baseadas na requisição do *software* cliente. Deve-se salientar que a comunicação entre servidor e cliente é possível apenas quando ambos utilizam o mesmo protocolo, como por exemplo o HTTP (WORLD WIDE WEB CONSORTIUM, 2004b).

Há duas principais abordagens para a elaboração de um *web service*, baseados em SOAP ou baseados em REST. O primeiro é a abreviação de *Simple Object Access Protocol,* ou Protocolo Simples de Acesso a Objetos. A comunicação utilizando este protocolo utiliza um formato de encapsulamento orientado à mensagem, ou seja, às informações transmitidas. Por outro lado, REST é a abreviação de *Representational State Transfer*, ou Transferência Representacional de Estado, em que a comunicação é determinada pelo recurso o qual se está compartilhando. As duas abordagens utilizam o protocolo HTTP para transmitir os dados, entretanto, *web services* baseados em SOAP utilizam apenas o XML como tipo de formado de dados, enquanto que aqueles baseados em REST podem utilizar outros formatos, como JSON, arquivos de texto, imagens, dentre outros.

Sistemas embarcados na automação e suas aplicações

A principal definição para o termo "sistema embarcado" diz que são dispositivos que combinam soluções de *hardware* e *software* projetadas para uma finalidade específica. Existem inúmeras aplicações para sistemas embarcados, dentre as quais estão diversos aparelhos eletrônicos (televisores, tocadores de Blu-Ray e MP3, GPS e Smartphones) e também sistemas complexos que exigem respostas em tempo real (freios ABS, sistemas de injeção de combustível, sistemas de controle de voo em aviões).

Grande parte dos dispositivos implantados em sistemas de automação industrial são uma especialização dos sistemas embarcados, concebidos para proporcionar entradas e saídas digitais e/ou analógicas, assim como métodos de comunicação para automatizar processos técnicos, os quais são chamados de Controladores Lógicos Programáveis (PLC). Estes dispositivos eletrônicos executam um *firmware* desenvolvido por seus respectivos fabricantes com apenas três tarefas em repetição: lê-se as entradas (digitais e/ou analógicas), depois executa-se o programa armazenado na memória interna com base na informação lida e, finalmente, atualiza-se as saídas com base na execução do programa (ROSARIO, 2005).

Por conta do uso destes equipamentos em indústrias, os mesmos são construídos para operar continuamente e suportar condições severas de funcionamento, tais como temperaturas elevadas, vibrações e partículas suspensas no ar. Entretanto, antes de serem aplicados na indústria, os CLP precisam ser programados e configurados para realizarem o controle de um sistema de automação industrial. Faz-se os programas com as linguagens padronizadas pela Norma IEC 61131-3: Ladder (LD), Diagrama de Blocos Funcional (*Functional Block Diagram* – FBD), Texto Estruturado (*Structured Text* – ST), Lista de Instruções (*Instruction List* – IL) e Gráfico de Funções Sequenciais (*Sequential Function Charts* – SFC), em ambientes de

programação específicos para cada fabricante. Por fim, gravação do programa desenvolvido é feita através de barramentos como RS-232, RS-485 ou Ethernet, os quais também podem ser usados para comunicação entre diferentes CLP.

Aplicar CLP no controle de processos técnicos grandes e complexos requer a utilização de redes industriais e protocolos de comunicação. A definição destes dois componentes beneficia a comunicação entre os diferentes dispositivos presentes no sistema de automação industrial, pois determinam a resistência a interferência eletromagnética na comunicação gerada por máquinas pesadas e pela rede elétrica, a velocidade com que dados são trocados entre os dispositivos, o gerenciamento da topologia de rede em que são ligados os dispositivos para evitar perda de pacotes e otimizar a comunicação.

Aproveitando-se das redes industriais, os sistemas SCADA são programas de computador projetados para agregar informações sobre o sistema de automação e exibi-las para o usuário através da Interface Homem-Máquina (HMI). Por meio destes programas de *software,* o operador avalia e controla os sistemas de automação (ROSARIO, 2005), o que permite aos departamentos de manutenção avaliarem problemas e tomarem, rapidamente, medidas preventivas, ou corretivas, além de também permitir gerencia da eficiência produtiva da fábrica.

Trabalhos relacionados

A seleção dos trabalhos relacionados ateve-se a modelos e arquiteturas capazes de realizar monitoramento e controle de sistemas de automação industrial remotamente. A solução proposta de cada obra deve permitir a ligação com aplicações SCADA, ou propor a sua própria interface de supervisão e controle. A análise de cada trabalho foi feita considerando melhorias de flexibilidade, modularidade e compatibilidade. Em geral, uma solução é considerada flexível caso possa ser facilmente adaptada a diferentes aplicações industriais, ou seja, com baixos esforços de desenvolvimento. Além disso, a análise também considera as seguintes características:

- Flexibilidade define a tolerância da solução ao uso de diferentes dispositivos para monitoramento e controle do sistema de automação industrial.
- Modularidade define a condição dos componentes de serem independentes uns dos outros, o que permite o uso de diferentes módulos, dispositivos ou adaptadores sem exigir nova construção do sistema.
- Compatibilidade define o foco da comunicação sobre os dados, ao invés do método de acesso, o que possibilita o uso de software diferentes para o mesmo fim.

Tabela 0.1: Tabela de comparação baseada na análise dos parâmetros de flexibilidade, modularidade e compatibilidade.

Trabalho relacionado	Flexibilidade		Modularidade		Compatibilidade
	Multiplos barramentos	Multiplos processos técnicos	Independência de dispositivos remotos	Independência de barramentos de comunicação	Software do terminal remoto diferente
(Hashimukai, 2002)	não mencionado	não mencionado	sim	não mencionado	sim
(Figueiredo & da Costa, 2007)	não	não[3]	sim	não	SCADA e celular J2ME
(Albrecht & Grosse-Plankermann, 2004)	-	sim[1]	sim	não necessário	Web browser
(Ozdemir & Karacor, 2006)	não	não[3]	sim	não	SCADA e celular J2ME
(Albrich, 2011)	RS-485 & Ethernet	não	sim	não	sim
(Antony *et al* , 2011)	-	sim	sim	não necessário	sim
(Truong & Vu, 2012)	não	não	sim	não	sim
(Stopper & Katalinic, 2009)	sim, com server/gateway[2] UA	sim[1]	sim[2]	se disponível no servidor	sim
This work	sim, com adaptador	sim[1]	sim	se pre-compiledo	sim

[1]: depende do sistema de automação a que esteja conectado

[2]: fabricante deve fornecer esta funcionalidade

[3]: requer novo projeto do sistema

A análise mostra a dependência das soluções de supervisão dos processos técnicos aos quais estão conectadas, assim como de comunicações e equipamentos específicos do fabricante. Soluções que são independentes de *software* cliente e permitem a integração com vários programas são mais propensas a serem implantadas em diferentes aplicações, como o caso do OPC UA. No entanto, não há foco em reutilização de *hardware*, ou seja, nos dispositivos que se conectam a sistemas de automação para coletar dados e alimentar programas de gestão de fábrica, além de permitirem o controle remoto dos mesmos. Na Tabela 0.1 é apresentada o resumo da análise de cada trabalho relacionado e a comparação com a solução proposta deste trabalho.

Os resultados de comparação na Tabela 0.1 foram baseados na melhoria das propriedades flexibilidade, modularidade e compatibilidade. Para as melhorias de flexibilidade, foi avaliado se o sistema de supervisão pode desempenhar as suas funções por meio de vários barramentos diferentes, e se o mesmo sistema poderia ser implantado em vários processos técnicos sem a necessidade de reconstruir todos os seus componentes de *software* e *hardware*. Em relação à melhoria modularidade, foi avaliado se o sistema poderia desempenhar suas tarefas independentemente do terminal remoto e do barramento de comunicação conectados ao

sistema. A melhora compatibilidade avaliou a possibilidade de realização de supervisão e controle de diferentes programas de *software*.

Este trabalho propõe então uma solução que se aproxima do OPC UA, mas com mudanças fundamentais para melhorar a reutilização de hardware e, consequentemente, reduzir os esforços de implantação em diferentes aplicações na indústria. Herdam-se os modelos para acesso remoto via HTTP e *web services* empregados pela maioria dos trabalhos relacionados, mas concentra-se nas realizações de desempenho de (PRÜTER, 2009) para fornecer supervisão e controle do sistema de automação industrial remoto.

Conceito da arquitetura

Sabe-se que os elementos básicos que compõem os sistemas de automação industrial são o sistema de controle, o sistema de supervisão (seja SCADA, seja proprietário) e a rede industrial. No entanto, muitos sistemas de automação não possuem supervisão remota, o que demanda a adição de outro dispositivo de hardware para extrair os dados do PLC, IPC ou qualquer dispositivo responsável pelo controle do processo técnico. Assim, é possível simplificar a arquitetura no diagrama mostrado na Figura 0.3, o qual é composto por quatro componentes e duas interfaces. Dentre os componentes estão o usuário, o terminal remoto, a plataforma embarcada e o sistema de automatização industrial, enquanto que as interfaces são o acesso remoto e o barramento de comunicação. O usuário e o sistema de automação industrial não são considerados neste conceito, pois o primeiro é responsável por inserir solicitações no sistema, já o segundo é onde essas solicitações serão executadas.

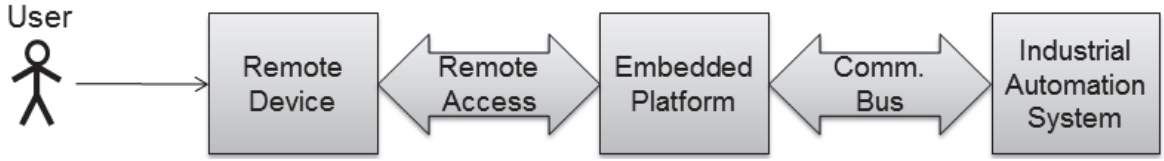


Figura 0.3: Diagrama de componentes e interfaces.

A arquitetura de *hardware* e *software*, então, deve melhorar a flexibilidade, modularidade e compatibilidade, visto que um sistema que a emprega deverá ser reutilizado em diversas aplicações, com qualquer terminal remoto e por meio de qualquer protocolo de comunicação industrial. Isso significa que as funções das interfaces e dos componentes são as mesmas, independente dos equipamentos utilizados terminal remoto, sistema embarcado ou sistema de automação industrial. Desta forma, uma solicitação de pedido de controle, ou monitoramento (diagnóstico), do sistema de automação industrial terá a seguinte sequência: o

usuário interage com a interface do terminal remoto e seleciona uma opção. Os dados que representam o pedido do usuário são enviados para a plataforma embarcada através da interface de acesso remoto. Esses dados são então recebidos pela plataforma embarcada e encaminhados para o sistema de automação industrial utilizando o barramento de comunicação. Por fim, todos os dados recebidos do sistema de automatização industrial são armazenados na plataforma embarcada para uso posterior.

Terminal remoto

O primeiro componente da arquitetura é o terminal remoto. Este componente busca melhorar a modularidade e a flexibilidade do conceito da arquitetura no processo de controle e monitoramento o sistema de automação industrial. Como função básica, os terminais remotos permitem a interação com o usuário. No entanto, a complexidade deste componente é dependente da disposição de centralizada ou descentralizada dos métodos relativos ao sistema de automação industrial. Se a disposição descentralizada é utilizada, os métodos de controle e diagnóstico do sistema de automação ficam presentes neste componente, enquanto que os métodos para gerenciamento do protocolo de comunicação estão presentes em outro componente.

Os métodos de controle e diagnóstico referem-se aos procedimentos necessários para controlar e monitorar o respectivo sistema de automação, ou seja, a sequência de dados que devem ser trocados, e ao barramento de comunicação pelo qual informação deverá ser enviada. Estes dados são, na verdade, parâmetros relacionados ao protocolo de comunicação do respectivo sistema de automação, necessários para executar corretamente o procedimento desejado, como por exemplo: endereços de memória, endereço no barramento de comunicação, valores para escrever ou ler, assim como as suas dimensões, protocolo usado para se comunicar, tipos de mensagens, dentre outros.

A principal diferença na organização de métodos centralizada está no fato do terminal remoto não possuir somente os métodos de controle de diagnóstico, mas também uma grande parte dos métodos do barramento de comunicação – ainda mais detalhado na seção Plataforma embarcada. Apesar dessa concentração de informação, nem todos os métodos podem ser transferidos para o terminal remoto, visto que parte dele é responsável por controlar os barramentos de comunicação de baixo nível, os quais fazem conversão bidirecional da informação digital para os níveis de elétricos usados pelos protocolos de automação de processo.

O terminal remoto, então, não tem conhecimento acerca da operação do barramento de comunicação. Para permitir que diversos terminais remotos utilizem apenas um meio de comunicação para de enviar dados a outro componente, sem que haja interferência entre eles, necessita-se de uma interface que seja comum a todos dispositivos e que permita o encapsulamento de dados.

Acesso remoto

A interface acesso remoto é uma de duas interfaces que estão presentes neste conceito. Projetou-se esta forma de comunicação entre componentes para melhorar a flexibilidade e compatibilidade do conceito da arquitetura, em que se utiliza protocolos e formatos de dados padronizados para permitir a comunicação entre o terminal remoto e plataforma embarcada.

Para que diferentes terminais remotos possam conectar-se a esta interface, definiu-se que a troca de informações seja feita através de uma estrutura que encapsule os dados dentro de um formato comum, tal que as informações contidas naquela mensagem sejam interpretadas apenas pelas estruturas responsáveis pelo processamento do protocolo de comunicação desejado. Desta forma, os protocolos padronizados garantem o meio de comunicação comum entre terminais remotos e plataforma embarcada, enquanto que a formatação de mensagens padronizada garante que a mensagem seja encaminhada para a respectiva estrutura de processamento do protocolo de comunicação.

Portanto, essa interface permite que as mensagens sejam geradas em qualquer dispositivo, independentemente de arquitetura do processador, sistema operacional e linguagens de programação, mas seu processamento correto está condicionado a algumas restrições impostas pela plataforma embarcada.

Plataforma embarcada

O segundo componente presente neste conceito é a plataforma embarcada. Suas funcionalidades estão diretamente ligadas ao terminal remoto, como definido anteriormente. Dentro da organização descentralizada de métodos, este componente proporciona melhorias na modularidade e na flexibilidade, pois fornece as regras para o formato obrigatório de mensagem, assim como também as regras para o gerenciamento de qualquer recurso fornecido pela pilha de protocolo. Isto significa que a plataforma embarcada é responsável pela execução de qualquer método intrínseco ao protocolo, ou seja, que permaneça inalterado quando o mesmo protocolo de comunicação é usado em diferentes aplicações industriais, como por exemplo: a

composição de pacotes do protocolo, os cálculos de erro (CRC, paridade, *etc*.) para pacotes recebidas e enviadas, recursos de prevenção de colisões de pacotes, se disponível, dentre outros.

Quando a arquitetura é disposta na organização centralizada, estas melhorias são fornecidas pelo terminal remoto, visto que possui a maior parte das características acima listadas. Apenas os métodos relacionados ao controle dos *drivers* do da interface barramento de comunicação estão presentes na plataforma embarcada, juntamente com o conceito de troca de mensagens formatadas.

Os *drivers*, aos quais se faz referência acima, são responsáveis por controlar o acesso físico ao protocolo de comunicação do sistema de automação industrial, e estão presentes independente da organização centralizada e descentralizada.

Barramento de comunicação

A segunda interface é chamada de barramento de comunicação. O propósito desta interface é melhorar as características de modularidade e flexibilidade da arquitetura, independentemente da organização de métodos. Esta interface requer um mecanismo para traduzir informação digital para os níveis de tensão definidos pela especificação do protocolo de comunicação.

Sabe-se que existem diversas maneiras de realizar a comunicação através do protocolo de comunicação, mas melhora-se a modularidade e a flexibilidade ao utilizar barramentos de dados padronizados para acessar o dispositivo responsável pela tradução de dados. Complementa-se os benefícios de modularidade e flexibilidade com o uso de placas de circuito impresso (PCI) permutáveis, as quais possuem um circuito integrado (CI) transceptor responsável pela tradução dos dados digitais. Na perspectiva de *software*, estes CI são acessados pelos *drivers* em execução na plataforma embarcada, e usam barramentos de baixo nível padronizados, tais como RS-232, USB, RS-485, dentre outros. As melhorias de flexibilidade e modularidade desta interface encontram-se no fato de qualquer transceptor poder ser utilizado, desde que o *driver* conecte-se ao CI por meio de um dos barramentos de baixo nível, além de também permitir o intercâmbio de diferentes placas com transceptor, de acordo com a necessidade da aplicação industrial.

Solução proposta

Antes de apresentar a solução para a arquitetura proposta neste trabalho, deve-se ressaltar que as soluções de *hardware* e *software* empregadas na arquitetura utilizam a organização descentralizada, visto que os métodos concentrados no terminal remoto exigiriam

mais de desenvolvimento de *software* para cada nova aplicação do sistema. A centralização também requer elevado tráfego de dados entre os dois componentes para gestão remota de pilhas de protocolo, além de realizar a tarefa através de redes com poucas, ou nenhuma, restrições de tempo real.

A solução para a arquitetura de *hardware* e *software* é definida por sete casos de uso que, em conjunto, são responsáveis por permitir o usuário realizar solicitações de controle e monitoramento, e executar as respectivas solicitações no sistema de automação industrial. Cada solicitação representa uma ação a ser executada pelo sistema de automação industrial, como a leitura de sensores, ou a ativação de uma sequência de atuadores. O usuário também é responsável pelo consumo de informações lidas do sistema de automação industrial, tais como valores de sensores, entradas digitais, etc.

Solução de *software*

A arquitetura de *software* é composta por seis elementos, que juntos contribuem para o aumento da modularidade, flexibilidade e compatibilidade do sistema de supervisão e controle. Estes elementos, e suas respectivas funções, são os seguintes:

- Interface de usuário: identifica a ID e faz a aquisição dos parâmetros específicos da solicitação. É no decorrer da aquisição que se pode encontrar API e *frameworks* externos que processem voz, processem vídeo, ou utilizem outros mecanismos para fornecer os parâmetros da solicitação do usuário.
- Base de conhecimento: gera os procedimentos que determinam a sequência mensagens a serem ser executadas para o cumprimento da respectiva solicitação, construídas com base nos parâmetros fornecidos pelo usuário. Os procedimentos, entretanto, podem ser de envio de dados para o sistema de automação industrial, como também de leituras de dados presentes na plataforma embarcada.
- Cliente *web service*: para envio ao sistema de automação industrial, formata os dados em XML ou JSON – dependendo do servidor – e encapsula essa informação no método POST do protocolo HTTP. Porém, quando se trata de leitura de dados presentes na plataforma embarcada, a informação é encapsulada no método GET do protocolo HTTP.
- Servidor *web service*: utiliza a estrutura REST (*Representational State Transfer*) para oferecer os serviços de acesso aos protocolos de comunicação industrial. Para cada protocolo em funcionamento na plataforma embarcada, cria-se um recurso de acordo com a estrutura REST, ou seja, para conectar um barramento de dados para

o protocolo CANOpen, por exemplo, tem-se um recurso /request/canopen associado ao endereço do servidor (http://ip:porta). Enquanto isso, também existem outros recursos para gerenciar o funcionamento da plataforma embarcada, com o objetivo de verificar processos em execução, disponibilidade de serviços, dentre outros. Dados recebidos por este elemento de *software* são armazenados em memória compartilhada e acessados pela respectiva pilha de protocolo de comunicação.

- Gerenciamento de *drivers*: este elemento de *software* sempre está associado a um protocolo de comunicação instalado na plataforma, pois tem a tarefa de gerenciar o funcionamento do protocolo de comunicação. O gerenciamento realizado neste elemento consiste do tratamento de erros de comunicação, do acesso às mensagens da memória compartilhada e, também, do armazenamento das respostas do sistema de automação para análise posterior. Portanto, sua implementação está intimamente ligada à pilha de protocolo a qual controla.
- Pilha de protocolo: implementa as funções do protocolo de comunicação, tal como determinam suas especificações, ou seja, funções que permitam construir um pacote de mensagens dentro do padrão determinado, gerenciar topologias de rede, calcular os erros no recebimento de mensagens, dentre outras. Neste elemento de *software* também é realizado o acesso à camada física do protocolo de comunicação utilizando uma das interfaces padronizadas e disponíveis na plataforma embarcada.

Solução de *hardware*

A arquitetura de hardware é elaborada com a intenção de cumprir as especificações que não foram cobertas pela solução de *software*. Um protótipo é apresentado para demonstrar a aplicação da arquitetura proposta e realizar a integração do sistema supervisório a um sistema de automação industrial. É importante notar que o protótipo não é a única implementação possível da arquitetura de *hardware*, pois também é possível executar as mesmas tarefas de controle e monitoramento de sistemas de automação em computadores, desde que sejam atendidas as diferenças de linguagem de programação, acesso a *drivers* e/ou portas de comunicação, características específicas do sistema operacional, dentre outras particularidades.

Existem dois componentes de *hardware*, assim como determina o conceito da arquitetura. O primeiro, cujo papel é ser o terminal remoto, pode ser qualquer dispositivo computacional capaz de conectar-se à mesma rede Ethernet ou Wi-Fi em que a plataforma embarcada estiver também conectada. O segundo componente é a plataforma embarcada, cujo

papel é fornecer os serviços descritos na solução de *software*. Divide-se este último em quatro partes responsáveis por algumas funções importantes para a plataforma de *hardware*:

- Conexões remotas: possibilitam o acesso à Ethernet via PHY presente no *System-on-Chip* (SoC), ou às redes Wi-Fi via adaptador conectado a uma porta USB.
- Placas de comunicação: são os módulos de comunicação que conectam o núcleo do processador ao barramento de comunicação do sistema de automação industrial, permitindo que a plataforma embarcada troque pacotes de dados através do protocolo de automação de processos, ou seja, é responsável por fornecer acesso à interface de barramento de comunicação descrita no conceito da arquitetura. Este elemento de *hardware* é a concretização do conceito de placas com transceptor mencionadas no conceito de arquitetura.
- Unidade de processamento: consiste de uma PCI que contém o processador, a memória RAM, a memória não-volátil, Ethernet PHY, cristais osciladores e outros periféricos.
- Placa-mãe: é a PCI responsável pelo fornecimento dos circuitos eletrônicos complementares à unidade de processamento, além das conexões para os adaptadores de comunicação e conexões remotas. Nesta placa é onde encontram-se a fonte de alimentação, conector Ethernet, conector USB, conectores para os adaptadores de comunicação (SPI, I²C e Serial), interface de depuração, entradas e saídas de propósitos gerais (GPIO), cartões de memória e conector para a unidade de processamento.

Casos de uso e resultados

O propósito da arquitetura é o máximo reaproveitamento de componentes *hardware* e *software* possível. Se novas aplicações industriais usam o mesmo protocolo de automação, então apenas os subcomponentes interface do usuário e base de conhecimento serão necessários implementar. Caso contrário, os subcomponentes pilha de protocolo e placa com transceptor deverão também ser implementadas, assim como mostra a Tabela 0.2 para os dois casos de uso em que a solução foi testada.

Por exemplo, considera-se uma nova aplicação industrial que use CANOpen, ou ZigBee. Não seria necessário implementar os subcomponentes pilha de protocolo e placa com transceptor novamente, pois estes já estariam presentes na arquitetura do sistema. Apenas a interface de usuário e a base de conhecimento seriam necessários para controlar e monitorar

remotamente a nova aplicação industrial. Esta funcionalidade está disponível para qualquer protocolo de automação industrial que por ventura fosse adicionado ao sistema usando a arquitetura proposta. Consequentemente, a camada de comunicação é preservada de uma aplicação para a outra, enquanto que apenas o mecanismo de controle, e a interface, são alterados em cada caso.

Tabela 0.2: Tabela de comparação de implementações dos casos de uso.

	Terminal Remoto		Plataforma Embarcada	
	Interface de usuário	Base de conhecimento	Pilha de protocolo	Placa com transceptor
Coffee machine	Dragon Naturally Speaking + jFuzzy	Procedimentos da máquina de café	CanFestival	PCAN-USB
ZigBee network	Sem funcionalidade especial	Procedimentos de Liga/desliga + leitura de sensores	xbee-api	XBEE Pro S1

Usando o exemplo acima, é possível avaliar as métricas de estabelecidas nos objetivos (subseção Objetivos) para que fossem alcançadas pelo sistema desenvolvido. A primeira métrica de avaliação está relacionada à interface universal que permite acesso transparente para múltiplos sistemas sem priorizar um em detrimento do outro. Começando pelos exemplos descritos na Tabela 0.2, pode-se visualizar que mudanças em ambos os casos estão restritas apenas a algumas partes da arquitetura de software, deixando os componentes relacionados ao *web service* - a interface de acesso remota - inalterados, como mostra a Figura 0.4. Além disso, partindo das características descritas nos capítulos 5 e 6, as características padronizadas desta interface permitem a transmissão de múltiplos dados de protocolo de qualquer sistema operacional, desde que esteja encapsulado dentro de uma requisição HTTP e que utilize a estrutura definida. A referida interface atende então às métricas propostas, pois foi construída utilizando as características acima.

A segunda métrica de avaliação está relacionada ao uso de módulos de comunicação através de um conector comum para permitir a permutabilidade entre eles, assim garantindo acesso físico aos múltiplos protocolos de automação de processos. A descrição dessas estruturas de *hardware* – módulos de comunicação e conector comum – são encontrados no apêndice A. Visto que estes módulos de comunicação dependem diretamente do protocolo de automação de

processos ao qual conecta-se, não há necessidade de explicar sua construção. Entretanto, estes módulos devem usar a forma de conexão comum apresentada na Tabela 0.3. Estes conectores possuem os barramentos padronizados I²C, SPI e Serial para fornecer o máximo de opções possíveis para enviar e receber dados dos circuitos integrados transceptores. Ademais, a porta USB também pode ser utilizada para conectar módulos de comunicação caso esteja disponível. Portanto, os conectores e os módulos podem fornecer acesso físico para virtualmente qualquer protocolo de automação de processos, porque é através dos barramentos mencionados acima que os componentes de *software* podem alcançar a camada física dos protocolos de comunicação.

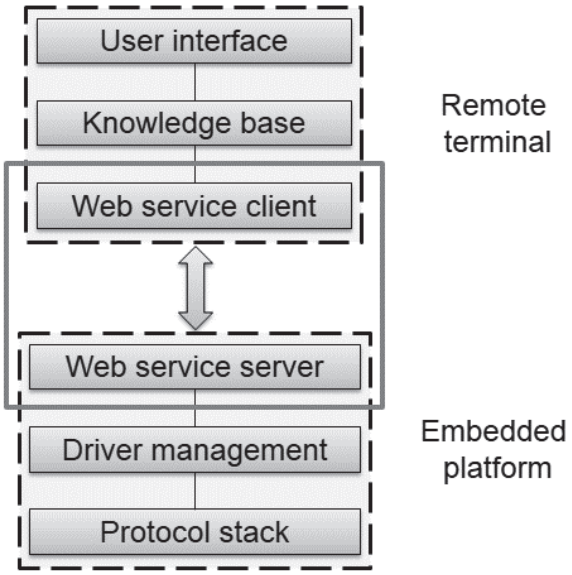


Figura 0.4: Componentes de software com a interface universal destacada.

Tabela 0.3: Configuração de pinos da interface para transceptores.

UART_TX	1	2	UART RX
I²C_SDA	3	4	I²C_SCL
SPI_MOSI	5	6	SPI_MISO
SPI_CLK	7	8	SPI_SSEL
+5V	9	10	GPIO
+3V3	11	12	GND

A última métrica relaciona-se com o desenvolvimento de uma plataforma que una os resultados da primeira e segunda métricas. Isso significa que esta plataforma deve reduzir os esforços de desenvolvimento para permitir o mecanismo proposto para supervisão e controle remotos de múltiplos sistemas de automação industrial. Como descrito no apêndice A e no capítulo 4 a plataforma de *hardware* possui o sistema operacional Linux e conecta-se via

Ethernet ou Wi-Fi às redes -- acesso ao protocolo HTTP -- e também possui conectores em pino -- acesso aos barramentos I²C, SPI, Serial e USB -- assim como mostra a Figura 0.5.

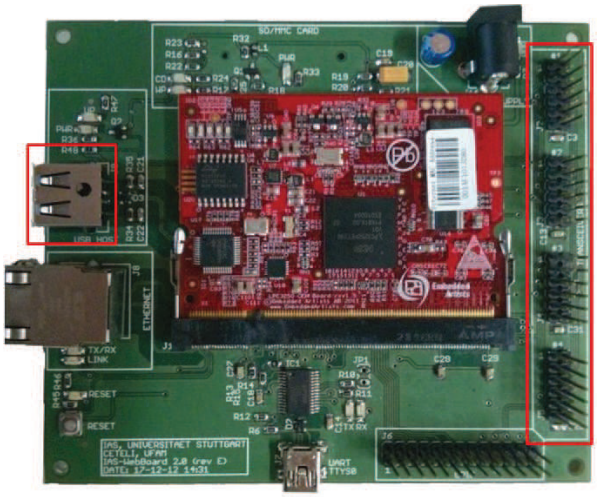


Figura 0.5: Plataforma de hardware com as conexões para módulos de comuniação destacados.

Chapter 1 Introduction

Control and Automation is the engineering field responsible for researching methods and mechanisms that make technical processes automatic and efficient by reducing human interference, or that increase precision of tools used in those processes. The systems that automatize technical processes are called automation systems, and are composed solely of mechanical and electronic devices, being able to perform tasks more accurately and quickly than a human (IAS, 2010a).

In response to increasing market demands, industrial plants and technical processes became larger and more complex, which assured the presence of automation systems in modern industry. Therefore it created the demand for managing information from the various levels of the factory in order to improve the operational efficiency of the industrial plant. The Computer Integrated Manufacturing (CIM) concept in 1970's was the first attempt to achieve the goal of standardizing the management of all factories' departments, from administration to production lines (SAUTER, 2011, p. 36). Later came other models, such as Supply-Chain Operations Reference (SCOR) from the Supply-Chain Council, Manufacturing Execution Systems (MES) from the Manufacturing Enterprise Solutions Associations (MESA) International, and Ready, Execute, Process, Analyze, and Coordinate (REPAC) from AMR Research, to define manufacturing applications from a functional point of view (UNGER, 2001, p. 46).

These models are used by ANSI/ISA-95 standard to define the five-level architecture for factory automation shown in the example of Figure 1.1. Each level represents part of the factory floor: in level 1 is found the process itself; in level 2, the process control equipment is found and is used to ensure proper work of the technical process; in level 3, the process supervision provides mechanisms to monitor parameters of the process; in level 4 is found the production supervision, which monitors production from the perspective of the end product; and, in level 5, the management of the factory plant is found.

The pyramid structure from Figure 1.1 was deployed in many industrial plants, resulting in an overall environment complexity increase. Consequently, more complex automation systems began to appear, requiring more supervision of machine operation and increasing number devices for control. This implied the increase on number of failures and maintenance requirements. Thus, engineers and technicians must be constantly prepared to react fast to both

planned and unplanned situations in order to keep these systems running. However, such reactions require mechanisms that both assure quick access to reliable information about the automation system, as well as to provide fast intervention in the technical process, if needed.

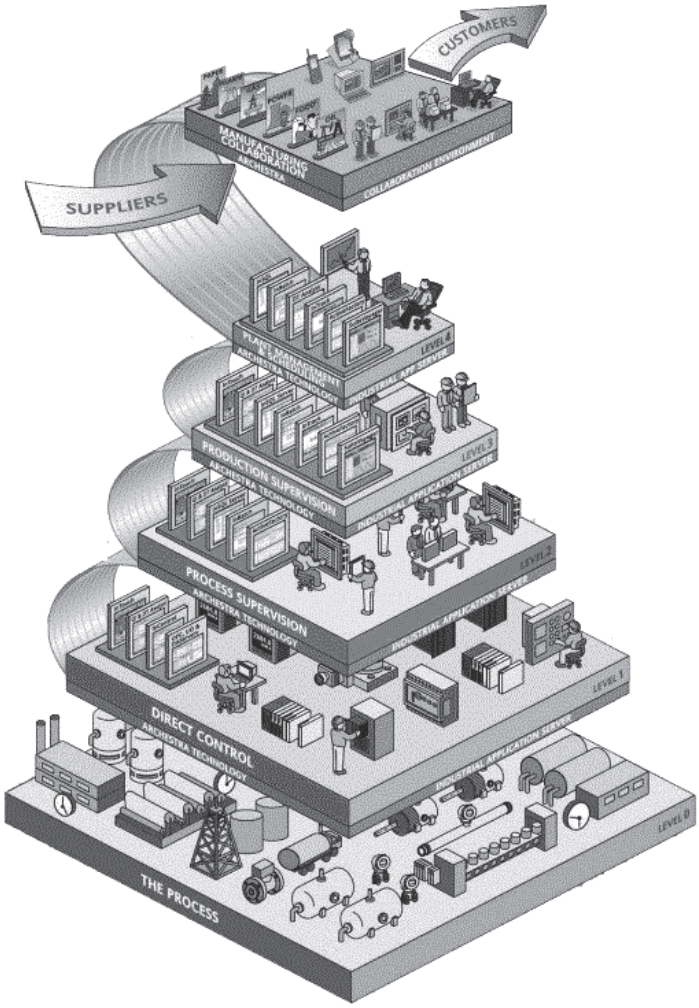


Figure 1.1: The ANSI/ISA-95 standard's levels of factory automation using the ArchestrA technology (WONDERWARE).

The construction of these complex automation systems, and of the supervision tools, often use pre-defined proprietary solutions from the manufacturers of industrial devices. This means the solutions work only with pre-determined Human-Machine Interface (HMI) devices, sensors, actuators, Programmable Logic Controller (PLC) and communication protocols. Even though this approach provides reliable and robust control of technical processes, it does not allow the automation system and its supervision mechanism to be transferred for other industrial applications. This requires a whole new project design to enable supervision in different

application cases, as little to none standardization is applied in the development of industrial automation systems.

These characteristics define automation systems and supervision tools as highly specialized industrial solutions and highly dependent of the manufacturer. The dependability decreases flexibility, modularity and compatibility between automation devices and factory plant levels (WANG, 2002).

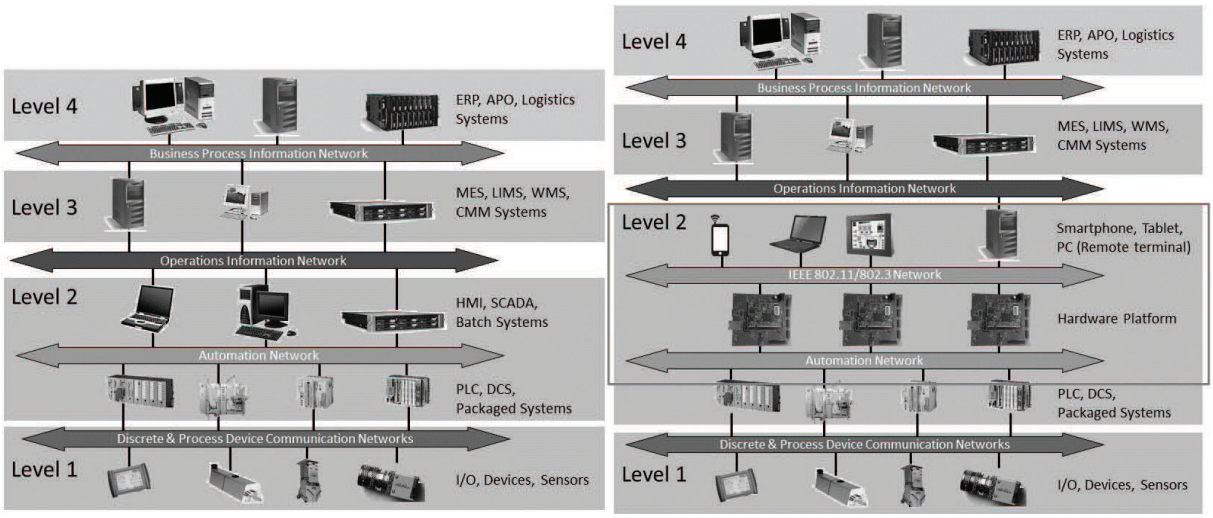


Figure 1.2: Current (left) and proposed (right) IT infrastructure of industrial networks (BR&L CONSULTING).

Currently, industrial networks in compliance with ANSI/ISA-95 have the infrastructure as shown in Figure 1.2. However, the desired improvements are not achieved using the current infrastructure. Therefore, an approach to increase flexibility, modularity and compatibility shall add a combination of hardware and software to the industrial network, as show in Figure 1.2, in order to isolate current process control devices from supervision networks. The isolation would naturally bring benefits such as the use of common computing devices instead of specialized industrial ones, and enable the inclusion of new features to industrial automation systems.

1.1 Motivation

Using flexible, modular and compatible automation system in the industry, and its supervision tools, allow factories to be easier to operate, design, assemble and adapt to variable production lines or to new products (LASTRA, 2008). Without flexibility in the factory plant, the hierarchical model shown in Figure 1 1 becomes static and has issues to adapt to changes of costumer orders, to the placement of newer field devices or to react to exceptions in levels zero and one (BRATUKHIN, 2010; STARKE, 2013). This is also an agreement between many

experts of the industry, which state that field level components, such as sensors, actuators, devices and cells, as well as whole production lines should be Plug & Play or easy to setup (CUCINOTTA et al, 2009).

In an effort to provide remote access to automation systems as mentioned above, an architecture shall provide the basis for integrating levels 1 and 2, as shown in Figure 1 2, by improving modularity, flexibility and compatibility of supervision. As a result, this integration may be used to build higher level features such as software reconfiguration during runtime, remote diagnostics, remote monitoring, and advanced human-machine interfaces.

The hardware and software structures used to provide the integration reduces developing effort to connect supervisory tools to automation systems, and enable it to be remotely monitored and controlled. The hardware structure enables access to communication buses of several industrial communications protocols, which grants technicians and engineers the flexibility to use the same hardware platform as a gateway to automation system data. In the meantime, the software structure reduces the integration process of protocol stacks that are not present inside it, even the proprietary ones. It also enables multiple devices to be connected as remote access terminals.

Together, both approaches ensure seamless remote access to reach out process information or to remotely control automation systems by employing standardized and platform-independent technologies, which give technicians and engineers the flexibility to use any desired programming language and development platform in the remote terminal. Thus, the architecture boils down to a single hardware platform that may be attached to any supported communication bus, and a modular software structure that enables virtually any device with computing capability to perform the task of being a remote terminal, as shown in Figure 1 2.

A benefit of the architecture is the use of any device capable of connecting to WLAN or LAN as remote access terminal, namely Industrial PC (IPC), smartphones, tablets or single board computers. Moreover, the supervision software can use any type of API or library to generate, or process information related to the industrial automation system. In this case, it may be used Fuzzy Logic, Neural Network algorithms, voice processing APIs, or solely plain programming logic.

1.2 Objectives

As a general objective, this work proposes an architecture for supervision and control of industrial automation systems, which uses both standardized hardware and software

technologies to improve modularity, flexibility, and compatibility, reducing integration efforts with in different industrial applications.

- The primary objective is achieved by successfully accomplishing the five specific ones determined by an analysis of the solution:
- Analyze different architectures for better arrangement of software and hardware for supervision and control.
- Elaborate a universal interface to enable remote supervision and control of industrial automation systems. This interface shall enable seamless access for multiple systems, without prioritizing one over another.
- Elaborate communication modules as interfaces to the industrial automation system based on the chosen architecture arrangement. This interface shall use a common connection to all modules in order to allow exchangeability, and thus enabling access to multiple process automation protocols.
- Develop a hardware platform to communicate with industrial automation systems, which possesses the universal interface and the communication modules. This platform shall reduce efforts to enable remote supervision and control for multiple industrial automation systems.
- Experiment with multiple use cases to control and monitor industrial automation systems.

1.3 Organization

This thesis is organized in seven chapters and are presented as follows: in Chapter 2 presents the concepts of web services used in machine to machine communications, more specifically SOAP and REST, as well as the concepts of devices and technologies, such as PLC and SCADA, used in industrial automation; in Chapter 3 is carried out the analysis of the related work based on the flexibility, modularity and compatibility of the proposed methods to supervise and control automation systems, which is then compared to the proposed solution of this work; in Chapter 4 the architecture concept is explained in terms of the characteristics each component and interface shall have, which may involve standardized communication methods, data formats as well as software structures to solve the problem.

The realization of the architecture concept is then explained in Chapter 5 by describing technologies and the working principles of both hardware and software integrated in the prototype developed to solve the problem; in Chapter 6 is described the applications of the

prototype in real life cases. The explanation addresses the application-specific software structures and the hardware modules used to enable communication with the automation system. It also describes the process of integrating the prototype to new industrial applications. At the end of the chapter, the results are discussed through a comparison of each application case in relation to the modifications required for each one; and finally, in Chapter 7, this work is summarized by addressing important aspects and obstacles encountered during the development. Future work is also presented, describing not only technical improvements to enable commercial use of the prototype, but also the evolution of the architecture, which may integrate layers 1 and 2 of the ISA-95 standard with cloud computing services for flexible factory management.

Chapter 2 Background

In order to understand the concepts and solutions discussed in this work, it is necessary to understand how some technologies work, as well as to comprehend some components used in industrial automation systems. The first technology to be described is the web service, which is a communication method that allows two distinct devices to exchange information over a network. Next, a branch of computing devices are going to described, called embedded systems, and their usage as components of industrial automation systems. This topic approaches mainly devices used to automate technical processes and communication methods between them.

2.1 Web services

With the popularization of the Internet, it became common in distributed computing applications to use of services with technical specifications provided by the Web standard (WANG, 2002; SAUTER, 2011; HASHIMUKAI, 2002). These standards, also called Internet Standards, are based on the specifications of the Internet Protocol. Together they define communication interfaces in the form of protocols by specifying syntax, semantics, and constraints for message exchange sequencing. These protocol-based characteristics make these communication interfaces not only platform independent, but also programming language independent, qualifying them as frameworks (WORLD WIDE WEB CONSORTIUM, 2004a; WORLD WIDE WEB CONSORTIUM, 2004b).

Thus, any software that implements these interfaces has the web service function, which enables communications between electronic devices over a network, i.e., over an IP network. Messages exchanged through these interface must be in a machine-processable format, which are usually conveyed using HTTP application layer protocol with XML serialization or other standards (WORLD WIDE WEB CONSORTIUM, 2004b).

These communications work in server-client architecture, in which the client is responsible for sending and receiving messages through the access methods provided by the web service function, whereas the server provides the access methods and is responsible for processing and returning data based on the request made by the client agent. As a requirement to run the access successfully, both client and server must have the same protocol for the

communication, for example, the HTTP application protocol (WORLD WIDE WEB CONSORTIUM, 2004b)

The XML is an extensible, flexible, and standardized data format, and is one of the most used in web services, because of the XML Infoset, XML Schema and XML Namespace specifications. The XML Infoset, above the others, has greater importance because it defines, for each part of an XML document, a set of information items and associated properties, which ensures accurate mechanisms to reference any data inside the document. The XML Namespaces provide means to uniquely name elements and attributes in a XML document. Finally, there is the XML Schema which is a description of a XML document in terms of data types, ensuring that data is read correctly when the XML document is deserialized.

The specification of communication between server and client is described in the WSD (Web Service Description), done by defining message format, data types, transport protocols, and transport serialization formats that should be used to exchange data (WORLD WIDE WEB CONSORTIUM, 2004b). The WSDL is the language responsible for describing the public interface of the web service in terms of messages that are exchanged between server and client, which are described independently of a specific wire format using a type system, typically XML Schema. Each description associates a message exchange pattern to the messages, which identifies the sequence, cardinality, and consumer of messages sent and/or received. This description is made so both client and server must agree in order to exchange messages, i.e., independently of programming language the agents were developed, as long as they agree on the service description.

Currently, there are two major approaches for creating a web service, using SOAP-based web services or REST-based web services. SOAP stands for Simple Object Access Protocol and it is a message-oriented envelope format. On the contrary, REST stands for Representational State Transfer and it is developed to be resource-oriented.

2.1.1 SOAP

As mentioned before, SOAP is an envelope format that uses the HTTP application protocol to exchange messages between server and client. It employs a standard, extensible, and composable framework to successfully exchange these messages, and it is divided in three parts:

- The SOAP envelope is responsible for defining the data that goes inside a message, the agent who will consume that data and if that data is optional or not.

- The SOAP encoding rules define the serialization mechanisms to exchange application-defined data types, i.e., the objects that are transferred on each call.
- The SOAP RPC representation is the convention that represents remote procedure calls and responses, i.e., the methods that are provided by the server.

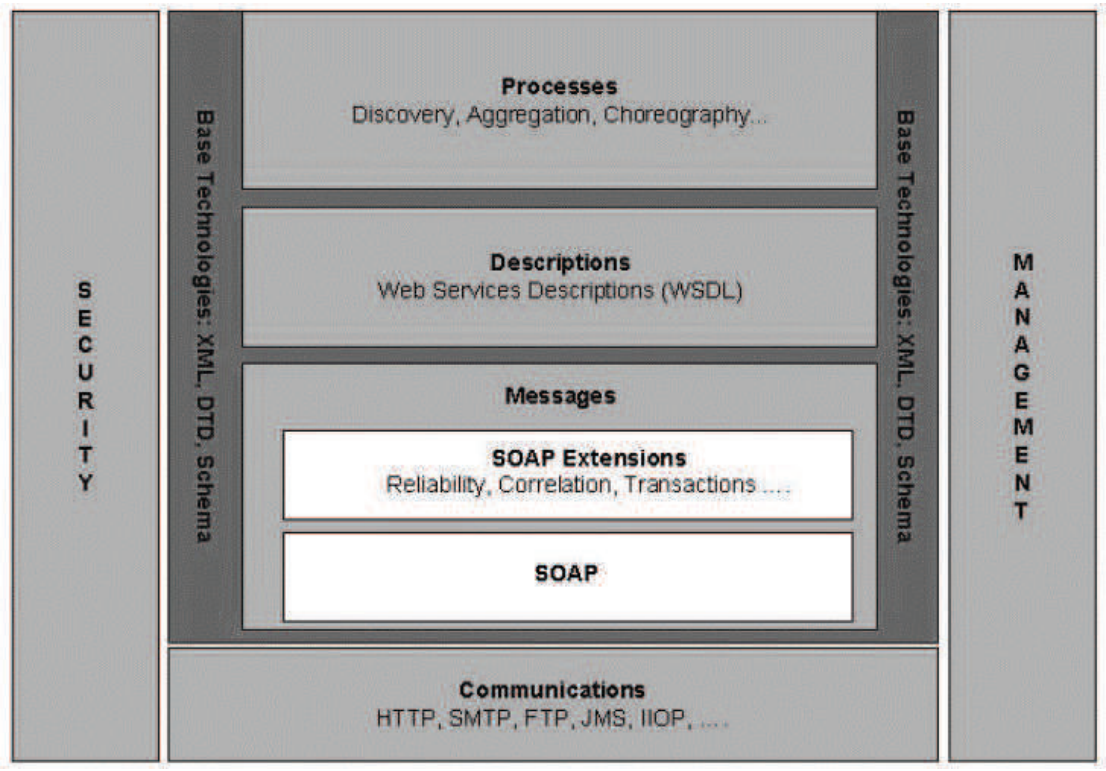


Figure 2.1: Stack Architecture for Web Services (WORLD WIDE WEB CONSORTIUM, 2004b).

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2.2: SOAP Message Request and Response (WORLD WIDE WEB CONSORTIUM, 2000).

This Web Technology uses WSDL and XML, shown in Figure 2.1. It defines a standardized framework for packaging and exchanging XML-based messages over different network protocols, like HTTP, SMTP and FTP, and the service is described in WSDL.

An example of a SOAP request and response can be seen in Figure 2.2. It is clearly visible that the message is embedded in a HTTP request, and that most of exchanged data is protocol-related information (protocol overhead) rather than actual data itself.

2.1.2 REST

It is important to say that REST is neither a framework, nor an API, but a set of constraints that creates a software architectural style when used (FIELDING, 2000). Some examples of these constraints are:

- Must be a client-server system.
- Has to be stateless, which means the server does not keep users sessions.
- Has to be uniformly accessible.

REST alone does not dictate which technology should be used to implement server and clients, instead it defines how data is exchanged, which means that it could be deployed on top of any network architecture available and use already established technologies and protocols. Hence, RESTful architecture is maintainable, extendable and distributable. Its first and most well-known example is the static World Wide Web, because it follows all constraints: web infrastructure supports caching, uses stateless connections, has unique hyperlink to resources, operates in client-server system, available documents are the resources, and the representations are defined by the web browser in the moment of access when HTML files are read. However, the Dynamic World Wide Web is not normally built using REST architecture, because these type of websites are not stateless for it is required to track users (SANDOVAL, 2009). As a result, it is possible to enumerate the abstractions needed by servers to be categorized as RESTful: resources, representations, URI (Universal Resource Identifier), and HTTP application protocol.

Resources are anything that can be addressable through the Web and transferrable between client and server (FIELDING, 2000). This abstraction is a logical representation of the problem around which the solution is being implemented. Each resource is developed to respond to the HTTP methods PUT, GET, DELETE, and POST. Representation is the data exchanged between the software agents, i.e., servers and clients. It represents the state of data that is stored in a device at the time of request and it is capable of being transferred in innumerous data format types, such as XML stream, JSON stream, text file, images, and so on.

URI is the identifier used by REST Web Services to reach for resources and exchange representations. If the server runs through the Web, then these identifiers can be hyperlinks, otherwise they would be different, but would have to maintain the same characteristic of addressing one single resource on the server.

Differently from SOAP, the HTTP application layer protocol is the mechanism used to transfer the data representation of resources in REST web services. By using the request methods GET, POST, UPDATE, and DELETE, the client points an action to be performed by the server for the accessed URI. A special feature of REST is that requests can be easily understood simply looking at them, as shown in Figure 2.3. It is showed that REST web services are deeply integrated with the HTTP.

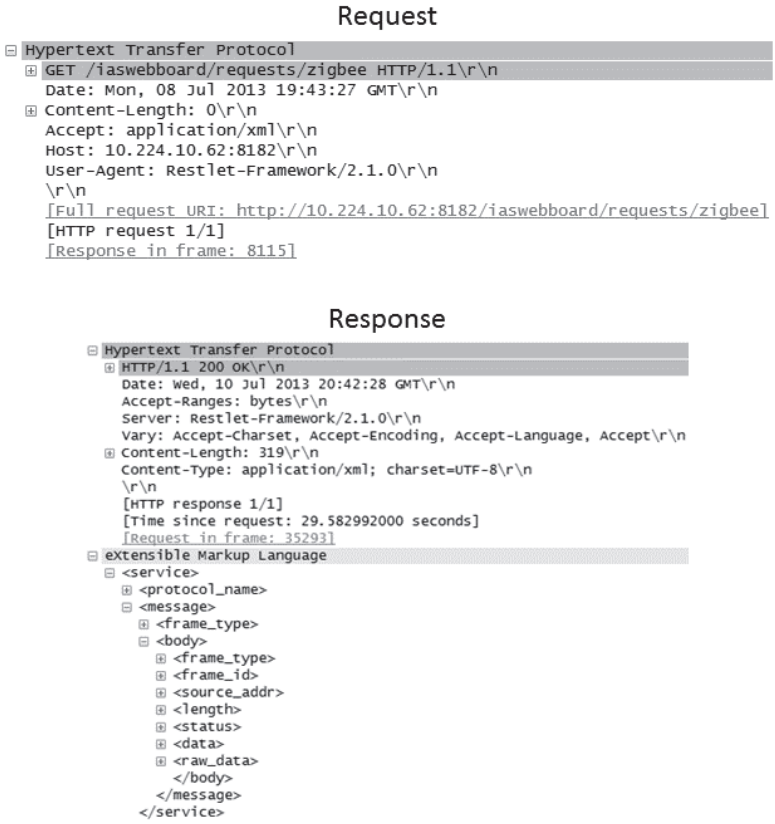


Figure 2.3: HTTP request and response using REST constraints.

2.2 Embedded systems in automation and its applications

The main definition for the term embedded system says they are devices that combine hardware and software solutions designed for a specific purpose. It is known that PCs are general-purpose computing devices with powerful processors, large storing capabilities and many peripherals, designed to be used for many common tasks. Such tasks include checking email, editing text, watching movies, and browsing the Internet, but PCs may also be used for

highly-demanding tasks, such as Computer-Aided Design, Computer-Aided Manufacturing, Computer-Aided Engineering, film post-production, and playing games, etc. In contrast, embedded systems use limited hardware solutions in terms of processing capabilities, storage and interfaces, and its resources are dedicated to a single task, to which the system was developed for. Also it does not mean that these systems cannot use powerful processors nor have a few peripherals, it is up to the system designer to choose the features for the desired purpose.

Applications for embedded systems are numerous and it is possible to find them in many consumer devices, like Televisions, Blue Ray Disk and MP3 players, GPS, and Smartphones, as well as part of large complex systems that demand real-time responses, like car's ABS brakes and fuel injection systems, and airplanes' FCS. Some examples are shown in Figure 2.4.

Designing embedded systems require good analysis of the application, because every implementation has special features, such as size, cost, performance and power, which specify which microcontroller to use, how much power it will consume, how fast data will be processed, how big the final printed circuit board will be and how much it will cost. Most of the devices deployed in industrial automation systems are a specialization of the embedded systems, designed to provide inputs, outputs and communication methods to automate technical processes, namely the Programmable Logic Controllers (PLC). However, these devices alone are not capable of automating large and complex processes, requiring more than one to achieve the goal.

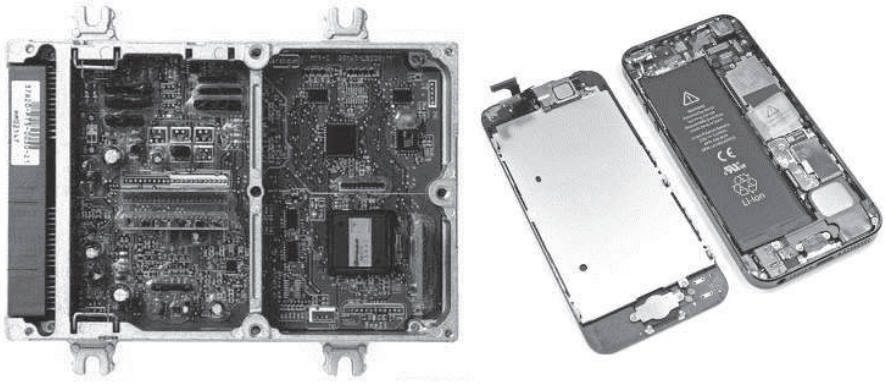


Figure 2.4: Examples of embedded systems. On the left there is an ECU from a car, and on the right, an iPhone 4.

2.2.1 Programmable logic controllers

The Programmable Logic Controllers (PLC) are electronic devices that run indefinitely a firmware that controls the execution of three tasks: reads the digital or analog inputs, than

executes the user-developed program stored in the internal memory using the readings information, and finally, updates the outputs based on the program's execution (ROSARIO, 2005). An example of this device can be seen in Figure 2.5.

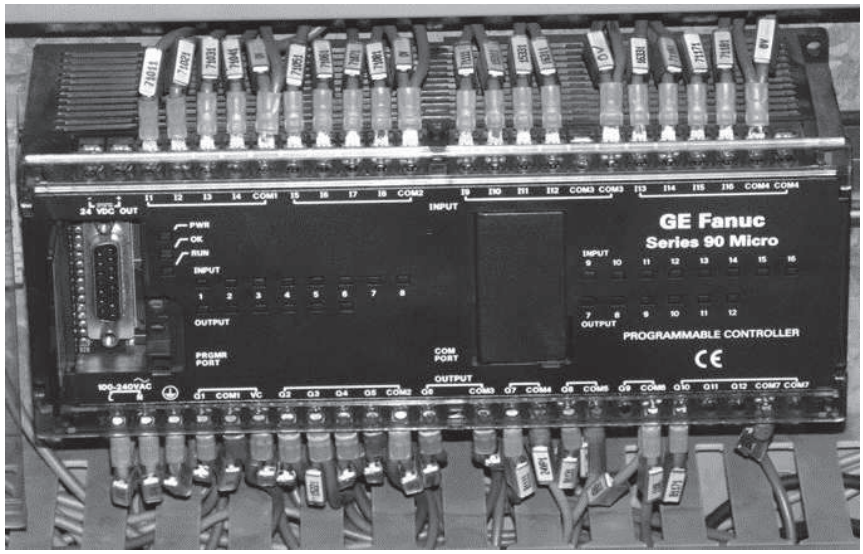


Figure 2.5: Example of an industrial PLC from GE Fanuc.

In order to be deployed in the industry, PLCs are built to sustain harsh conditions, such as resist dust, vibration, high temperatures, found in the industrial environment, which is ensured by the casing used. Since these devices, when deployed, must operate for long time periods non-stop, the manufacturers design the components and the software to fulfill this heavy demand. The major components of a PLC are the following:

- CPU, which involves the processor, the RAM memory and the programmable memory.
- Power supply, which solely provides power to the internal components and outputs.
- Inputs and Outputs, which read electrical signals from sensors and write signals to actuators. The signals may be digital or analog ones.

Another important aspect of PLC is the connection to the computer for programming the device, which happens, in most cases, through a communication port of the serial protocol RS-232. However, there some PLC models that use RS-422 as the connection interface to the computer. As a result, a RS-232 to RS-422 converter is required to communicate with the device. Moreover, some manufacturers may use proprietary protocols to communicate with the device, which demands the installation of specific hardware equipment on the computer to successfully connect to the PLC.

Connecting the PLC to the computer is a fundamental part in the process of utilizing this type of device in industrial applications. The software provided by the manufacturer, which usually has its own communication protocol, uses the communication port to manage all features of the PLC:

- Write and read programs to the programmable memory.
- Configure device's properties.
- Debug the execution of the program stored in the programmable memory.
- Exchange data during execution to expand communication possibilities.

In order to program these devices, the conventional programming languages, such as C/C++ or Java, are not used, but there are yet other languages that are employed for this purpose. Described in the IEC 61131-3 standardization, the languages are the following: Ladder (LD), Functional Block Diagram (FBD), Structured Text (ST), Instruction List (IL) and Sequential Function Charts (SFC, also known as Grafcet). The programming environment for these languages is provided by the manufacturer, as the programmable memory of the PLC must be written to execute the program. In Figure 2.6 is shown examples of these programming languages.

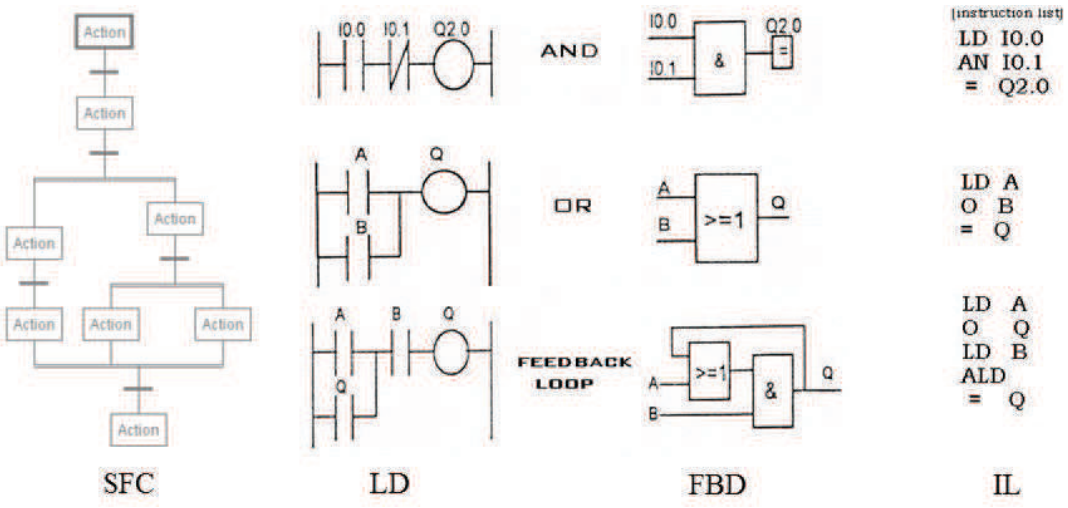


Figure 2.6: Examples of programming languages used to program PLC.

When deployed to complex and large technical processes, more than one PLC may be used, creating a communication network between them. The communications protocols may vary, but they shall always be protected against the electromagnetic interference caused by industrial electrical installations and motors. For this purpose, there are several communication protocols available, which are going to be explained in the next section.

2.2.2 Industrial networks and communications protocols

As technical processes became larger and more complex, more control devices (PLC and Industrial PC) were required to automate them. However, controls of automation systems controls were centralized, demanding cables to be spread through great distances around the factory floor, leading towards the introduction of distributed control systems. These systems were composed of several individual, less powerful and intelligent devices, capable of performing control tasks directly on the technical process, without the need of a centralized system. Still, these devices required mechanisms to exchange information between each other and perform the automation process. Thus, communications protocols began to be applied in industrial automation systems, forming the industrial networks (ROSARIO, 2005).

Industrial communications protocols are designed based on the OSI model's abstraction layers. The physical layer is usually developed to sustain heavy electromagnetic interference from high-voltage electrical installations used to power the large variety of machines on the factory plant's floor, as well as from the noise generated by the operation of the same machines. Whereas the upper layers provide other features, such as package CRC, time-critical response mechanisms, network topology management, and many others.

When deploying distributed control systems, the network topology is an important characteristic to be evaluated, as it defines the best communications protocol that suites the connection requirements. Also, the access method of the communications protocol is to be taken account, for instance master-slave, token-passing, time slicing, and carrier sensing (collision avoidance and collision detection), because it determines how high is the availability of the devices to respond to an event of the technical process (IAS, 2010b). In Figure 2.7 is shown some examples of network topologies that can be applied to industrial automation systems.

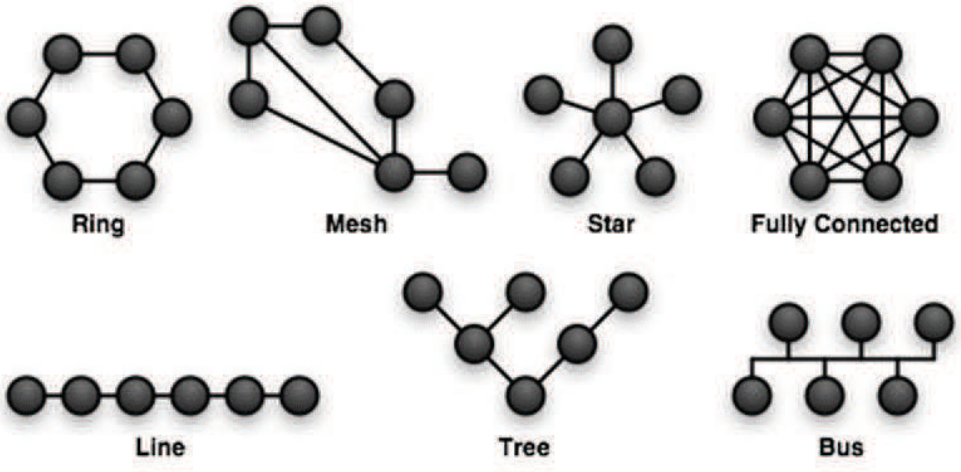


Figure 2.7: Examples of network topologies.

As a benefit of deploying distributed control systems and industrial networks to automation systems, remote access to the information regarding all processes at the factory plant became possible by using supervision systems. These mechanisms not only gave more control over the automation system, but also optimized process management, and are going to be described in the next section.

2.2.3 SCADA systems

SCADA systems are computer programs designed to aggregate information about the automation system and display it in user interfaces, also known as Human-Machine Interface (HMI), for the operator to evaluate, and also to provide methods to control the automation system (ROSARIO, 2005). The computers where SCADA systems run are always connected to a PLC or industrial network, and are developed to display a graphical representation of the technical process, as shown in Figure 2.8.

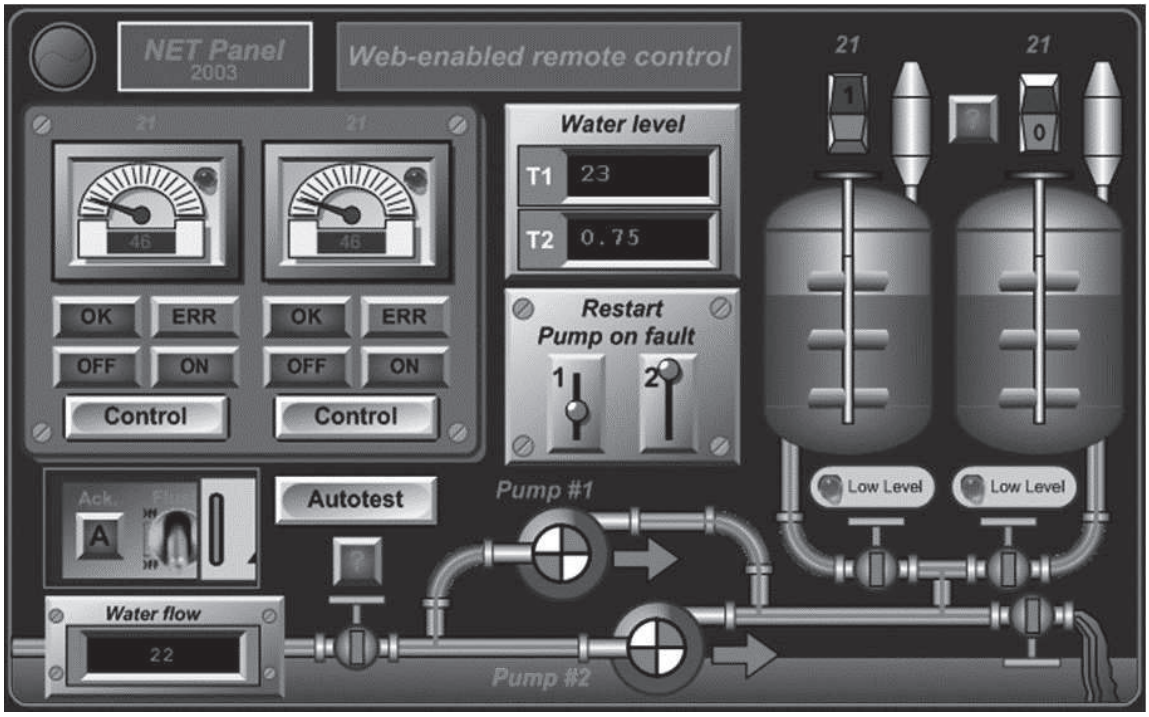


Figure 2.8: Example of a SCADA system interface.

The use of supervisory systems became important in the industrial field, mainly because maintenance departments can assess potential problems on the automation systems and take preventive action faster. It also allows the engineering department of the factory plant to ensure that production rates meet the demands. Since every data collected from the automation system is stored in data bases, reports can be generate on-demand to show production efficiency,

process cycle times, machine failures and many others relevant knowledge about the automation system.

2.3 Summary

In this chapter, at first were described the concepts and technologies involving the web services, which are frequently applied in machine-to-machine communications over a network. This type of communication demands the implementation of at least two software programs: the server who provides the services and the client who consumes the remote services. These agents use XML as a data format to exchange messages and WSDL to describe the service, but the proper web service implementation is defined by either SOAP or REST approaches.

Later, the application of embedded systems in automation was described by first giving a general overview of the composition and applications of these devices. Next, some features of PLC were described, such as the hardware composition and the standardization of the programming languages by the IEC 61131-3, followed by a brief explanation of industrial networks, which are used in decentralized control of technical process in the industry. Finally, the general concept of SCADA system is defined as tools for supervision and control of automation systems, which help technicians and engineers to keep track of operation and efficiency of the shop floor by using both PLC and industrial networks as mechanisms to interact with the automation systems.

Comprehension of these two subjects are required for both the analysis – performed in the next chapter – and the main objective of this work, since the proposed architecture is compared to other works in Chapter 3, comparing how remote supervision and control is performed by using web services, PLC and/or dedicated embedded systems.

Chapter 3 Related work

The selection of related work sought models and architectures that performed remote monitoring and control of industrial automation systems. The proposed solution of each work should either connect to SCADA applications, or propose its own supervision and control interface. The analysis is made by evaluating each related work in terms of flexibility, modularity and compatibility improvements. In general, a solution is considered flexible if it is easily adaptable to different applications, i.e., without too much development effort. Moreover, the analysis also considers the following:

- Flexibility defines the solution's tolerance to use different devices for monitoring and controlling the industrial automation system.
- Modularity defines the components' condition of being independent of each other, which allows the use of different modules, devices or adapters without demanding a new system build.
- Compatibility defines the communication's focus on the data instead of the access method, which enables the use of different software for the same purpose.

3.1 Analysis of similar architectures

Hashimukai presents (2002) a supervision method based on Web technologies that use Java2EE language for programming web services to perform the M2M (Machine-to-Machine) communication via SOAP-RPC. These services exchange maintenance data between information systems using the XML meta-language, which increases flexibility and compatibility of solutions focused on remote supervision and control. However, there is no mention on which hardware devices run the proposed solution.

The supervision system presented by Figueiredo and Da Costa (2007) defines a two-layer structure to manage production lines using PLCs interconnected through Profibus-DP network, which are coordinated by a Siemens PLC that acts as a factory plant manager. This proposition, as described, uses one PC server that provides data to remote terminals through the Web and communicates to the manager PLC through RS232-MPI. It is possible to analyze that this solution has little flexibility and modularity increase, because it is reusable for different applications, but demands developing effort to change PLC programming. On the contrary, the

analysis of the arrangement of remote terminals – namely SCADA – shows improvement in compatibility and flexibility, as it uses Internet-compatible technologies to remotely display data on several devices.

Albrecht and Grosse-Plankermann define (2004) a model of Web browser-driven application to perform control and supervision of industrial automation systems. Information is displayed using a Web browser as a client, which receives data from an ACPLT/KS server. Flexibility and compatibility are increased in this proposal given the features of the ACPLT/KS, although it is limited to browsers. Since the server remains attached to the automation system, it is assumed that modularity is not increased since the industrial automation system must be integrated to the supervision software.

The proposed solution of Ozdemir and Karacor (2006) provides features of SCADA systems via remote and mobile terminal units. The application case is developed around a crane model that is controlled by a Siemens S7 300-312 IFM PLC, which is then accessed by a CP5611 MPI bus card connected to a PC. The solution uses a Java-enabled mobile phone to access supervisory data through the Internet, as well as Java-enabled web browsers to visualize data through the LAN. This solution grants flexibility increase for remote supervision systems, since it uses Internet/Intranet infrastructure to provide information about the industrial automation system. In counterpart, the SCADA server accesses the control PLC through the MPI-enabled PC card, which limits the level of compatibility and flexibility with other automation systems, as only devices with this type of communications protocol can be monitored and controlled.

Albrich (2011) presents an application case of remote monitoring deployed in the packing industry, in which an Industrial PC manages connection hubs via Ethernet network. These hubs enable communication to RS485-capable equipment and expand the connection to other machinery that uses Ethernet as well. The solution uses web services to provide access to remote clients, using SOAP to exchange information encoded in XML format. It can be seen that this system improves flexibility and compatibility in the communication to remote terminals, because of the web services' use as content servers. However, since no mention was made about different industrial communications protocols aside the Modbus/TCP, this solution is limited to the application case described.

The work proposed by Antony *et al* (2011) features an approach to monitor and control industrial automation systems that uses a Web PLC with an embedded web server inside, turning it accessible from several programming languages. This solution shows a flexible and compatible interface via methods of the HTTP that mimics the World Wide Web structure, in

which each device either provides data representations directly to client applications, along with the secondary function of automatically storing data in remote databases. The HTTP protocol enables client applications to be programmed in any programming language that has access to this application-layer protocol. However, the Web PLC lacks communication to industrial communication networks. Instead, it has direct accesses to sensory input from the technical process through digital I/O or analog inputs, which would require the substitution of current PLC in order to enable the use of this supervision and control mechanism.

Truong and Vu (2012) describe a system for remote control and monitoring of an industrial automation system via wireless networks. The application case uses an Android application to remotely manage a CNC (Computer Numeric Control) machine connected to a Windows-based computer, all through SOAP-based web services and network sockets. Although the proposed solution does not focus on flexibility, modularity, and compatibility improvements, it does propose an infrastructure solution to remotely access industrial automation systems, which enables low-latency data exchange.

Stopper and Katalinic (2009) describe the design aspects of the OPC Unified Architecture (UA) in process control applications. This specification uses TCP/IP, HTTP, SOAP and XML to provide the same features as older specifications of OPC did, but enabling the use of more programming languages and operating systems other than Microsoft's, which expands the possibilities for its application to embedded devices as well. Thus, the OPC UA is a specification that enables devices to remotely monitor and control industrial automation systems that support this technology, aggregating improvements in flexibility, compatibility and modularity. However, access to industrial communications protocols is possible only through gateways that map protocol's parameters to OPC UA tags, which are usually charged for each protocol.

Differently from previous works, Prüter, Golatowski and Timmermann (2009) present the Resource-Oriented Device Architecture (RODA) to remotely manage robotic systems, and expand the application to industrial automation systems. The proposal is based on the REST style with some additions, which shows good results regarding performance issues over the SOAP-based Devices Profile for Web Services (DPWS).

The analysis shows how tightly dependent the supervision solutions are to the technical processes where they are deployed, and to manufacturer-specific communication and equipment. Solutions that are independent of client software and allow integration with multiple programs are more likely to be reused for different applications, such as the OPC UA. However, there is no focus on reusable hardware that attaches to automation systems to also feed factory

management programs with supervision data, and to enable remote control of these systems. In Table 3.1 is shown the analysis' summary for each related work and a comparison with the proposed solution of this work.

Table 3.1: Comparison chart based on the analysis of flexibility, modularity, and compatibility parameters.

Related work	Flexibility		Modularity		Compatibility
	Multiple buses	Multiple technical processes	Remote device independency	Communication bus independency	Different remote terminal software
(Hashimukai, 2002)	not mentioned	not mentioned	yes	not mentioned	yes
(Figueiredo & da Costa, 2007)	no	no[3]	yes	no	SCADA and J2ME mobile phone
(Albrecht & Grosse-Plankermann, 2004)	-	yes[1]	yes	not required	Web browser
(Ozdemir & Karacor, 2006)	no	no[3]	yes	no	SCADA and J2ME mobile phone
(Albrich, 2011)	RS-485 & Ethernet	no	yes	no	yes
(Antony *et al*, 2011)	-	yes	yes	not required	yes
(Truong & Vu, 2012)	no	no	yes	no	yes
(Stopper & Katalinic, 2009)	yes, with UA server/gateway[2]	yes[1]	yes[2]	if server sofware available	yes
This work	yes, with adapters	yes[1]	yes	if pre-compiled	yes

[1]: depends on the automation system to which is connected to

[2]: manufacturer must support this feature

[3]: requires redesigning the system

The comparison results in Table 3.1 were based in properties of the flexibility, modularity, and compatibility improvements. For the flexibility improvements, it was evaluated whether the supervision system could perform its tasks over multiple buses, and whether the same system could be deployed to multiple technical processes without having to rebuild all its software and hardware components. Regarding the modularity improvement, it was evaluated whether the system could perform its tasks independently of the remote terminal and the communication bus. The compatibility improvement had the possibility of performing supervision and control from different software programs evaluated.

3.2 Summary

In this chapter, the parameters for comparison of related work were established and an analysis has been made. At first, it was explained the meaning behind flexibility, modularity,

and compatibility improvements for automation systems: flexibility defines the tolerance to perform supervision and control in different devices; modularity defines the components' condition of being independent of each other; and compatibility defines the focus of the communication on the data instead of the access method. Later, the related work were exposed and led to the conclusion that most supervision tools are highly dependent on the automation system where they are deployed, and also upon manufacturer-specific communication and equipment. A summary is shown in Table 3.1 comparing the related work to the proposed solution of this work.

This thesis then proposes a solution that approximates to the OPC UA, but with fundamental changes to enhance hardware reusability and, consequently, the cost of deployment to the industry. It inherits models for remote access via HTTP and web services detailed in most cases, but focuses on the performance achievements from (PRÜTER, 2009) to provide remote supervision and control of the industrial automation system. The architecture concept behind it is detailed in the next chapter, where its interfaces and components are defined in order to achieve the flexibility, modularity, and compatibility improvements.

Chapter 4 Conception of the architecture

This chapter will start by analyzing the problem mentioned of low flexibility, low modularity, and low compatibility in automation systems and supervision tools. The focus is on the ability to reuse them for different industrial applications and devices from multiple manufacturers. Later each element will be described individually to define how flexibility, modularity, and compatibility can be improved. The chapter ends with an overview of all concepts that were explained and shows their contribution to facilitate the process of finding technologies that fulfill the requirements for increasing the parameters mentioned above.

4.1 Problem analysis and overview

In Chapter 2, the basic elements that compose industrial automation systems and their supervision systems were described: the automation systems itself, the SCADA system and the industrial network. However, to perform supervision of already deployed automation systems, another hardware device must be added in order to extract the data from the PLC, IPC or any particular device, which controls the technical process. Thus, it is possible to simplify the architecture in an element diagram, as shown in Figure 4.1, in which there are four components and two interfaces. The components are the user, the remote terminal, the embedded platform, and the industrial automation system; and the interfaces are the remote access and the communication bus. An example of how the architecture could be assembled in the industry is shown in Figure 4.2.

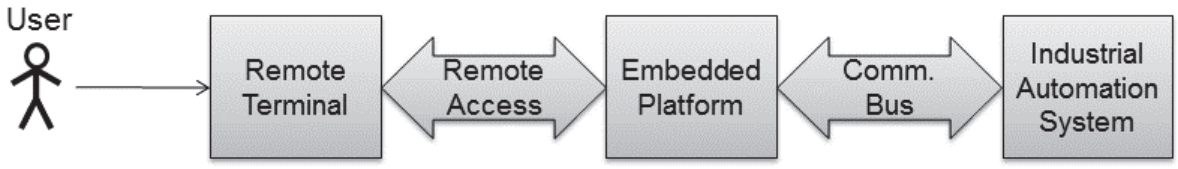


Figure 4.1: Diagram of the components and interfaces.

It must be noted though that both the user and the industrial automation system components are not defined in the architecture. The reason is that the latter is the target of the supervision system and the first is who places requests for controlling and monitoring the target. Moreover, the working principles of the industrial automation system components must be known, along with its characteristics, since it is going to be supervised. With the consideration

above, it is possible to affirm that the architecture is mainly composed of the remote terminal, the embedded platform, the remote access, and the communication bus, which are going to be explained individually further in this chapter.

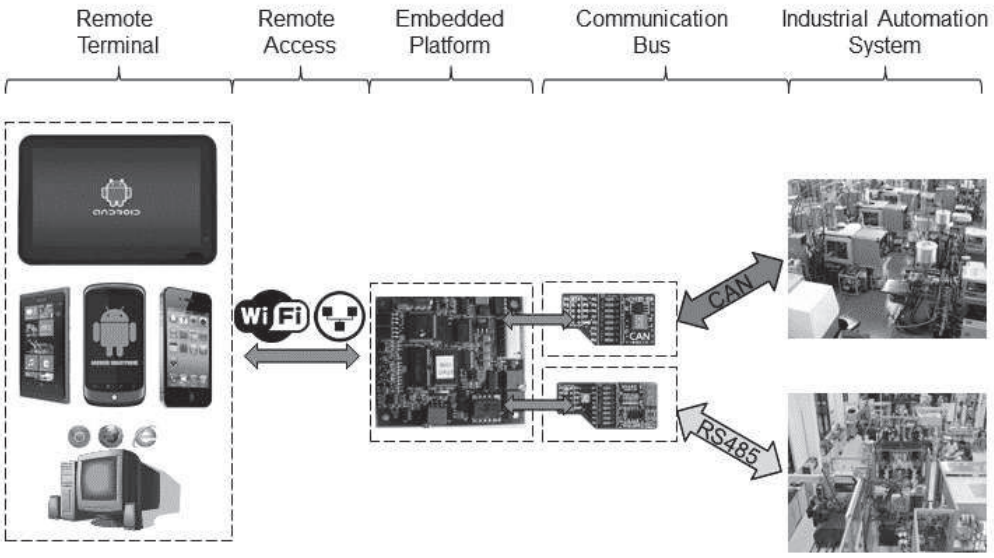


Figure 4.2: Overview of the system architecture based on the requirements.

An evaluation of the proposed elements for the architecture indicates the following: the embedded platform is a hardware platform that operates as a device-independent gateway between the remote terminal and the industrial automation system; the remote access is a universal communication method between the embedded platform and the remote terminal; the communication bus is an interface between the embedded platform and the industrial automation system that converts the industrial protocol to a processable data standard; and the remote terminal is where the user places requests of diagnosis, maintenance, and operation of an industrial automation system.

The hardware and software arrangement must improve flexibility, modularity, and compatibility, because it has to be reused in several different applications, with any remote terminal, and through any industrial communication protocol. This means that interfaces and components work the same way even when the remote terminal, embedded system or industrial automation system are switched to different ones. In Figure 4.3 is shown an overall scenario composed of two remote terminals, two industrial automation systems, and one embedded platform. This scenario is going to be used to explain two different arrangements of the software architecture for the proposed elements – components and interfaces – in order to achieve the desired improvements for controlling and monitoring industrial automation systems. The

arrangements being described in the following topics are named distributed and centralized arrangements.

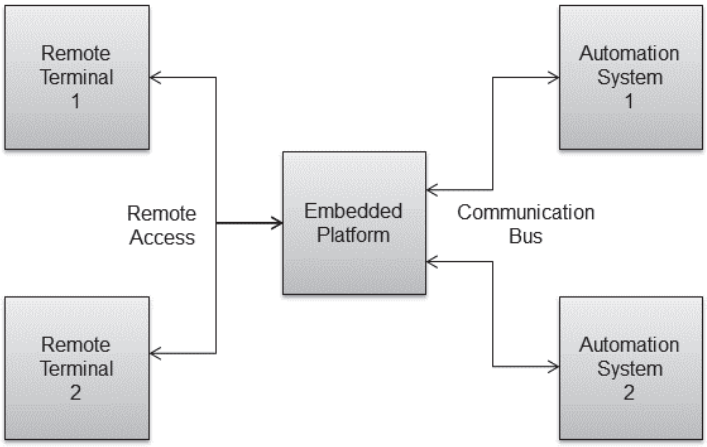


Figure 4.3: Use case scenario.

Regardless the arrangement, a request to control or monitor the industrial automation system shall have the following sequence: the user interacts with the interface in the remote terminal and selects an option to either control or monitor; the data representing the user's request is sent to the embedded platform over the remote access interface; this data is then received by the embedded platform and forwarded to the industrial automation system using the communication bus; finally, any data received from the industrial automation system is stored in the embedded platform for later use by the remote terminal.

In the first arrangement, called distributed arrangement, the software required to properly control and monitor the industrial automation system is divided between the remote terminal and the embedded platform. This arrangement focuses on reducing data traffic between remote terminal and embedded platform. It also reduces the amount of software add-ons to be implemented when the architecture is deployed for new applications.

4.2 Remote terminal

The first component of the architecture is the remote terminal. This component improves the modularity and flexibility of the architecture concept to control and monitor the industrial automation system. With the decentralized arrangement, it separates knowledge on the system from the knowledge on the communication protocol, as shown in Figure 4.4. Each remote terminal has the knowledge for controlling and monitoring an automation system. As can be seen by the color code, the remote terminal 1 (RT1) has information about the automation

system 1 (AS1) and the remote terminal 2 (RT2) has information about the automation system 2 (AS2).

The knowledge consists of the procedures required to control and monitor the respective automation system, i.e., the sequence of data that must be exchanged and processed with the intention to execute the desired requests of the user. These exchanged data is, in fact, a few parameters associated with the respective automation system's communication protocol, which are required to correctly execute the desired procedure. As for the parameters, a few can be enumerated:

- Memory addresses and, if available, sub-indexes.
- Address in the communication bus.
- Values to write or read, as well as their sizes.
- Protocol used to communicate and its message types.

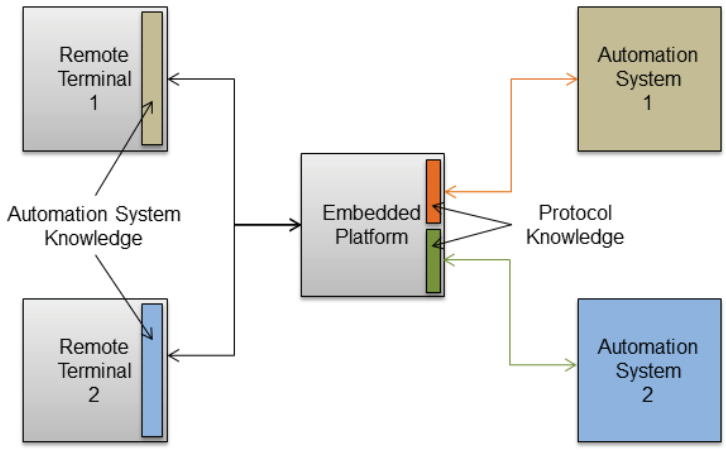


Figure 4.4: Organization of knowledge for the distributed arrangement.

Regarding the protocol information that is stored inside the remote terminal, it is used only to know through which communication protocol the automation system is accessed. This information is also sent to the embedded platform to specify that the respective request must be executed using that communication bus. However, the remote terminal has no knowledge on how the communication bus operates and how the process of exchanging messages through it is. Thus, in Figure 4.4, every time messages are sent to the embedded platform, both RT1 and RT2 specify inside the message package that the request must be executed through, respectively, orange and green buses.

By using this arrangement, RT1 cannot control or monitor AS2, as well as RT2 cannot control or monitor AS1, unless the knowledge base on the other automation system is present. This arrangement also allows different devices to work as an interface for the same automation

system just by presenting the same knowledge on the automation system, enabling more elaborate or simpler interfaces. As can be seen in Figure 4.5, both RT1 and RT3 have the knowledge on how to control and monitor the AS1. Supposing RT1 and RT3 are different devices, for instance one Smartphone and one PC respectively, it could grant mobility for the user operating the RT1, but more features for the user operating the RT3.

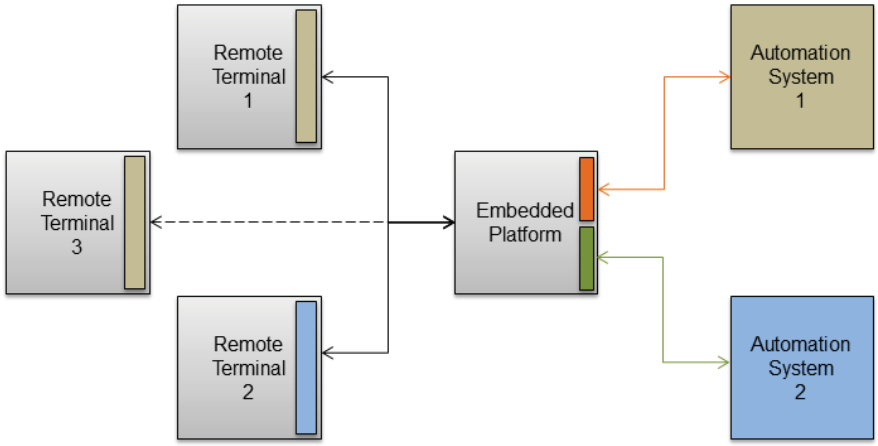


Figure 4.5: Example of multiple devices to control and monitor automation systems.

The major difference between the arrangement above and the centralized one is that with the latter, the remote terminal packs not only the knowledge on the automation system, but also a great portion the knowledge on the communication bus – further detailed in section 4.4 – as shown in Figure 4.6. However, not all of the knowledge can be moved to the remote terminal as some of it is responsible for controlling low-level communications buses, which convert digital information to the voltage levels required by the process automation protocols, and vice-versa. As for other aspects, such as the restriction of access and addition of multiple devices, the arrangements have no difference at all.

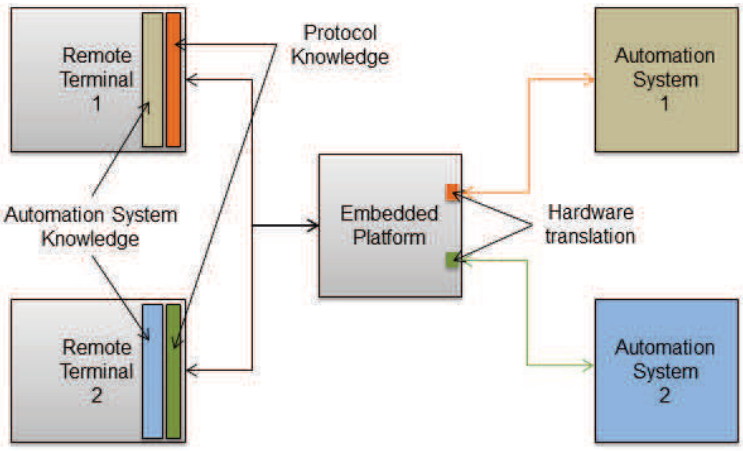


Figure 4.6: Organization of knowledge for the centralized arrangement.

Nevertheless, the remote terminal alone is not capable of providing the desired improvements on flexibility, modularity, and compatibility, even if the centralized arrangement is selected. This component requires a communication method to the embedded platform – the remote access interface – that operates independently of programming languages, operating systems, and hardware. The interface is going to be explained in the following topic.

4.3 Remote access

The remote access is one of two interfaces that are compose this concept. It is designed to improve flexibility and compatibility of the architecture concept by using standardized protocols and data formats. This design would allow seamless communication between the remote terminal and embedded platform. This interface does not depend of any arrangement.

The proposed concept for the remote access is illustrated in Figure 4.7. Both remote terminals carry knowledge about different AS, but whenever data is transmitted, it is done through one protocol that is common to the RT1, the RT2, and the embedded platform. Using standardized protocols ensures that communication is always the same independently of processor architecture, operating system, and programming languages.

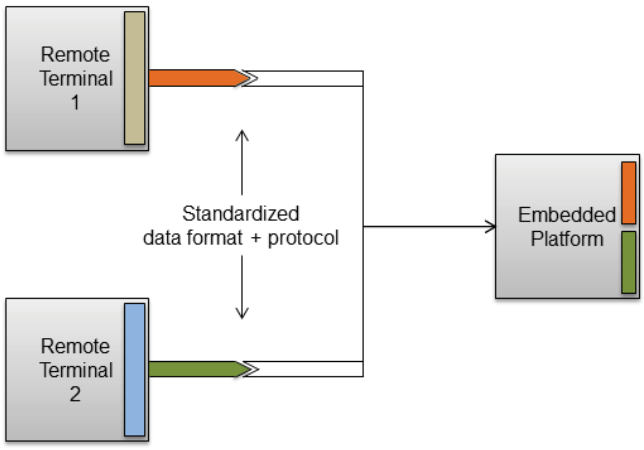


Figure 4.7: Remote access's compatibility and flexibility illustration with the distributed arrangement.

On top of the protocol, the data must be transmitted using standardized formats that can be processed by the embedded platform independently of its programming language of origin. This means that the focus of the communication is actually on data being transmitted, instead of how or where it is generated.

This approach keeps the remote access interface compatible with any remote terminal by using formats common to any programmable device, as well as keeps it flexible by allowing

any programming language to be used to generate, and process data for controlling and monitoring AS.

Thus, as mentioned in section 4.2, information about communication bus used by the AS is sent in every request through the remote access interface. However, this information must follow some constraints in order to be correctly processed by the embedded platform, which is going to be explained in the next topic.

4.4 Embedded platform

The second component of this concept is the embedded platform. As mentioned in section 4.1, its concept is tightly connected to the remote terminal. It was also mentioned in the referred section the improvements brought by the separation of the knowledge on the system and the knowledge on the communication protocol. Improvements in modularity and flexibility, as shown in Figure 4.4, are present when the architecture is organized in the decentralized arrangement, which has the system's knowledge stored in the remote terminal, and has the protocol's knowledge stored in the embedded platform.

The knowledge on the communication protocol consists of the rules for exchanging the mandatory frame format for that protocol, and for managing any feature provided by the protocol stack. This means that the embedded platform is responsible for running any computation that is specific to the protocol, i.e., which remains unchanged in case the communication protocol is used to access a different AS. A few of these characteristics can be exemplified:

- Protocol frame composition for each frame type.
- CRC calculation for received and sent protocol frames.
- Package collision avoidance features, if available.

In Figure 4.8 is exemplified the concept for the embedded platform when receiving three request messages from remote terminals, as depicted by the use case in Figure 4.4. As part of the formatted request message comes an indication of which communication protocol must be used to access the desired AS. In messages 1 and 2 there is an indication that both messages should be sent through the communication protocol represented by the orange color, whereas the message 3 has an indication that it should be sent through the communication protocol represented by the blue color. Consequently, one knowledge base is required for each protocol the embedded platform has access to.

Improvements in flexibility of the embedded platform are seen when it is reused in different applications involving the same communication protocol, as shown in Figure 4.9 by the orange-colored communication protocol. Since the embedded platform has no knowledge on how the AS3 works, it isn't relevant the information that is received from the remote terminals, as long as it can be correctly exchanged through the communication bus.

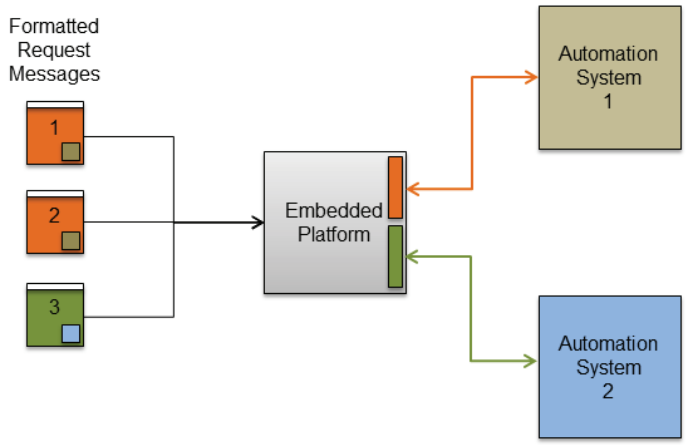


Figure 4.8: Overview of the architectural concept for the embedded platform.

Modularity improvement can also be seen in Figure 4.9. For accessing the AS4 a different communication protocol is required, but employing software techniques in the embedded platform it possible to attach new protocols stacks and enable the communication with new AS. This approach allows as many communication protocols as possible to be managed by this component.

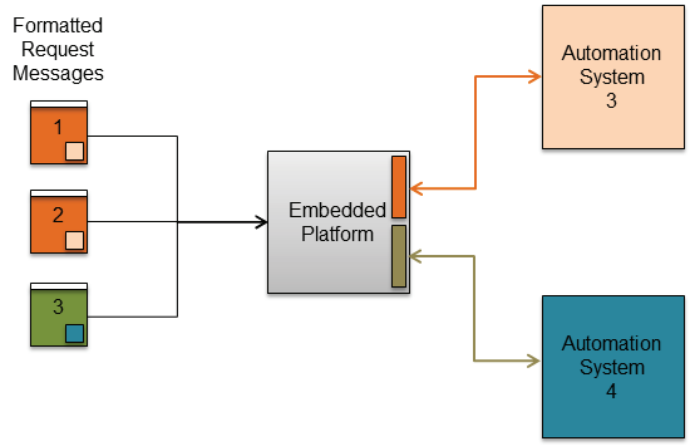


Figure 4.9: Overview of the concept for different AS applications.

When the architecture is arranged around the remote terminal, most of the improvements are provided by that component, as it holds most of the features shown above. As shown in Figure 4.6, the knowledge present inside the embedded platform is about how to manage drivers

for the communication bus interface. Yet, the concept of formatted messages remains the same when exchanging data with the remote terminal.

Although the embedded platform has features that enable several communication protocols to be attached, it still needs a physical interface to exchange information in a voltage level comprehensible by the AS – especially if the architecture is organized in the centralized arrangement. Thus, the low-level communication bus interface was employed to follow the proposition defined by previous elements, and it is going to be explained in the next section.

4.5 Communication bus

The second interface is called communication bus. It is designed to improve modularity and flexibility characteristics of the architecture, regardless the arrangement. This interface assumes that industrial communication protocols use physical layers to exchange data between devices, thus requires a mechanism to translate digital information into the voltage levels defined by the specification of the communication protocol.

There are several ways to accomplish this, but modularity and flexibility are achieved by using standardized low-level data buses and transceiver boards to exchange data with the AS. Transceiver boards are attachable small PCB that use an Integrated Circuit (IC) called transceiver, and its auxiliary circuitry, to write data to or read data from the bus connecting the embedded platform and the AS. On the software side, these IC are accessed by drivers running in the embedded platform that use one of four low-level standardized buses.

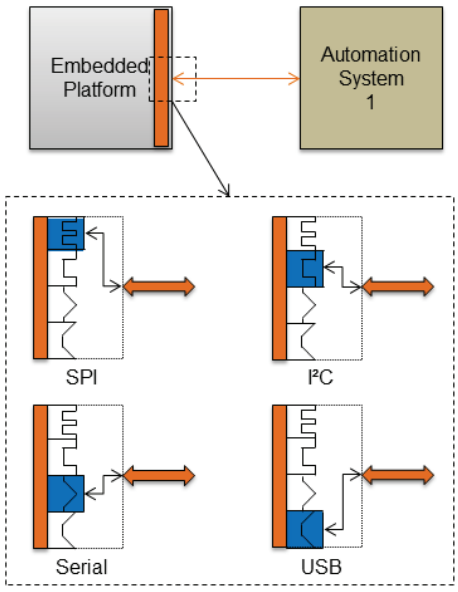


Figure 4.10: Concept illustration for flexibility and modularity increase of the communication bus.

The representation of the concept proposed for this interface is shown in Figure 4.10, such that the orange rectangle represents the protocol stack managed by the embedded platform, explained in section 4.4 , and the blue rectangles represent the transceiver boards connected to one of the low-level buses.

Flexibility is increased because access to the physical layer is conditioned to four different data buses that, along with the respective transceiver boards, enable communication with the AS. This means that any IC can be used, as long as the embedded platform is able to actuate on it. Meanwhile, modularity is increased because transceiver boards are built to be easily replaceable and reused for different applications, following the same principle of the protocol stack managed by the embedded platform.

4.6 Summary

In order to improve flexibility, modularity, and compatibility of supervision systems, this architecture concept proposes different solutions that, together, accomplish the desired goal, which comes from the use of standardized technologies and segregation of information storage. The architecture is presented with two different arrangements, decentralized and centralized, which share software and hardware components, but arrange them differently, as states the name.

The components of the architecture concept, the embedded platform and remote terminal, are defined to cope with the remote access interface between them. This interface then uses standardized technologies that must follow the rules defined by the organizations that created and maintain them, e.g., IEEE, Wi-Fi Alliance, and World Wide Web Consortium. Moreover, the embedded platform must also manage the interface with the industrial automation system through the communication bus interface, which also uses standardized technologies to interact with process automation protocols.

The next step is to define how the architecture concept may be implemented by analyzing the different solutions that can fulfill the specifications performed in this chapter, many of which are provided freely by the open-source community. Even though most solutions have predefined packages for several operating systems and hardware architectures, selecting the one that fulfills best the specification is also a non-trivial task, therefore the decisions of which hardware tool and software models were used are detailed in Chapter 5.

Chapter 5 Proposed solution

In order to show the implementation of the concept exposed in Chapter 4, this chapter is divided in three parts: model of the solution, software solution, and hardware solution. At first is the modeling, which explains how the proposed concept evolved to a structure that is more likely to be implemented. Next, the architecture and development are shown for the software, explaining guidelines for coding and some actual implemented source-code. Later the hardware is detailed with design decisions and electronic circuits employed in the developed prototype.

This solution uses the decentralized arrangement explained in the previous chapter. The choice was made because concentrating all knowledge on the remote terminal would require more software development for every new application when compared to distributing it. The centralized arrangement also requires larger amounts of data traffic between the two components, since the management of protocol stacks is done remotely, over networks with lose or none real-time constraints.

5.1 Model of the solution

Based on the architecture concept, in which data is exchanged depending on the user's requests, it is possible to identify two actors and seven use cases, which together build the use case diagram shown in Figure 5.1.

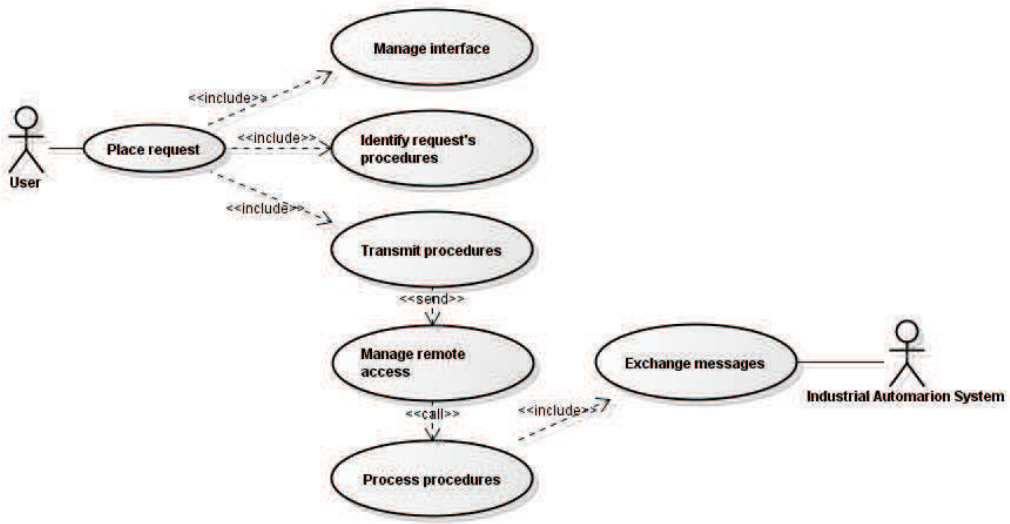


Figure 5.1: Use case diagram.

The first actor is the user, which is responsible for inputting data in the system in the form of requests. Each request represents an action to be executed by the industrial automation system, such as reading sensors or activating a sequence of actuators to produce a product. The user is also responsible for consuming information read from the industrial automation system, such as sensors values, digital inputs, etc. The second actor is the industrial automation system, which is a passive actor as it does not start any use cases. In fact, it is this actor that provides the environment to execute the procedures that describe the user's requests.

Considering the concept from Chapter 4, this use case diagram is then more accurate if shown as in Figure 5.2. The remote access interface discussed previous chapter is found in this diagram as part of use cases transmit procedures and manage remote access. The other interface discussed, communication bus, is found as part of the use case exchange messages and of the actor industrial automation system.

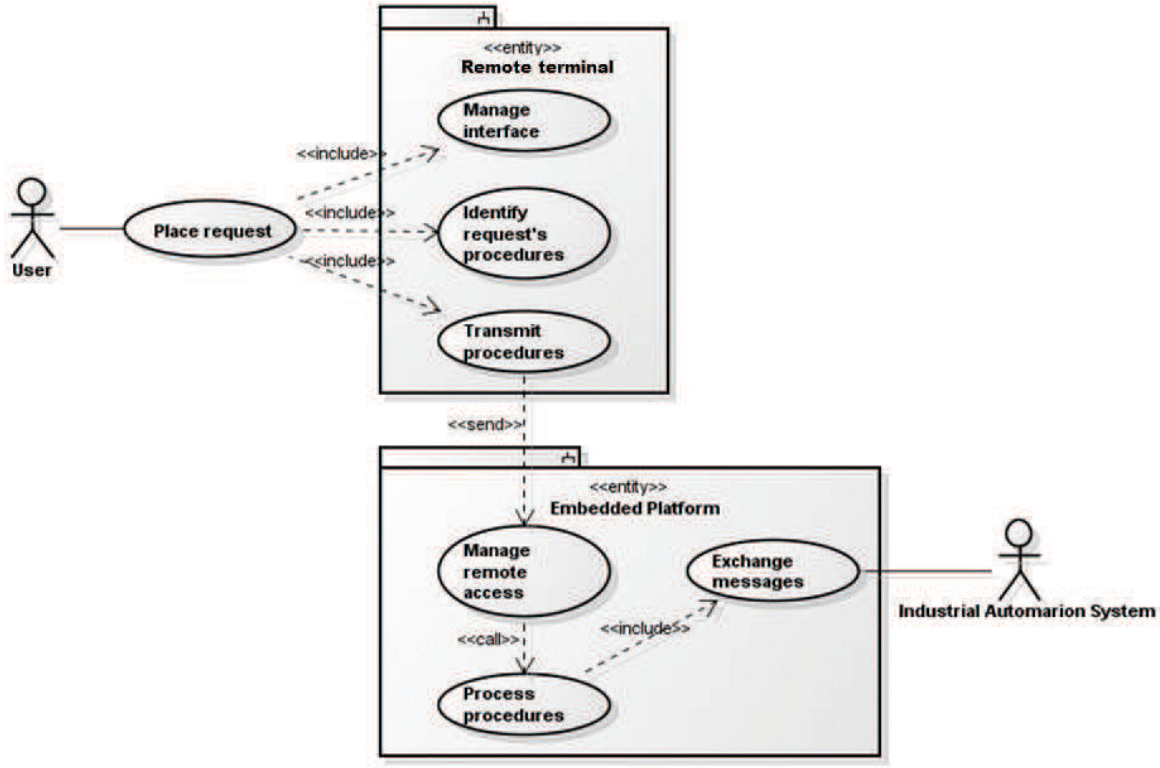


Figure 5.2: Use case diagram with the components' boundaries

The place request use case represents the interaction with the user for placing a request to control or monitor the industrial automation system. The solution is then composed of six use cases, which abstract the functions required to accomplish the requested task, and are distributed between the remote terminal and embedded platform components.

An evaluation of this use case diagram shows the workflow of this architecture, as in Figure 5.3. After a user requests an action, the remote terminal identifies which procedures need

to be executed for the particular requisition, loads the information, and transmits them to the embedded platform. When the procedures arrive at the latter, the parameters feed the processing mechanism, which builds protocol messages and manages protocol-related software that exchanges data with the industrial automation system.

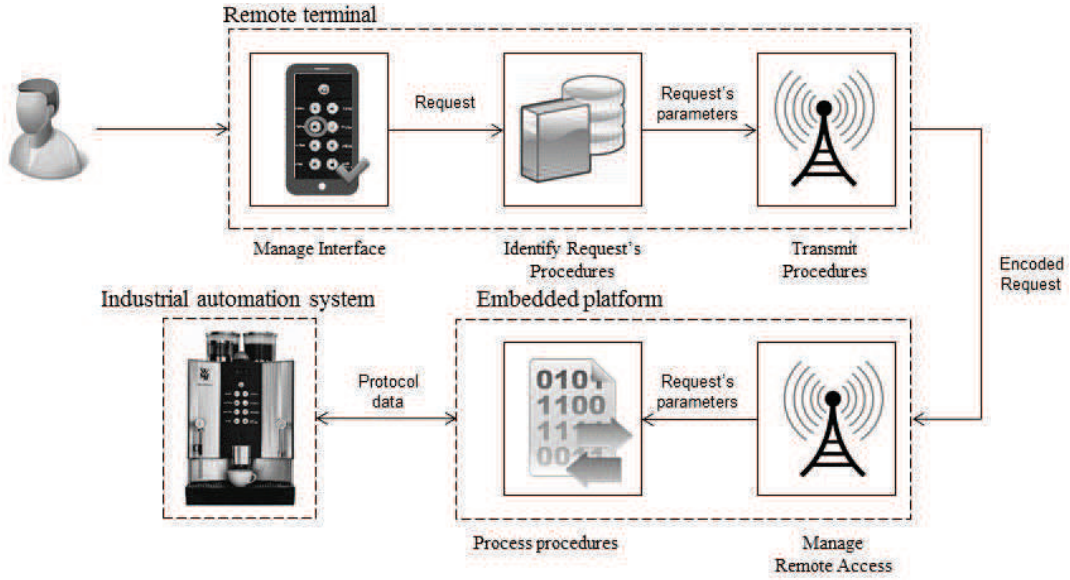


Figure 5.3: Proposed workflow based on the use case diagram.

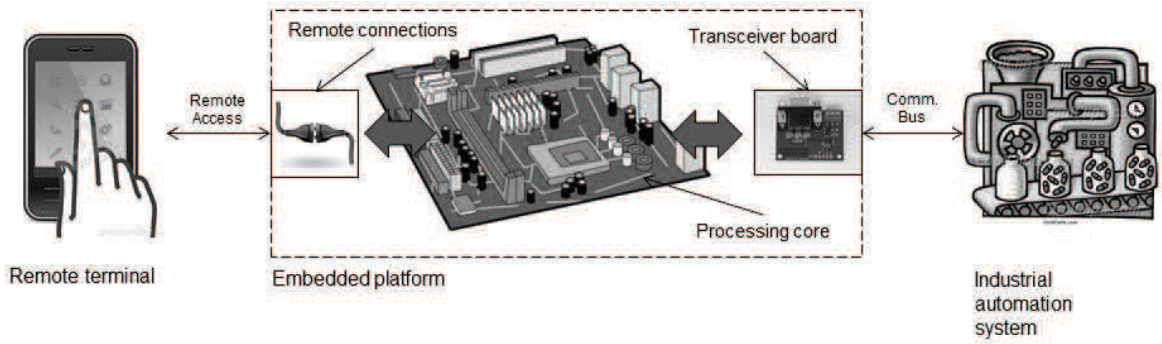


Figure 5.4: Hardware components diagram.

The use case diagram and its proposed workflow provide an overview of the proposed solution, from which more specific requirements can be established to help define the software and hardware elements of the components. However, these two diagrams are not enough to determine all requirements and for that a hardware diagram is drawn, as shown in Figure 5.4, to provide a more tangible view of the proposed solution. The hardware solution is basically composed of three functional components – previously defined in Chapter 4 – of which the most critical (embedded platform) is divided in three: the processing core and two boundary components.

Based on the diagrams shown in Figure 5.2, Figure 5.3, and Figure 5.4, a few important attributes must be pointed out in order to form the foundation for the software and hardware solutions. These features either affect the way software and/or hardware architectures are designed, or just fill blanks for execution environment in order to make the solution feasible.

5.1.1 Design decisions

The first characteristic to be discussed is about the protocol used in the remote access interface. As defined by the concept in Chapter 4, this interface has to exchange data in standardized formats over a standardized protocol. Technologies such as Wi-Fi and Ethernet meet these constraints: both can be used with the Internet Protocol suite (also known as TCP/IP), which provides several protocols capable of carrying standardized data formats; every device that supports these technologies must comply with the respective standards, IEEE 802.11 for Wi-Fi and IEEE 802.3 for Ethernet, ensuring the communication throughout the network.

The use of such technologies raises the next feature, which is the ability to manage wired or wireless networks connections. Aiming towards a platform that supports *ad-hoc* and encrypted wireless networks, a full-featured wireless protocol stack and software tools to manage network connections are required. This task can be accomplished by using one of the following approaches:

- Use a reference driver source-code from a selected Wi-Fi and Ethernet chip manufacturer, and develop the managing application and other system's features around this single solution.
- Use an embedded operating system that has support for several Wi-Fi and Ethernet chip manufacturers and provides off-the-shelf network managing tools.

For the approaches above, the embedded operating system option is the one that preserves the proposed improvements of flexibility, compatibility, and modularity. An embedded operating system has software tools to manage network connections, such as DHCP management and WPA supplicants, while its kernel provides seamless access the respective protocol stacks. There are two commonly used operating systems in embedded systems, the embedded Linux and the Windows CE. Both have been applied in industrial automation systems in recent years, but as Linux is distributed under GLP copyright license, its open-source characteristic and multiple programming language support aggregates more flexibility to the solution, which is not possible with Windows CE.

However, to run an embedded operating system the hardware device must have external memory controlled through a memory management unit (MMU), persistent storage and, most importantly, a supported CPU architecture to run upon. For running embedded Linux OS there are some processor architectures that are supported by the Linux kernel, such as Super Hitachi, x86, PowerPC, ARM, and many others. These architectures have great performance, but this system has to be low-cost. Therefore the only one that fulfills both low-cost and high performance constraints is the ARM architecture. It has de advantage to be widely used for prototyping purposes and to have a big support from Linux open-source community and silicon manufacturers. The possibility of finding ARM processors in OEM boards is also an advantage of this design decision, because these boards save hardware developing time, are easy to purchase, and are not expensive in comparison to boards with different processors.

The last aspect relates to the use of wireless networks as the remote access interface. As previously mentioned in this section, the Wi-Fi technology meets the requirements of the architecture concept. However, adding the circuitry to the embedded platform would increase the manufacturing cost. Then, an USB connector is used to provide full wireless connection over Wi-Fi and save hardware design, since Linux kernel already has all drivers and the management capabilities.

5.2 Software solution

Explanation of the software solution is based on the model described in section 5.1 and starts with the software structure that fulfills the proposed concept. The software realization follows with an in-depth description of the software components that grant system's functionality.

5.2.1 Software architecture

Based on previous definitions and diagrams, the software architecture is specified as shown in Figure 5.5. The remote terminal and the embedded platform together have six software components that are designed to perform the task proposed by the concept and to represent the use cases in Figure 5.2.

Inside the remote terminal, the software subcomponents user interface, knowledge base, and web service client are responsible for translating the user's desires into the standardized format for the protocol. The first, as states the name, is responsible for displaying an interface to the user, through which it's possible to request a service or evaluate a request result. This subcomponent depends on the application where the system is used together, as it can use inputs

from voice processing, mechanical interfaces, and touch interfaces, and output results to displays and LEDs.

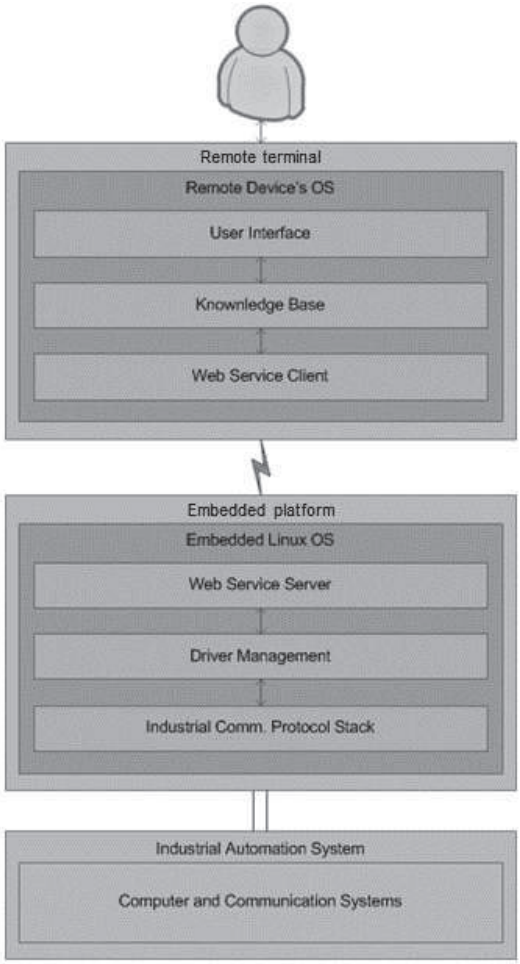


Figure 5.5: Software architecture.

The data of the desired request is forwarded to the next software subcomponent, the knowledge base, which generates a list of objects based on the received data. Each of these objects is an abstract version of the protocol message required to execute the respective request and it contains the descriptors of the protocol, e.g., slave address, memory references or memory registers, sub-references (if available for the protocol), description of data type to be read or written, data to be read or written, and many other possible descriptors. When a request result is queried, the resulting data is also formatted in a list of abstract objects that represent the protocol's response message, and it may contain message identification code, response values, execution status, error code if applied, and many other descriptors.

This software subcomponent is also dependent on the application to which the remote terminal desires to control and monitor. Every industrial automation system requires different procedures to execute tasks and uses different industrial communication protocols to exchange

these procedures. However, if applications share one protocol, i.e., provide the same protocol stack to exchange data with external devices, the knowledge base subcomponent is going to generate similar list of objects with different values and parameters.

Inside the last software subcomponent of the remote terminal, it is where the list of abstract protocol messages will be formatted to the standard data format defined for the respective protocol, and then sent to the embedded platform. This subcomponent uses a web service client to perform communication between the remote terminal and the embedded platform. By using this approach, the remote terminal's software can be implemented in different operating systems and programming languages.

In the embedded platform there are three software subcomponents, the web service server, the driver management, and the protocol stack. As shown in Figure 5.5, they run on top of the embedded Linux operating system, and together their main purpose is to work as a gateway between the remote terminal and industrial automation system.

Whenever a request is made in the remote terminal, the web service server receives the information. This subcomponent is responsible for managing the remote access interface, and also for offering support for access from multiple remote terminals due to its web service-like design. Every received request is stored in the execution queue, inside shared memory addresses.

The driver management subcomponent, as states the name, has the task of managing the drivers that build the protocol messages from the data stored in the shared memory. Therefore, each protocol requires one wrapper layer to translate the abstract data that represents the procedures into protocol-specific messages and feed them the protocol stack.

Actual communication to the industrial automation system occurs only in the protocol stack by accessing the device drivers installed in the Linux's kernel as modules, through which the protocol's physical bus is accessed. Consequently, the implementation of this subcomponent is unique for each industrial communication protocol because it depends on libraries or APIs to access the respective kernel module. All protocol features, like frame assembly, collision detection, parity validation, CRC calculation, and many others, are present only in this block.

Some Protocol Stacks are available as open-source libraries, e.g., CANFestival for CANOpen, XBEE-API for ZigBee, and Jamod for Modbus, that need to be compiled to run on the microcontroller used in the embedded platform, but some are commercial ones that require one-time license or annual fee to be used.

Thus this software architecture is composed of two programs that together work to provide the desired control and monitoring capabilities to the user. The remote terminal does not have fixed architecture, meaning that for each application it is possible to have a different interface and knowledge base, but the communication method to the embedded platform is fixed. On the contrary, the latter remains unaltered in order to provide seamless communication between the industrial automation system and the remote terminal, acting as a gateway.

5.2.2 Software realization

Before explaining the implementation of the software architecture, some observations must be done in advance. First, the remote terminal's software subcomponents, detailed in the previous section, may have its own implementation in compliance with the flexibility concept detailed in Chapter 4, but must preserve the communication method with the embedded platform in compliance with the compatibility concept. Another observation is regarding the embedded platform's software subcomponents, which, for the purpose of this thesis, were implemented in Java programming language because of strong community support, greater availability of web service libraries, and easy support for multitask applications.

The software realization is going to be done by describing each software component individually, starting with the ones in the remote terminal, followed by the ones in the embedded platform. The descriptions in this section show the several methods used to achieve the improvements of the proposed concept, but usability of the system is going to be shown in Chapter 6, where the application cases are going to be described.

5.2.2.1 User Interface

This software subcomponent is one of which that actual implementation may differ from one application to another, since its main purpose is to provide means for the user to input his requests to the system and this can be done in many different ways, e.g., speech processing, text input, option selection, etc. These requests may be an execution of diagnostic procedures, an operation or a configuration of the system, depending only of what the remote terminal application is defined to control and monitor on the industrial automation system.

Whenever an option is selected, this subcomponent inquires the user for any information required to define the desired request, which may be done through many methods, as mentioned earlier. This information is needed by the knowledge base subcomponent to generate the request procedures. The working principle is shown in Figure 5.6.

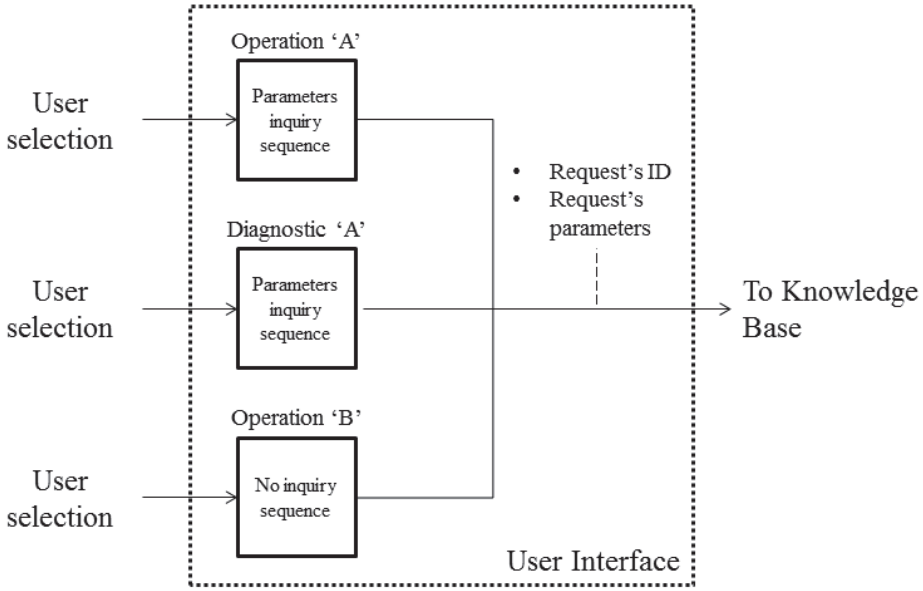


Figure 5.6: User interface's realization diagram.

For every available option in the user interface, a sequence of inquiries may or may not be triggered, as some options may not require additional parameters, and the user inputs information. The output of this software subcomponent is a set of data, containing the request's ID and parameters, which feed the knowledge base subcomponent.

Since displaying information back for the user depends exclusively on the selected request, the results of diagnostics are not triggered by the user interface, instead, this action is triggered by the knowledge base, which is the software subcomponent where information can be processed correctly.

5.2.2.2 Knowledge Base

All requests are processed and managed inside the knowledge base subcomponent, following the concept proposed in Chapter 4, in which the information regarding the working principle of the industrial automation system is stored in the remote terminal. As mentioned earlier, every request fed to the knowledge base has an ID and optional parameters that define how the respective procedures should be generated. These two elements are used to trigger the respective procedure sequence and feed it with specific values.

In Figure 5.7 is illustrated an example of organization for the knowledge base subcomponent. As it is shown, each request is composed by a set of procedures which boils down to a sequence of reading and writing messages, along with any logic required to execute the designated request. These messages are abstract representations of the actual protocol message, and they are required to correctly execute the request, as shown in Figure 5.8. A

CANOpen data frame is used to compare the abstract data frame with the actual one. It exemplifies the separation of knowledge about the working principles of the industrial AS and the industrial communication protocol.

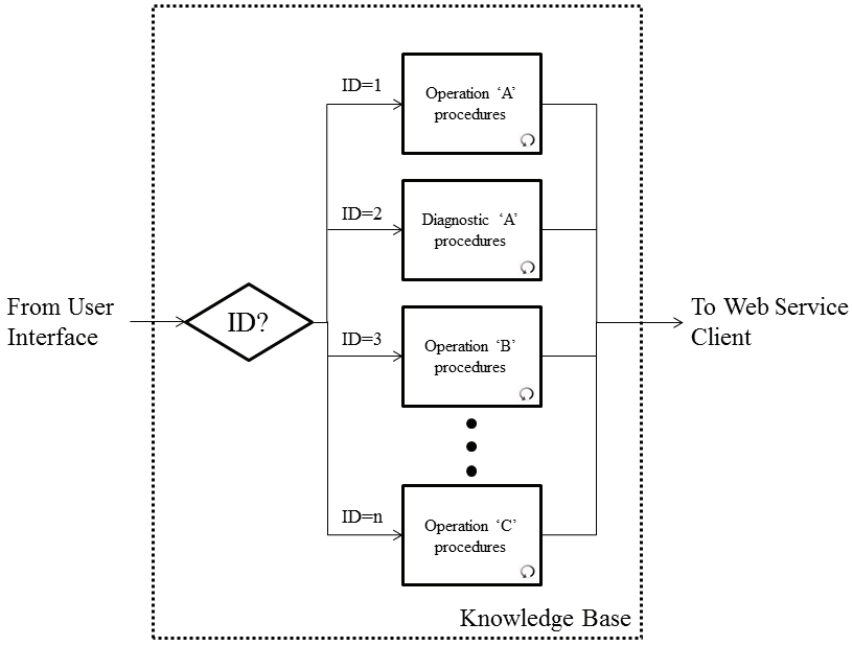


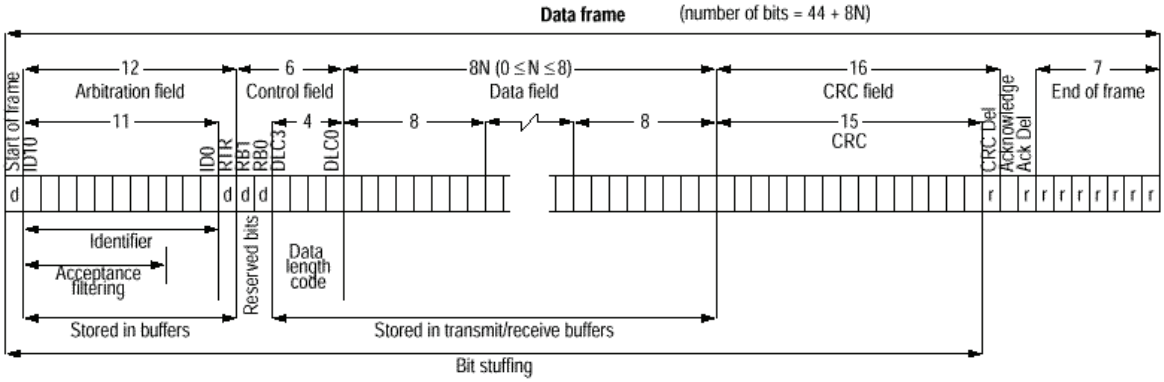
Figure 5.7: Knowledge Base's realization diagram.

As shown in Figure 5.8 (a), the abstract data frame contains information regarding addresses and values, but no information regarding other protocol-specific parameters, such as Identifier (Function code + Node ID), RTR, CRC and others. Meanwhile, the actual data frame, as shown in Figure 5.8 (b), needs those parameters to ensure the message exchange happens correctly and to provide its real-time features. The information present in the abstract data frame may be parameters generated by the user interaction, or it may be constants values, depending exclusively on the procedure in question.

Moreover, each set of procedures in the knowledge base is actually a control loop that generates several abstract messages with the intention to execute the desired request. These loops may require readings from and/or writings to the embedded system, which are resolved by the next subcomponent, the web service client.

```
CANOpenFrame frame = new CANOpenFrame(
        CANOpenFrame.DOWNLOAD_FRAME,    //Frame type
        0x1,                            //Node ID
        0x4580,                         //OD's index
        0x02,                           //OD's subindex
        0x05,                           //Value
        CANOpenFrame.UINT8);            //Value size
```

(a)



(b)

Figure 5.8: Abstract data frame (a) and actual data frame (b) comparison.

5.2.2.3 Web Service Client

The web service client is an important software subcomponent which its sole purpose is to exchange data with the embedded platform; it provides the link between the remote terminal's component and the remote access interface. As stated in the concept from Chapter 4, this subcomponent should provide access to the standardized protocol using standardized data formats, which must be independent of programming languages and operating systems.

In order to solve both requirements, a solution based on web Service communication method is proposed, as it benefits of the following characteristics:

- This solution uses the Ethernet or Wi-Fi, along with the Internet Protocol suite (also known as TCP/IP), to provide machine-to-machine interoperability, as mentioned in section 5.1.1 .
- Some of the TCP/IP application protocols enable the use of standardized data formats, such as XML and JSON. This adds programming language independency and compatibility as devices uses the text-based formats to exchange data.

This solution can be conceived by using one of two architectural styles: the REST style, based on the Resource-Oriented Architecture (ROA), and the SOAP, based on the Service-Oriented Architecture (SOA) style. A brief comparison between the two is displayed in Table 5.1. The data package parameter evaluates whether data is exchanged between devices in simple, comprehensible packages; the data type parameter evaluates the number of different

types of data can be used as message conveyor; the selection tendency parameter evaluates the preference of developers to choose one technology over the other; and the security parameter evaluates whether communication is safe from undesired access.

After analyzing Table 5.1, it can be seen that the REST-based web services offer more advantages than the SOAP-based ones. However, REST is an architectural style of software for communication, therefore, it requires the implementation of a software agent to perform the task. Thus, the web service client is implemented using frameworks to fulfill the constraints defined by the REST architectural style.

Table 5.1: Comparison between SOAP and REST approaches for Web Services.

	SOAP	**REST**
Data package	complex	simple
Data types	one	multiple
Selection Tendency	closed network services	scalable Internet services
Security	X.509, Kerberos, SAML, etc. (WS-Security)	OAuth1.0a, OAuth2

Every time the web service client is called by the knowledge base, it either triggers a POST method or a GET method of the HTTP application protocol, as demands the REST style, to exchange data with the web service server. Sending write and read messages to the industrial automation system uses the POST method, whereas the GET method is used to retrieve data already stored in the embedded platform.

Because of the POST method's purpose, abstract messages must be converted to the compatible XML format before being sent. In Figure 5.9 is shown how the messages generated in the knowledge base subcomponent are converted to the XML format, which is processable by the embedded platform. The XML is composed of five tags:

- The tag service represents a complete request and contains as many messages as needed to execute the procedure correctly.
- The tag protocol_name specifies which protocol must be used to execute the procedure.
- Each tag message represents a single message that is part of the procedure, therefore, it varies with the procedure in execution.

- The tag frame_type specifies an important feature of the respective messages, which may be the difference between message types, e.g., read/write or synchronous/asynchronous.
- The tag body contains the protocol frame's data itself, thus it varies with each protocol used.

In Figure 5.10 is shown how the XML conversion is performed for different industrial communication protocols. As a result, the protocol_name and body tags are different from the ones in Figure 5.9, as each protocol needs a unique representation of its parameters.

```
CANOpenFrame frame = new CANOpenFrame(
        CANOpenFrame.DOWNLOAD_FRAME,    //Frame type
        0x1,                            //Node ID
        0x4580,                         //OD's index
        0x02,                           //OD's subindex
        0x05,                           //Value
        CANOpenFrame.UINT8);            //Value size
```

```
<xml version="1.0" encoding="UTF-8">
<service>
  <protocol_name>canopen</protocol_name>
  <message>
    <frame_type>0</frame_type>
    <body>
      <nodeid>1</nodeid>
      <index>17792</index>
      <subindex>2</subindex>
      <value>5</value>
      <valuetype>UINT8</valuetype>
    </body>
  </message>
</service>
```

(a)

```
CANOpenFrame frame = new CANOpenFrame(
        CANOpenFrame.UPLOAD_FRAME,  //Frame Type
        0x1,                        //Node ID
        0x5114,                     //OD's index
        0x01);                      //OD's subindex
```

```
<xml version="1.0" encoding="UTF-8">
<service>
  <protocol_name>canopen</protocol_name>
  <message>
    <frame_type>1</frame_type>
    <body>
      <nodeid>1</nodeid>
      <index>20756</index>
      <subindex>1</subindex>
    </body>
  </message>
</service>
```

(b)

Figure 5.9: Example of CANOpen abstract message's conversion to XML format: (a) writing message and (b) reading message.

```
ZigBeeFrameVO frame = new ZigBeeFrameVO(
        ZigBeeFrameVO.TX_REQUEST_16,                    //Frame API code
        ZigBeeFrameVO.HAS_RESPONSE,                     //Frame Type
        buildSetOutputsZigBeeFrames(0x1234, 0x7B0A),    //Address and Payload
        "TX_STATUS_RESPONSE"                            //Response type
        );
```

```
<xml version="1.0" encoding="UTF-8">
<service>
  <protocol_name>zigbee</protocol_name>
  <message>
    <frame_type>1</frame_type>
    <body>
      <frame_api_id>TX_REQUEST_16</frame_api_id>
      <identifier_data>
        <remoteAddr16>4660</remoteAddr16>
        <option>UNICAST</option>
        <payload>31498</payload>
      </identifier_data>
      <expected_response>TX_STATUS_RESPONSE</expected_response>
    </body>
  </message>
</service>
```

Figure 5.10: Example of ZigBee abstract message's conversion to XML format.

The XML format is also used when retrieving data from the embedded platform. Whilst the GET method's request does not exchange any XML data, the HTTP response contains XML data. The reply represents each protocol messages received from the industrial automation system as a response to the ones sent using a POST method. In the example in Figure 5.11, it can be seen that the response XML data uses the same tags as the ones generated after the conversion of abstract messages. This also keeps compatibility between remote terminals bearing different operating systems and programming languages.

ZigBee response

```
<service>
  <protocol_name>zigbee</protocol_name>
  <message>
    <frame_type>1</frame_type>
    <body>
      <frame_api_id>RX_16_RESPONSE</frame_api_id>
      <frame_id>1</frame_id>
      <source_addr>123</source_addr>
      <length>7</length>
      <status>true</status>
      <data>45 11</data>
      <raw_data>00 07 81 01 23 24 00 45 11</raw_data>
    </body>
  </message>
</service>
```

CANOpen response

```
<service>
  <protocol_name>canopen</protocol_name>
  <message>
    <frame_type>1</frame_type>
    <body>
      <nodeid>1</nodeid>
      <abortcode>0</abortcode>
      <data>1</data>
      <size>2</size>
      <status>true</status>
    </body>
  </message>
</service>
```

Figure 5.11: Examples of XML data returned in HTTP GET method responses.

```
Hypertext Transfer Protocol
  POST /iaswebboard HTTP/1.1\r\n
  Date: Mon, 08 Jul 2013 19:42:58 GMT\r\n
  Content-Length: 352\r\n
  Content-Type: text/plain; charset=UTF-8\r\n
  Accept: application/xml\r\n
  Host: 10.224.10.62:8182\r\n
  User-Agent: Restlet-Framework/2.1.0\r\n
  \r\n
  [Full request URI: http://10.224.10.62:8182/iaswebboard]
  [HTTP request 1/1]
  [Response in frame: 8006]
Line-based text data: text/plain
  <service><protocol_name>zigbee</protocol_name><message>\n
  <frame_type>1</frame_type>\n
  <body>\n
  <frame_type>TX_REQUEST_16</frame_type>\n
  <frame_id>1</frame_id>\n
  <identifier_data><remoteAddr16>4660</remoteAddr16><option>UNICAST</option><payload>31500</payload></identifier_data>\n
  <expected_response>RX_16_RESPONSE</expected_response>\n
  </body>\n
  </message>\n
  </service>
```

(a)

```
Hypertext Transfer Protocol
  GET /iaswebboard/requests/zigbee HTTP/1.1\r\n
  Date: Mon, 08 Jul 2013 19:43:27 GMT\r\n
  Content-Length: 0\r\n
  Accept: application/xml\r\n
  Host: 10.224.10.62:8182\r\n
  User-Agent: Restlet-Framework/2.1.0\r\n
  \r\n
  [Full request URI: http://10.224.10.62:8182/iaswebboard/requests/zigbee]
  [HTTP request 1/1]
  [Response in frame: 8115]
```

(b)

Figure 5.12: Examples of POST (a) and GET (b) HTTP requests.

Finally, the requests for both POST and GET methods can be seen in Figure 5.12. As mentioned earlier, the POST method, in Figure 5.12 (a), carries the XML data derived from the conversion, and the GET method, in Figure 5.12 (b), carries no XML data. The respective responses are shown in Figure 5.13. Whenever a POST method is used to send data, the response contains the confirmation about which protocol has to be used and the number of messages that have to be sent, as shown in Figure 5.13 (a). The GET method, as mentioned earlier, retrieves data from the server, thus, the response contains the representation of the protocol message exchanged with the industrial automation system, as shown in Figure 5.13 (b). The URIs used by each request in the same figure are going to be detailed in section 5.2.2.4 , as the organization of resources is related to the web service server subcomponent.

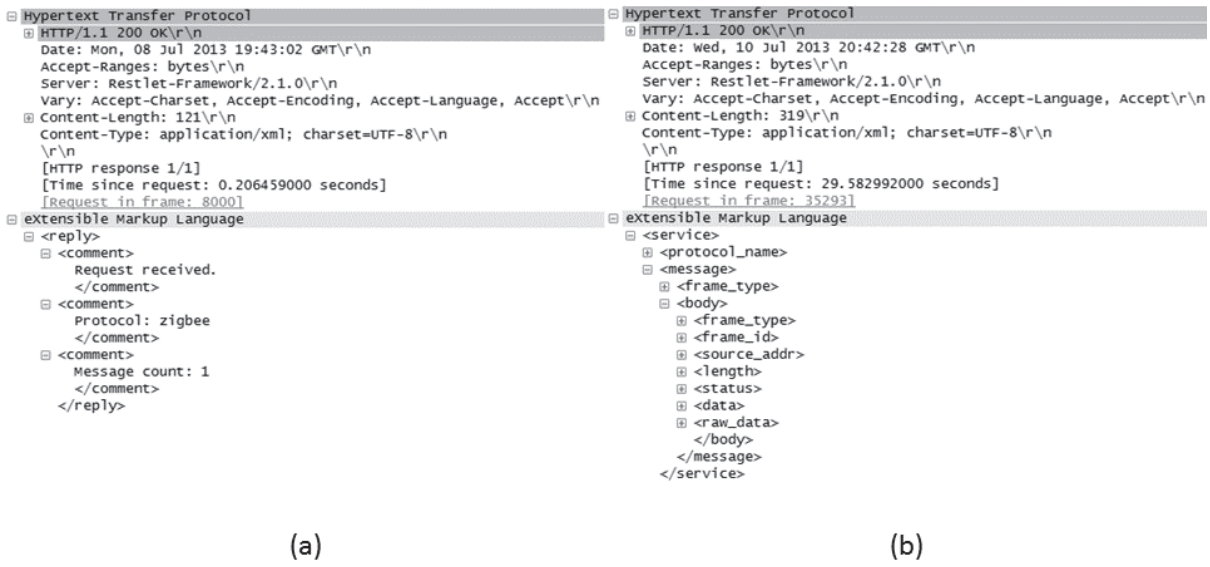


Figure 5.13: Examples of POST (a) and GET (b) HTTP responses.

5.2.2.4 Web Service Server

The web service server is the embedded platform's software subcomponent that supplies the resources consumed by the web service clients, and thus closing the final endpoint of the concept's remote access interface. This implies the server must present the same network features regarding the TCP/IP protocol suite as the client, which were detailed in the previous section.

This subcomponent has its tasks divided between two elements, the RESTlet framework and the REST resources and business. The purpose of the first is to manage incoming HTTP requests at the designated IP and port of the embedded platform's network interface, e.g., http://192.168.0.123:8080/. Whenever a HTTP request is received, this element performs three

tasks: it assembles the parameters of the request as a representation variable; it processes the URI to identify which resource shall process the data; and it calls the respective resource function from the REST resource and business element, as shown in Figure 5.14. Resources are identified by the URI string, but they can be distinguished by the sequence of characters immediately after the IP and port:

- In http://10.224.10.62:8182/iaswebboard, the resource is /iaswebboard.
- In http://10.224.10.62:8182/iaswebboard/requests/zigbee, the resource is /iaswebboard/requests/zigbee.

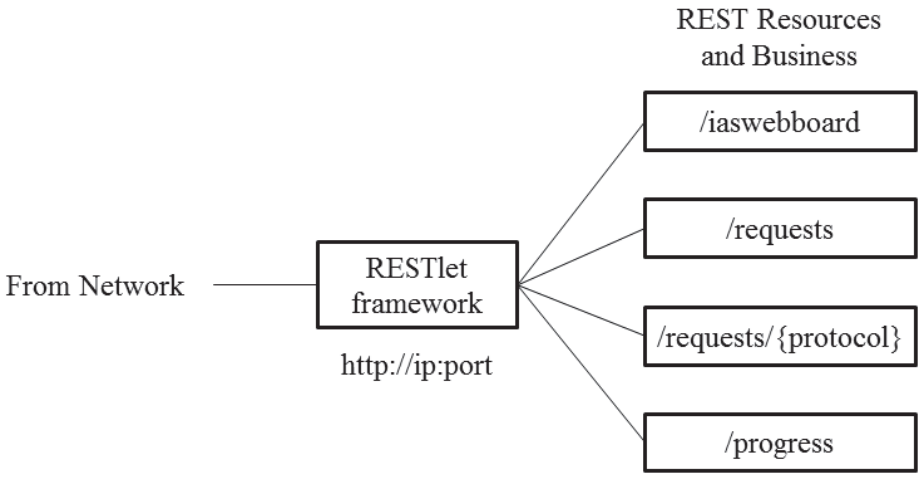


Figure 5.14: Web Service server's URI organization.

The second element is the element REST resource and business, which is responsible for two tasks: process the HTTP representation and manage the shared memory. Every resource has a function that is able to process the GET, POST, DELETE, and PUT methods of the HTTP application protocol. Thus, when a resource is called by the REST framework, one of these functions is called. If a POST, DELETE or PUT method is called, the representation is passed to it as an argument; otherwise, an empty representation is used. The resource part is responsible for calling the business function capable of handling the received request.

The four resources shown in Figure 5.14 are used provide the remote terminal with remote access to the industrial automation system:

- iaswebboard resource is the root resource of the server, and it is used to organize the resources under relevant URIs. When a GET method is used, this resource returns relevant information about the software running on the embedded platform, such as versioning and inner structure.
- requests resource provides the methods to exchange data with the industrial automation system. The POST method of this resource allows the remote terminal

to send the XML data to the embedded platform. The GET method returns all messages that are stored in the shared memory regardless the communication protocol. The other methods are disabled.

- requests/{protocol} resource replies to every GET method the response of a communication protocol to a previous request. The URI of this resource defines from which communication protocol the response must be returned, e.g., /requests/canopen returns the response of a previous CANOpen protocol data exchange. The data returned by this resource is formatted as mentioned in section 5.2.2.3 .
- progress resource is accessed only by GET methods, and it returns whether the data exchange process is busy.

Actual processing happens at the business part of the element, where the communication rules are applied, and the software extracts the XML data, for the POST method, or retrieves data from a previous request, for the GET method. The extraction of data, at this point, is limited only to common information, such as the tags protocol name, number of message tags, and type of frame, leaving the content of the tag body to be processed by the protocol-specific elements.

The result of the extraction is formatted into an object, the ServiceVO, which contains one variable for each of the XML mentioned tags: a string for the protocol name, a Boolean for the service status, and a list of message objects called MessageVO. For the response to a request, the same format is obeyed as the tag body contains the data regarding the parameters received from the industrial automation system.

In order to share data between the web service and the driver management threads, a shared memory is build using the data structure provided by the ConcurrentHashMap, which supports concurrency of access to prevent the resource to be deadlocked. This structure maps one ServiceVO object to each driver available for use by using the protocol name parameter. As a result, this data structure is used to store both the responses and the requests for each communication protocols being monitored by the embedded platform, as shown in Figure 5.15 in numbers 1 and 2.

Thus, the GetResult() function searches the data structure after the ServiceVO object related to the specified protocol name. It then returns the gathered data in the XML format, as shown in the example from Figure 5.11. Meanwhile, the CreateNew() function checks whether the protocol name is being monitored by the embedded platform. Granted, it puts the ServiceVO object in the data structure. Then, after the object is stored, a flag is set to the driver management software subcomponent to signal that data is available to be exchanged.

As shown in Figure 5.15, the web services are managed by the web service thread, but proper communication with the industrial automation system happens inside the driver management thread, which is going to be detailed in the next section.

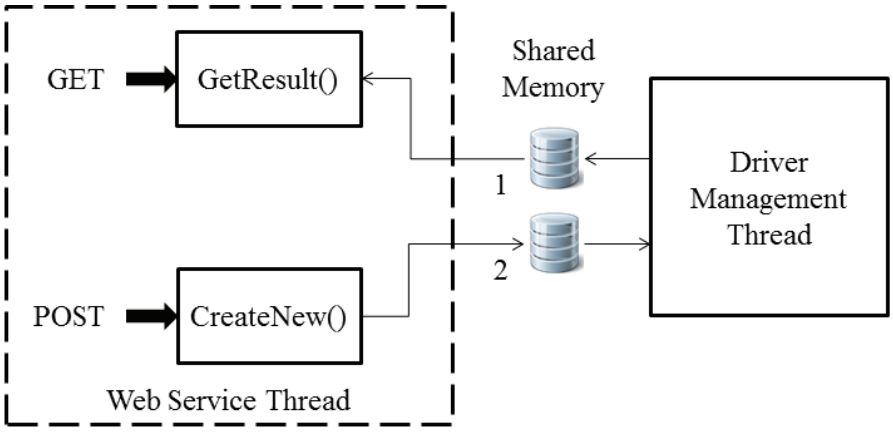


Figure 5.15: Thread diagram and shared memory.

5.2.2.5 Driver Management

The driver management subcomponent is the software responsible for managing the available communication protocols that grant access to the industrial automation system. It is divided in two elements: the driver business and the driver access. They manage together the protocol stack used to exchange data with the communication bus interface.

The driver business element is the foundation of this software subcomponent. It implements the driver management thread, and controls which communication protocol shall be managed by the thread. The creation of the threads starts when the embedded platform's software starts, as it reads a configuration file that defines which communication protocols that shall be managed, as shown in Figure 5.16. The software recognizes each tag driver, and calls upon an instance of the driver business element to manage the respective communication protocol. The tag name represents the string that is associated with the driver managed by the thread, and it is also equal to the string received from the remote terminal as the protocol_name parameter. As for the tag type, it is used log which drivers are being started by the embedded platform's software.

Since every communication protocol requires specific parameters to properly work, such as the baudrates, device names inside the kernel, and own bus address, this information must be configured in order to the protocol stack initialize communications. As shown in Figure 5.16, the tag node meets this requirement, for it presents the required parameters of each driver shown in the example. Thus, when the driver management thread is created, this element uses

the tag name to create the respective driver access element, which handles all protocol-specific functions.

Figure 5.16: Example of the preferences XML file.

Naturally, given the thread implementation in this element, it has the task of accessing the shared memory to retrieve the ServiceVO associated with the configured tag name, as shown in Figure 5.15, and pass this object to the driver access element for processing and execution.

The driver access element implements the unique features of each communication protocol. It is responsible for calls to protocol-specific features: starting or stopping the protocol stack; sending or receiving data; decoding from or encoding in protocol's data standard; managing the rules to identify errors or retry communication; and any other function demanded to accomplish the data exchange.

Whilst the starting, stopping, sending, receiving, and managing features are self-explanatory, the decoding and encoding features of this element translate data into the standard data frame of the communication protocol, and vice versa. The first processes the string from each MessageVO that represents the tag body, and generates a representation object of a

protocol data frame. The other transforms each received response in a string, and creates the MessageVO equivalent to the respective protocol data frame. The protocol stack's functions can only be used after the protocol's representation object is created, because it is at this point that all information about the protocol data frame can be processed by the software.

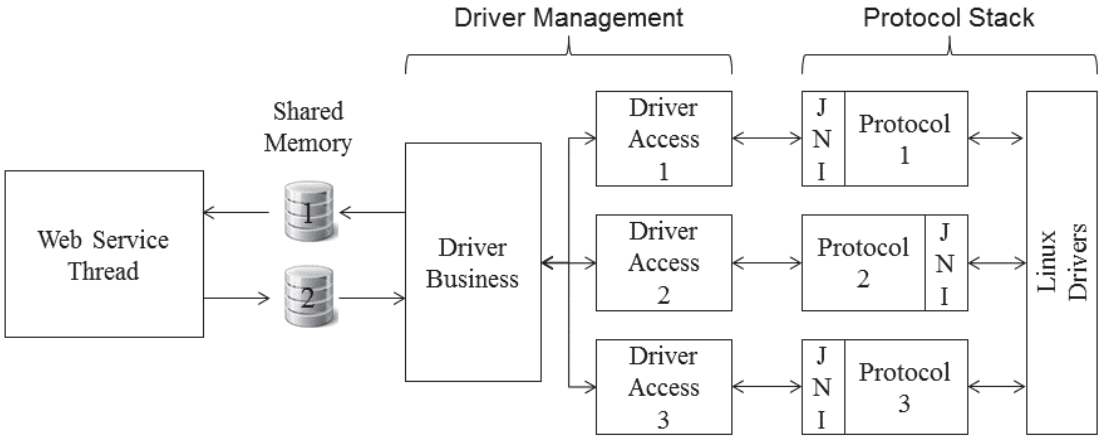


Figure 5.17: Relation between the Driver Management and the Protocol Stack subcomponents.

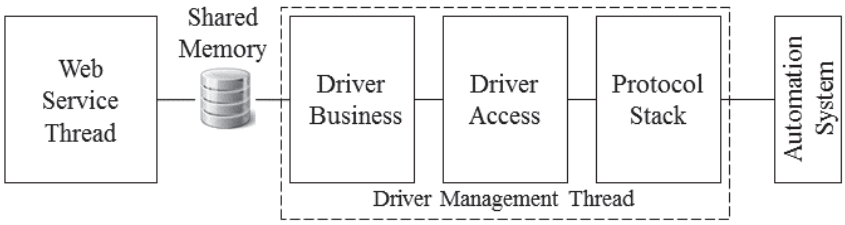


Figure 5.18: Structure of an initialized Driver Management thread.

The threshold between the driver management and the protocol stack is shown in Figure 5.17. As can be seen in driver access 1 and driver access 2, there are a few possibilities for this threshold to be developed, which depends on the library or API used by the protocol stack (detailed in the next section). Then, whenever the driver management thread is initiated, it will possess the structure shown in Figure 5.18. The driver business takes care of the shared memory access and thread execution; the driver access acts as the bridge that gives access to the communication protocol; and the protocol stack implements all the communication protocol's layers, down to the proper device driver, required to access the physical communication bus.

5.2.2.6 Protocol Stack

Although the driver management subcomponent manages communication rules, proper communication with the industrial automation system happens through the protocol stack. This software subcomponent is the last one required to comply with the concept from Chapter 4, as it works as the link between the embedded platform and the communication bus interface. It

uses the embedded Linux's device drivers to access the physical bus of the protocol to send or receive data. Consequently, this subcomponent demands a unique implementation for each communication protocol, because of two reasons: Linux's device drivers require a specific libraries or API; and it manages the specific features that render the communication protocol appropriate for industrial use.

The specific libraries or API used in this subcomponent must use the USB, SPI, I²C or Serial device drivers, as shown in Figure 5.19, to comply with the concept for the communication bus interface, as these low-level data buses provide access to the transceiver device (detailed in section 5.3) that connects to the industrial automation system's bus. Many implementations of communication protocols follow this definition to allow the use of several hardware devices to be connected as the physical layer.

Due to the fact that Linux's kernel is programmed in C, the device drivers require the Java Native Interface (JNI) to be accessed. This characteristic was previously mentioned in section 5.2.2.5 and shown in Figure 5.17. However, it does not mean that the entire protocol stack's source code must be written in C/C++. Instead, it may take advantage of Java's features to implement part of the communication protocol's abstraction layers, and call the Linux's kernel modules when needed. There is no standardization for this approach, consequently each API or library realizes the abstraction layers of a communication protocol in its own way, e.g., the CANFestival for CANOpen protocol is developed completely in C, whereas the XBEE-API for ZigBee protocol has part of the source code written part in Java and part in C.

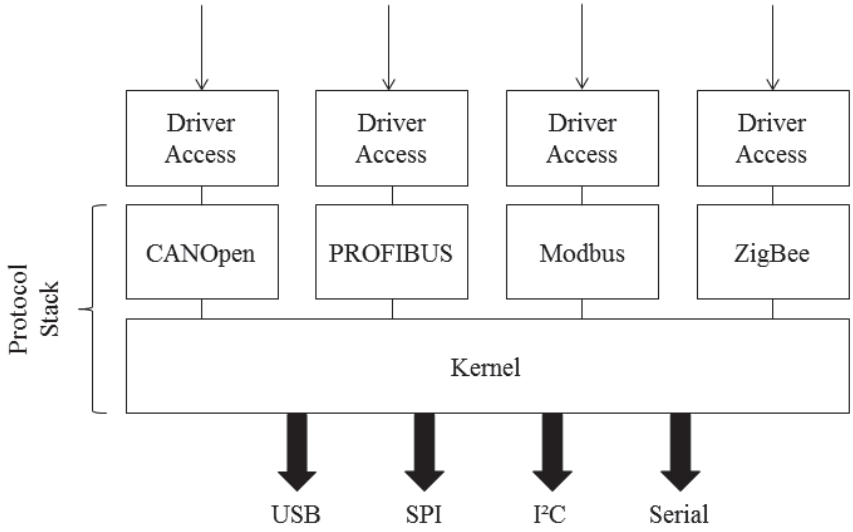


Figure 5.19: Composition of the Protocol Stack for different communication protocols.

The protocol stack, as the last subcomponent of the software architecture, receives the representation object from the driver management, and passes the data through its multiple

abstraction layers to communicate with the industrial automation system, while the reverse path happens every time data is received from the industrial automation system. The abstraction layers present in this subcomponent perform the tasks defined by the communication protocol's specification, such as collision avoidance, frame assembly, collision detection, parity check, CRC calculation, and so on, and build the respective data frame, as shown for the CANOpen protocol in Figure 5.8 (b).

5.3 Hardware solution

Explanation of the hardware solution starts with diagrams to explain where hardware solutions are deployed, in accordance to the architecture concept proposed in Chapter 4. Next, the hardware architecture is elaborated with the intention to fulfill the specifications that were not fully covered by the software solution. And last, it is presented the prototype hardware that allows the system to be integrated in any industrial automation system. It important to note that the prototype is not the only possible implementation to the hardware architecture, as it possible to perform the same tasks of controlling and monitoring automation systems on computers, taking into account the differences related to programming languages access methods to drivers and/or ports, as well as to features specific of operating system, namely, library management.

5.3.1 Hardware architecture

By using the hardware diagram in Figure 5.4 and the design features detailed in section 5.1.1 , the hardware architecture may be drawn as shown in Figure 5.20. As states the architecture concept, the solution has two functional components that are yet to be defined, the remote terminal and the embedded platform. The third one, the industrial automation system, is a real industrial process or a simulation model of one, which was stated that it was not part of the hardware components, since it is the target of supervision and control. Even though it can be controlled by PLCs, Microcontrollers or Industrial PCs, the hardware capabilities of this component have no influence in the rest of the solution. However, it is required only an interface through which the embedded platform can exchange data packages with the industrial automation system.

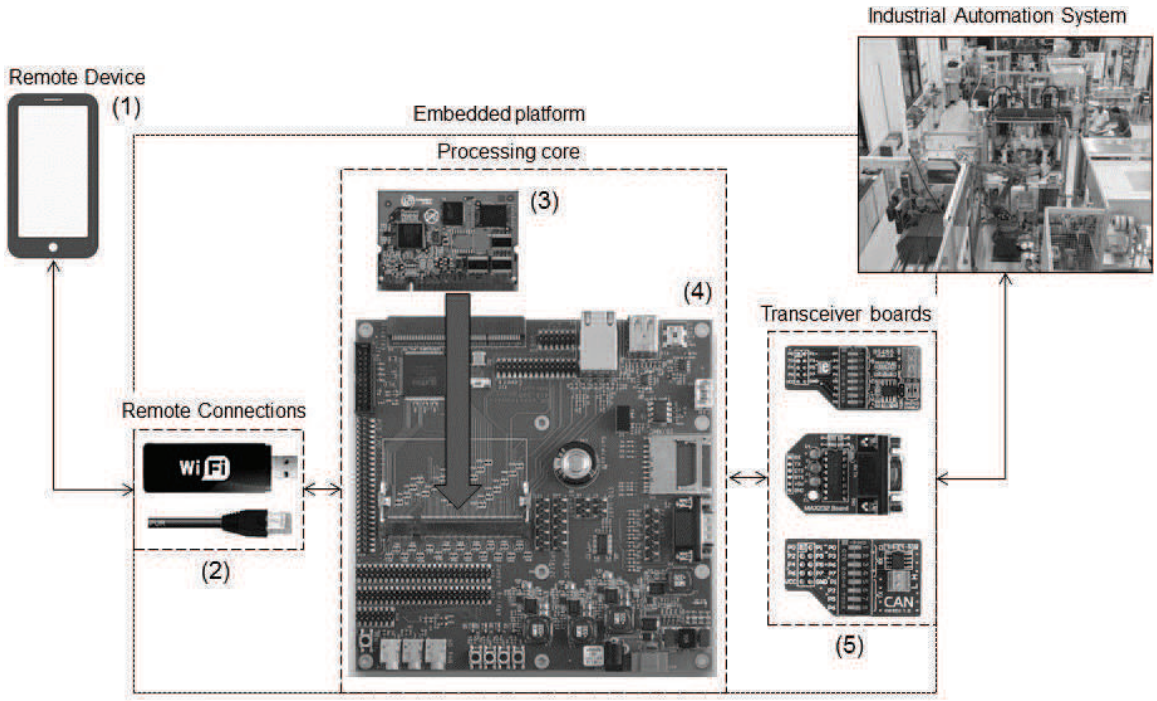


Figure 5.20: The hardware components expanded.

Shown as (1) in Figure 5.20, the remote terminal functional component is described by the architecture concept as a device that provides an interface to the user perform requests, generates a sequence of procedures based on user input, and transmits the produced information in standardized format over standardized protocols, which led, in sections 5.2.2.1 , 5.2.2.2 and 5.2.2.3 , to the description of the remote terminal's software subcomponents. Moreover, in section 5.1.1 , the Wi-Fi and Ethernet technologies were selected, because they support the requirements established by the architecture concept for the remote access interface. As a result, virtually any device can prosecute the tasks charged to the remote terminal, as long as it is able to use HTTP protocol, and has minimal computing capabilities to offer a user interface and manage de procedures of each request. Thus, it leads to the conclusion that Smartphones, Tablets and Computers can be used as remote terminals.

The embedded platform is the most complex functional component of this solution. As shown in Figure 5.20, this device is divided into four parts: the remote connections (2), the processing core (3) and (4), and the transceiver boards (5). Parts (2) and (5) are the simpler ones, since they are used as adapters that give the processing core ability to access the physical layer of their respective communication protocols.

The remote connections (2), as defined in section 5.1.1 , has the sole purpose of granting access to Ethernet and/or to Wi-Fi networks, via on-chip PHY or via dongle connected to a USB port. Meanwhile, the transceiver boards (5) are the communication modules that connect

the processor core and the industrial automation system's communication bus together. They allow the embedded platform to exchange data packages through the process automation protocol, and therefore, provide access to the communication bus interface.

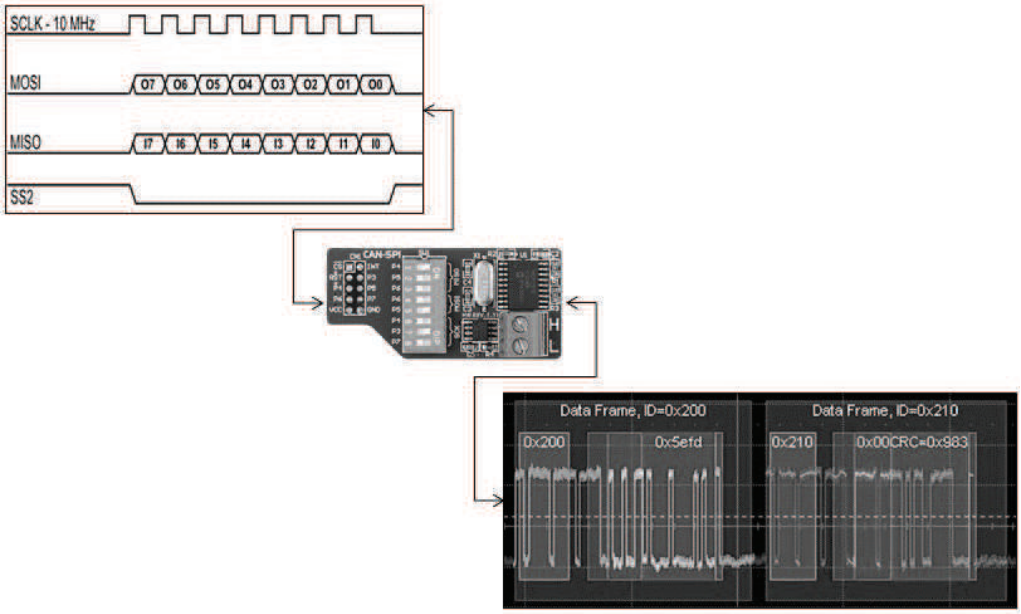


Figure 5.21: Example of transceiver board for CAN protocol.

However, there is no transceiver IC capable of providing communications simultaneously to CAN, RS-485, LIN, RS-232 or any other industrial protocol. There are ICs available for each protocol individually, though, which implies that each protocol requires a PCB that connects to the processing core via a common interface, and is capable of translating incoming and outgoing data to the desired industrial interface. An example of these PCB is shown in Figure 5.21, which uses the CAN-SPI board from MikroElektronica.

The common interface is designed to comply with the concept's requirements of modularity and flexibility improvement for the communication bus interface: use transceiver boards as communication modules that can be plugged in to enable access to the industrial communication protocol provided by that PCB; and support the commonly used low-level buses provided by the processing core. As a result, the USB, SPI, I²C, and Serial (via UART) are chosen to integrate the common interface, for these buses are largely used in data exchange between ICs. In conclusion, this hardware architecture provides modularity to the proposed solution for both the remote access and the communication bus interfaces, not only because it defines an interface, but also allows the use of different transceivers and adapters. Therefore, the hardware architecture makes it possible to connect to several industrial communications protocols and local area networks.

The processing core, shown in Figure 5.4, is divided in two parts, (3) and (4), to improve modularity of the prototype. Part (3) is the CPU board, which consists of a PCB that contains the processor, RAM memory, non-volatile memory, Ethernet PHY, crystals, and other peripheral ICs, as shown in the examples present in Figure A.1. Despite the lack other circuits, namely, power supply, RJ45, and USB connectors, this option not only adds modularity to the solution, but also minimizes the developing effort of the hardware design.

Part (4), called motherboard, is a PCB responsible for supplying the CPU board with the complementary electronic circuits, mainly for integrating it with the transceiver boards and the remote connections. In this board is where the power supply, Ethernet, USB, debugging interfaces, general purposes IOs, memory cards, common interface, and connector of the CPU board are found. Since these circuits constitute several electronic components, each one is considered to be subparts of the motherboard:

- The power supply circuit supplies the hardware with the required voltage and amperage to keep it running.
- The Ethernet circuit possesses the electronic components needed to connect the embedded platform to local area networks.
- The USB enables the embedded platform to use this bus, which widens the range of devices that can be used (Wi-Fi dongles, USB-to-protocol adapters).
- The common interface concentrates the low-level buses mentioned earlier (USB, SPI, I²C, and Serial) and provides them in edge connectors, enabling the transceiver boards to be connected.
- The connector of the CPU board, as states de name, simply serves to plug the PCB in to the motherboard.
- The debugging interfaces are extra connections required to program the embedded platform's operating system.
- Another extra connection is the general purpose inputs and outputs (GPIO), which adds the possibility to control certain devices that do not use the previously mentioned low-level data buses.
- The last extra circuit relates to the memory card interface, which expands the already available persistent storage of the CPU board.

Based on the parts described above, a diagram that describes the motherboard can be drawn as in Figure 5.22. As mentioned before, the architecture is not only limited to be used with the proposed prototype, but also with computer or any other device that fulfills the

specification, as long as it has a processing unit, Ethernet and/or Wi-Fi, and access to the communication modules.

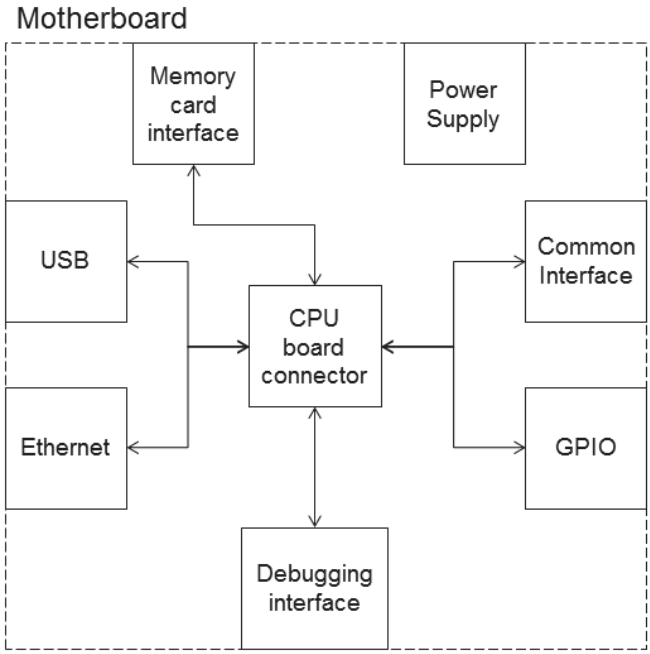


Figure 5.22: Diagram of electronic circuits present in the motherboard.

5.3.2 Hardware realization

A possible realization of the hardware architecture is achieved with the proposed prototype of this work, which is designed to have a small form factor, be cheaper than an IPC, and provide simultaneous connections to different communication buses. It was developed following guidelines and examples from other commercially-available boards that provide their schematics and layouts free-of-charge. The prototype is composed of the parts shown in Figure 5.20, with exception of the part representing the remote terminal, because it is a complete different device in the architecture. The details are available in Appendix A.

5.4 Summary

In this chapter, it was described the processes taken to define a feasible architecture that improves flexibility, modularity, and compatibility. Starting from the use case diagram, the workflow diagram and hardware components diagram are defined. They are used together as the foundation for specifying the software and hardware architectures.

The software architecture focuses in two main aspects to achieve the desired improvements: the separation of system knowledge and protocol knowledge, and the use of standardized data formats and protocols to exchange data. In summary, the three-layer software

architecture of the remote terminal stores all information about the automation systems, such as memory addresses, bus addresses, values, types of values, and type of message. It also performs remote requests over HTTP with XML-formatted data when needed. Meanwhile, the three-layer architecture of the embedded platform stores all information about managing the process automation protocol, e.g., topology management, CRC calculation and validation, and frame sequencing. It also provides the HTTP resources for remote access through an implementation of the REST architectural style.

The hardware architecture provides the last feature defined in the architecture concept, which is proper communication through the process automation protocol bus. Since the architecture proposes that remote access has to be done by virtually any device, the constraints are much looser for the remote terminal. Differently, the embedded platform has strict constraints that define the use of low level data buses to access the communication bus of a given automation system, mainly USB, SPI, I²C, and Serial. This part of the architecture also has a prototype that is detailed in Appendix A.

This complete solution was designed to be reusable and portable, as it can be used in different hardware devices, such as computers and even other development boards with enough computing power to handle the protocol stack and web server. The Java programming employed in the embedded platform's software can run on other compatible industrial equipment, as long as drivers for the transceiver boards are ported.

Finally, this architecture does not represent the actual integration with an automation system, since some software and hardware components are still required, such as the user interface, knowledge base, and transceiver boards. Thus, the procedures performed to realize these components and enable the supervision and control solution to be used are detailed in the next chapter.

Chapter 6 Case studies

In Chapter 4, it was discussed the improvements that can be applied to industrial supervision systems in order to provide more flexibility, compatibility, and modularity, and, as a result, an architecture concept was created. Then, in Chapter 5, a solution has been proposed to comply with the definitions of the architecture concept in a way it was cheaper, compact, and reusable. However, as mentioned in the respective chapter, some components are application dependent, more specifically the user interface, the knowledge base, the protocol stack, and the transceiver board.

This chapter shows how the software and hardware components mentioned above were developed, or shall be, in two application cases. The first case, a voice activated coffee machine, has been completely developed and tested with this control and monitoring systems. The other, a wireless sensors and actuators network gateway, is yet to be completed, lasting only the knowledge of the automation system to be implemented. At last, there is a third case which is an overview on how to develop all system components.

Figure 6.1: The CombiNation S from WMF.

6.1 Voice activated coffee machine

The industrial coffee machine CombiNation S from WMF, shown in Figure 6.1, allows external devices to perform diagnostics and controls of its internal process via a communication port for the CANOpen protocol (PFEIFFER, 2008). This machine is part of the process models present at the Institute for Industrial Automation and Software Engineering (IAS) in Stuttgart, Germany.

In summary, the user orders a coffee through the voice interface in which he/she speaks normally without any specific command word. This vague input is first converted to a comprehensible format, such as text, then it is processed, and the resulting parameters are sent to the coffee machine.

It must be noted that a mobile approach for the remote terminal was mandatory, therefore, an Android smartphone was used as the remote terminal. Naturally, the REST-based web service client was implemented for the respective OS. In Figure 6.2 is shown how the components and interfaces are represented in this application case.

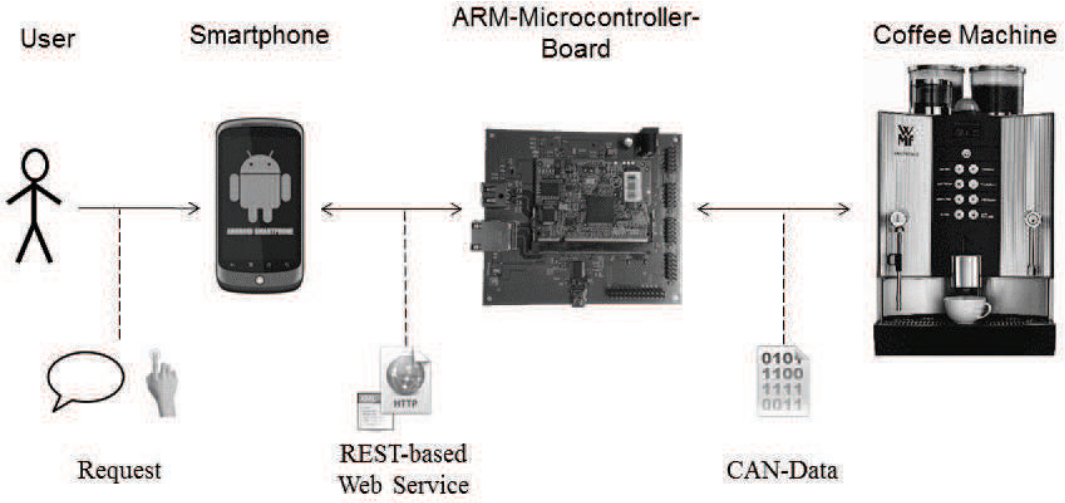


Figure 6.2: Connection diagram of the Voice activated coffee machine.

To control the coffee machine through voice using this device, an API was required to process the speech input and generate its equivalent in a comprehensible way to the software. Throughout the market, there are a few off-the-shelf speech-processing API available for the Android OS, but the most accurate to process audio, and also to generate text-based audio, is the Mobile Dragon Naturally Speaking API from Nuance (DRAGON). The next element required for this application case is one that generates parameters based on vague inputs. Fuzzy Logic enables this feature for the user interface component, thus it was developed using JFuzzy API.

These two API make the user interface subcomponent. They allow the user to place an order in which he/she speaks to the Android Smartphone using natural sentences, e.g., “I would like a big cup of coffee, but not too hot” or “Please make me a warm and strong espresso”. The speech input generated by the Dragon API is converted to text, and then processed by the Fuzzy Logic to generate parameters for making the desired cup of coffee, which are temperature,

coffee strength (powder amount), and coffee cup size (water amount), as developed by Parvaresh (2012).

The parameters are passed on to the application-specific knowledge base, which retains all information on how to monitor and control the Coffee Machine. In other words, this software component contains all frame sequences and parameters (node addresses, indexes, subindexes, values, value size) that need to be modified in the coffee machine's Object Dictionary, as well as how to process data read from the Coffee Machine. The set of abstract messages containing the indexes, at a total of 12 abstract messages, is then sent in XML format to the embedded platform, as explained in Chapter 5.

At the embedded platform, after the information extraction, the driver management thread in charge of the CANOpen protocol builds the data frames. It is inside this hardware component that the protocol stack subcomponent is implemented. The CANOpen protocol stack is implemented by the open-source framework CANFestival (CANFESTIVAL). As the transceiver board there is the Peak System's PCAN-USB (PEAK SYSTEM), which is an adapter that connects to CAN buses via DB9 connector, and to USB hosts via USB. The frames are assembled by the PCAN-USB's driver according to the CANOpen specification, and sent through the communication bus interface to start the process of making the cup of coffee.

Other procedure performed in this application case is the execution of diagnostics in the coffee machine. Differently from the procedure above, multiple diagnostics can be selected from the user's touch interface of the Android device, each generating the parameters to execute the desired reading, as shown by Eberdard (2004).

6.2 Wireless sensors and actuators network gateway

Another application case of this supervision and control system is to use the embedded platform as a gateway to ZigBee wireless networks. This provides a bridge for reading sensor data and activating actuators. Given the features of the communication protocol, this application could be used in both industry and home automation to provide user interfaces to manage those scenarios. Each node of the network is connected to a Rogercom's RCOM-HOMEBEE automation board (ROGERCOM), shown in Figure 6.3, which provides digital inputs and outputs, as well as relay outputs.

This application scenario uses a tablet or a computer to manage all nodes, along with their respective sensors and/or actuators. The remote terminal reads sensory input in time intervals and, according to the input, executes the respective procedure, driving actuators on or off. Similarly to the application described in section 6.1 , this application also focuses on the

Android OS for the tablet, which means that most of the Java application can run on the computer with slight modification of parameters. An illustration of this network is displayed in Figure 6.4.

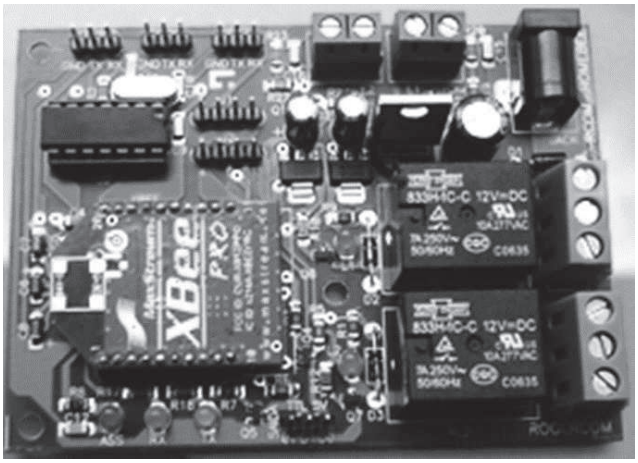


Figure 6.3: RCOM-HomeBee automation board from Rogercom (ROGERCOM).

Each board has its outputs and inputs mapped to a certain driver or sensor inside the user interface subcomponent. The knowledge base subcomponent for the application is generated after the mapping is completed. When the user selects one of them, the corresponding 16-bit address for that ZigBee device is retrieved. The software was designed to provide the user with evaluation and intervention methods over the manufacturing line, meaning that he/she is capable of reading the current state of line's components, and deactivate them if necessary. The application is also ready to generate automated tasks for the nodes by creating new entries on the knowledge base. When executed this way, the tasks make the activation and deactivation process independent of user inputs.

For validation purposes, two automation boards were used, each one containing two relays, two digital outputs, and two digital inputs. These inputs and outputs were connected to sensors and motor drivers along an experimental manufacturing cell. Regardless of on-demand or of automatic readings, the knowledge base builds the abstract message with the respective 16-bit address and the "return I/O current digital level" command as the payload. The abstract message list is sent to the embedded platform, where it is translated to the format used by the XBee-API (XBEE) working as the protocol stack subcomponent. The transceiver board for this application case is an XBee PRO RF module from Digi International (DIGI INTERNATIONAL).

Information to the user interface is updated whenever a ZigBee package returns from the requested board. Only after this update the automated tasks can be executed, or the user can

evaluate and change the state. This is performed by sending a different payload content to the same 16-bit address carrying the “change output” command along with the desired output.

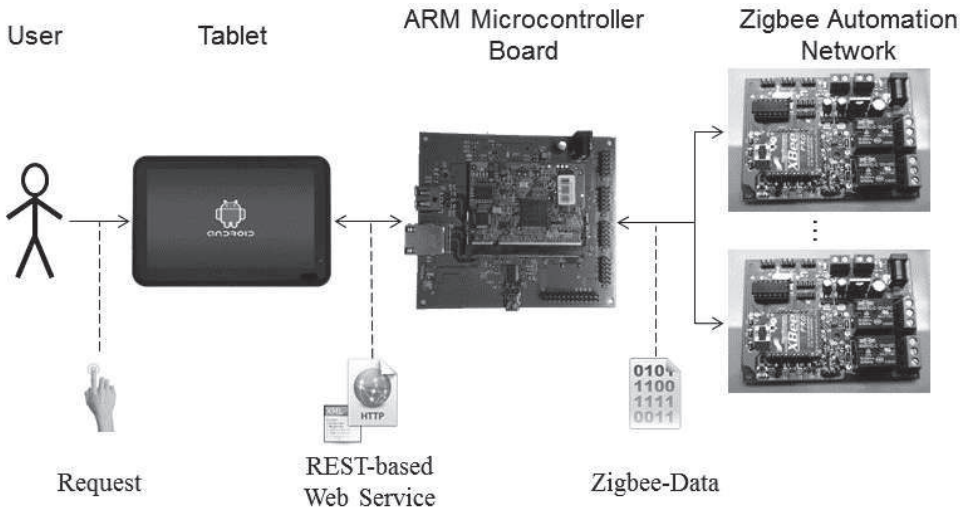


Figure 6.4: Connection diagram of the Voice activated coffee machine.

6.3 Deploying for different applications

If there is the need to perform supervision and control of a different industrial application using the proposed architecture, then there are some aspects that must be considered in order to integrate the prototype with the automation system, which determines the number of components that require modifications.

The first step that must be taken is to verify which process automation protocol is used to communicate with the automation system, because this influences whether or not it will be required changes to the transceiver board (communication module) and the protocol stack component. If the process automation protocol is not part of the software package available for the embedded platform, then it is required to add this software component to the source-tree of the software. This means that the driver access subcomponent (section 5.2.2.5) and the protocol stack component (section 5.2.2.6) must be developed to enable message exchange through the communication bus. It is also required a transceiver board (section 5.3.1 and Appendix A) to send data to and receive data from the process automation protocol.

There are two ways to define this communication aspect of the architecture, whether by selecting first a protocol API and then a communication module that is compatible to it, or the other way around. Either way the driver access subcomponent is always developed based on the interfaces provided by the protocol API. However, if these elements are already part of the system, this step can be skipped.

The other step relates to which procedures shall be generated to perform the supervision and control tasks. This involves directly the user interface and knowledge base subcomponents. At first it must be defined the procedures that the remote terminal will request execution of, such as the ones from the previous application cases (execute diagnostics, read sensors, move actuators or make a coffee). Next it must be defined which parameters are required by each chosen operation, as well as their input methods, e.g., touch input, voice input, and gesture input. With these three definitions, it is possible to specify the sequence of procedures for each operation in the knowledge base (section 5.2.2.2), as well as which API or framework to use in the user interface (section 5.2.2.1) to receive user inputs.

6.4 Overall analysis of the cases

The purpose of the architecture focuses on reusing as much software and hardware components as possible. If newer industrial applications use the same process automation protocol, then only the user interface and knowledge base subcomponents are required to be implemented. Otherwise the protocol stack and the transceiver board should also be implemented, as shown in Table 6.1 for the two previous application cases.

Table 6.1: Comparison chart of the implementations for both application cases

	Remote Terminal		Embedded Platform	
	User interface	Knowledge base	Protocol stack	Transceiver board
Coffee machine	Dragon Naturally Speaking + jFuzzy	Coffee machine procedures	CanFestival	PCAN-USB
ZigBee network	No special feature	On/off + sensor reading procedures	xbee-api	XBEE Pro S1

As an example, consider a new industrial application that uses either CANOpen or ZigBee. It would not require the implementation of the protocol stack and transceiver board again, because these subcomponents were already added to the system's architecture. Only the user interface and knowledge base would be required in order to remotely control and monitor the new industrial application, and this feature is available for every process automation protocol that would eventually be added to the system using the proposed architecture.

Consequently, the communication layer is preserved from one application to another, while only the control mechanism and interface to the user are modified in each case.

Using the examples above, it is possible to evaluate the achievements of the developed system in comparison to what was defined for the proposed architecture in section 1.2 as objectives. The first achievement is the universal interface that enables seamless access for multiple systems without prioritizing one over another. Starting at the examples depicted in Table 6.1, it can be seen that changes in both application cases are restricted to the some parts of the software architecture. This leaves the software components related to the web services – the remote access interface – unaltered, as shown in Figure 6.5. Moreover, from the characteristics described in previous chapters, the standardized feature of this interface allows the transmission of multiple protocol data from any operating system, as long as it is transmitted over HTTP and uses the predefined structure. The referred interface fulfills then the proposed achievement, as it is constructed using the features described above.

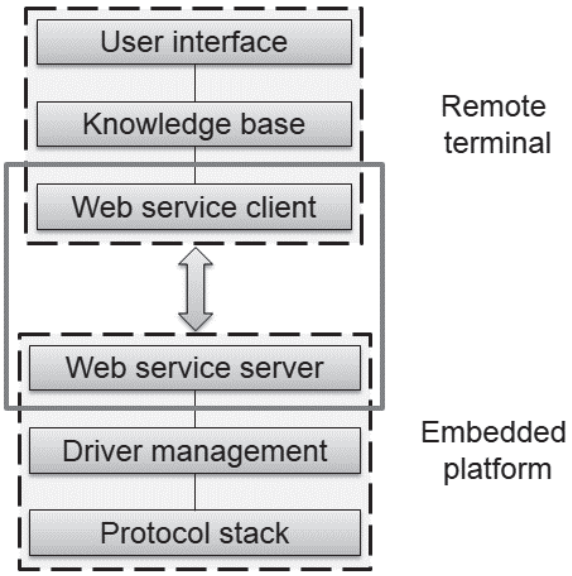


Figure 6.5: Software components with the universal interface highlighted.

The second achievement is the use of communication modules via common connection to allow exchangeability of modules, thus enabling physical access to multiple process automation protocols. The description of these hardware structures – communication modules and common connection – are found in Appendix A. Because the communication modules depend directly of the process automation protocol to which it connects, there is no need to explain their construction. However, the modules must use the common connection shown in Table A.1. These connectors have the standardized I²C, SPI, and Serial buses as possible options to send to and receive data from the transceiver ICs. Furthermore, the USB port can

also be used to connect communication modules, if available. Therefore, the connectors and modules can provide physical access to virtually any process automation protocol, because it is through the buses mentioned above that the software components can reach the physical layer of the communication protocols.

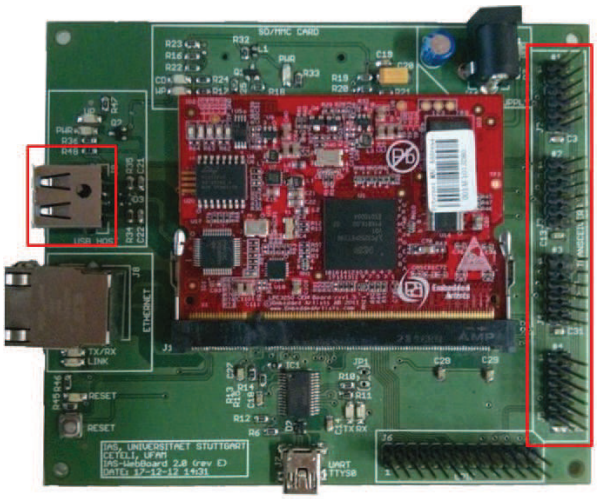


Figure 6.6: Hardware platform with highlighted connections for communication modules.

The final achievement is the development of a platform unites the results of the first and second achievements. This means that it must reduce efforts to enable the proposed methods of remote supervision and control of multiple industrial automation systems. As described in Appendix A and in Chapter 5 , the hardware platform runs Linux operating system and connects to Ethernet and Wi-Fi networks – granting access to HTTP protocol – and also has the connectors shown in Figure 6.6 – granting access to the I²C, SPI, Serial, and USB buses.

6.5 Summary

The application cases, altogether, demonstrate the effort to deploy the solution to different industrial automation systems. In the first two application cases, the changes were made to the remote terminal (user interface and knowledge base) and to the embedded platform (protocol stack and transceiver board). Meanwhile, in the general application, it was explained the process to integrate the prototype with a new automation system, i.e., the evaluations of components and impacts on development.

Next it was shown that the focus of this architecture is the possibility to reuse it in different industrial applications. Thus, if an automation system used either CANOpen or ZigBee protocols, then there would be no need to redo the development process inside the embedded platform, i.e., the communication part of software package would not need to be

developed again, and it would leave only the user interface and knowledge base to be developed. Consequently, the integration process of the system would be reduced.

Finally, it was evaluated the fulfillment of the achievements defined by the objectives of this thesis: the implementation of a universal interface that enables protocol data to transit over HTTP protocol; the implementation of a common interface for multiple communication modules; and the development of a hardware platform that supports both previous achievements.

Chapter 7 Conclusions

In the early chapters of this thesis, some definitions of industrial automation systems were presented and explained to describe the state of these systems in their application environments. It could be seen that many related work are proposing the use of service-oriented architecture, i.e., SOAP-based web services, in supervision systems as the remote access method for gathering data. However, the proposals do not focus on reusing the hardware different application, as shown by the summarized analysis in Table 3.1.

The architecture concept's definition begins at the problem analysis of supervision and control systems. Flexibility, compatibility, and modularity improvements come after following some guidelines about the use of standardized technologies in the communication interfaces, and about the way hardware and software are built. The assertions are summarized in the following characteristics:

- Separation of protocol knowledge and automation system knowledge.
- Standardized communications protocol and data format.
- Standardized low-level data buses.

Based on these assumptions, the proposed solution is formed using a remote terminal and an embedded platform, which interconnect via the remote access interface. Moreover, the automation system is accessed by the embedded platform via the communication bus interface. These devices use a few software and hardware components to successfully meet the demands established by the architecture concept, as described in Chapter 5 .

By standardizing the remote access interface, the solution enables any device to perform the task of remote terminal. The only requirement is to be able to exchange the procedure information over HTTP using the defined XML format. As for the communication bus interface, the use of data buses as SPI, USB, I²C, and Serial limits the number of transceiver boards that can be attached to the embedded platform, but ensures that drivers are always available. Meanwhile, the embedded platform is designed to operate as a gateway between both interfaces, whereas the remote terminal enables the user to monitor and to control the automation system.

Finally, real life applications of the conceived prototype are briefly described in Chapter 6, where two cases were used to demonstrate the proposed improvements and reusability of the

solution. In one application case, the solution was deployed to remotely control a coffee machine using voice inputs, while in the other, to monitor and control a wireless network of sensors and actuators. In both cases it was possible to remotely perform diagnostics and control requests from a smartphone, a tablet, and a computer, using the same Java program configured to the operating systems of each device. Moreover, as a hardware platform, a computer substituted the embedded platform in both cases also running the same Java program calibrated to use files and kernel structures of the respective operating systems.

The main contribution of this thesis is the definition of a hardware and software architecture to perform remote access on automation systems. It enables support for different process automation protocols, thus, proposing a flexible and reusable system to monitor and control industrial automation systems. Despite the cost of 200 euros of electrical components and PCB production, the board, as provided in this thesis, may be used in demonstrators of industrial process models, for it allows software development in different programming languages on its Linux environment. However, usage of this hardware platform in industrial applications requires further EMC testing and software optimization to shorten booting times and to ensure operating system's continuous operation.

During the progression of this thesis, many challenges were encountered. At first the embedded system platforms were evaluated in terms of architecture and interfaces availability. Furthermore, the software mechanisms were also being evaluated in terms of framework availability and implementation-independent flexibility.

The hardware development presented many obstacles to surpass, such as design of high-frequency signal trails and design of PCB with multiple signals layers. It was then followed by some software availability problems, since some drivers were only available to the x86 architecture and had to be compiled to the ARMv5 architecture.

At the end of development, the first application case which the solution was deployed, integrating the embedded platform and remote terminal, was the coffee machine one. This process model is part of the Institute of Industrial Automation and Software Engineering from the University of Stuttgart.

7.1 Future work

There are two types of proposals for future works proposals for this work. At first, there are those ones that suggest integration of more technologies with the prototype, instead of the architecture, in order to improve its performance and functionalities, and thus, intention of

increasing the possibilities of application cases in which it can be integrated with. These proposals are the following:

- The addition of real-time features to the Linux's kernel to enable communication with time-critical industrial automation systems.
- The increase of processing capabilities and hardware IO to convert the embedded platform into a Web-enabled PLC.
- Rewrite the embedded platform's software related to remote access to use the OPC UA specification and enable integration with current MES, ERP software.
- Embedded as many process automation protocols as possible to ensure connectivity for every industrial application.

The other type of proposals suggests the application of the architecture as one element of a larger architecture for flexible automation systems and flexible manufacturing systems. As mentioned during the beginning of this work, this architecture is the basis for working with the evolution of the ISA-95 standard, sometimes called Industry 4.0, which seeks to modify the pyramid structure that is widely spread in the industry. Researches focus on pushing the levels two, three and four of the ISA-95 standard (as seen in Figure 1.1) to cloud computing services, as described in (GIVEHCHI, 2013; LANGMANN, 2013; NEUGSCHWANDTNER, 2013), and the architecture would fit between control devices (level one) and the cloud computing services (other levels), acting as a flexible mechanism to access data from lower levels.

References

ALBRECHT, H.; GROSSE-PLANKERMANN, H. *An infrastructure for browser-located applications in industrial automation*. In: IEEE International Workshop on Factory Communication Systems, p. 373- 376, September 22-24, 2004.

ALBRICH, L.M. M. *GirControl Plus: Manufacturing process control & remote maintenance via Internet*. In: 50th FITCE Congress (FITCE), p. 1-7, Aug. 31-Sept. 3, 2011.

AN186. *SMSC Ethernet Physical Layer Layout Guidelines*. 2008. Available at: <http://www.smsc.com/Downloads/SMSC/Downloads_Public/Application_Notes/an186.pdf> . Accessed at September 21st 2012.

ANTONY, J.; MAHATO, B.; SHARMA, S.; CHITRANSHI, G. *A Web PLC Using Distributed Web Servers for Data Acquisition and Control: Wed Based PLC*. In: International Conference on Information Science and Applications (ICISA), p. 1-4, April 26-29, 2011.

ARDUINO. *Arduino Ethernet*: reference design and schematics. Available at: < http://arduino.cc/en/Main/ArduinoEthernetShield>. Accessed at August 21th, 2012.

KENNEDY, B.; MILLER, D. *PHYTER® Design & Layout Guide*. April 29, 2008. Available at: <www.ti.com/lit/an/snla079d/snla079d.pdf>. Accessed at October 8th 2012.

BEAGLEBOARD. *Beagleboard-xM reference documents*: schematics, layout, bill of materials, and user manual. 2011.

BR&L CONSULTING. *A Tutorial on the ANSI/ISA95 Enterprise/Control*. Available at: <www.brlconsulting.com/Files/2003-09%20IEE%20Cambridge-V03.ppt>. Accessed at September 19th 2013.

BRATUKHIN, A.; SAUTER, T. *Distribution of MES functionalities for flexible automation*. In: 8th IEEE International Workshop on Factory Communication Systems (WFCS), p. 157-160, 18-21 May 2010.

CANFESTIVAL. *CANFestivel framework for CANOpen protocol.* Available at: <http://www.canfestival.org/>. Accesses at December 21st 2012.

CUCINOTTA, T.; MANCINA, A.; ANASTASI, G. F.; LIPARI, G.; MANGERUCA, L.; CHECCOZZO, R.; RUSINÀ, F. *A Real-Time Service-Oriented Architecture for Industrial Automation*. In: IEEE Transactions on Industrial Informatics, v.5, n.3, p. 267-277, Aug. 2009.

DIGI INTERNATIONAL. *Product manual:* XBee™/XBee-PRO™ OEM RF Modules. Minnetonka, 2009.

DOBJANSCHI, V. *Developing Android REST Client Applications.* In: Google I/O 2010. 3, 2010, San Francisco, Proceedings. San Francisco: 2010. Available at: <http://www.google.com/events/io/2010/>. Accessed at November 18th, 2012.

DRAGON MOBILE. *Dragon Mobile Software Development Kit*. Available at: <http://dragonmobile.nuancemobiledeveloper.com/public/index.php?task=home>. Accessed at September 2012.

EAGLE. *Manual for the Easily Applicable Graphical Layout Editor, version 6*. Pembroke Pines, 2011.

EBERHARD, J. *Remote Diagnostic System for an Industrial Coffee Machine*. 2004. 125 p. Dissertation (Master), Institute for Automation and Software Engineering, University of Stuttgart, Stuttgart, 2004.

EMBEDDED ARTISTS. *Getting started with Linux on the LPC3250 OEM board*. Malmö, 2012a.

EMBEDDED ARTISTS. *LPC3250 Development kit v2 reference documents*: schematics and user manual. Malmö, 2012b.

EMBEDDED ARTISTS. *LPC32X0 OEM Board schematics*: revision 1.5. Malmö, 2011.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. 162 p. Thesis (Doctorate), Institute of Software Research, University of California, Irvine, 2000.

FIGUEIREDO, J. M. G.; DA COSTA, J. M. G. *A Concept for an Operational Management System for Industrial Purposes*. In: IEEE International Symposium on Intelligent Signal Processing WISP 2007, p. 1-6, October 3-5, 2007.

FURASTÉ, P. A. *Normas técnicas do trabalho científico*: Explicação das normas da ABNT. 15 Ed, Porto Alegre: s.n., 2009.

HASHIMUKAI, H. *Web based de-facto standard for manufacturing*. In: Proceedings of the 41st SICE Annual Conference, v. 2, p. 905-908, August 5-7, 2002.

IAS. *Industrial Automation Lectures*: Chapter 1 - What is Industrial Automation?. Institute for Industrial Automation and Software Engineering, Universität Stuttgart, Stuttgart, 2010a.

IAS. *Industrial Automation Lectures*: Chapter 2 - Automation Device Systems and Structures. Institute for Industrial Automation and Software Engineering, Universität Stuttgart, 2010b.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

GARGENTA, M. *Learning Android*. Sebastopol: O'Reilly Media, 2011.

GIVEHCHI, O.; TRSEK, H.; JASPERNEITE, J. *Cloud Computing for Industrial Automation Systems - A Comprehensive Overview*. In: 18th Conference on Emerging Technologies and Factory Automation. 10-14 Septemper 2013.

LANGMANN, R.; MEYER, L. *Architecture of a Web-oriented Automation System*. In: 18th Conference on Emerging Technologies and Factory Automation. 10-14 Septemper 2013.

LASTRA, J.L.M.; DELAMER, I.M. *Automation 2.0: Current trends in factory automation*. In: 6th IEEE International Conference on Industrial Informatics, INDIN 2008, p. 1321-1323, 13-16 July 2008.

LOUVEL, J.; TEMPLIER, T.; BOILEAU T. *Restlet in Action*: *Developing RESTful web APIs in Java*. Shelter Island: Manning Publications, 2012.

NEUGSCHWANDTNER, G.; REEKMANS, M.; LINDEN, D. van der. *An open automation architecture for flexible manufacturing*. In: 18th Conference on Emerging Technologies and Factory Automation. 10-14 Septemper 2013.

OZDEMIR, E.; KARACOR, M. *Mobile phone based SCADA for industrial automation*, ISA Transactions, v. 45, n. 1, p. 67-75, January 2006.

PANDABOARD. *Pandaboard ES reference documents*: schematics, layout, bill of material, and user manual. Available at < http://pandaboard.org/content/resources/references>. Access at: August 8th, 2012.

PARVARESH, P. *Konzeption und Implementierung einer Software zur Sprachsteuerung eines industriellen Kaffeeautomaten unter Berücksichtigung vager Aussagen*. 2012. 76 p. Dissertation (Master), Institute for Automation and Software Engineering, University of Stuttgart, Stuttgart, 2012.

PEAK-SYSTEM. *User manual v2.1.4:* PCAN-USB USB-to-CAN interface. Darnstadt, 2012.

PFEIFFER, O.; AYRE, A.; KEYDEL, C. *Embedded Networking with CAN and CANOpen*. Greenfield: Copperhill Media, 2008.

PRÜTER, S.; GOLATOWSKI, F.; TIMMERMANN, D. *Adaptation of resource-oriented service technologies for industrial informatics*. In: IECON Annual Conference of Industrial Electronics, p. 2399-2404, November 3-5, 2009.

RICHARDSON, L.; RUBY, S. *RESTful Web Services*. Sebastopol: O'Reilly Media, 2007.

ROGERCOM. *Instruction manual:* RCOM-HOMEBEE. March 15th 2013.

SANDOVAL, J. *RESTful Java Web Services: Master core REST concepts and create RESTful web services in Java*. Birmingham: Packt Publishing, 2009.

SAUTER, T.; SOUCEK, S.; KASTNER, W.; DIETRICH, D. *The Evolution of Factory and Building Automation*. IEEE Industrial Electronics Magazine, v. 5, n. 3, p. 35-48, September 23rd, 2011.

STARKE, G.; KUNKEL, T.; HAHN, D. *Flexible collaboration and control of heterogeneous mechatronic devices and systems by means of an event-driven, SOA-based automation concept*. In: IEEE International Conference on Industrial Technology (ICIT), p. 1982-1987, 25-28 February. 2013.

STOPPER, M.; KATALINIC, B. *Service-oriented Architecture Design Aspects of OPC UA for Industrial Applications*. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, Hong Kong, v. 2, March 18 - 20, 2009.

ROSÁRIO, J. M. *Princípios de Mecatrônica*. São Paulo: Prentice Hall, 2005.

TRUONG, N.-V.; VU, D.-L. *Remote monitoring and control of industrial process via wireless network and Android platform*. In: International Conference on Control, Automation and Information Sciences (ICCAIS), p. 340-343, November 26-29, 2012.

UNGER, K. *Manufacturers 'Needs Not Changing–But Acronyms Are*. In: Industrial Computing, p. 46-48, October 3rd, 2001.

WORLD WIDE WEB CONSORTIUM. *Simple Object Access Protocol (SOAP) 1.1*. 2000. Available at: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Accessed at March 23rd, 2012.

WORLD WIDE WEB CONSORTIUM. *Architecture of the World Wide Web, Volume One*. 2004a. Available at: <http://www.w3.org/TR/2004/REC-webarch-20041215/>. Accessed at March 23rd, 2012.

WORLD WIDE WEB CONSORTIUM. *Web Services Architecture*. 2004b. Available at: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>. Accessed at March 23rd, 2012.

WANG, L. *Factory Automation Systems: Evolution and Trends*. In: IEEE AUTOTESTCON Proceedings, p. 880-886. October 16-17, 2002.

WONDERWARE. *Wonderware's ArchestrA™ technology and the ISA-95 standard are helping SA companies ease successfully into MES*. Available at: <http://www.wonderware.co.za/live/content.php?Item_ID=475>. Accessed at September 2012.

XBEE. *XBEE Application Programming Interface for XBee/Xbee-PRO ZigBee OEM RF modules*. Available at: <https://code.google.com/p/xbee-api/>. Accessed at November 7th, 2012.

Appendix A - Hardware prototype

The hardware prototype described in this segment provides an implementation of the software and hardware architectures described in Chapter 5, more precisely the implementation of the embedded platform component, and its hardware layout and schematics were done using EAGLE 6.2 from CADSoft (EAGLE). It follows the guidelines from datasheets provided by IC manufacturers and references from other hardware platforms, which are distributed under Creative Commons Share Alike licenses. The four parts of the prototype, the CPU board, the motherboard, the transceiver boards, and the remote connections, are going to be described individually regarding the electronic components that compose their schematics, as well as the technologies involved in its realization.

CPU board

The CPU board is where the processor, working memory, storage memory, and peripherals are found. In the prototype, this hardware part uses OEM products with the SO-DIMM form factor, as shown in Figure A.1, since these type of equipment are more common amongst the ones available in the market. The LPC3250 OEM Board v1.5 from the company Embedded Artists, shown in Figure A.2, was selected because of the amount of peripherals available in a small 200-pin SO-DIMM form factor. This board fulfills the requirements established in section 5.3.1 stating that this part shall have an ARM processor, RAM memory, persistent storage, Ethernet, and USB PHY. The SO-DIMM connector allows the interchangeability of this PCB in case of malfunction or, if required, in case of upgrades. Its features are as follows:

- NXP's ARM926EJ-S LPC3250 microcontroller.
- 64MByte external DDR SDRAM.
- 128MByte NAND FLASH.
- 4MByte SPI-NOR FLASH.
- 100/10Mbps Ethernet interface based on DP83848 ETH-PHY.
- On-board ISP1301 USB OTG transceiver.
- 13.000 MHz and 32.768 kHz crystals, but internal PLL creates frequencies from 208MHz to 266MHz.
- SO-DIMM with 1.8V keying.
- Linear voltage supply of 3,3V.

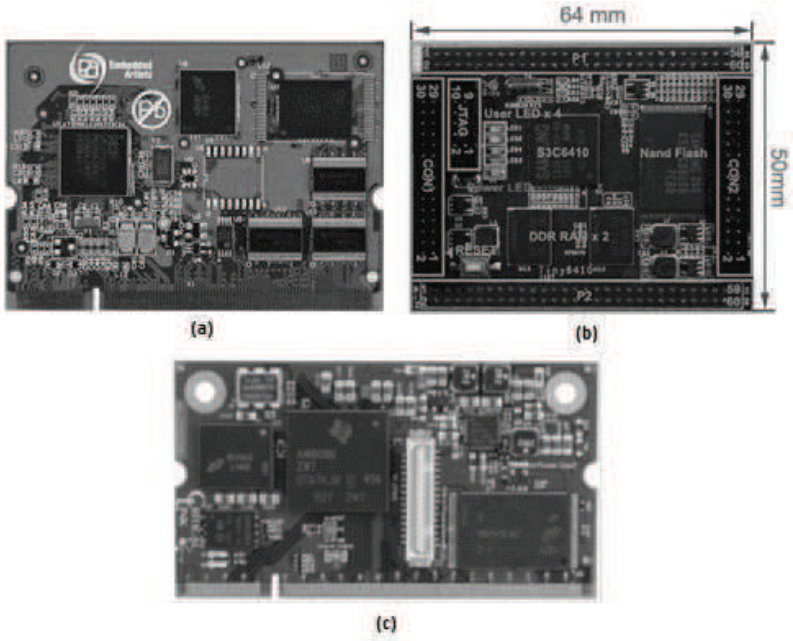


Figure A.1: OEM Boards examples: (a) Embedded Artists' EA3250, (b) FriendlyArm's Tiny6410 and (c) Critical Link's MityARM-1808.

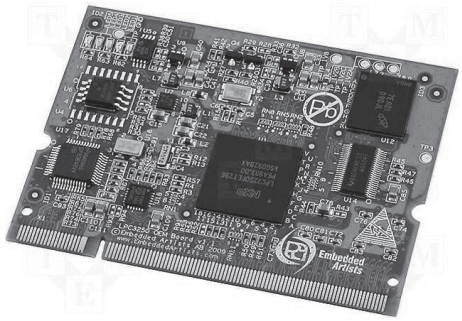


Figure A.2: Embedded Artists' LPC3250 OEM board.

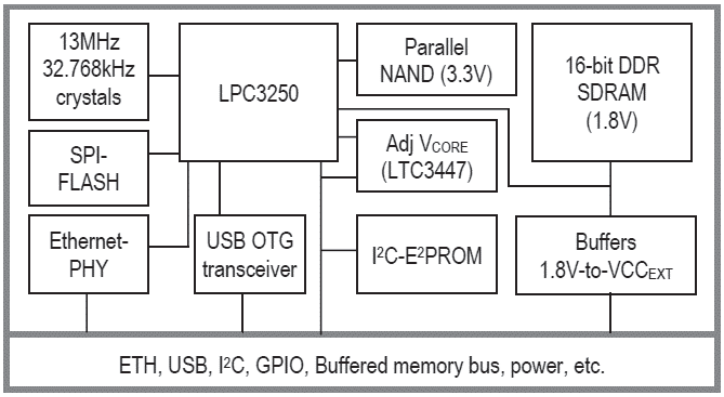


Figure A.3: OEM board's components diagram.

The layout of this board is not shown in this section, because of its complexity derived from the RAM memory and NAND Flash memory ICs, as well as the 7-page electric schematics. However a simpler diagram of its components is shown by the diagram in Figure A.3.

Motherboard

In order to design the motherboard PCB, the most important requirements are the dimensions of the CPU board. According to the datasheet provided by the manufacturer, the CPU board has 67.6 mm of width, 49.5 mm of height, and 4.8 mm of depth, as shown in detail in Figure A.4 and Figure A.5. These dimensions are needed to define the clearance between the OEM board and the motherboard.

The hardware design of the motherboard was based on other hardware platforms with open schematics, namely, the Beagleboard (BEAGLEBOARD), the Pandaboard (PANDABOARD), the Arduino Ethernet (ARDUINO), and the LPC3250 Development Kit v2 (EMBEDDED ARTISTS, 2012b). These references were used to find part numbers for the electronics circuit commonly deployed in marketable hardware platforms, as well as to identify methods employed to draw the layout of components and signal traces.

Then, the specification of each subpart of the motherboard is realized around the signals available at CPU board's SO-DIMM connector, aiming to attend to all specifications from section 5.3.1 . Note that small components such as capacitors and resistors are not described for each subpart as they used in large numbers.

The power supply is responsible for powering the motherboard. The manufacturer's specification states that the CPU board requires 3.3V, 500mA, but to power USB Wi-Fi adapters it is required more current and voltage, as these devices are specified to work with 5V, 500mA. Consequently, a 5V, 2A power adapter is shall be used to power not only the CPU board and the Wi-Fi adapter, but also transceiver board and any auxiliary circuit connected to the GPIO. The selected components to this subpart are:

- One low-dropout voltage regulator in TO-263 packing, such as MIC29150 or SPX2950.
- One jack connector with center pin positive.
- One LED for power indication.

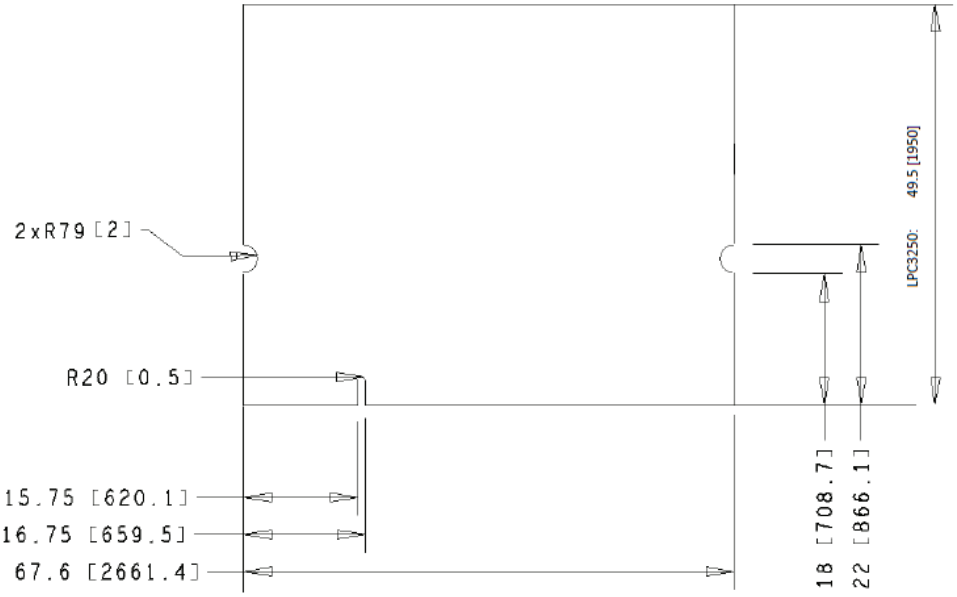


Figure A.4: OEM board's width and height measurements (frontal view).

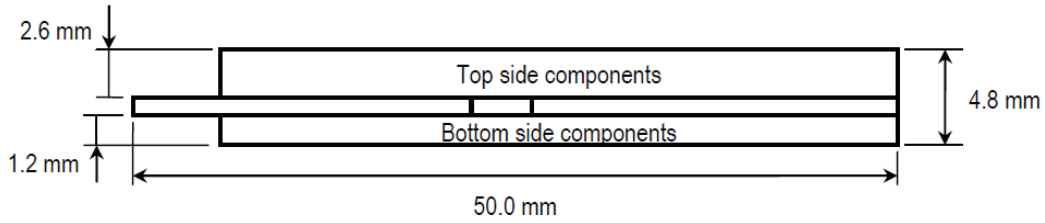


Figure A.5: OEM board's depth measurements (side view).

Since the power adapter provides 5V to any circuit that requires it, such as the USB, the voltage regulator provides 3,3V to the OEM board and other circuits to a maximum of 1,5A of drainage.

The Ethernet subpart of the motherboard consists only of connector, because the CPU board already has the Ethernet PHY in the Reduced Media Independent Interface (RMII) mode onboard. Thus, the motherboard shall have the following components:

- One RJ45 connector in 90°.
- Two LEDs for link and activity indicators.
- One magnetic transformer.

Next, there is the USB subpart of the motherboard. Similarly to the Ethernet, the CPU board also has the USB PHY onboard, demanding only circuitry for USB bus' power supply, ESD protection, and connector, which implies that the motherboard has the following components:

- One USB type A horizontal connector with 4 contacts and through-hole fixation.

- One electrostatic discharge (ESD) protection IC, such as TPD4S012 or PRTR5V0U2X.
- One power driver to control USB's power consumption, such as LM3529-L.

The power driver LM3529-L is the type of IC used to manage the voltage and current drainages, keeping the consumption to a maximum of 500mA, as states the USB specification.

Table A.1: Transceiver interface's pin configuration

UART_TX	1	2	UART RX
I²C_SDA	3	4	I²C_SCL
SPI_MOSI	5	6	SPI_MISO
SPI_CLK	7	8	SPI_SSEL
+5V	9	10	GPIO
+3V3	11	12	GND

As mentioned in section 5.3.1 , the common interface uses USB, SPI, I²C, and Serial to connect different transceiver boards to the CPU Board. The USB, though, uses a different connector and, if a Wi-Fi adapter is used, the transceiver board shares the USB connection via a hub. Thus, a 12-pin connector is proposed to enable access to the remaining buses simultaneously, and the pinout is shown in Table A.1. Despite availability of four connectors in the motherboard, it won't be possible to connect, at the same time, multiple SPI transceivers because only one SSEL (CS) pin is available. The presence of a GPIO in this connector is due to the fact that some transceivers chips may have a pin for controlling the connection with the communication bus, thus giving more control to the application.

The LPC3250 board connects to the motherboard through a standard 200-pin SO-DIMM connector that fits the form factor of the PCB. Embedded Artists, the manufacturer, specifies that the connector must have, aside the 200 pins, 1.8V keying and the minimal distance clearance from the motherboard's signal layer to avoid unwanted contacts. This connector is available from manufacturers such as Tyco and Foxconn, though Embedded Artists recommends the 0-1473005-4 connector from Tyco Electronics, show in Figure A.6.

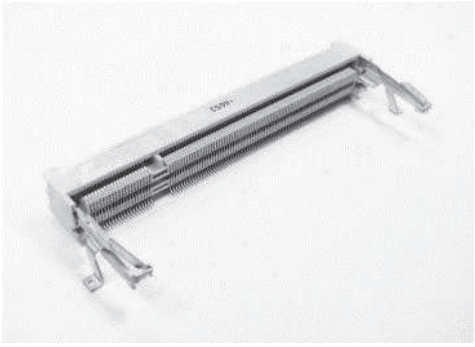


Figure A.6: SO-DIMM connector from Tyco Electronics.

Even though the main subparts of the motherboard have been specified, the software cannot be uploaded to the embedded platform without the debugging interface. It consists of a USB-to-Serial connection used to upload binaries to the embedded platform, as well as for accessing the Linux operating system's terminal. The components required for this circuit are the following:

- One FT232RL USB to Serial IC.
- One ESD protection IC, such as TPD4S012 or PRTR5V0U2X.
- Three LED for power indication, RX and TX data lines.
- One USB type mini-B horizontal connector.

The GPIO and the memory card interface are extra circuits, as mentioned in section 5.3.1 , which were added to enhance the capabilities of the embedded platform. The first requires no components, just a pin header connector. Meanwhile, the second requires a couple of transistors to drive the power and the standard SD/MMC connector, since the OEM board also supports these devices.

Based on the components specification above, the schematics of the motherboard is divided in four pages, shown in Appendix B. Thus, the layout of the motherboard is designed to have four layers, in a form factor with 116, 06 mm wide, 100 mm high, and 1.536 mm thick. The PCB was manufactured by the Fischer-Leiterplatten GmbH Company using the industry standard FR4 material.

The layers of the board were designed according to high-frequency signal traces, which require isolation and shielding from other traces because of magnetic induction. This technique was used to trace only Ethernet and USB differential traces (AN186; KENNEDY, 2009), for they carry electric signals at 100MHz and 480MHz, respectively, whereas remaining signals work at much lower frequencies. The traces are distributed among four layers: layer 1, Ethernet and USB high-frequency signals traces, ground shielding/distribution plane, low-frequency

signal, and power distribution traces; layer 2, ground shielding/distribution plane; layer 3, 5V and 3,3V power distribution planes, and ground shielding/distribution plane; layer 4, low-frequency signal traces, power distribution traces, and ground shielding/distribution plane.

The finished embedded platform's hardware is seen in Figure A.7, where the red PCB is the CPU board and the green PCB is the motherboard. The motherboard's subparts are identified in Figure A.8: (1) SO-DIMM connector, (2) RJ-45 connector, (3) USB type A connector, (4) debugging interface, (5) GPIO, (6) common interface, (7) Power jack for 5V power supply, and (8) memory card interface connector.

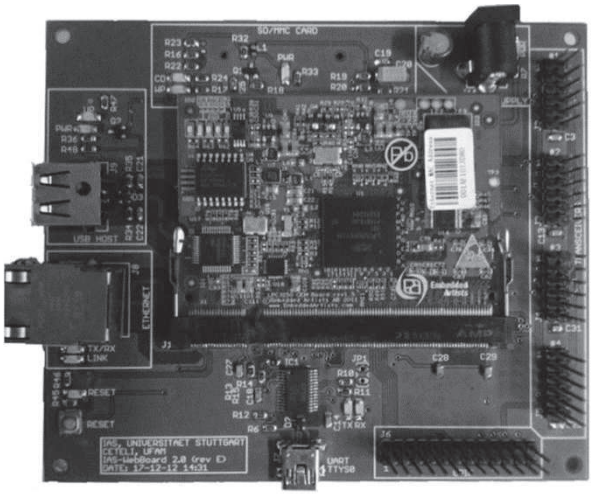


Figure A.7: The embedded platform, top view.

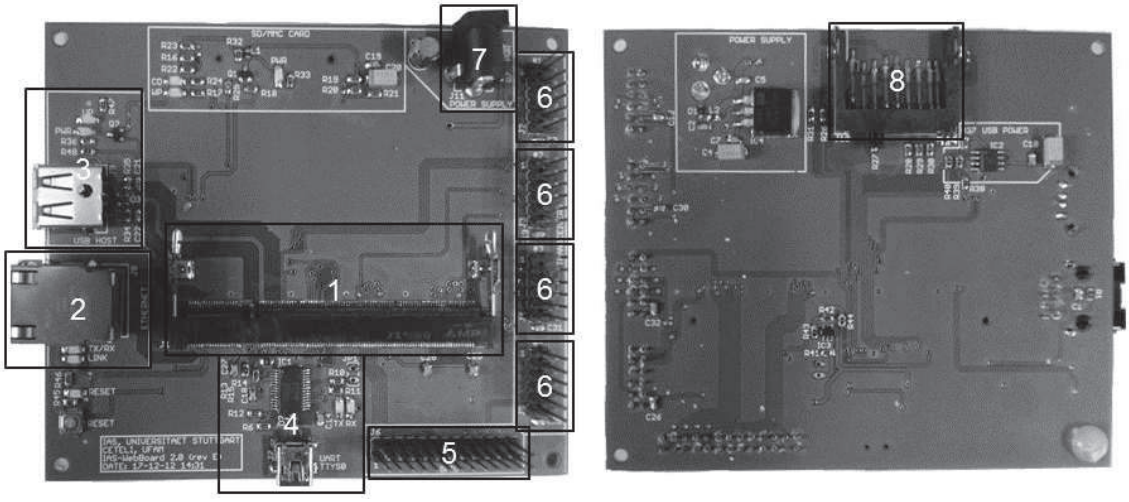


Figure A.8: The motherboard, top and bottom views.

Transceiver board

The use of these small PCB depends strictly on the process automation protocol to which it is desired to connect the embedded platform to. Thus, the hardware realization presented in this section is not the only possible one, as there are commercially available adapters, such as the Peak System's PCAN-USB and the Digi International's XBee PRO RF module. In this section is proposed the transceiver boards for CAN, RS232 and RS485, where each single board has a transceiver IC responsible for converting data from both the SPI's and the UART's serial buses to the signal level of the respective bus voltage level, and vice versa.

Besides both power supply voltage provided in the common interface connector, the transceiver ICs must have serial input (Rx) and output (Tx) tolerant to 3,3V or 1,8V, because that is the voltage which the UART, SPI, and I²C interfaces operate. If a transceiver IC does not attend to this requirement, a voltage level translator, such as TXS0104E, must be used as a two-way translator to voltage levels of 5V.

The CAN2.0 protocol specification (PFEIFFER, 2008) defines that the physical layer is governed by two IC, the controller and the transceiver. Consequently, the Texas Instruments' SN65HVD234 transceiver, together with Microchip's MCP2515 CAN Controller, are selected to be part of the transceiver board for CAN protocol. Following the respective manufacturers' suggestions of circuit for both IC, the assembled circuit for the CAN transceiver board is drawn as in Figure A.9.

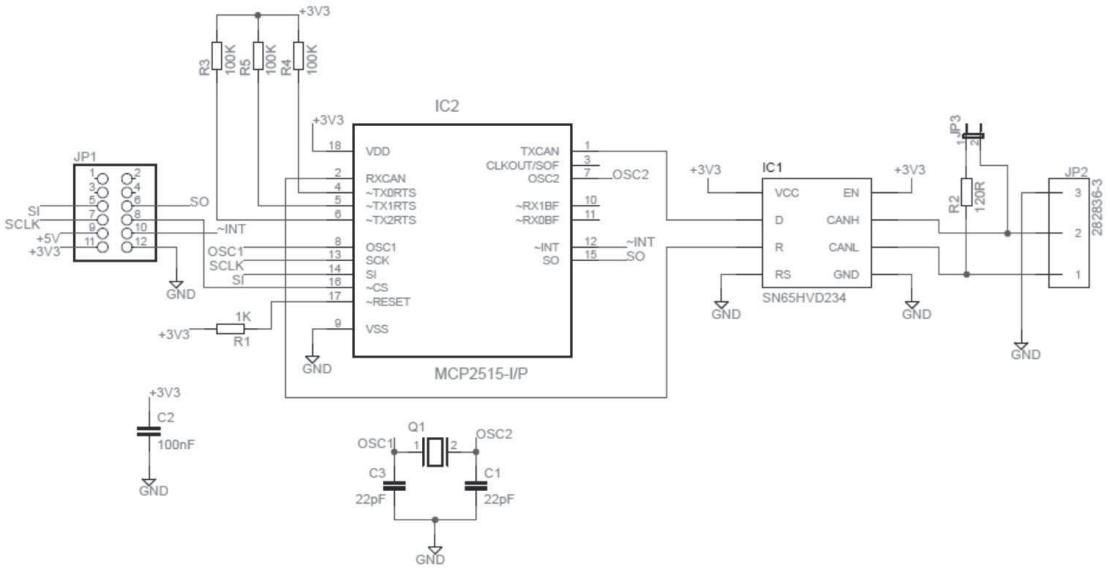


Figure A.9: CAN transceiver board using SPI interface.

Even though the RS232 is a standard from 1969, it is still used as a serial communication method, acting as the physical layer of some protocols, such as Modbus RTU. Thus, the MAXIM's MAX3221 is selected as the transceiver IC for its 3,3V tolerance. The assembled circuit, as suggested by the manufacturer, is shown in Figure A.10.

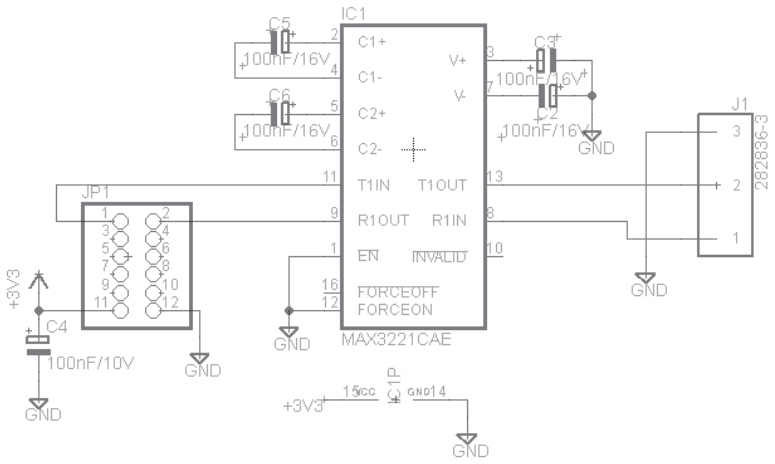


Figure A.10: RS232 transceiver using UART's serial interface.

The RS-485 standard is used in communications as physical layer of some industrial communications protocols, such as Modbus RTU and Profibus DP. Thus, for a RS-485 transceiver board the MAXIM's MAX13431E IC was selected. And as suggested by the manufacturer, the circuit is shown in Figure A.9.

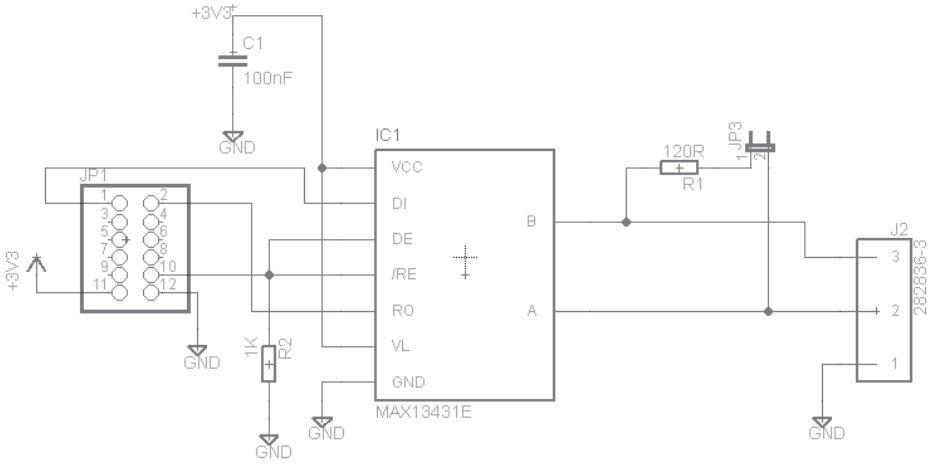


Figure A.11: MAXIM'S MAX134301E RS485 transceiver.

Remote connections

The devices used in the remote connections are responsible for providing the physical medium of the communications between the remote terminal and the embedded platform, which

may be through either Ethernet or Wi-Fi networks. The motherboard provides, by itself, the physical layer for Ethernet communications via the RJ45 connector. Therefore, only a network cable is required to enable this type of remote connection. However, the Wi-Fi network requires USB adapters to be attached to the embedded platform in order to access enable this type of connections. Schematics and layout of these devices are completely unknown, as they are commercial products available in the market, but they must follow specifications from the regulatory organizations for USB and Wi-Fi devices, respectively USB-IF and Wi-Fi Alliance, which are responsible for evaluating the compliance of these devices with the IEEE's standards. More specifically, the USB adapters must be certified to IEEE 802.11b/g/n standards, as are most smartphones, tablets, and computers.

Appendix B - Motherboard schematics

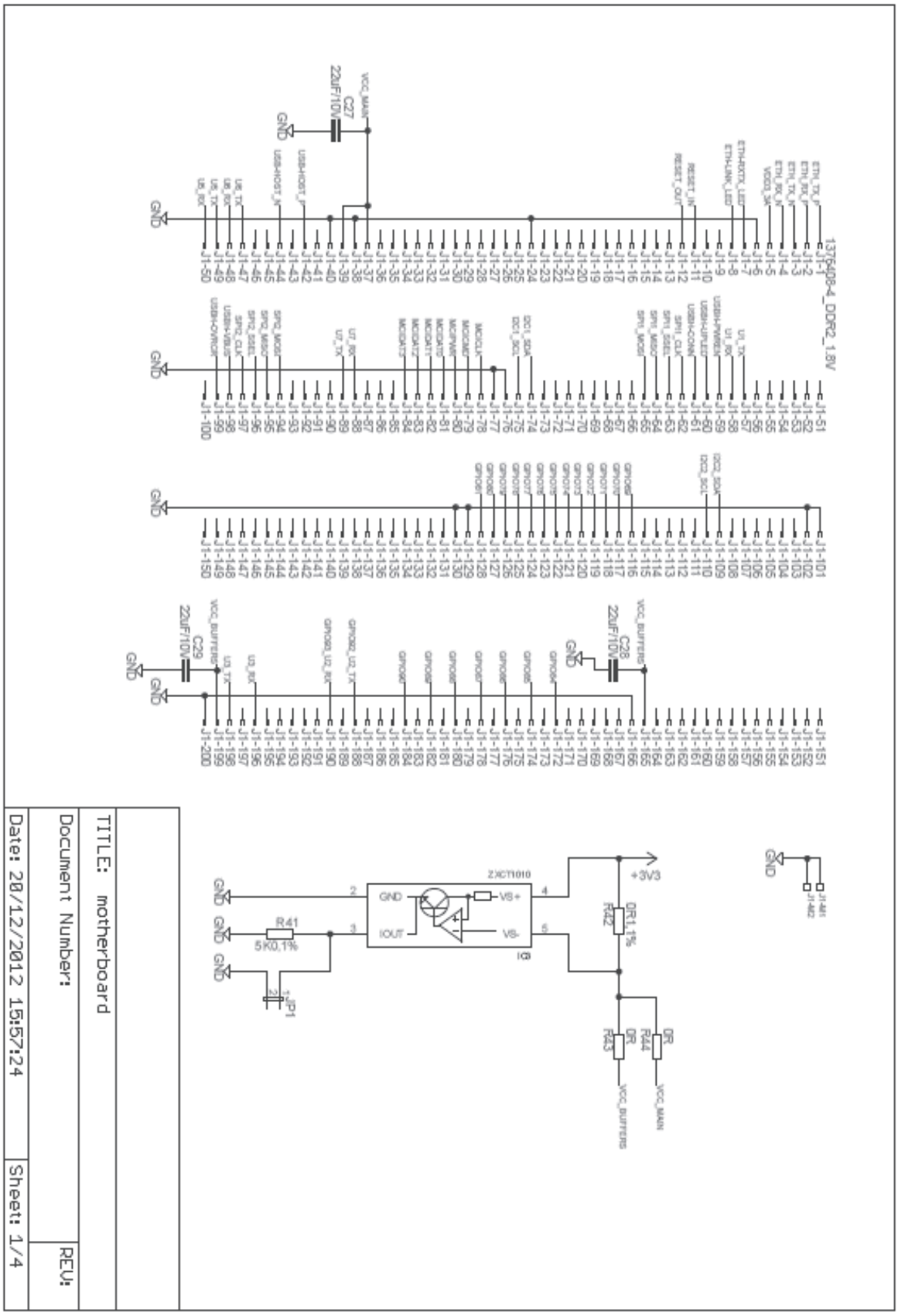


Figure B.1: Motherboard schematics, page 1: SODIMM connector and current measurer.

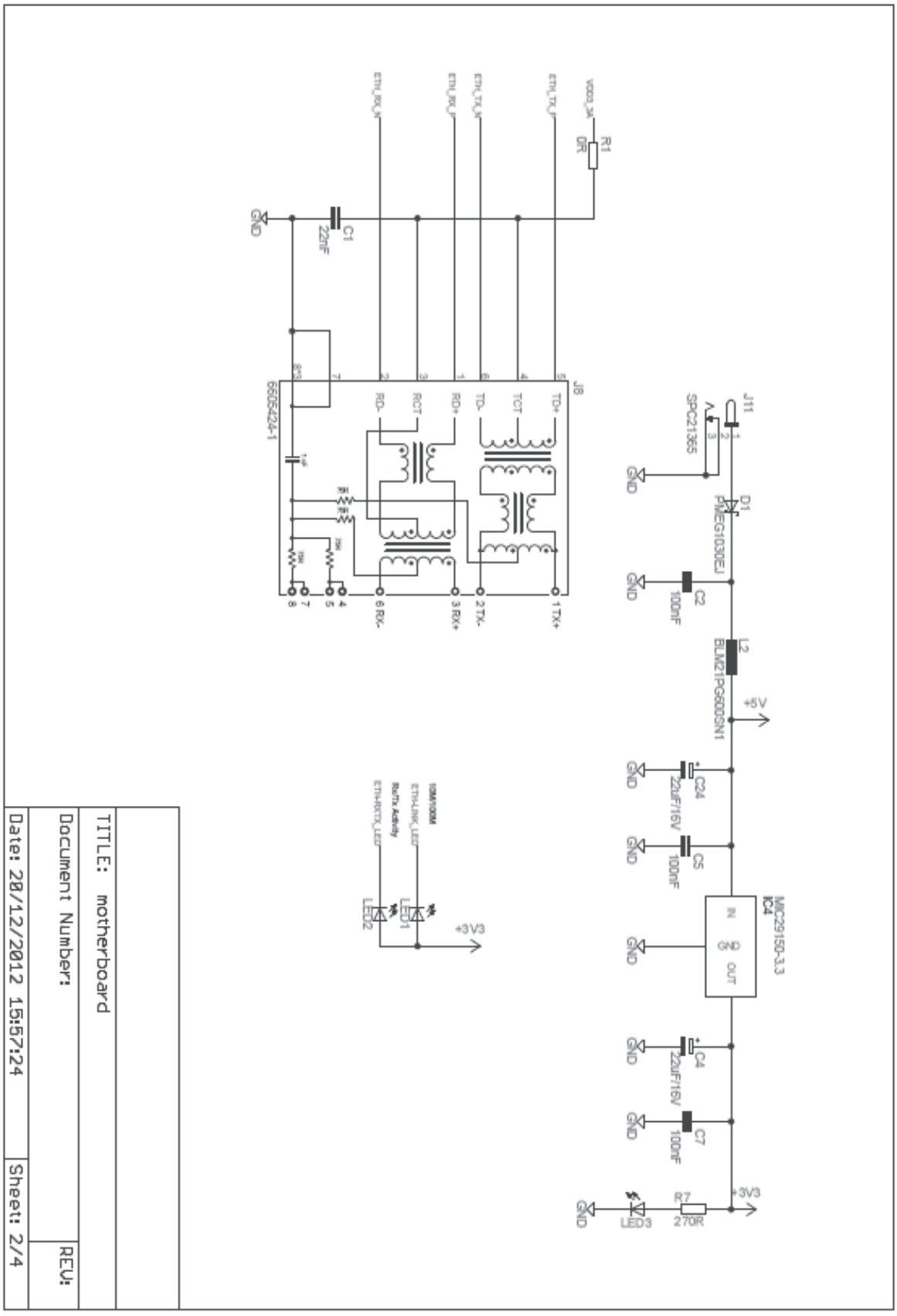


Figure B.2: Motherboard schematics, page 2: power supply and Ethernet connector.

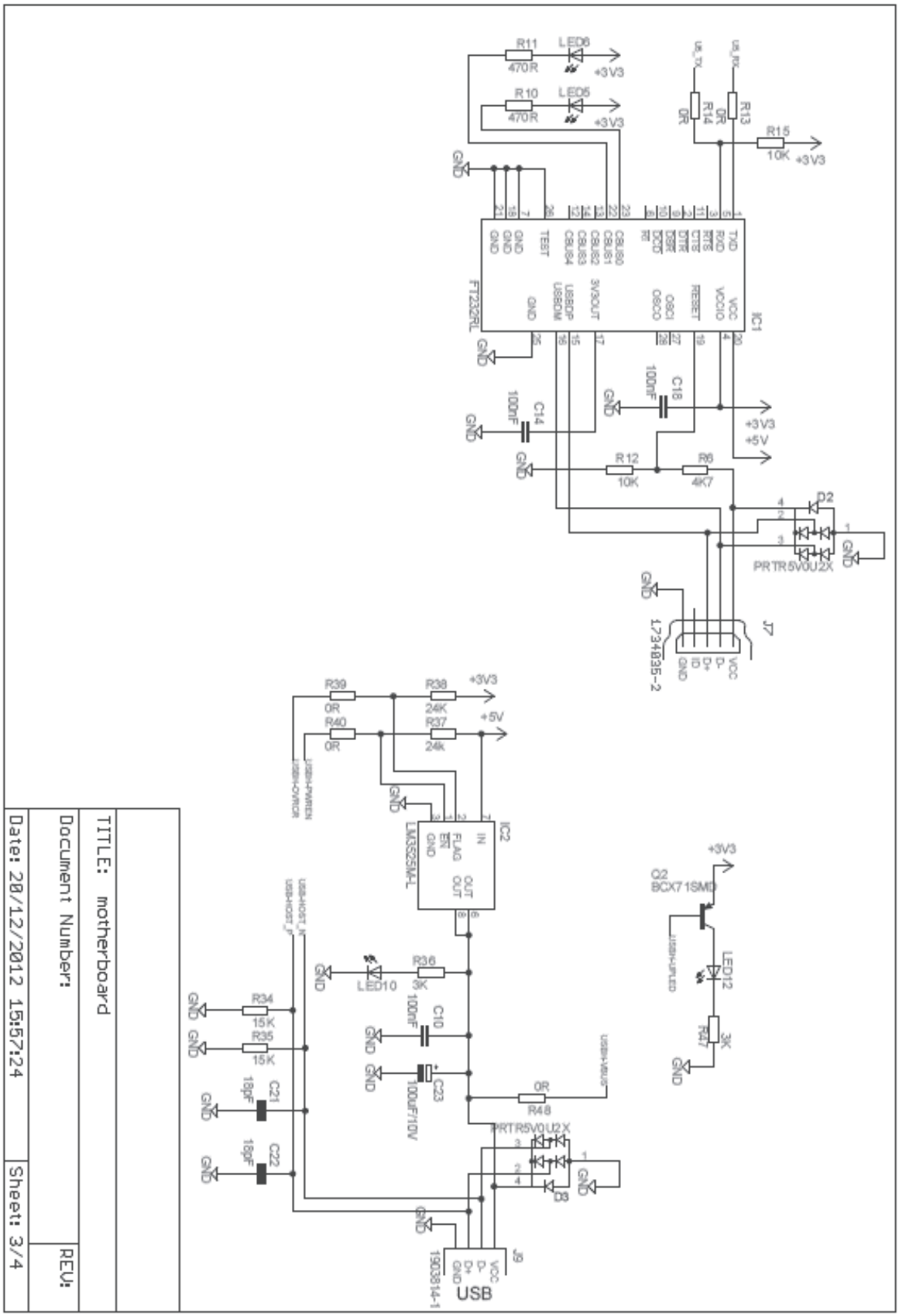


Figure B.3: Motherboard schematics, page 3: USB host and USB debugging port.

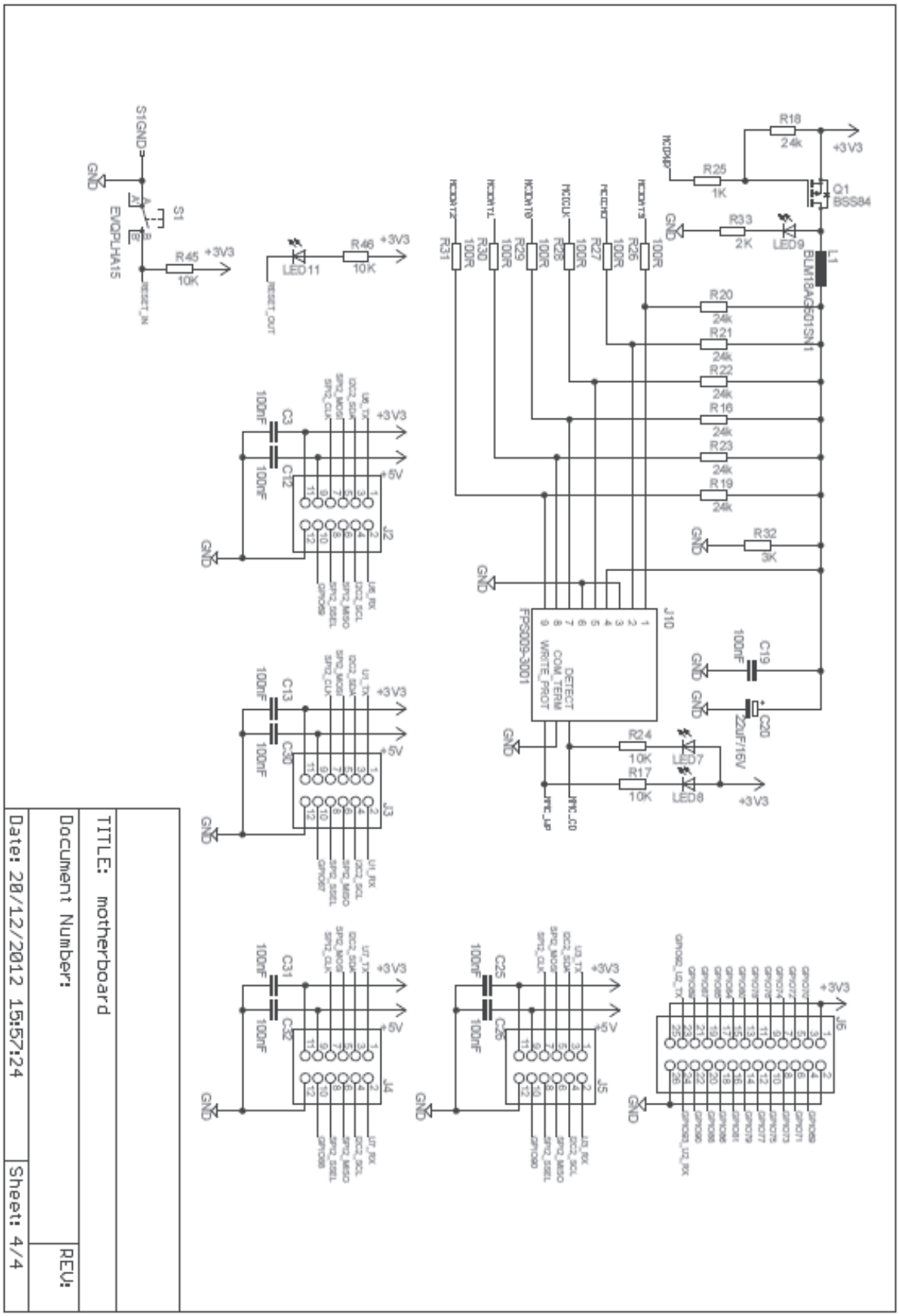


Figure B.4: Motherboard schematics, page 4: GPIO, SD/MMC card interface and transceiver interfaces.