Departamento de Computación

Modular Framework for Ambient Intelligence Systems

Tesis que presenta

MC. Rafael Baquero Salaquardova

Para obtener el Grado de

Doctor en Ciencias

en

Ciencias de la Computación

Director de la Tesis:

Dr. José Guadalupe Rodríguez García.

México, D.F. Enero 2014

Acknowledgements

Many people have supported and helped me throughout the development of this work. Above all I wish to thank my wife and daughters, Paty, Aleksandra and Karolina for their unconditional support and love. Girls, without your love I would have never been able to do this.

I also have an enormous debt to my parents, Mrs. Libuse Salaquardova de Baquero and Dr. Rafael Baquero. Thank you for all your help and love throughout these years and all my life. I also want to thank my in laws, Ing. Amalia Hajducek B. and Ing. Angel Tellez W. Thank you for all your support and encouragement.

I wish to thank my PhD advisor Dr. José Guadalupe Rodríguez G. Thank you José for all your guidance, advice and encouragement. I also want to thank all my classmates, professors and friends for their support and help. Thanks guys!

Finally, I wish to express my gratitude to CONACyT for their support to perform this work.

Table of Contents

Table of Figures

Glossary of Abbreviations

2PC	Two-Phase Commit Protocol
6LowPAN	IPv6 over Low-power PAN
AAL	Ambient Assisted Living
ACID	Atomic, Consistent, Isolated, and Durable
ACK	Acknowledge
AI	Artificial Intelligence
AL	Application Layer
AmI	Ambient Intelligence
APL	Application
APS	Application Support
ASHRAE	American Society of Heating, Refrigerating and Air- Conditioning Engineers
BA	Building Automation
BAS	Building Automation System
BPEL	Business Process Execution Language
bps	Bits per Second
BPSK	Binary Phase Shift Keying
CMPS	Component Side Part
CMPS-CR	CMPS Component Registration Module
CMPS-EQ	CMPS Local Event Queue

CMPS-MD	CMPS Message Dispatcher/MEI Supervisor
CMPS-MR	CMPS Message Receiver
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CS	Component Supervisor
CSMA	Carrier Sense Multiple Access
CTRL	Controller
DBMS	Database Management Systems
DCE	Distributed Computing Environment
DCOM	Distributed Component Object Model
DCS	Distributed Control Systems
DL	Data Link Layer
DS	Distributed System
DTI	Data Type Id field
DTP	Distributed Transaction Processing
EC	Execution Control
ECC	Execution Control Chart
ECM	External Communications Modules
EHS	European Home System
EIB	European Installation Bus
FBN	Function Block Network

FFD	Full Function Device
FM	Function Module
FMR	Function Module Repository
FSK	Frequency Shift Keying
FSO	Full Scale Output
FTI	Failure Type Id
GDP	Gross Domestic Product
HA	Home Automation
HCI	Human Computer Interface
HL7	Health Level 7
HTN	Hierarchical Task Network
HVAC	Heating, Ventilation and Air Conditioning
ICT	Information and Communication Technology
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
iHCI	Implicit Human Computer Interaction
IP	Internet Protocol
IS	Intelligent Systems
ISM	Industrial, Scientific and Medical
ISO	International Organization for Standardization

ISTAG	Information Society and Technology Advisory Group
IT	Information Technology
LAN	Local Area Network
LSIE	Large Scale Intelligent Environment
mA	Miliampere
MAC	Medium Access Control
MCI	Middleware Communications Interface
MDLW	Middleware
MEI	Maximum Event Interval
MFR	MAC Footer
MHR	MAC Header
MHz	Megahertz
MOM	Message Oriented Middleware
MPDU	MAC Payload Data Unit
MSAC	Module and Sensor/ Actuator Catalog
MSAD	Module and Sensor/ Actuator Description record
MSDU	MAC Service Data Unit
NI	Number of Interests
NL	Network Layer
NP	Number of Provides
NWK	Network

ODP	Open Distributed Processing
OOM	Object-Oriented Middleware
O-QPSK	Offset Quadrature Phase Shift Keying
OSGi	Open Services Gateway initiative
OSI	Open Systems Interconnection
PDA	Personal Digital Assistant
PHR	Physical Header
PHY	Physical
PLC	Programmable Logic Controller
PM	Procedural Middleware
PPDU	Physical Protocol Data Unit
PSDU	Physical Service Data Unit
PSI	Pounds per Square Inch
QoS	Quality of Service
REI	Register Event Id
RF	Radio Frequency
RFD	Reduced Function Device
RFID	Radio Frequency Identification
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SA	Sensor, Actuator

SAM	SOPRANO Ambient Middleware
SD&I	Service Discovery and Interaction
SFCS	Store and Forward/Component Supervising
SFCS-CR	SFCS Component Register
SFCS-CS	SFCS Component Supervisor
SFCS-EQ	SFCS Event Queue
SFCS-GC	SFCS Garbage Collector
SFCS-MD	SFCS Message Dispatcher
SFCS-MR	SFCS Message Receiver
SFSK	Spread Frequency Shift Keying
SHR	Synchronization Header
SIP	Session Initiation Protocol
SLP	Service Location Protocol
SMS	Short Message Service
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TL	Transport Layer
TM	Transactional Middleware
TP	Transaction Processing
TTL	Time-To-Live

UbiComp	Ubiquitous Computing
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UIA	User Input Analyzer
UPnP	Universal Plug and Play
VDC	Volts Direct Current
WPAN	Wireless Personal Areas Network
WS-Discovery	Web Services Discovery
ZDO	ZigBee Device Objects

Abstract

Ambient Intelligence (AmI) is a compelling vision of smart environments that are reactive to people and able to make our actions safer, more efficient, more informed, more comfortable or simply more enticing. In this vision our environments will be embedded with visual, audio or many other types of sensing systems, pervasive devices, and networks that can perceive and react to people, sense ongoing human activities and proactively respond to them.

During the last decade an explosive growth in the technologies required to implement Ambient Intelligence systems has occurred. However, due to the multidisciplinary nature of Ambient Intelligence systems and the distinct requirements of different user groups, integrating these developments into full-scale systems is not an easy task. To facilitate the development of AmI systems several frameworks have been, or are currently being, developed but as of yet none of the proposed frameworks enjoys widespread use. One of the main drawbacks of currently available AmI frameworks is their complexity and the learning curve required to apply them in a specific project.

In this thesis work we propose a minimalist modular framework for the development of Ambient Intelligence systems based on the function module abstraction used in the IEC 61499 standard for distributed control systems. This framework allows for the development of AmI systems through the integration of modules loosely joined by means of an event-driven middleware and a Module and Sensor/ Actuator catalog. The modular design of the framework allows the development of AmI systems which can be easily customized to a wide variety of usage scenarios.

Resumen

La Inteligencia Ambiental (Ambient Intelligence, AmI) es un atractivo concepto de entornos inteligentes que reaccionan a los usuarios y que hacen más seguras, más eficientes, mejor informadas, más cómodas o simplemente más agradables nuestras actividades. En esta visión nuestros entornos se encuentran dotados con dispositivos sensores visuales, de audio, y de otros tipos, dispositivos pervasivos, y redes que pueden percibir y reaccionar a las personas, determinar el desarrollo de actividades humanas y responder de forma proactiva a ellas.

En la última década se ha producido un avance vertiginoso en las tecnologías necesarias para la implementación de sistemas AmI. Sin embargo, debido a la naturaleza multidisciplinaria de los sistemas de Inteligencia Ambiental y a los diversos requerimientos de los diferentes grupos de usuarios, integrar estos desarrollos en sistemas completos no es tarea sencilla. Para facilitar el desarrollo de sistemas AmI se han desarrollado, o se encuentran en desarrollo, varios frameworks pero hasta el momento no se ha difundido el uso de ninguno de ellos. Una de las principales desventajas de los frameworks AmI disponibles en la actualidad es la curva de aprendizaje requerida para emplearlos en algún proyecto concreto.

En esta tesis proponemos un framework modular minimalista para el desarrollo de sistemas AmI basado en la abstracción de módulos funcionales empleada en el estándar IEC 61499. Este framework permite el desarrollo de sistemas AmI mediante la integración de módulos levemente acoplados unidos mediante un middleware basado en eventos y un catálogo de Módulos y Sensores/ Actuadores. El interés en el diseño modular del framework es permitir el desarrollo de sistemas AmI que pueden ser fácilmente personalizados a una amplia variedad de escenarios.

1 Introduction

In this chapter Ambient Intelligence is described along with some of the challenges that must be overcome for widespread deployment of AmI systems. An overview of the proposed Modular Framework for Ambient Intelligence Systems is also provided.

1.1 Ambient Intelligence. The Vision

Ambient Intelligence (AmI) is a vision of smart environments that are reactive to people and able to make our actions safer, more efficient, more informed, more comfortable or simply more enticing. In this vision our environments will be embedded with different types of sensing systems, pervasive devices and networks that can perceive and react to people, sense ongoing human activities and proactively respond to them [1], [2], [20]. Ambient Intelligence (AmI) has emerged in the past 10 years as a multidisciplinary field which involves many areas of computer science and engineering [27].

A fundamental and distinguishing aspect of AmI is interactivity. Ambient Intelligence places emphasis on greater user-friendliness, more efficient services support, user-empowerment, and support for human interactions. People are surrounded by intelligent intuitive interfaces that are embedded in all kinds of objects and an environment that is capable of recognizing and responding to the presence of different individuals in a seamless, unobtrusive way. Technology becomes invisible, embedded in our natural surroundings, present whenever we need it, enabled by simple, effortless and natural interactions, autonomously acting and adaptive to users and context [3], [4].

Different appliances have successfully become integrated to our daily life surroundings to such an extent that we use them without consciously thinking about them. Today many of these appliances have become enhanced with computing devices which provide them with some degree of “intelligence”. Washing machines, heating systems and even toys come equipped with some capability for autonomous decision making. Cars have embedded sensing mechanisms that allow them to make decisions for a safer and less expensive journey. To achieve these “intelligent” functions, cars have dozens of sensor and actuator systems that make decisions to assist the driver. Public spaces have become increasingly occupied with tracking devices which enable a range of applications such as sensing shoplifted objects and crowd behavior monitoring in a shopping mall [5].

Advances in the miniaturization of electronics are allowing computing devices with various capabilities and interfaces to become part of our daily life. Sensors, actuators, and processing units, all linked through different types of networking technologies, can now be purchased at very affordable prices. This technology can be used with the coordination of “intelligent” software that understands the events and relevant context of a specific environment, takes sensible decisions on behalf of the user and adjusts the environment according to the user’s needs and wishes [6].

Whether it is our home anticipating our arrival at night and helping us get ready in the morning, an airport facilitating a commute, or a hospital room helping to care for a patient, we will witness great changes in the next decades by the introduction of a wide range of devices which will enhance diverse environments with computing power [7].

1.2 Description of Ambient Intelligence

Although AmI involves areas such a computer networks, sensors, human-computer interfaces, ubiquitous computing, robotics and artificial intelligence it should not be confused with any of those in particular. Even though such areas are an important part of AmI none of them conceptually fully covers Ambient Intelligence. It is AmI which brings together all these resources as well as other areas, such as education, health and social care, entertainment, sports, transportation and many others to provide flexible and intelligent services to users acting in their environments [8], [78], [79], [83].

The IST Advisory Group of the EU identified the following five technology requirements for AmI [20]:

1. Vey unobtrusive hardware.
2. A seamless mobile/fixed communications infrastructure.
3. Dynamic and massively distributed device network.
4. Natural feeling human interfaces.

5. Dependability and security.

Ambient Intelligence is a digital environment that proactively, but sensibly, supports people in their daily lives. Therefore two characteristics that an AmI system must possess are to be sensitive and responsive. Through sensors, AmI systems gather information about the environment, process that information to determine the activities and needs of the users, and finally through actuators modify the environment in a form that will benefit such users [75], [136]. A control system is a device or set of devices to manage, command, direct or regulate the behavior of other devices or systems [128]. As such, an Ambient Intelligence system can be viewed as a type of control system. Furthermore, AmI systems can be viewed as an evolution of Home Automation (HA) and Building Automation (BA) systems.

1.3 Problems with Ambient Intelligence

In 2001 the Information Society and Technology Advisory Group (ISTAG) released the report titled *"Scenarios for Ambient Intelligence in 2010"*. In this report four different scenarios are presented with the aim of describing what living with Ambient Intelligence might be like for ordinary people in 2010. Each scenario has a script designed to illustrate the key developments in technologies, society, economy, and markets necessary to arrive to such scenario [20]. After more than a decade of intense research which includes dozens perhaps even hundreds of conferences, thousands of research articles and large financial investments we do not yet live in homes where smart refrigerators keep track of our food, smart agents negotiate carpooling on our behalf without user intervention, or in environments that interact with us by recognizing voice commands or our daily activities [27]. Why don't we live in such environments yet?

A very active research field within Ambient Intelligence is the area of Ambient Assisted Living (ALL). AAL is targeted at people with specific needs such as the handicapped, the aged and the sick. In an AAL enabled smart home disbursement of medicine can be monitored and managed, accidents that can occur to the elderly can be detected and appointed caregivers summoned if necessary. However, for users without the specific needs of the handicapped or elderly the complexity and cost of the technologies involved in setting up a smart home demand a very high level of perceived benefit before they become appealing [9].

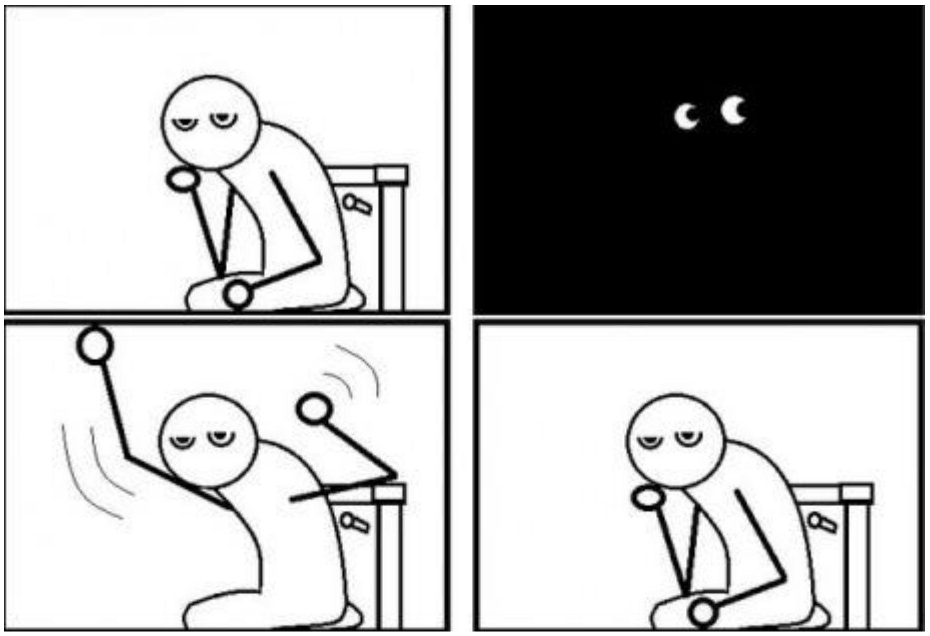


Figure 1-1. Problems with Home Automation.

Despite the many advantages for everyday living offered by Ambient Intelligence there has not been a widespread offer or demand of AmI. Home Automation technologies can be seen as the stepping stones on the road to full Ambient Intelligence deployments. Although the devices required to enable HA, such as motion sensors, programmable lighting and video surveillance cameras have been available to consumers since the 1970s, the technology has not been adopted in such a wide scale as, for example, cellular phones or personal computers. While some HA technologies like lighting control are gaining acceptance in commercial settings, broader adoption is severely lacking. According to estimates by ABI Research only 204, 000 home automation systems were shipped globally in 2009, although recent studies suggest that this figure may be increasing [26], [81], [82], [135].

The problem of wider HA acceptance cannot be placed on a single cause but stems from multiple factors. Brush *et al.* [135] conducted semi-structured visits to 14 households that currently employ HA. The interviews involved both households of Do-It-Yourselfers (DIYs) who have installed the automation themselves and households that have outsourced the installation and management to professionals. In contrast to previous research, the people involved in this study were not living in a home developed specifically as a lab but were people who had opted to introduce HA technologies into their homes and daily lives. The study showed four major challenges people face when deploying and using HA systems:

1. *High cost of ownership.* High cost of ownership involves both the monetary cost of the technology and the time required to install and setup the systems. Money spent by the households on home automation varied widely from about \$200 to \$120,000 US. Monetary cost was one of the most frequently mentioned considerations determining both the brand and the amount of functionality to install. Outsourced households require an outside consultant to come when their system needs adjustment or repair. The cost of these consultant visits did not seem to concern the households as much as expected, probably because the visits were relatively infrequent and had a low cost relative to the initial installation cost. After initial setup, consultants were primarily called only when problems occurred. For DIYs households, in addition to monetary cost, the time cost currently required should not be underestimated.
2. *Inflexibility.* Current installations are inflexible, often requiring a choice between a single integrated system or flexibility, as well as the need for structural changes in many installations, which limits when automation can be installed and raises concerns about moving. Several participants, especially DIYs, did not want to be locked in one specific vendor and expressed resistance to certain brands, e.g. Creston and Control4, because of a perceived lack of personal control. However, choosing to use multiple brands meant dealing with the challenge of integrating separate systems. Integration capabilities prevented the connection of some of the devices with network capabilities to the HA system. Perhaps one of the biggest challenges to broad adoption is the structural changes needed to install

Home Automation systems. The most common trigger to install HA among the participants was building or remodeling a house. As a result of the structural changes performed the challenge of moving was a concern. Several participants felt adding HA would make their house more difficult to sell.

3. *Poor manageability*. Living with a Home Automation system requires managing it. The experiences of participants suggest challenges that would need to be addressed before broader adoption of Home Automation such as support for the iteration necessary to customize, issues with reliability, complex user interfaces and concerns raised by reliance on consultants. Most households, even those with unchanging set-ups, described an initial period of iteration right after they installed their HA systems during which they customized their set-ups to their homes and household needs. A few households described scaling back their initially installed functionality as they became aware of a difference between what they thought they wanted before installing automation and what functionality they actually wanted.

   Fourteen participants explicitly mentioned problems with their current system's reliability, which typically resulted in unpredictable behavior. Participants felt rules in general were hard to debug when they did not work and so participants lived with problems or turned off the rules. Lack of responsiveness was a related frustration. Four households described waiting several minutes for a system's response.

   The augmentation strategy many households adopted, particularly for lighting scenes, aimed for a simple interface that could be used by anyone including guests. While some households were more successful than others at achieving simplicity, problems with complex user interfaces faced by participants, particularly the technology consumers, and guests illustrate that user interface challenges exist. Eight participants mentioned complex user interfaces as one of the things they most dislike about home automation. The complexity of the user interfaces could be confusing or even frightening to guests.

   To handle manageability households relied on consultants, either the DIY guru who served as the on-site consultant or professional consultants in Outsource households. Participants described downsides to being reliant on a consultant including inability to fix their own system, inability to customize and password management.

4. *Difficulty achieving security*. Most automation participants had enabled, such as lighting or media, could be used by anyone present in the house. Because households had augmented the physical controls the functionality was available within the house using wall switches or a remote control device. The main exceptions were home security systems which always required passwords, and interfaces for writing rules, which often did.

   Remote access was a double-edged sword for people. The functionality was appealing but participants worried about introducing a security risk. Households used remote access for a variety of tasks including remotely controlling lights to make a house appear occupied when no one was home, turning on heating before they arrived home, checking the state of the house using cameras, or verifying the doors had been locked. In general, participants perceived remote access to be valuable particularly for vacation scenarios. However participants also expressed concerns about security when enabling remote access, for example, that it would make their house vulnerable. The level of concern of some participants was directly related to what technology was in the house, cameras and door locks raised higher concern compared to other devices.

Mennicken and Huang [26] identify a similar set of problems when they surveyed three key groups of stakeholders in commercial HA offerings:

1. Inhabitants of homes equipped with automation technology.
2. People in the process of planning or building automated homes.
3. Providers of existing commercial solutions for Home Automation.

Their objective was to understand how a smart home currently develops, from the initial idea to instrument the home to the emergent uses of its technology by household members. The study was divided into two phases. The first phase involved interviews with seven Home Automation professionals while the second phase involved interviewing seven household which inhabit HA equipped homes and three households who are in the process of planning homes with HA. The following issues with Home Automation technologies were found as a result of the study:

1. *Even Full Automation of Control is Not a Game Changer.* Participants mentioned that they obtained very few direct benefits or major impacts on their lives or practices despite the cost and effort required. They described the effects of the technology as small conveniences rather than substantial support for routines or tasks. People perceived the technology as enhancing their comfort level but pointed out that the technology was limited in the help it could provide.
2. *The Challenge of Planning for Unfamiliar Technology.* At the start of the planning phase some participants reported not understanding potential benefits of technologies and therefore had difficulties prioritizing those technologies against other needs in the home. Participants without technical backgrounds reported having to rely upon other people's experiences and expertise, and therefore feeling powerless. Professionals provided an interesting perspective on what this challenge represents for users. They reported that customers have difficulties understanding the available technology and options. One of the professionals said: *"It just doesn't make sense to people… [that] they need power line switches if they [just] want to have access with their smart phone. They don't see the connection"*. Another professional illustrated this challenge by contrasting HA with more familiar technologies: *"The whole issue of home automation is still so remote. For cars, everyone knows what's possible"*
3. *The Challenge of Getting High-Level Expert Advice.* Professional system integrators typically only provided information on the systems that they offered, and other types of home experts, such as electricians and architects, were rarely able or willing to provide information about home automation technology. As a result, participants reported frustration over being unable to access authoritative and expert advice for high-level decision-making despite the existence of experts.
4. *The Tension between Comfort and Control.* Although participants felt that automation resulted in a gain in comfort for some aspects of the home, they also perceived a loss of control with increased automation. One participant described

the override functions he had created for the home while another said she feared becoming *"a prisoner of the system"*.

5. *Experimenting and Testing.* Participants with a strong technical background often considered the installation and iterating to be a hobby.

Holroyd *et al.* [25] identify the following key areas as part of the problem of adoption of HA technologies:

1. *Lack of a killer feature.* The lack of one clear benefit is also something that is required for HA to penetrate the mass market.
2. *Usability.* Usability refers to the way an end user will interact with various smart devices and system preferences. If a user finds a system too complicated or unintuitive they will simply not use it.
3. *Interoperability.* Although various HA protocols exist, such as X-10, Lonworks and KNX, there still exist problems when different devices are connected together. Universal connectivity rather than individual designed and controlled smart devices can only happen when manufacturers use open and common standards.
4. *Retrofitting.* Currently Home Automation technology is frequently installed during construction of the building or during a major renovation. Expensive cabling and planning is required to insure that all systems and devices are connected together and therefore introducing the technology in older buildings is usually not an easy task.
5. *Cost.* Cost has long been a problem for HA manufacturers. The requirement that all non-smart devices in the home be replaced along with expensive installation and cabling cost means that these technologies are generally only available to the wealthy.

1.4 The Need for Ambient Intelligence

In view of the challenges described previously for Home Automation adoption we have to ask ourselves an important question: Do we really need Home Automation or Ambient Intelligence technologies?

Yes we do, and soon.

The 12th of October of 1999, a little more than a year before the ISTAG published its 2001 scenarios report, was declared by the UN as the “Day of 6 Billion” [10]. This was the day when according to UN estimates the world population reached 6 billion people. A bit over a decade later, on October 31st 2011, the Population Division of the United Nations estimated that the world population hit the 7 billion mark [11]. The population of Mexico rose from 97.5 million people in the year to 2000 to 108.4 million in 2010 [12].

Although the rate of growth of the world population is decreasing it is still positive. The United States Census Bureau estimates that the 8 billion marker will be reached by the year 2025, and that the world population will have reached roughly 9.4 billion people by the year 2050 [13]. UN estimates reach roughly the same figures [14]. This increase in the world population translates into higher demands on our society’s infrastructure, on our planet’s environment and on natural resources.

According to the United States Energy Information Administration the world’s total energy consumption rose from 406 quadrillion Btu in the year 2000 to 524 quadrillion Btu in 2010. The world energy consumption is estimated to reach 630 quadrillion Btu by 2020 and 820 quadrillion Btu by 2040. This increase in energy demand results in an increase in greenhouse gas emissions. The US EIA estimated that world-energy related carbon dioxide emissions increased from 23.6 billion metric tons in the year 2000 to 31.2 billion metric tons in 2010. Energy-related carbon dioxide emissions are expected to reach 36.4 billion metric tons in 2020 and 45.5 billion metric tons in 2040 [15].

Energy awareness has become a topic of interest within Ambient Intelligence research. Home owners have a high interest in reducing energy consumption because this translates into monetary savings [16], [26]. Usage awareness alone has the potential to reduce consumption by 15% in private households. However standard electricity meters and the suppliers’ analog billing systems lack the feedback capabilities that are necessary to increase energy awareness and positively affect customers’ behavior. Smart metering helps to alleviate this situation by considerably reducing the time from energy consumption to user billing. Yet current solutions to smart metering are proprietary and generally not generic or flexible, which makes it necessary to research generic solutions that are independent of any proprietary hardware or software. Besides pure monitoring, AmI could help users increase their monetary savings and reduce their energy consumption by using energy pricing information to control devices.

The increase in world population places additional stress on the world's cities infrastructure. In a study of the impact of transport decisions on the economy and the environment of the United Kingdom, it is estimated that traffic congestion may cost the economy of England £22 billion a year in lost time by 2025 [17]. In the United States, the Texas Transportation Institute estimated that in the year 2011 passengers and drivers experienced 5.5 billion hours of travel delay, resulting in 2.9 billion gallons in wasted fuel resulting in a congestion cost of $121 billion US dollars. It also estimated that the annual congestion cost for each commuter was approximately $818 US dollars [18]. Traffic congestion is increasing in major cities and delays are becoming more frequent in smaller cities and rural areas.

World increase in traffic not only translates to traffic congestion. According to the World Health Organization about 1.24 million people die each year as a result of road traffic crashes. Road traffic injuries are the leading cause of death among young people, aged 15 to 29 years. Half of those dying on the world's roads are "vulnerable road users": pedestrians, cyclists and motorcyclists. Without action, road traffic crashes are predicted to result in the deaths of around 1.9 million people annually by 2020. Additionally, every year between 20 and 50 million people suffer non-fatal injuries, with many incurring a disability as a result of their injury. Road traffic injuries cause considerable economic losses to victims, their families, and to nations as a whole. An estimate carried out in 2000 suggests that the economic cost of road traffic accidents was approximately $518 billion US. National estimates have illustrated that road traffic crashes cost countries between 1 and 3% of their gross national product [19].

Traffic management and safety has been an interest area of AmI since the conception of the field. One of the scenarios described in the 2001 ISTAG Scenarios for Ambient Intelligence in 2010 report, the *Carmen: traffic, sustainability & commerce* scenario, was targeted specifically to promote research in the areas of traffic management and safety among others [20].

Although a reduction in the growth of the world's population is definitely necessary, such growth rate reduction brings additional problems regarding the needs of the elderly. According to the World Health Organization, between 2000 and 2050 the proportion of the World's population over 60 years of age will double from about 11% to 22%. The absolute number of people aged 60 years and over is expected to increase from 810 million to 2 billion over the same period. In other words, by the year 2050 one out of every five persons in the world will be over 60 years of age; currently the proportion is one out of every 9. In 2050 older persons will outnumber the population of children (0 – 14 years) for the first time in human history. Additionally the older population is itself ageing. Presently, the oldest old population (aged 80 years or over) accounts for 14 percent of the population aged 60 years or over. The oldest old is the fastest growing age segment of the older population. By 2050, 20 percent of the older population will be aged 80 years or over. The number of centenarians (aged 100 years or over) is growing even faster, and is projected to increase tenfold, from approximately 343,000 in 2012 to 3.2 million by 2050 [21].

This shift in the age distribution of the population is one of the more pressing matters for Ambient Intelligence and, more specifically, the field of Ambient Assisted Living. On the one hand people prefer to age at home [22], on the other, there may not be enough caregivers available to provide support to the elderly. In a recent study by the AARP Public Policy Institute, in 2010 there were more than seven potential caregivers for every person in the high risk years of 80 plus. By 2030, it is projected that the ratio will be reduced to 4:1, and by 2050 it is expected to further fall to 3:1 [23]. It is necessary to develop the technologies that will allow people to age well and productively at home.

1.5 Developing Intelligent Environments

A core concept in Ambient Intelligence is the idea of a caring environment that senses and intelligently reacts to people, anticipating their desires and intentions. This part of the AmI vision has given context-awareness a prominent role and generated the assumption that Artificial Intelligence should be able to detect, model, and understand daily situations in a way that would allow the system to autonomously take the most appropriate actions. This particular notion of intelligence is an integral part of some of the most enticing AmI scenarios and has inspired a large body or research into new techniques for improving the sensing, inference and reasoning process. However it has also become one of its most challenged assumptions, generating a growing level of criticism that essentially questions its feasibility in the near future [27].

Scaling up from prototypes that work in restricted environments to solutions that reliably and robustly work in the full complexity of human environments is currently one of the challenges of Ambient Intelligence. According to Leahu *et al*. [24], the problem of how computational systems can make sense of, and respond sensibly to, a complex, dynamic environment filled with human meaning is identical to that of Artificial Intelligence (AI). The history of AI has been marked by numerous highs followed by crashing lows in terms of success, recognition and funding. One such low was triggered by the undelivered promises of early AI research, resulting in the AI funding winter of the 1970s. As the goal of computational intelligence proved to be harder to achieve than initially expected, a number of key areas of interest for AI began to emerge. These areas targeted individual intelligent capabilities: planning, computer vision, natural language understanding, machine learning, etc. In what later became known as classical AI, these problems were approached by identifying real-world subjects relevant to the area, modeling them using symbolic representations, and using rules of inference to manipulate those representations to derive answers. Based on the principle of divide-and-conquer, progress was to be made in each of these intelligent capabilities through functional decomposition and the functional modules would eventually be combined into a complete intelligent system. It was anticipated that this divide-and-conquer strategy would allow for measurable, incremental improvements in AI algorithms that could eventually be merged into a single system. However researchers slowly realized that these functional modules were related, each requiring others in order to work. Because of this, such problems were sometimes termed AI-complete to suggest that a solution to them would lead to a solution to the general problem of intelligence. As a workaround to the challenge posed by AI-completeness some researchers focused on solutions that worked in specific, situated contexts rather than solving the general problem of intelligence.

Using a similar approach might aid in achieving a wider acceptance of Ambient Intelligence. In [26] participants stated that they considered “smart” to be that which fits, speeds up or improves their routines while avoiding unnecessary work. Another aspect of “smartness” was that technology, no matter how powerful, needs to fit into everyday life. Participants reported they did not see a benefit to automation if they could still perform the same task faster or better manually. Merely being convenient was not sufficient for automation to be considered “smart”. Technology itself is not smart, but applications of technology could be smart. Participants felt that adding the functionality and mapping functions to the different components was what resulted in instances of intelligence.

The development of “pseudo-intelligent” modules may lead to a wider demand of Ambient Intelligent solutions. As noted by Mennicken and Huang [26], potential users have difficulties understanding the benefits that this technology may bring. At the same time, the lack of “field testing” limits the understanding by researchers of “real world” requirements of Ambient Intelligence. “Pseudo-intelligent” modules which solve specific problems may be the “killer app” that will bring AmI to everyday life.

Potential users also require time to adapt to the technology. In a study on smart home interface preferences between US and Korean users, Jeong *et al*. found that participants preferred to interact with a smart home using a physical device (computer, mobile phone or remote control) rather than communication modalities such as speech or gesture. Perhaps with frequent exposure to speech and gesture recognition interfaces people will get accustomed to this form of interaction or, if preferences do not change, this opens opportunities for new research in alternative natural user interfaces [25].

A key factor in the development of AmI solutions may be hobbyists and “hackers”. Mennicken and Huang [26] and Brush *et al.* [135] found in their study that “hacking” of the home was a primary motivation and a perceived major benefit for installing smart home technology for some participants. They conclude that there may be an important open research direction on providing support to those who want to engage with hacking the home. Providing appropriate tools would not only support the hobby aspect of smart homes but would also facilitate experimentation, innovation and, possibly, solutions better fitted to the needs of individual households.

1.6 Simplicity for Multidisciplinary Development

Developing Ambient Intelligence systems from scratch requires a significant investment of time and money [27], [28], [75]. As a result it would be very beneficial for hobbyists, researchers and businesses interested in developing AmI systems to have a basic development platform which could be used as a starting point in the development of their own applications. We argue that the main features of such a basic development platform should be the following [136], [81], [82], [139]:

- *Flexibility.* The development platform should make the least possible amount of assumptions about the applications that will be developed. This will allow the development platform to be used in a wide range of applications. Flexibility is particularly important as AmI projects scale up from the vehicle, household and office levels to building, city, country-wide and even global Ambient Intelligence systems. These so called Large Scale Intelligent Environments (LSIE) [29] will most likely consist of several sub-spaces each with their own individual requirements and characteristics. As a result, a development platform for Ambient Intelligence systems should possess the ability to adapt to many different sub-spaces and must also scale seamlessly from small single-apartment environments to multi-subspace scenarios.
- *Ease of Use:* Ambient Intelligence is a multidisciplinary field which can benefit from contributions from researchers outside of the field of Computer Science and from hobbyists. Therefore to encourage contributions by such researchers and hobbyists it is very important that the development platform be as easy to use as possible.
- *Modular Design.* To encourage incremental contributions it is necessary for developers and researchers who use the development platform to be able to reuse as easily as possible the work done by others. A modular design would provide an easy way for developers and researchers to share their work.

There is evidence to suggest that such a development platform would promote work reuse. An example of the benefits that a platform with such features provides can be found in the Arduino development platforms.

The Arduino Uno is a microcontroller board based on the Atmel ATmega328. It has 14 digital input/output pins, six analog inputs, a 16-MHz crystal oscillator, a USB connection, a power jack and an In Circuit Serial Programming header. It can be powered through the USB connection or with an external power supply. The ATmega 328 has 32 KB of program memory, 2KB of RAM and 1 KB of EEPROM. Software development for the board is done by means of a standard programming language based on a simplified version of C++ and with a simple Processing-based IDE. Software is transferred to the board using the USB connection [30], [31].

One of the great advantages of the Arduino is its ease of use. It takes around 20 to 30 minutes to set up the development environment and make a simple “Blinky” app (the embedded systems equivalent of the Hello World application). As a result the Arduino has gathered an enthusiastic user base. The Arduino is used to develop anything from psychological and neurophysiological lab equipment to artistic light displays. Many users, frequently from fields not related to Computer Sciences, make code and hardware designs available freely on the Internet. Due to the modular oriented design of the Arduino, there are many accessory boards, known as shields, available to developers. The Arduino can be easily customized by developers by connecting shields with the desired functionality or by developing their own shields.

It would be a great benefit to Ambient Intelligence research if a development platform equivalent to the Arduino were available.

1.7 IEC 61499

Until recently, industrial control systems have fallen into one of two main camps, either based on traditional distributed control systems (DCSs) or on programmable logic controllers (PLCs). Distributed control systems, commonly used in petrochemical plants and refineries, are structured around a few large central processors that provide supervisory control and data acquisition and which communicate via local networks with multiple controllers, instruments, sensors and actuators located throughout the plant. A system may have both discrete instruments and out-stations with clusters of instruments with local controllers. In a DCS, the main supervisory control comes from one or more central processors. Instruments positioned out in the plant typically provide local closed loop control, such as for PID control.

In contrast, for many machine control and production processes, particularly in automotive production lines, systems have generally been designed using programmable logic controllers (PLCs). A large PLC system will generally have a number of PLCs communicating via one or more proprietary high-speed networks. Each PLC will typically be connected to a large number of input and output (I/O) signals for handling sensors and actuators. In some cases discrete instruments, for example for temperature and pressure control, are also connected to PLCs.

With both design approaches, systems have tended to be developed by writing large monolithic software packages, which are generally difficult to re-use in new applications and are notably difficult to integrate with each other. The data and functionality of one application are not readily available to other applications, even if the applications are written in the same programming language and running in the same machine. Significant system development time is concerned with mapping signals between devices and providing drivers to allow different types of instruments and controllers to communicate. Both types of systems, DCS and PLC, tend to be difficult to modify and extend and do not provide the high degree of flexibility required for advanced and flexible automation [133].

The Function Block, first introduced by the IEC 61131 standard on programming languages for programmable logic controllers, is a well-known and widely used construct by control engineers. IEC 61499 defines a general model and methodology for describing function blocks in a format that is independent of implementation. The methodology can be used by system designers to construct distributed control systems. It allows a system to be defined in terms of logically connected function blocks that run on different processing resources [129].

1.8 Modular Framework for Ambient Intelligence Systems

Current AmI systems are not “researcher friendly”. After more than a decade of considerable research the field has not yet matured to the point of enabling incremental research. Ambient Intelligence is a highly multi-disciplinary field which involves communications, control systems, electronics, artificial intelligence, human-computer interfaces, distributed systems and others. However there is not a common set of tools which researchers from different disciplines can use to contribute to the field of AmI research.

Several frameworks have been developed in an attempt to find a set of tools which facilitate the development of commercial AmI systems and also to allow a “standard base” for researchers of different fields to contribute to Ambient Intelligence research. So far none of those frameworks are in widespread use. Although proposed frameworks address important issues, such as modularity, which are vital for the development of flexible AmI systems, they also tend to be rather complex platforms which require high investment of both time and effort. As a result AmI systems continue to be developed on many occasions “from scratch”. We consider this to be one of the main roadblocks towards incremental research in the field.

To address these issues we propose a minimalist modular framework for the development of Ambient Intelligence systems. This framework is based on the function block abstraction used in the IEC 61499 standard for distributed control systems. Through the use of function blocks, the framework promotes component reuse which allows researchers from different fields to easily apply previous results in new developments.

This minimalist modular framework is also targeted at the development of commercially viable AmI systems. Using the framework, system integrators can develop highly customizable AmI systems where end users can add new functions with a degree of difficulty analogous to hooking up a common home appliance or installing a new program on a computer.

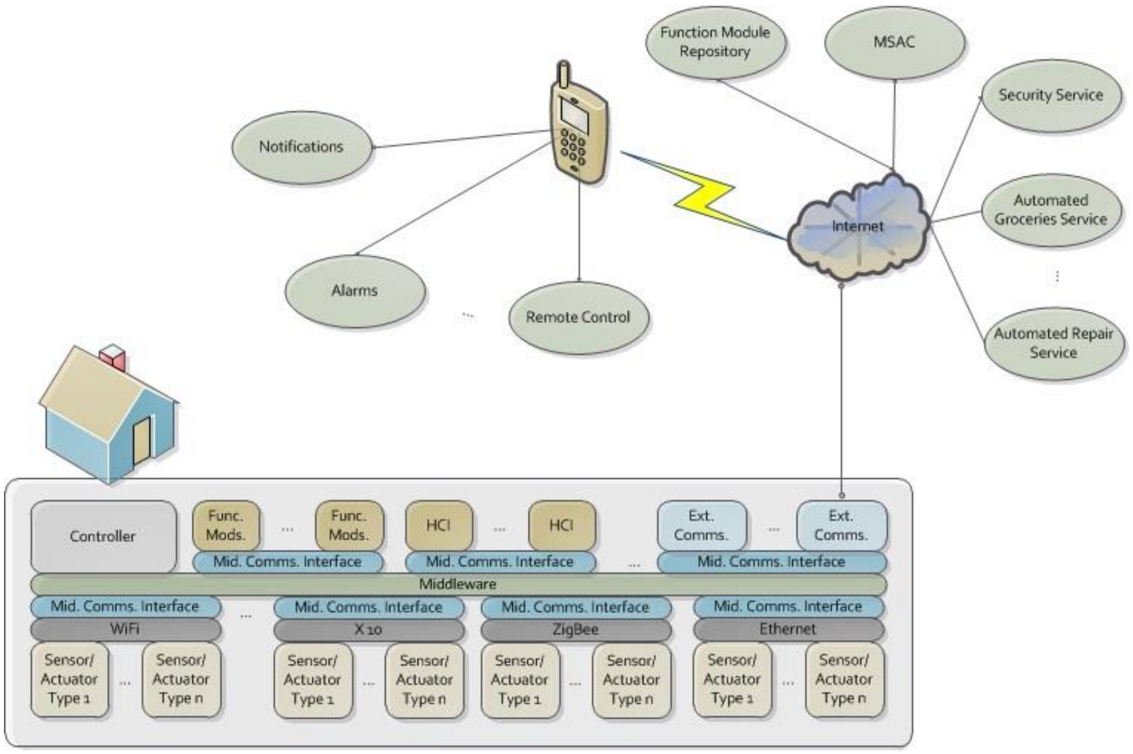


Figure 1-2. Funblocks Diagram.

1.8.1 Thesis Proposal

Currently the development of AmI systems is frequently performed "from the ground up", meaning that different sensor and actuator technologies have to be integrated with embedded platforms and communications systems in order to have a development environment on which to test research ideas. Unless such research is specifically focused on AmI systems platform development, this approach is far from ideal.

The development of custom research platforms for AmI systems has two main drawbacks. First of all sensors, actuators, embedded systems and communications can be complex and heterogeneous thus making the development of a system with these technologies neither simple nor brief. On the other hand, the use of custom made research platforms discourages the reuse and third party testing of research results.

The Modular Framework for Ambient Intelligence Systems described in this proposal aims at providing a standard and vendor neutral platform for AmI systems development, both for research and commercial purposes. To encourage the reuse of research results and therefore promote incremental research in AmI systems, our framework is based on the use of function blocks linked through an event driven middleware. Given that our framework is based on the use of the function blocks abstraction we call this framework Funblocks [81], [82]. Correspondingly the middleware, which can be used in other systems that benefit from a middleware with component supervising capabilities, is called Midblocks [139].

1.8.2 Modular Framework Components

The Funblocks framework is made up of the following components (Figure 1-2):

- *Controller.* Coordinates installation/removal of modules, sensors, actuators and other components. Process system failures through the reception of failure events, and performs communications with other systems such as Automated Repair Services.
- *Middleware.* Distributes events generated by system components and provides a link between diverse communications systems such a RS-485 and Ethernet.
- *Function Modules.* Process sensor data and generate events for other function modules or actuators.
- *Sensor/Actuators.* Sensors and actuators allow an AmI system to gather information about the environment and to perform adjustments to said environment.
- *Human Computer Interfaces.* Human Computer Interfaces allow users to interact with the AmI system.
- *Module and Sensor/Actuator Catalog.* Stores Module and Sensor/Actuator Records. These records contain a description of the services provided by a class of sensor, actuator or function block.

2 Related Work

Ambient Intelligence systems are required to interact with a wide variety of other systems and devices. AmI systems have to interact with different types of sensors and actuators in order to gather information from the environment, and to modify the environment in a meaningful way. On occasions these sensors and actuators will come in the form of a domotic or building automation system. Using these different types of sensors, actuators and systems poses a challenge in the development of Ambient Intelligence systems [136].

AmI systems have diverse applications, for example Ambient Assisted Living (AAL), and each of these applications has its own requirements. Several frameworks, many of them aimed at specific applications have been developed or are currently under development. In this section a description of some of the sensor, actuator, and building automation technologies currently available is provided. Some frameworks for the development of Ambient Intelligence systems are also briefly described along with a review of currently available middleware.

2.1 Building Automation Systems

Building automation system (BAS) is an umbrella term used to refer to a wide range of computerized building control systems. From special-purpose controllers to standalone remote stations, to larger systems including central computer stations and printers, a BAS comprises several subsystems which are connected in various ways to form a complete system. The system has to be designed and engineered around the building itself to serve the services systems for which it is intended. Consequently, although the component parts used may be identical, no two systems are the same, unless they are applied to identical buildings with identical services and identical uses [32].

Building services include HVAC systems, electrical systems, lighting systems, fire and security systems and lift systems. In industrial buildings they may also include the compressed air, steam and hot water systems used for the manufacturing process. A BAS may be used to monitor, control and manage all or just some of these services. In the following sub-sections we briefly describe some of the most commonly used communication standards in the BA industry.

2.1.1 BACnet

BACnet is a data communication protocol for BA and control networks. BACnet has been developed under the endorsement of the American Society of Heating, Refrigerating and Air- Conditioning Engineers (ASHRAE). It is an American national standard, a European standard, an ISO global standard and the national standard in more than 30 countries. It is the only open protocol that was designed originally for BA and supports functions such as scheduling, alarming and trending [32].

To achieve interoperability across a wide spectrum of equipment, the BACnet specification consists of three major parts. The first part describes a method for representing any type of BA equipment in a standard way. The second part defines messages that can be sent across a computer network to monitor and control such equipment. And the third part defines a set of acceptable LAN architectures that can be used to convey BACnet communications.

BACnet provides a standard way of representing the functions of any device, such as analogue and binary inputs and outputs, schedules, control loops, and alarms, by defining collections of related information called ‘objects’, each of which has a set of ‘properties’ that further characterize it. Each analogue input, for instance, is represented by a BACnet ‘analogue input object’ which has a set of standard properties such as present value, sensor type, location, alarm limits and so on. One of the object’s most important properties is its identifier, a numerical name that allows BACnet to unambiguously access it. BACnet defines 25 standard object types. A BACnet device does not need to support all object types, but if an object type is supported, it must comply with the standard object model for that object type.

BACnet employs the OSI Model as its reference model. The BACnet protocol defines a number of data link / physical layers, including Ethernet, BACnet/IP, Point-To-Point over RS-232, Master-Slave/Token-Passing over RS-485, ZigBee and LonTalk.

2.1.2 LonWorks

LON technology is used primarily for the decentralized processing of automation functions in room automation. LON can carry out monitoring, controlling and regulating functions for building services such as heating and ventilation systems. The focus of using LON technology at the automation level is not to decentralize individual functions, but to provide a standardized integrated bus system [33].

The heart of the LonWorks system is an integrated circuit called Neuron Chip. The Neuron Chip was developed by Echelon Corporation and comprises three processors that provide both communication and application processing capabilities. The Neuron Chip has to be used in conjunction with a transceiver for a specific medium; twisted pair, power line, radio frequency and fiber-optics. The most common medium employed in LonWorks is the twisted pair while the types of Neuron Chip most commonly used are the 3120 and the 3150 made by Toshiba and Cypress. The Neuron Chip type 3120 is usually used in simple devices that do not carry out complex functions, while the type 3150 is intended for more sophisticated applications.

Both types of Neuron Chips have three internal processors where each processor carries out different functions:

- CPU 1 is responsible for physical accessing the transmission medium. A network interface provides access to the transceiver. This represents layer one and two of the ISO/OSI model.
- CPU 2 is responsible for transmitting network variables and represents layers three to six of the ISO/OSI model.
- CPU 3 processes application programs but does not access the network. This is done by the other two CPUs.

The Neuron Chip 3120 has read-only memory which stores the LonTalk protocol, the Neuron operating system and the predefined operating routines for input/output (I/O) conditioning. On the other hand the Neuron Chip 3150 does not have internal ROM, but relies instead on external memory to store the aforementioned functions and the application program.

2.1.3 KNX

The KNX standard describes an open system concept for distributed home and building automation and control. KNX covers the full scope of home and building automation and control including lighting, shading, shutters and blinds, heating, ventilation, and air conditioning (HVAC), and remote meter reading. KNX emerged in 2002 as a merger between the European Installation Bus (EIB), Batibus and the European Home System (EHS) standards. The aim of this merger was to create a single European home and building electronic control systems standard [34], [35].

The KNX standard defines the network protocol specification, rules and definitions of how a KNX system is managed and how devices implemented by different vendors have to behave to achieve internetworking. The KNX protocol stack is based on the ISO/OSI reference model. Since different communication media are supported, the KNX protocol stack is divided into a medium-dependent and medium-independent parts. The medium-dependent part of the protocol stack consists of the physical layer and the lower level of the data link layer (DL) while the medium-independent part includes the upper level of the DL, the network layer (NL), the transport layer (TL), and the application layer (AL).

For physical media KNX provides a choice of dedicated twisted-pair cabling, power line transmission, and RF communication. Communication over IP networks as a first class medium is currently in the draft stage.

The main KNX medium is the twisted-pair cabling variant known as KNX TP1. The single twisted pair carries the signal as well as 29VDC link power. Data is transferred in a character oriented manner via half-duplex bidirectional communication at a transmission rate of 9600 bps. TP1 allows free topology wiring with cable lengths of up to 1000 meters per physical segment. Up to four segments can be concatenated using bridges, called line repeaters, forming a line. A line can contain up to 256 devices and up to 16 lines can be interconnected by main lines to form an area. Finally, up to 15 main lines can be interconnected by a common backbone line using routers called backbone couplers. Medium access on TP1 is controlled using CSMA with bit-wise arbitration on message priority and station address. Four priority levels are provided.

KNX Powerline 110 (PL110) uses the 230V/50Hz electrical power supply network for data and power transmission (in compliance with EN 50065-1). Half-duplex bidirectional communication is supported. KNX data is modulated using spread frequency shift keying (SFSK) with a center frequency of 110 kHz and a maximum transmission rate of 1200 bps. The signal is injected between phase and neutral and is superimposed on the sinusoidal oscillation of the mains. Repeaters can be installed in three-phase networks if passive phase coupling is no sufficient. Medium access control is based on a slotted technique to reduce the probability of collisions: After the minimum silence period between two frames has elapsed, two time slots are reserved for pending high-priority transmissions, followed by seven more from which nodes with pending standard priority transmissions choose one at random as their starting time.

KNX RF uses a sub-band in the 868 MHz frequency band reserved for short-range devices by European regulatory bodies. Data is transmitted at a rate of 16.4 kbps using frequency shift keying (FSK) modulation and Manchester encoding. The KNX RF frame format is based on FT3 as specified in IEC 60870-5. To minimize hardware requirements KNX RF not only supports bidirectional communication but unidirectional transmit-only devices are also supported. KNX RF devices communicate peer-to-peer.

KNXnet/IP currently focuses on scenarios for enhancing central and/or remote management. The KNXnet/IP Core Services define the packet structure and methods required for discovery and self-description of a KNXnet/IP server and for setting up and maintaining a communication channel between the client and the server. KNXnet/IP specifies several service protocols. KNXnet/IP tunneling describes the point-to-point exchange of KNX data over the IP network. Its main purpose is to replace USB or EIA-232 connections between KNX network interfaces and PC workstations or servers by tunneling L_Data frames. Acknowledgements, sequence counters and a heartbeat mechanism are used to ensure robustness. KNXnet/IP routing is a point-to-multipoint protocol for routing messages between KNX lines over a high-speed IP backbone. KNXnet/IP routers send UDP/IP multicast messages to other KNXnet/IP routers on the same IP network, which in turn filter the messages according to their destination or group address and pass them to the native KNX segment. KNXnet/IP device management allows configuration and diagnostic of KNXnet/IP tunneling interfaces or routing devices via the IP network. While KNXnet/IP is designed for devices with one IP and one traditional KNX network interface, the upcoming KNX IP is also intended for end devices that are solely connected to the IP medium.

The DL defines services to send and receive frames over the network. Acknowledgement is available as an option for some network media. The two most important services defined are L_Data for peer-to-peer data frame transfer and L_Poll_Data for a master collecting data from slaves in a so-called polling group. Furthermore, the DL defines a generic addressing scheme that is common to all available network media.

The NL uses the services provided by the DL and offers four different NL services: a unicast service, a multicast service, a domain-wide broadcast service, and a system broadcast service. For all four services, the individual address of the sender is used as the source address while the destination address depends on the used service. Additionally the NL introduces a hop count which is decremented and examined by routers and repeaters to perform filtering based on the amount of elapsed hops of a packet.

The TL uses the NL services and adds a connection-oriented unicast service. Using this service a device can establish a reliable unicast connection to another device. The state machine used for this service implements an acknowledgement mechanism such that data packets are retransmitted in case of a negative of absent acknowledgement. The four NL services are also present unchanged in the TL.

The AL layer on top of the stack provides several AL services which can be broadly classified in two different classes: data process exchange (process data communication) and configuration and maintenance tasks (management communication).

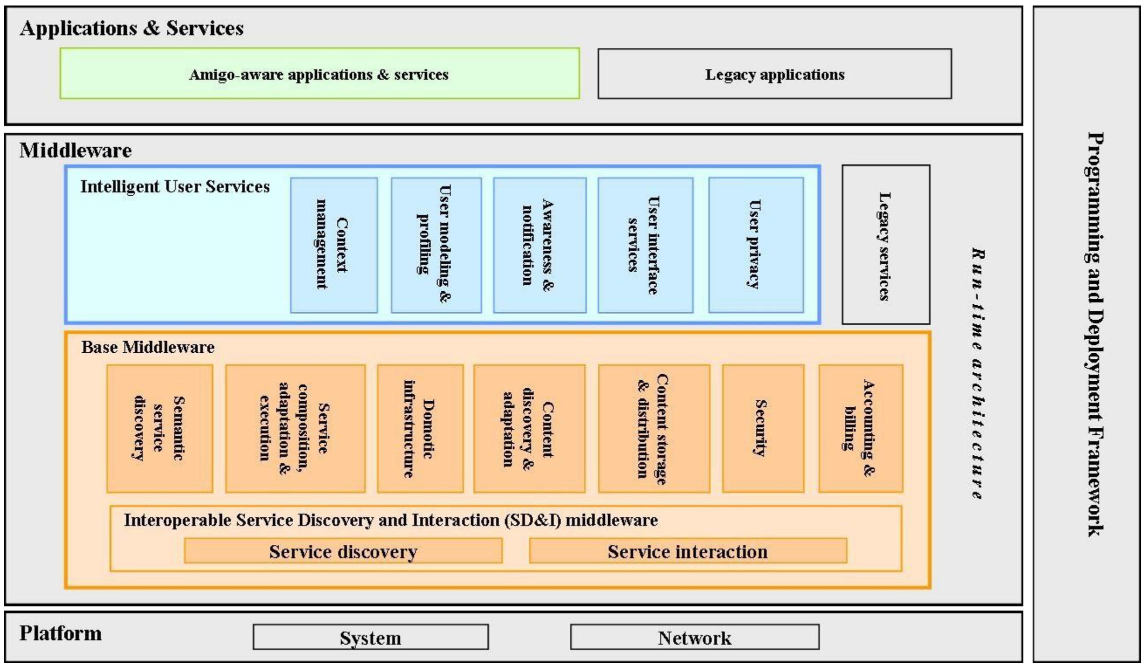
The internetworking and application model specifies how data is represented in KNX and how it is accessible via the network. Data points associated with functional blocks (FBs) are a central concept in this model. A KNX data point may be related to a sensor value/actuator state or it may be a parameter that controls the behavior of the user application. The necessary association between data points is established via bindings. KNX also specifies the application interface that is presented to user applications for interacting with remote data points. Finally, profiles define which parts of the KNX specification have to be implemented and ensure interworking of devices with the same profile that are provided by different manufacturers.

2.2 Ambient Intelligence Frameworks

Ambient Intelligence has been a field of intense research during the last decade. As part of the research done in the field of AmI systems several frameworks have been developed. In this section we provide a brief description of some of these frameworks.

2.2.1 AmIGo

The AMIGO Project was a project developed by fifteen European companies and research organizations [36–41]. The project was concluded in 2008.



Applications & Services
Amigo-aware applications & services
Legacy applications
Middleware
Intelligent User Services
Context management
User modeling & profiling
Awareness & notification
User interface services
User privacy
Legacy services
Base Middleware
Semantic service discovery
Service composition, adaptation & execution
Domotic infrastructure
Content discovery & adaptation
Content storage & distribution
Security
Accounting & billing
Interoperable Service Discovery and Interaction (SD&I) middleware
Service discovery
Service interaction
Run-time architecture
Platform
System
Network
Programming and Deployment Framework

Figure 2-1. AmIGo Architecture.

The aim of the AMIGO Project was to develop a middleware that dynamically integrates heterogeneous systems to achieve interoperability between services and devices. Through this middleware devices such as home appliances, multimedia players that communicate through UPnP, and personal devices are connected in the home network to work in an interoperable way. This interoperability across different domains can also be extended across different homes and locations. AMIGO focused on four application domains: Personal Computing, Mobile Computing, Consumer Electronics and Home Automation.

The AMIGO architecture follows the paradigm of Service orientation, which allows developing software as services delivered and consumed on demand. Discovery mechanisms can be used to find and select the functionality that a client is looking for. The AMIGO architecture is formed by the following three main components (see Figure 2-1):

Base Middleware layer. The Base Middleware layer contains the functionality necessary to integrate a networked environment such as discovery, interoperability, security, quality of service, content distribution, billing, etc. This solution is based on the semantics that are used to communicate and discover available services and devices in the network, including those based on existing communication and discovery standards. Existing hardware and software and new services can be discovered and composed independently. Supported service discovery protocols are UPnP, SLP and WS-Discovery while supported service interaction protocols are SOAP and RMI.

AMIGO's Middleware layer is further subdivided into the following components:

a) *Interoperable Service Discovery and Interaction (SD&I) middleware.*

b) *Semantic service discovery.*

c) *Service composition, adaptation and execution.*

d) *Domotic infrastructure.*

e) *Content discovery & adaptation.*

f) *Content storage & distribution.*

g) *Security.*

h) *Accounting and billing.*

Intelligent User Services layer. The Intelligent User Services layer contains the functionality needed to facilitate an ambient in-home network. This layer brokers between users and service providers, provides context information, combines multiple sources of information and makes pattern-based predictions. Information is tailored to user profiles and adapts to the user’s situation and changes in context.

The Intelligent User Services layer of the AMIGO Project is formed by the following components:

a) *Context Management.*

b) *User modeling and profiling.*

c) *Awareness and notification.*

d) *User interface services.*

e) *User privacy.*

Programming and Deployment framework. The Programming and Deployment framework contains the resources necessary for programmers to develop AMIGO aware services. This .NET/OSGi based programming framework contains libraries for service description, encryption, etc.

2.2.2 Home OS

The goals of HomeOS are to simplify the management of home networks and the development of applications [42], [43], [44], [45], [46], [47], [48]. It accomplishes these goals as follows. First, it provides one place to configure and secure the home network as one connected ensemble. Users do not have to deal with multiple different interfaces and semantics. Second, it provides high-level abstractions to applications. Developers do not have to worry about low-level details of devices and about device inter-connectivity. HomeOS is responsible for enforcing user preferences for device access and coordination, which does not have to be supported by individual applications. For example, if a user dislikes noise at night, she can disable night-time access to all speakers; HomeOS will then automatically deny access to all applications that try to use the speakers.

With HomeOS, users enable new tasks by installing new home applications. Because homes are heterogeneous, this process must be streamlined such that users do not inadvertently install applications that will not work in their homes. For instance, if an application for keyless entry requires a fingerprint scanner, users without such devices should be warned against purchasing such an application.

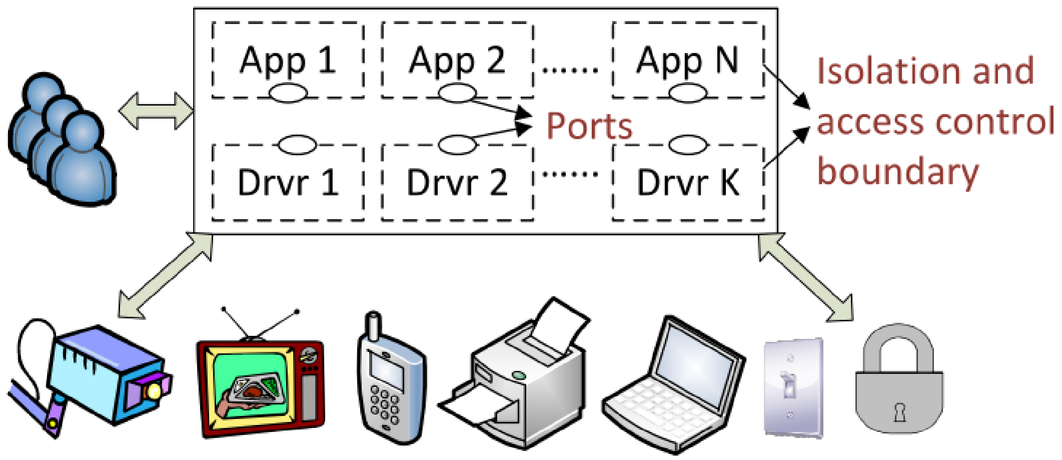


Figure 2-2. Home OS Architecture.

Inspired by the iPhone model, HomeOS is designed to be coupled with a HomeStore to simplify the distribution of applications and devices. The HomeStore verifies compatibility between homes and applications. Based on users' desired tasks, it recommends applications that work in their homes. If a home does not have devices required for those tasks, it recommends appropriate devices as well. For instance, if a user wants integrated temperature and window control, the HomeStore can recommend window controllers if there exists an application that combines those window controllers with the user's existing thermostat. In addition, the HomeStore can perform basic quality checks and support rating and reviewing to help identify poorly engineered applications and devices. HomeStore is not intended to become the sole gatekeeper for home applications. Towards this end, Home OS allows for multiple HomeStores and users can visit the one they trust most.

2.2.3 MPOWER

The MPOWER project developed a middleware platform to support the rapid development and deployment of integrated services for the elderly and cognitively disabled. MPOWER provides a domain specific architecture that facilitates interoperable, integrated, secure and standardized solutions that speed up the development process and create market possibilities for interested parties. Work was aligned with the standardization activities within the HL7 (Health Level 7) organization which was a key prerequisite for exchanging messages between different health providers [49], [50], [51], [52], [53].

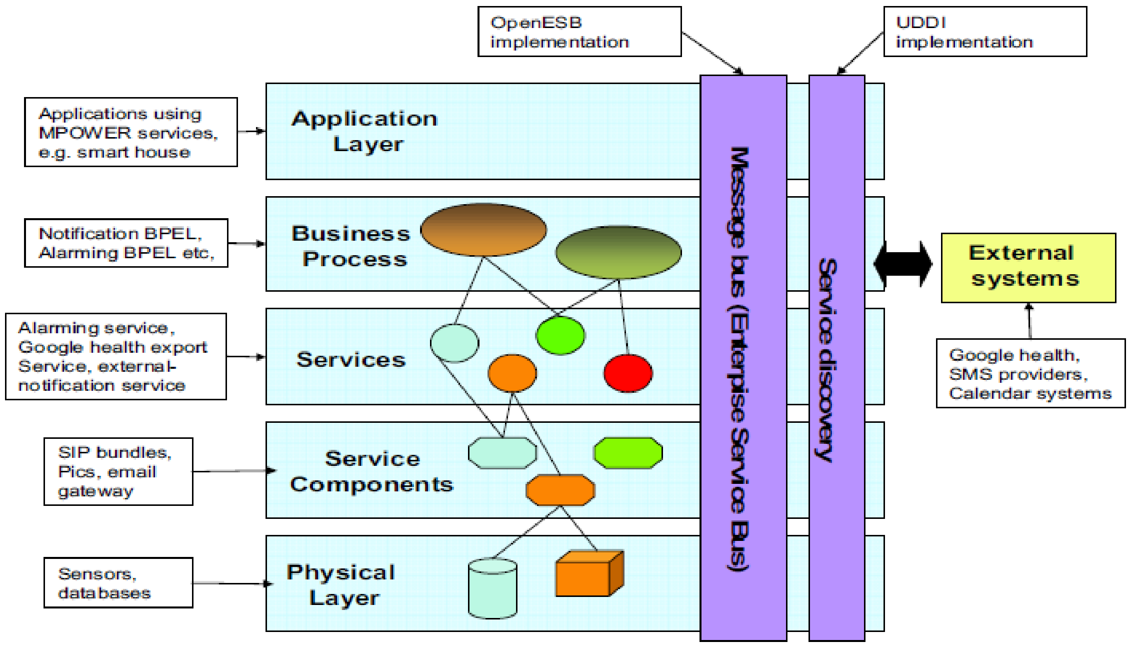

OpenESB implementation
UDDI implementation
Applications using MPOWER services, e.g. smart house
Application Layer
Notification BPEL, Alarming BPEL etc,
Business Process
Alarming service, Google health export Service, external-notification service
Services
SIP bundles, Pics, email gateway
Service Components
Sensors, databases
Physical Layer
Message bus (Enterprise Service Bus)
Service discovery
External systems
Google health, SMS providers, Calendar systems

Figure 2-3. Mpower Architecture.

As shown in Figure 2-3, the MPOWER SOA architecture consists of five layers:

- *Application layer.* Provides graphical user interfaces and serves as an entry point for using the services.
- *Business Process layer.* Defines the business rules of the applications that are created using the MPOWER middleware.
- *Services layer.* Contains the implementations of the services that are created within the MPOWER platform.
- *Service Components layer.* Service components expose the functionality of the components and databases in the Resource layer.
- *Resource layer.* This layer consists of existing custom built applications, such as databases storing patient-administrative, medication, and management information. Other relevant resources in this layer are smart sensors such as physiological monitoring devices, temperature sensors and burglar alarm systems.

The services provided are grouped into the following five categories:

- *Information (Medical and Social) services.* These services enable management of social information and events of the patient. For instance, these services handle information regarding patients' schedules, personal information, and social contacts.
- *Interoperability services.* These services enable interoperability of MPOWER platform with the other platforms that are relevant for health-care application. Primarily these services enable integration of medical services with existing medical systems of hospitals or particular state medication systems.
- *Sensor services.* Sensor services provide functionality to add, remove, and adjust devices as well as retrieve sensor information. The services expose both a management mechanism and data access thorough an easy to use and standardized interface.
- *Contextual services.* These are services related to the monitoring and management of the context of a patient. For instance, these services provide information about location of the patient in his house. The context services are realized through a set of sensors and actuators located in the patient's premises.
- *Security services.* Ensure sufficient protection for any of the MPOWER enabled services when they are used. This implies that security middleware is orthogonal to the

other services in a way that is an implicit part of each service, ensuring a satisfactory security level of any combination of services in the MPOWER platform.

2.2.4 Soprano

SOPRANO's goal is to build an AAL system for elderly people with functional impairments [54], [55], [56], [57]. SOPRANO seeks to provide flexible and personalized IT services that maximize the independence of elderly people with functional impairments and help them in retaining their dignity. Examples of such services are medication reminding, home automation, coping with increasing frailty, home safety and security, activity monitoring, keeping healthy and active, coping with cognitive ageing and forgetfulness, combating social isolation, and countering boredom.

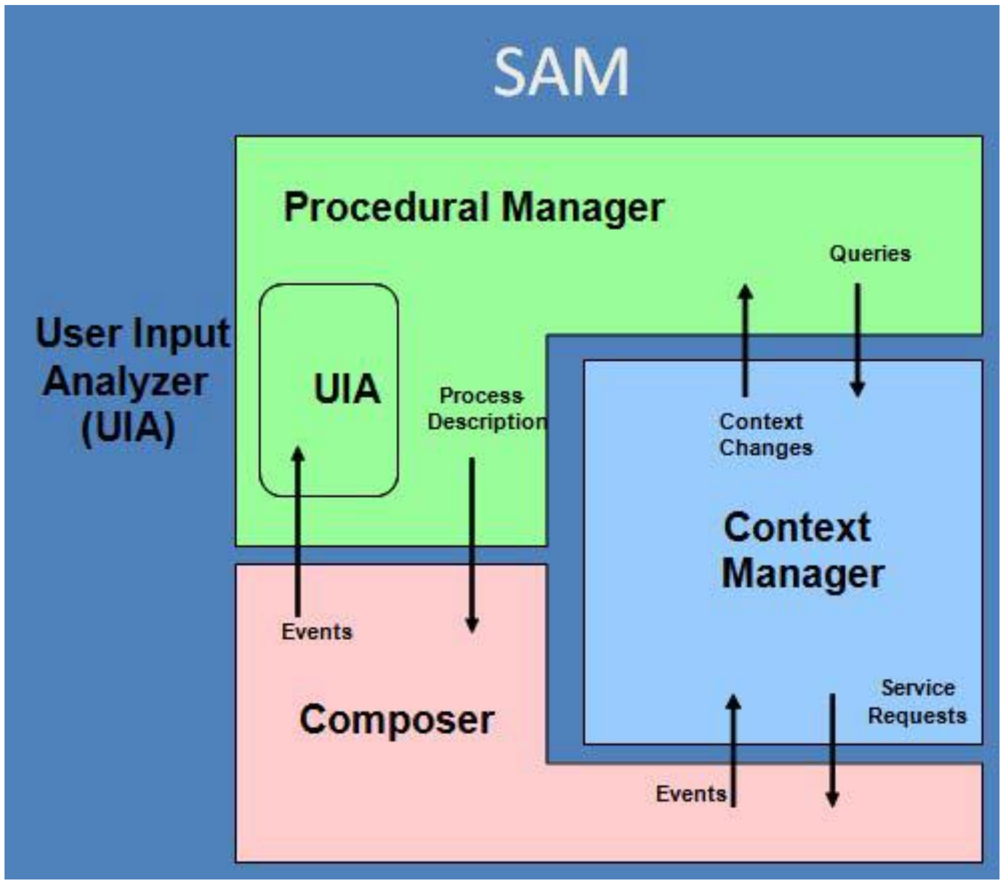


Figure 2-4. Soprano Ambient Middleware.

The core of the SOPRANO system is SAM, the SOPRANO Ambient Middleware, which provides its intelligence by receiving user commands and data from sensors, enriches them semantically and provides appropriate reactions via actuators in the house. Planned sensors are, for example, smoke, temperature, door status, location of the user by Radar or RFID, its health status and so on. Planned actuators are speech synthesizers, digital TVs with avatars, device regulators (for switching devices on/off or modifying their behavior), emergency calls to a central and more. Additionally the more static context of the house and the user shall be taken into consideration when performing concrete actions.

SAM can be divided into the following three main components (Figure 2-4):

- *Context Manager*. The Context Manager constantly analyzes incoming sensor events as well as the status of networked devices and appliances and tries to deduce higher-level context information. The results of this analysis are context parameters change events of a high semantic level. Context is defined as the user's situation in terms of all the temporal, personal, organizational, environmental, and even global conditions surrounding the user at a certain instant in time. Examples are the user's current activity and long lasting profile, the user's environment like lighting and temperature, connectivity parameters and so on. The context parameters change events are the input of the Procedural Manager.
- *Procedural Manager*. The Procedural Manager provides meaningful reactions to contextual changes or explicit user requests. Explicit user requests are handled by a subcomponent called the User Input Analyzer or UIA. By analyzing the new situation, the Procedural Manager compiles an abstract process description based on a repository of process templates. The abstract process description is a standard workflow description which contains abstract service requests instead of concrete service bindings. It can be parameterized with contextual variables, is based on pre-defined templates and is annotated with context-aware metadata. The procedural manager can obtain state information by invoking ad-hoc queries to the context manager. The result from this second step is a "plan of goals" which serves as input for the composer.
- *Composer*. The Composer has two objectives. First it serves as SAM's interface to the "real world" through sensor information. All incoming and outgoing service calls are

handled by the Composer. Second, it receives the “plan of goals” from the Procedural Manager, intelligently searches, compares, composes and parameterizes adequate concrete services in order carry out the process in a concrete manner and execute it through actuators.

2.2.5 universAAL

The universAAL project seeks to combine the advantages and strengths of still ongoing or already finished research projects to create a universally applicable AAL platform [58–62]. universAAL reuses components and concepts from the AMIGO, GENESYS, OASIS, MPOWER, PERSONA and SOPRANO projects. universAAL seeks to achieve its goals through the development of an open source middleware targeted towards health, home automation, entertainment, and energy efficiency applications and services. universAAL will also provide reference use cases and a tool chain for extending platform capabilities. Support for the creation of AAL services and applications will be provided through the universAAL Developer Depot and uStore.

Based on the GENESYS and PERSONA projects universAAL makes use of a layered design. The layers that make up the universAAL platform are the following:

- *Middleware.* The Middleware extends the native system layer of the different physical nodes participating in an AAL system. It hides the distribution of these nodes as well as the possible heterogeneity of their native system layers. Additionally this layer acts as a container for the integration of the components from the above layers and facilitates the communication among them.
- *Generic Platform Services.* The Generic Platform Services layer provides basic platform services like context management, service management, and a framework for supporting complex user interactions.
- *AAL Platform Plug-Ins.* On this layer special platform services can be introduced to extend the basic functionality of the framework. This might be needed in case high-level services have specific demands on, for example, data-mining of context reasoning.

- *AAL Applications and Services.* The AAL Applications and Services layer encapsulates all applications and services that directly provide support and assistance to the end user.

3 Technical Background

In this section we provide some background material necessary for a proper understanding of the Modular Framework for Ambient Intelligence Systems.

3.1 Ubiquitous Computing

Technology in computing has experienced great changes over the last years. In the past, mainframe computers dominated the computing scene based on the principle of one computer serving many people. In the 1980s, mainframe computers were replaced by personal computers where the emphasis was one computer to one person. Currently, with increased computing power available at always more affordable prices, it is common place for multiple computers to serve one person.

As a result we inhabit an increasingly digital world, populated by a profusion of digital devices designed to assist and automate more human tasks and activities, to enrich human social interaction and enhance physical world interaction. The physical world environment is being increasingly digitally instrumented and populated with embedded sensor based control devices. These can sense our location and can automatically adapt to it, easing access to localized services, e.g., doors open and lights switch on as we approach them. Positioning systems can determine our current location as we move. They can be linked with other information services to, for example, propose a map of a route to our destination. Devices such as contactless keys and cards can be used to gain access to protected services, situated in the environment. E-paper and e-books allow us to download current information onto flexible digital paper, over the air, without going into any physical bookshop [63], [64], [73].

A world in which computers disappear into the background of an environment consisting of smart rooms and buildings was first articulated over fifteen years ago in a vision called ubiquitous computing by Mark Weiser in 1991 [67]. The term ubiquitous, meaning appearing or existing everywhere, combined with computing to form the term Ubiquitous Computing (UbiComp) is used to describe ICT (Information and Communication Technology) systems that enable information and tasks to be made available everywhere, and to support intuitive human usage appearing invisible to the user. Ubiquitous computing represents a powerful paradigm shift in computation, where people live, work, and play in a seamless computer enabled environment, interleaved into the world. UbiComp postulates a world where people are surrounded by computing devices and a computing infrastructure that supports us in everything we do.

There might seem to be a paradox within the UbiComp vision in, how can something be everywhere yet be invisible? The point here is not that one cannot see (hear or touch) the technology but rather that its presence does not intrude into the environment, either in terms of the physical space or the activities being performed. This description of transparency is strongly linked to the notion that devices and functions are embedded and hidden within larger interactive systems.

Human computer interaction (HCI) with ICT systems has conventionally been structured using a few relatively expensive access points. This primarily uses input from keyboard and pointing devices which are fairly obtrusive to interact with. Weiser’s vision focuses on digital technology that is interactive yet more non obtrusive and pervasive. His main concern was that computer interfaces are too demanding of human attention. Unlike good tools that become an extension of ourselves, computers often do not allow us to focus on the task at hand but rather divert us into figuring out how to get the tool to work properly. Weiser used the analogy of writing to explain part of his vision of ubiquitous computing. Writing started out requiring experts such as scribes to create the ink and paper used to present the information. Only additional experts such as scholars could understand and interpret the information. Today, hard copy text printed on paper and soft copy text displayed on computer based devices are very pervasive. The majority of people can access and create information without consciously thinking about the processes involved in doing so.

Summing up, UbiComp systems are situated in human centered personalized environments, interacting less obtrusively with humans. UbiComp systems are part of, and used in, physical environments, sensing such physical environment. Since they are aware of the physical environment they can adapt to it, and are able to act on and control it. Hence, Weiser's vision for ubiquitous computing can be summarized in the following core requirements [65], [66], [67], [73]:

- Computers need to be networked, distributed and transparently accessible.
- Human computer interaction needs to be hidden more.
- Computers need to be context aware in order to optimize their operation in their environment.
- Computers can operate autonomously, without human intervention, be self-governed, in contrast to pure human computer interaction.
- Computers can handle a multiplicity of dynamic actions and interactions, governed by intelligent decision making and intelligent organizational interaction. This may entail some form of artificial intelligence in order to handle:
    - incomplete and non-deterministic interactions,
    - cooperation and competition between members of organizations,
    - richer interaction through sharing of context, semantics and goals.

3.1.1 Distributed Systems Aspects of Ubiquitous Computing

Pervasive computers are networked computers. They offer services that can be locally and remotely accessed. As a result ICT systems are naturally distributed and interlinked. Multiple systems often behave as, and appear as, a single system to the user, i.e., multiple systems are transparent or hidden from the user. Individual systems may be heterogeneous and may be able to be attached and detached from the ICT system infrastructure at any time [64], [73].

A key property of distributed systems is openness. Openness allows systems to avoid having to support all their functions at design time thus avoiding closed implementation. Distributed systems can be designed to support different degrees of openness to dynamically discover new external services and to access them. For example, a UbiComp camera can be set to discover printing services and to notify users that these are available. The camera can then transmit its data to the printer for printing.

On the other hand, openness often introduces complexity and reduces availability. When one function is active, others may need to be deactivated, e.g., some devices cannot record one input while displaying another one. Openness can introduce heterogeneous functions into a system that are incompatible and make the complete system unavailable. Openness can reduce availability because operations can be interrupted when new services and functions are set up.

Many systems are still designed to restrict openness and interoperability even when there appears to be strong benefits not to do so. For example, messages stored in most home answering machines cannot easily be exported, for auditing purposes or as part of a discourse with others. Vendors may deliberately and selectively reduce openness, e.g., transparently ignore the presence of another competitor's services, in order to preserve their market share.

Distributed ICT systems are typically designed in terms of a layered model comprising:

- A hardware resource layer at the bottom, e.g., data source, storage and communication.
- Middleware and operating system services in the middle, e.g., to support data processing and data manipulation
- A human computer interaction layer at the top.

Such a layered ICT model oversimplifies the UbiComp system model because it does not reflect heterogeneous patterns of systems' interaction. This ICT model typically incorporates only a simple explicit human interaction and simple physical world interaction model.

3.1.2 Implicit Human Computer Interaction

Much of human device interaction is designed to support explicit human computer interaction which is expressed at a syntactical low level, for example to activate particular controls in a particular order. In addition, as more tasks are automated, the variety of devices increases and thus more devices need to interoperate to achieve a given set of tasks. The sheer amount of explicit interaction can easily disrupt, distract and overwhelm users. Interactive systems need to be designed to support greater degrees of implicit human computer interaction [64], [65], [66], [67], [73].

The concept of the calm or disappearing computer model has several dimensions. It can mean that programmable computers as we know them today are replaced by something else, for example, by human brain implants that are no longer physically visible. It can mean that computers are present but are hidden, e.g., they are implants or miniature systems. Alternatively, the focus of the disappearing computer can mean that computers are not really hidden; they are visible but are not noticeable as they form part of the peripheral senses. They are not noticeable because of the effective use of implicit human computer interaction. The forms and modes of interaction to enable computers to disappear will depend in part on the target audience because social and cultural boundaries in relation to technology drivers may have different profile clustering attributes. For some groups of people, ubiquitous computing is already here. Applications and technologies, such as mobile phones, email and chat messaging systems, are considered as a necessity by some people in order to function on a daily basis.

The original UbiComp vision focused on making computation and digital information access more seamless and less obtrusive. To achieve this requires in part that systems do not need users to explicitly specify each detail of an interaction to complete a task. For example, using many electronic devices for the first time requires users to explicitly configure some proprietary controls of a timer interface. It should be implicit that if devices use absolute times for scheduling actions, then the first time the device is used, the time should be set. This type of implied computer interaction is referred to as implicit human computer interaction (iHCI). iHCI can be defined as ‘an action, performed by the user that is not primarily aimed to interact with a computerized system but which such a system understands as input’.

Reducing the degree of explicit interaction with computers requires striking a careful balance between several factors. It requires users to become comfortable with giving up increasing control to automated systems that further intrude into their lives, perhaps without the user being aware of it. It requires systems to be able to reliably and accurately detect the user and usage context and to be able to adapt their operation accordingly [68].

3.1.3 Context Awareness

A fundamental aspect of UbiComp is context awareness [69], [70], [71], [72]. There are three main types of context awareness supported in UbiComp:

- *Physical environment context.* This refers to context pertaining to some physical world dimension or phenomena such as location, time, temperature, rainfall, light level, etc.
- *Human context (or user context or person context).* In this context interaction is usefully constrained by users in terms of identity, preferences, task requirements, activities, experience, and knowledge.
- *ICT context or virtual environment context.* Particularly important in a distributed system is awareness of the services that are available internally, externally, locally and remotely in the system.

Generally, the context aware focus of UbiComp systems is on physical world awareness, often in relation to user models and tasks. Ubiquitous computers can utilize where they are and their physical situation or context in order to optimize their services on behalf of users. This is sometimes referred to as context awareness in general but more accurately refers to physical context awareness. A greater awareness of the immediate physical environment could, for example, reduce the energy and other costs of physical resource access.

User context awareness, also known as person awareness, refers to ubiquitous services, resources and devices being used to support user centerd tasks and goals. For example, a photographer may be primarily interested in capturing digital memories of people (the user activity goal) rather than capturing memories of places or of people situated in places. For this reason, a UbiComp camera can be automatically configured to detect faces and to put people in focus when taking pictures.

An important design issue for context aware systems is to balance the degree of user control and awareness of their environment, i.e. active versus passive context awareness. At one extreme, in a (pure) active context aware system, the UbiComp system is aware of the environment context on behalf of the user and automatically adjusts the system to the context without the user being aware of it. This may be useful in applications where there are strict time constraints and the user would not otherwise be able to adapt to the context quickly enough. An example of this is a collision avoidance system built into a vehicle to automatically brake when it detects an obstacle in front of it. In contrast, in a (pure) passive context aware system, the UbiComp system is aware of the environment context on behalf of the user but does not make any automatic adjustments. It just reports the current context to the user without any adaptation, for example, a positioning system reports the location of a moving object on a map.

3.1.4 Autonomy

Autonomy refers to the property of a system that enables a system to control its own actions independently [72], [73]. An autonomous system may still be interfaced with other systems and environments. However, it controls its own actions. Autonomous systems are defined as systems that are self-governing and are capable of their own independent decisions and actions. Autonomous systems may be goal or policy oriented: they operate primarily to adhere to a policy or to achieve a goal.

Autonomous systems can be designed so that these goals can be assigned to them dynamically, perhaps by users. Thus, rather than users needing to interact and control each low level task interaction, users only need to interact to specify high level tasks or goals. The system itself will then automatically plan the set of low level tasks needed and schedule them automatically, reducing the complexity for the user. The system can also replan in case a particular plan or schedule of tasks to achieve goals cannot be reached.

Building, maintaining and interlinking individual systems to be larger, more open, more heterogeneous and complex systems poses a great challenge. Some systems can be relatively simply interlinked at the network layer. However, this does not mean that these can be so easily interlinked at the service layer, for example, interlinking two independent heterogeneous data sources, defined using different data schemas, so that data from both can be aggregated. Such maintenance requires a lot of additional design in order to develop mapping and mediating data models. Complex system interaction, even for automated systems, reintroduces humans in order to manage and maintain the system.

Rather than design systems to focus on pure automation but which end up requiring manual intervention, systems need to be designed to operate more autonomously, to operate in a self-governed way to achieve operational goals. Autonomous systems are related to both context aware systems and intelligence as follows. System autonomy can improve when a system can determine the state of its environment, when it can create and maintain an intelligent behavioral model of its environment and itself, and when it can adapt its actions to this model and to the context.

Autonomous behavior may not necessarily always act in ways that human users expect and understand. For example, self-upgrading may make some services unresponsive while these management processes are occurring. Users may require further explanation and mediated support because of perceived differences between the current state of the system and the users' mental model of the system.

From a software engineering system perspective, autonomous systems are similar to functionally independent systems in which systems are designed to be self-contained, single minded, functional, systems with high cohesion that are relatively independent of other systems (low coupling). Cohesion means the ability of multiple systems or system components to behave as a single unit with respect to specific functions. Such systems are easier to design to support composition, defined as atomic modules that can be combined into larger, more complex, composite modules.

3.1.5 Intelligence

Intelligent systems (IS) are systems which use artificial intelligence (AI), also referred to as machine intelligence, computational intelligence and include (intelligent) agent based systems, software agents and robots [73], [74], [79], [80]. Intelligence can enable systems to act more proactively and dynamically in order to support behaviors such as the following in UbiComp systems:

Modeling of its physical environment. An IS can attune its behavior to act more effectively by taking into account a model of how its environment changes when deciding how it should act.

Modeling and mimicking its human environment. It is useful for an IS to have a model of a human in order to better support iHCI. IS could enable humans to be able to delegate high level goals to the system rather than interact with it through specifying the low level tasks needed to complete the goal.

Handling incompleteness. Systems may also be incomplete because environments are open to change and because system components may fail. AI planning can support re planning to present alternative plans. Part of the system may only be partially observable. Incomplete knowledge of a system's environment can be supplemented by AI type reasoning about the model of its environment in order to deduce what it cannot see is happening.

Handling non deterministic behavior: UbiComp systems can operate in open, service dynamic environments. Actions and goals of users may not be completely determined. System design may need to assume that the environment is a semi deterministic environment (also referred to as a volatile system environment) and be designed to handle this. Intelligent systems use explicit models to handle uncertainty.

Semantic and knowledge based behavior. UbiComp systems are also likely to operate in open and heterogeneous environments. Types of intelligent systems define powerful models to support interoperability between heterogeneous systems and their components, e.g., semantic based interaction.

3.2 Ambient Intelligence

It has long been a dream to have a home that responds to the occupant's needs, anticipating such needs and adapting to the occupant. The goals are generally to maximize comfort and safety, optimize energy usage, enhance general well-being, and eliminate strenuous repetitive activities. Research in this area takes on different labels in the wider research community around the world. Terms such as Ambient Intelligence, Intelligent Environment, Smart Spaces, and Smart Homes are often used interchangeably [78], [79].

Ambient Intelligence has been defined in several slightly different forms. The basic idea behind AmI is that by enriching an environment with technology (e.g., sensors and devices interconnected through a network), a system can be built that acts as an "electronic butler", which senses features of the users and their environment, then reasons about the accumulated data, and finally selects actions to take that will benefit the users within the environment. Therefore the operation of an AmI system can be divided into three stages [75], [83]:

- *Environment Sensing.*
- *Reasoning.*
- *Action.*

Advances in networking technology and the miniaturization of devices and sensors have the potential of making Ambient Intelligence a reality. A typical home today contains a large number of embedded computers, each with a dedicated function and more often than not situated at a fixed location. Embedded computers are found not only in high-tech equipment such as PDAs, iPods, and mobile phones but also in the traditional household appliances such as cookers, washing machines, and fridges. Although useful as individual appliances and devices, their utility is greatly enhanced if connected through a network to allow communication between the devices and appliances.

3.2.1 Environment Sensing

Because Ambient Intelligence is designed for real-world, physical environments, effective use of sensors and actuators is vital. Without physical components that allow an intelligent agent to sense and act upon the environment, an AmI system would have no practical use. Sensors and actuators are the key that link available computational power with physical applications. Ambient Intelligence algorithms rely on sensory data from the real world. AmI systems perceive the environment, use this information to reason about the environment and the actions that should be taken to change the state of the environment, and finally adjust the state of such environment [136], [81], [82], [83].

As mentioned previously, there is a wide array of sensors available. Sensors have been designed to measure many physical parameters such as humidity, light, radiation, temperature, sound, strain, pressure, position, velocity, direction, detection of chemicals and physiological sensing to support health monitoring. Many of these sensors have been available for several years and are well documented. Recently there has been emphasis in developing systems and devices which can determine user activities and needs or, as previously described, user context. Voice recognition, movement recognition, and facial expression recognition systems are all starting to emerge as commonly available technologies for AmI systems. User centered sensors are a fundamental aspect of AmI which distinguishes this field from other, similar fields, such as Home and Building Automation [76], [77], [83].

Making sense of sensor data is a complex task. They generate large volumes of data with unique features. Due to environmental interference and inherent variations of any manufacturing process sensor data can be noisy. Furthermore, if a sensor fails there may be missing values and frequently sensor data may have a spatial or temporal component to it. As a result data from sensors frequently has to be processed and filtered in order to obtain useful information. Kalman filters are a common technique for performing sensor data processing.

When analyzing sensor data, AmI systems may employ a centralized or distributed model. Sensors in the centralized model transmit data to a central server, which fuses and analyzes the data it receives. In the distributed model, each sensor has onboard processing capabilities and performs local computation before communicating partial results to other nodes in the network. The choice of model will have a dramatic effect on the computational architecture and type of sensor that is used for the task. In both cases, sensor data is collected from disparate sources and later combined to produce information that is more accurate, more complete, or more insightful than the individual pieces. Probabilistic approaches have been effective for modeling sensors and combining information from disparate sources.

Sensor data processing which is focused on filtering, disambiguation and interpretation of sensed data before it is used by the higher level decision-making modules usually happens at the middleware level of the system. There the various elements of the distributed technology are integrated. Given the importance that this work has to maximize the understanding of the environment through the sensed data, significant effort has been directed in the scientific community at this area in search of achieving efficient and robust middleware levels within their smart environment systems.

3.2.2 Reasoning

Sensing and acting provide links between AmI algorithms and the real world in which they operate. In order to make such algorithms responsive, adaptive, and beneficial to users, a number of types of reasoning must take place. These include user modeling, activity prediction and recognition, decision making, and spatial-temporal reasoning [78], [79], [80].

One feature that separates general computing algorithms from those that are responsive to the user is the ability to model user behavior. If such a model can be built, it can be used to customize the behavior of the AmI system toward the user. If the model results in an accurate enough baseline, such baseline can provide a basis for detecting anomalies and changes in resident patterns. If the model has the ability to refine itself, the environment can then potentially adapt itself to these changing patterns. AmI user modeling can be characterized based on the following parameters:

- *The data that is used to build the model.*
- *The type of model that is built.*
- *The nature of the model-building algorithm (supervised, unsupervised).*

Another aspect of reasoning is the ability to predict and recognize activities that occur in AmI environments. Because the need for activity recognition technology is great, researchers have explored a number of approaches to this problem. The approaches differ according to the type of sensor data that is used for classification and the model that is designed to learn activity definitions [80].

Different types of sensor information are effective for classifying different types of activities. When trying to recognize actions that involve repetitive body motions (e.g., walking, running, sitting, standing, climbing stairs), data collected from accelerometers positioned on the body has been used. In contrast, other activities are not as easily distinguishable by body position. In these cases, interaction with objects of interest such as doors, windows, refrigerators, keys, and medicine containers can be used to determine the user's activity [83].

Several machine learning models have been used for activity recognition, such as Nave Bayes classifiers and decision trees. Nave Bayes classifiers identify the activity that corresponds with the greatest probability to the set of sensor values that were observed. Decision trees to learn logical descriptions of the activities have also been employed. This approach offers the advantage of generating rules that are understandable by the user, but it is often brittle when high precision numeric data is collected. Other approaches that have been explored are to encode the probabilistic sequence of sensor events using Markov models, dynamic Bayes networks, and conditional random fields [83].

Decision making and control techniques allow AmI systems to automate tasks. For example AI planning systems could be employed to not only remind individuals of their typical next daily activity, but also to complete a task if needed. The use of temporal reasoning with a rule-based system to identify hazardous situations and return an environment to a safe state while contacting the resident has also been proposed.

The use of a Hierarchical Task Network (HTN) planner in AmI systems to generate sequences of actions and contingency plans has also been researched. For example, an HTN enabled AmI system may respond to a sensed medical emergency by calling a medical specialist and sending health vitals using any available device (cell phone, email, or fax). If there is no response from the specialist, the AmI system would then phone the nearest hospital and request ambulance assistance [80], [83].

One of the first, fully-implemented, AmI decision making technology applications is the Adaptive Home, which uses a neural network and a reinforcement learner to determine ideal settings for lights and fans in the home. This is implemented in a home setting and has been evaluated based on an individual living in the Adaptive Home. Another fully-implemented automated living environment is the iDorm project. iDorm is set in a campus dorm environment. The environment is automated using fuzzy rules learned through observation of resident behavior. These rules can be added, modified, and deleted as necessary, which allows the environment to adapt to changing behavior. However, unlike the reinforcement learner approaches, automation is based on imitating resident behavior and therefore is more difficult to employ for alternative goals such as energy efficiency [83].

Very little can be done within an AmI system without an explicit or implicit reference to where and when the meaningful events occurred. For a system to make sensible decisions it has to be aware of where the users are and have been during some period of time. These insights, together with other information, will provide important clues on the type of activities the user is engaged in and the most adequate response.

Spatial and temporal reasoning have been the subject of intense research for a couple of decades and there are well known formalisms and algorithms to deal with spatial, temporal, and spatiotemporal reasoning. The traditional frameworks for spatial reasoning and for temporal reasoning can be used to have a better understanding of the activities in an AmI application. In an environment such as an airport or a home, for example, such reasoning can be used to analyze trajectories of people within a room and classify them as "having a clear goal" or "being erratic" [83].

3.2.3 Acting

AmI systems tie reasoning to the real world through sensing and acting. Intelligent and assistive devices provide a mechanism by which AmI systems can execute actions and affect the system users. There is a large variety of actuators to adjust the environment such as heaters, air conditioners, motorized windows, lights, motorized blinds, humidifiers and others. Furthermore, music, television, and information displays can also be adjusted to better suit the user’s needs, therefore devices such as entertainment systems, television sets, wall displays, HCI mirrors and others can also be considered actuators [136], [81], [82], [83].

Another mechanism is through robots. Research in robotics has progressed to the point where users no longer need to provide detailed instructions for a robot to move to a specified location, but instead can formulate requests such as "bring me the medicine on the counter". Such robot assistants are already found in nursing homes [83].

Robots are able to provide an even wider range of assistive tasks to support AmI. They can monitor the vital signs humans and provide conversational stimulation. Robots are now capable of exhibiting much more human-like emotions and expressions than in the past and can even influence human decision. One such case is the museum traffic control project, where a robot generated cues that caused visitors to travel to portions of the museum that were normally avoided. Robots provide AmI systems with self-mobility and human-likeness, which facilitates human interaction and allows the influence of AmI to more greatly pervade human culture.

3.2.4 Ambient Assisted Living

The number of persons over 60 years is growing faster than any other age group. The number of people in this age group was estimated to be 688 million in 2006, and is projected to grow to almost two billion by 2050. By that time, the population of older people will be much larger than that of children under the age of 14 years for the first time in human history. People prefer to age at home and remain productive but there may not be enough caregivers available to provide support to the elderly. 89% of the older adults prefer to stay in the comfort of their own homes [89]. These changes in the world's demographics will also result in many challenges for society and the health care system, such as:

- *Increase in diseases.* It is expected that an increase in age-related diseases, such as Alzheimer's disease or Parkinson's disease, will occur in the near future.
- *Increase in health care costs.* Rising health care costs are expected. Currently more than 40% of the U.S. health care budget is destined to senior citizens while this age group for only 13% of the U.S. population. Therefore, the current model of health care will become strained as the aging population increases during the next decades.
- *Shortage of caregivers.* There will be a shortage of professionals trained to work with the aging population, which means more family members have to take the role of informal caregivers. Caring for dependent individuals at home will result in many complications for the informal caregivers, such as higher levels of emotional distress and physical health problems.
- *Dependency.* With an increase in age-related diseases, there will also be a rise in individuals unable to live independently. As the number of senior citizens increases there is the pressing matter of how to provide quality care to our aging population.
- *Larger impact on society.* Economists believe that as a society, we will be unable to provide the human resources required for the older individuals to live in assisted living or skilled nursing facilities. It is also estimated that annual loss to employers of various

working family care givers is about $33 billion ($2100 per employee) for absenteeism, workplace disruptions, and reduced work status.

Assisted living technologies based on Ambient Intelligence are called Ambient-Assisted Living (AAL) technologies [89], [87]. Ambient Assisted Living was conceived as one strategy for addressing the difficulties that this forthcoming demographic shift will give rise to. It proposes the use of Information & Communications Technologies (ICTs) to deliver innovative applications and services that would enable the elderly to live independently for longer, and reduce the need for long-term care. Independence is closely associated with health; the World Health Organization (WHO) regards health as a combination of physical, mental and social wellbeing, as distinct from the mere absence of disease. Thus it behold AAL to contribute to the maintenance of each. Moreover, it has been demonstrated that greater independence, implying fewer care needs, correlates with better housing conditions. It is here that a key justification for proceeding with AAL, rather than the mere financial, may be found.

AAL can be used for preventing, curing, and improving wellness and health conditions of older adults. AAL tools such as medication management tools and medication reminders allow older adults to take control of their health conditions [84], [87], [85], [89], [86], [88]. AAL technologies can also provide increased safety for the elderly by using mobile emergency response systems, fall detection systems, and video surveillance systems. AAL technologies can provide help with daily activities by monitoring activities of daily living (ADL) and issuing reminders, as well as helping with mobility and automation. AAL systems can also allow older adults to better connect and communicate with their peers, as well as with their family and friends.

Ambient Assisted Living (AAL) tools support assisted living by being sensitive and responsive to the presence of people [87], [88]. Ambient Assisted Living fosters the provision of equipment and services for the independent living of elderly people, via the seamless integration of information and communication technologies within homes and extended homes, thus increasing their quality of life and autonomy and reducing the need for being institutionalized. These include assistance to carry out their daily activities through smart objects, health and activity monitoring systems including wearables as well as context-aware services, enhancing safety and security, getting access to social, medical and emergency systems, and facilitating social contacts, in addition to context-based infotainment and entertainment.

Several recently emerging technologies are helping to make the vision of AAL a reality[89]:

1. *Smart Homes.* As mentioned previously, a smart home is a regular home which has been augmented with various types of sensors and actuators. Rich context information can be obtained by analyzing and fusing various types of sensor data. Most smart homes utilize such knowledge for automation and providing more comfort for the residents, as well as for assessing the cognitive and physical health of the residents.
2. *Mobile and Wearable Sensors.* Most smart phones are equipped with various sensors such as accelerometer, gyroscope, proximity sensor, and global positioning system (GPS), which can be used for detecting user activity and mobility. Also, recent advances in epidermal electronics and MEMS technology promise a new era of health-related sensor technology. Researchers have already developed noninvasive sensors in form of patches, small Holter-type devices, body-worn devices, and smart garments to monitor health signals.
3. *Robotics.* Assistive robots allow older adults to overcome their physical limitations by helping them in their daily activities. Assistive robots can be classified into three categories: robots assisting with ADL activities, robots assisting with instrumental activities of daily living (IADL), and robots assisting with enhanced activities of daily living (EADL). ADL tasks include self-maintenance activities, such as feeding, dressing, grooming, etc. IADL tasks include the ability to use instruments in daily living, such as the successful use of the telephone, preparing food, etc. EADLs include participation in social activities, such as engaging in hobbies.

3.3 Distributed Systems

A distributed system can be defined as a collection of independent computers that appears to its users as a single coherent system [90], [91], [120]. One important characteristic of a distributed system is that differences between the various computers and the ways in which they communicate are mostly hidden from users. The same holds for the internal organization of the distributed system. Another important characteristic is that users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.

Distributed systems must also be relatively easy to expand or scale. This characteristic is a direct consequence of having independent computers while simultaneously hiding how these computers actually take part in the system as a whole. A distributed system will normally be continuously available, although perhaps some parts may be temporarily out of order. Users and applications should not notice that parts are being replaced or fixed, or that new parts are added to serve more users or applications.

In order to support heterogeneous computers and networks while offering a single-system view, distributed systems are often organized by means of a layer of software that is logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems and basic communication facilities, such a distributed system is called middleware [92], [117].

There are four important goals that should be met by a distributed system [92]:

Ease of resource access. A distributed system must make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way. Typical examples of resources include things like printers, computers, storage facilities, data, files, Web pages, and networks. There are many reasons for wanting to share resources. One obvious reason is that of economics. For example, it is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user. Likewise, it makes economic sense to share costly resources such as supercomputers, high-performance storage systems, imagesetters, and other expensive peripherals. Connecting users and resources also makes it easier to collaborate and exchange information, as is clearly illustrated by the success of the Internet with its simple protocols for exchanging files, mail, documents, audio, and video.

Distribution transparency. An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers. A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent. There are several types of transparency:

- *Access.* Hides differences in data representation and how a resource is accessed.
- *Location.* Hides where a resource is located.
- *Migration.* Hides that a resource may move to another location.
- *Relocation.* Hides that a resource may be moved to another location while in use.
- *Replication.* Hides that a resource is replicated.
- *Concurrency.* Hides that a resource may be shared by several competitive users.
- *Failure.* Hides the failure and recovery of a resource.

Openness. Another important goal of distributed systems is openness. An open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services. Services are generally specified through interfaces, which are often described in an Interface Definition Language (IDL). Interface definitions written in an IDL nearly always capture only the syntax of services. In other words, they specify precisely the names of the functions that are available together with the types of the parameters, return values, possible exceptions that can be raised, etc. If properly specified, an interface definition allows an arbitrary process that needs a certain interface to talk to another process that provides that interface. It also allows two independent parties to build completely different implementations of those interfaces, leading to two separate distributed systems that operate in exactly the same way. Proper specifications are complete and neutral. Complete means that everything that is necessary to make an implementation has indeed been specified. Neutral means that specifications should not prescribe what an implementation should look like. Completeness and neutrality are important for interoperability and portability. Interoperability characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard. Portability characterizes to what extent an application developed for a distributed system can be executed, without modification, on a different distributed system that implements the same interfaces as the first one.

Another important goal for an open distributed system is that it should be easy to configure the system out of different components, possibly from different developers. It should also be easy to add new components or replace existing ones without affecting those components that stay in place. In other words, an open distributed system should also be extensible.

Scalability. Scalability is one of the most important design goals for developers of distributed systems. Scalability of a system can be measured along at least three different dimensions:

- *Size.* A system can be scalable with respect to its size, meaning that more users and resources can be easily added to the system.
- *Geographic.* A geographically scalable system is one in which the users and resources may lie far apart.
- *Administrative.* A system is administratively scalable if it can still be easy to manage even if it spans many independent administrative organizations.

Unfortunately, a system that is scalable in one or more of these dimensions often exhibits some loss of performance as the system scales up.

3.4 Sensors

A sensor is a device that receives a stimulus and responds with an electrical signal [93], [94], [97]. The stimulus is the quantity, property, or condition that is received and converted into an electrical signal. The term sensor should be distinguished from transducer. A transducer is a converter of any one type of energy into another, whereas a sensor converts any type of energy into electrical energy. Transducers may be used as actuators in various systems. An actuator may be described as an opposite to a sensor; it converts an electrical signal into generally nonelectrical energy. For example, an electric motor is an actuator; it converts electric energy into mechanical action.

3.4.1 Sensor Classification Schemes

Sensor classification schemes range from very simple to the complex. Depending on the purpose, different classification criteria may be selected.

Not all types of stimuli can be directly converted to an electrical signal, sometimes it is necessary to use one or more transducers before the stimulus can be converted to an electrical signal. Accordingly sensors can be broadly classified into one of two categories [96]:

- *Direct Sensor.* A direct sensor converts a stimulus into an electrical signal or modifies an electrical signal by using an appropriate physical effect.
- *Complex Sensor.* A complex sensor needs one or more transducers before a direct sensor can be employed to generate an electrical output.

Sensors can also be classified, depending on whether they require an external power source or not, into:

- *Passive.* A passive sensor does not need any additional energy source and directly generates an electric signal in response to an external stimulus. That is, the input stimulus energy is converted by the sensor into the output signal. Examples of sensors in this category are a thermocouple, a photodiode, and a piezoelectric sensor.
- *Active.* Active sensors require an external power signal for their operation, which is called an excitation signal. That signal is modified by the sensor to produce the output signal. Active sensors are sometimes called parametric because their own properties change in response to an external effect and these properties can be subsequently converted into electric signals. In this case a sensor’s parameter modulates the excitation signal and that modulation carries information of the measured value. For example, a thermistor is a temperature sensitive resistor. It does not generate any electric signal, but by passing an electric current through it (excitation signal) its resistance can be measured by detecting variations in current and/or voltage across the thermistor. These variations presented, in ohms, directly relate to temperature through a known transfer function.

Based on the selected reference, sensors can also be classified into the following [96]:

- *Absolute.* An absolute sensor detects a stimulus in reference to an absolute physical scale that is independent of the measurement conditions. An example of an absolute sensor is a thermistor. Its electrical resistance directly relates to the absolute temperature scale of Kelvin.
- *Relative.* Relative sensors produce a signal that relates to some baseline reference value. An example of a relative sensor is a thermocouple. A thermocouple produces an electric voltage, which is a function of a temperature gradient across its wires. Thus, a thermocouple's output signal cannot be related to any particular temperature without referencing to a known baseline.

Finally, sensors can also be classified depending on the stimulus to which they respond [94–97]:

- *Acoustic.* Wave amplitude, phase, polarization, spectrum, Wave velocity.
- *Biological.* Biomass types, concentration, states.
- *Chemical.* Substance identities, concentration, states.
- *Electric.* Charge, current, voltage (amplitude, phase, polarization, spectrum), conductivity, permittivity.
- *Magnetic.* Magnetic field (amplitude, phase, polarization, spectrum), magnetic flux, permeability.
- *Optical.* Wave amplitude, phase, polarization, spectrum, wave velocity, refractive index, emissivity, reflectivity, absorption.
- *Mechanical.* Position (linear, angular), acceleration, force, stress, pressure, strain, mass, density, moment, torque, speed of flow, rate of mass transport, shape, roughness, orientation, stiffness, compliance, viscosity, crystallinity, structural integrity.
- *Radiation.* Type, energy, intensity.
- *Thermal.* Temperature, flux, specific heat, thermal conductivity.

3.4.2 Sensor Characteristics

The following are some of the main sensor characteristics [94], [95], [96], [97]:

1. *Transfer Function.* The transfer function shows the functional relationship between physical input signal and electrical output signal. Usually, this relationship is represented as a graph showing the relationship between the input and output signal, and the details of this relationship may constitute a complete description of the sensor characteristics. For expensive sensors that are individually calibrated, this might take the form of the certified calibration curve.
2. *Sensitivity.* The sensitivity is defined in terms of the relationship between input physical signal and output electrical signal. It is generally the ratio between a small change in electrical signal to a small change in physical signal. As such, it may be expressed as the derivative of the transfer function with respect to physical signal. A thermometer would have "high sensitivity" if a small temperature change resulted in a large voltage change.
3. *Span or Dynamic Range.* The range of input physical signals that may be converted to electrical signals by the sensor is the dynamic range or span. Signals outside of this range are expected to cause unacceptably large inaccuracy. This span or dynamic range is usually specified by the sensor supplier as the range over which other performance characteristics described in the data sheets are expected to apply.
4. *Accuracy or Uncertainty.* Uncertainty is generally defined as the largest expected error between actual and ideal output signals. Sometimes this is quoted as a fraction of the full-scale output or a fraction of the reading. For example, a thermometer might be guaranteed accurate to within 5% of FSO (Full Scale Output).
5. *Hysteresis.* Some sensors do not return to the same output value when the input stimulus is cycled up or down. The width of the expected error in terms of the measured quantity is defined as the hysteresis. Typical units are percent of FSO.
6. *Nonlinearity.* The maximum deviation from a linear transfer function over the specified dynamic range. There are several measures of this error. The most common compares the actual transfer function with the "best straight line," which lies midway between the two parallel lines that encompass the entire transfer function over the specified dynamic range of the device. This choice of comparison method is popular because it makes most sensors look the best. Other reference lines may be used, so the user should be careful to compare using the same reference.

7. *Noise.* All sensors produce some output noise in addition to the output signal. In some cases, the noise of the sensor is less than the noise of the next element in the electronics, or less than the fluctuations in the physical signal, in which case it is not important. Many other cases exist in which the noise of the sensor limits the performance of the system based on the sensor. Noise is generally distributed across the frequency spectrum. Many common noise sources produce a white noise distribution, which is to say that the spectral noise density is the same at all frequencies.
8. *Resolution.* The resolution of a sensor is defined as the minimum detectable signal fluctuation. Since fluctuations are temporal phenomena, there is some relationship between the timescale for the fluctuation and the minimum detectable amplitude. Therefore, the definition of resolution must include some information about the nature of the measurement being carried out. Many sensors are limited by noise with a white spectral distribution.
9. *Bandwidth.* All sensors have finite response times to an instantaneous change in physical signal. In addition, many sensors have decay times, which would represent the time after a step change in physical signal for the sensor output to decay to its original value. The reciprocal of these times correspond to the upper and lower cutoff frequencies, respectively. The bandwidth of a sensor is the frequency range between these two frequencies.

3.5 Sensor and Actuator Communications Protocols

The purpose of a sensor is to respond to some kind of an input physical property (stimulus) and to convert it into an electrical signal that is compatible with electronic circuits. There is a wide array of communication protocols developed for use with sensors and actuators. In this section we describe some commonly used protocols.

3.5.1 X-10

X-10 is a domotics communication standard introduced in the 1970s. The original X-10 protocol was intended to control devices and thus did not incorporate any means of receiving data from sensors. The protocol was later expanded with the introduction of extended commands to allow its use in sensors. Due to the low cost and wide availability of X-10 based devices this standard still enjoys wide spread use.

X-10 is a power line based communication standard which employs 120 kHz bursts synchronized with the zero crossing of the power line's alternating current waveform. A binary ‘1’ is represented by a 1ms burst near the zero crossing while a binary ‘0’ is represented by the absence of such burst [98].

Standard X-10 messages are composed by a Start Code, followed by a House Code and finally a Key Code. The Start Code is represented by the unique sequence ‘1110’, while House Codes and Key Codes employ complementary pairs for each bit.

The House Code is a 4 bit identifier intended to avoid commands destined for one house from activating sensors or actuators in another house. The 5 bit Key Code can either indicate the address of an X-10 module, a standard command, or can indicate the beginning of an extended command. Addresses and commands are distinguished from each other by means of the Key Code's LSB. If the LSB is ‘1’ then the Key Code in question is a command, while if the LSB is a ‘0’ the Key Code is an address. When an X-10 module is addressed through an appropriate Key Code, it will continue to respond to X-10 commands until either a new address is issued or an “All Units Off” command is issued [99].

Standard commands are X-10 commands which allow a system to interact with actuators and perform functions such as turning lights or devices on or off, but do not provide a means to obtain readings from sensors. Obtaining data from sensors or sending more complex commands to actuators requires the use of extended commands. After the beginning of an extended command is signaled by an appropriate Key Code, the rest of the extended command is made up of a 4-bit address, an 8-bit data field and an 8-bit command.

In order to avoid collisions, before using the medium an X-10 transmitter must wait for a random interval between 8 and 10 zero-crossings. If during this interval there have not been any data ‘1’ bits transmitted, the device can begin transmission. During the transmission of each ‘0’ bit (no 120 kHz burst) the transmitter must monitor the medium for 120 KHz bursts to detect a collision. If a collision is detected, the transmitter must abort its transmission and recommence the transmission process.

3.5.2 TIA-485

The TIA-485 standard, titled Electrical Characteristics of Balanced Voltage Digital Interface Circuits and commonly known as RS-485, specifies a balanced data-transmission scheme for transmitting data over long distances in noisy environments [100]. The key features of RS-485 are:

1. Balanced Interface.
2. Multipoint operation from a single 5V supply.
3. -7V to +12V bus common-mode range.
4. Up to 32 unit loads.
5. 10 Mbps maximum data rate (at 40-feet).
6. 4000 foot maximum cable length (at 100 kbps).

RS-485 employs a bus topology which can be designed for full-duplex or half-duplex transmission (Figure 3-1). The full-duplex implementation requires two signal pairs and transceivers with separate bus access lines for transmitter and receiver. Both full-duplex and half-duplex implementations require operation of all nodes via control signals, such as Driver/Receiver Enable signals, to ensure that only one driver is active on the bus at any time. Having more than one driver accessing the bus at the same time leads to bus contention which should be avoided at all times through software control.

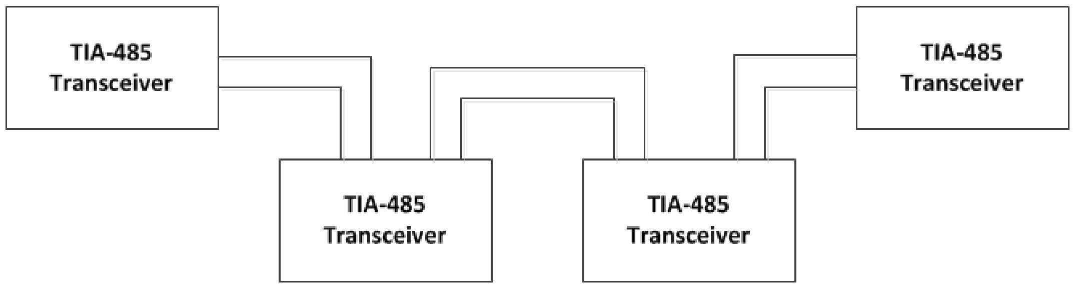


Figure 3-1. Typical TIA-485 Half-Duplex Network.

RS-485 specifies a differential driver output of 1.5V minimum across a 54Ω load while a receiver must detect a differential input down to 200mV. These values provide sufficient margin for reliable data transmission even with severe signal degradation across the cable and connectors. This robustness is the main reason why RS-485 is well suited for long-distance networking in noisy environments [101].

Since a driver's output depends on the current it must supply into a load, adding transceivers to the bus increases the total load current required. RS-485 specifies a load unit referred to as a Unit Load (UL). A UL represents a load impedance of approximately 12 kΩ. An RS-485 driver must be able to drive 32 ULs, however, current RS-485 transceivers commonly provide reduced unit loading which allows a larger number of devices to participate in an RS-485 bus.

RS-485 enjoys wide acceptance and is a physical layer found in popular industrial automation protocols such as Modbus and DMX 512.

3.5.3 IEEE 802

IEEE 802 is a family of standards for Local Area Networks and Metropolitan Area Networks published by the Institute of Electrical and Electronics Engineers. IEEE 802 contains descriptions of the networks considered as well as a reference model for protocol standards. The networks are intended to have wide applicability in many environments [102].

An IEEE 802 LAN is a peer-to-peer communication network that enables stations to communicate directly on a point-to-point, or point-to-multipoint, basis without requiring them to communicate with any intermediate switching nodes. LAN communications takes place at moderate-to-high data rates and with short transit delays, on the order of a few milliseconds or less. A MAN is optimized for a larger geographical area than a LAN. As with local networks, MANs can also depend on communications channels of moderate-to-high data rates.

IEEE 802 networks employ packet-based communications capabilities as opposed to cell-based or isochronous networks. That is, the basic unit of transmission is a sequence of data octets which can be of any length within a range that is dependent on the type of LAN. For all LAN types the maximum length is in excess of 1000 octets.

The IEEE 802 LAN/MAN Reference Model is patterned after the ISO/IEC 7498-1 Open Systems Interconnection (OSI) Basic Reference Model [102], [103]. These reference models are based on a scheme composed of layers in which the services of a layer are the capabilities it offers to a user in the next higher layer. In order to provide such services a layer builds its functions on the services it obtains from the next lower layer.

The IEEE 802 family of standards encompass the lower two layers of the OSI Reference Model, i.e. the Physical Layer and the Data Link Layer. In IEEE 802 the OSI Data Link Layer is further subdivided as two sub-layers: the Logical Link Control layer defined in the IEEE 802.2 standard and the Medium Access Control layer which is defined in each of the standards aimed at a specific medium type.

The IEEE 802 family of standards is comprised of a large number of individual standards covering many areas of device networking; however the most relevant standards for sensor and actuator communication protocols are those which cover Ethernet, Wi-Fi and WPAN.

3.5.3.1 IEEE 802.2

The services provided by the Data Link layer of the OSI reference model are aimed at providing the means of transferring data between network-entities contained within a network, i.e. network-entities identified by data-link-addresses [104]. These services are used by local network layer entities to exchange packets with remote network layer entities.

As mentioned previously, in the IEEE 802 family of standards the OSI Data Link layer is divided into a Logical Link Control sublayer (LLC) and a Medium Access Control sublayer (MAC). The LLC uses the services exposed by the MAC sublayer to provide the services of the Data Link layer to entities in the Network layer. The following three forms of services are provided by the LLC sublayer:

1. *Unacknowledged connectionless-mode services:* This set of data transfer services provides the means by which network entities can exchange link layer service data units (LSDUs) without the establishment of a data link level connection. The data transfer can be point-to-point, multicast, or broadcast.

2. *Connection-mode services:* This set of services provides the means for establishing, using, resetting, and terminating data link layer connections. These connections are point-to-point connections. The following are the connection-mode services:
    a. *Connection establishment service.* This service provides the means by which a network entity can request or be notified of the establishment of data link layer connections.
    b. *Connection-oriented data transfer service.* Provides the means by which a network entity can send or receive LSDUs over a data link layer connection. This service also provides data link layer sequencing, flow control, and error recovery.
    c. *Connection reset service.* This service provides the means by which established connections can be returned to the initial state.
    d. *Connection termination service.* Provides the means by which a network entity can request, or be notified of, the termination of data link layer connections.
    e. *Connection flow control service.* This service provides the means to control the flow of data associated with a specified connection across the network layer/data link layer interface.
3. *Acknowledged connectionless-mode services:* The acknowledged connectionless-mode data unit exchange services provide the means by which network layer entities can exchange LSDUs that are acknowledged at the LLC sublayer, without the establishment of a data link connection. The services provide a means by which a network layer entity at one station can send a data unit to another station, request a previously prepared data unit from another station, or exchange data units with another station. The data unit transfer is point-to-point.

The Medium Access Control sublayer performs the functions necessary to provide packet-based, connectionless-mode data transfer services to the LLC sublayer. The main functions performed by the MAC sublayer are the following [104]:

1. Frame delimiting and recognition.

2. Addressing of individual destination stations and groups of destination stations.
3. Transport of source-station addressing information.
4. Transparent data transfer of LLC Protocol Data Units (PDUs)
5. Protection against errors, generally by means of generating and checking frame check sequences.
6. Control of access to the physical transmission medium.

Other functions of the MAC sublayer, which apply to interconnection devices such as hubs of bridges, include the following:

1. Flow control between an end station and an interconnection device.
2. Filtering of frames according to their destination addresses to reduce the extent of propagation of frames in parts of a LAN or MAN that do not contain communication paths leading to the intended destination end station or stations.

Certain functions of the MAC sublayer, such as the control of access to the physical medium, are specific to every Physical layer type and therefore both the MAC and PHY sublayers are specified together in various other standards. The following IEEE 802 MAC/PHY standards are some of the most commonly used for sensor-actuator communications.

3.5.3.2 IEEE 802.3

The IEEE 802.3 Working Group develops standards for Ethernet-based Local Area Networks and Metropolitan Area Networks [105]. The term Ethernet refers to a family of computer networking technologies for wired LANs. The original Ethernet was an experimental coaxial cable network developed in the 1970s by Xerox at their Palo Alto Research Center (PARC). The original Ethernet was designed to operate with a data rate of 3 Mbps using a carrier sense, multiple access, collision detect access method for LANs with occasionally heavy traffic requirements [106], [107].

The original IEEE 802.3 standard was based on, and was very similar to, the Ethernet Version 1.0 specification [Ethernet Technologies – CISCO docwiki]. Since then the standard has evolved to cover different wired media types such as coaxial, twisted-pair or fiber optic cables. Currently speeds range from 1 Mbps to 100 Gbps using a common media access control specification.

IEEE 802.3 LANs consist of network nodes and interconnecting media. The network nodes fall into two major classes:

1. *Data Terminal Equipment (DTE):* These are devices that are either the source or the destination of data frames. Examples of DTEs are PCs, file servers, or print servers.
2. *Data Communications Equipment (DCE):* These are intermediate network devices that receive and forward frames across the network. DCEs may be either standalone devices such as repeaters, switches and routers, or communications interface units such as interface cards or modems.

The IEEE 802.3 standard provides for two distinct modes of operation: half duplex and full duplex. A given IEEE 802.3 network operates in either half or full duplex mode at any one time.

In half duplex mode stations share the common transmission medium through the Carrier Sense, Multiple Access, Collision Detect (CSMA/CD) media access method. In this method, to transmit, a station waits for a quiet period on the medium and then sends the intended message in bit-serial form. If, after initiating a transmission, the message collides with that of another station, then each transmitting station intentionally transmits for an additional predefined period to ensure propagation of the collision throughout the system. The station then remains silent for a random amount of time (backoff period) before attempting to transmit again.

Full duplex operation allows simultaneous communication between pairs of stations using point-to-point media, i.e. a dedicated transmission channel. Full duplex operation does not require for stations to wait until the medium is quiet, nor do the stations monitor or react to receive activity, as there is no contention for the shared medium in this mode. Full duplex mode can only be used when all of the following are true:

1. The physical medium is capable of supporting simultaneous transmission and reception without interference.
2. There are exactly two stations connected with a full duplex point-to-point link. Since there is no contention for use of a shared medium, the multiple access algorithms are unnecessary.
3. Both stations on the LAN are capable of, and have been configured to use, full duplex operation.

The most common configuration envisioned for full duplex operation consists of a network switch with a dedicated media segment connecting each switch port to a single device. With the increasing availability of low cost network switches full duplex operation is quickly becoming the norm.

3.5.3.3 IEEE 802.11

The IEEE 802.11 standard defines one medium access control (MAC) and several physical layer (PHY) specifications to provide wireless connectivity for fixed, portable, and moving stations within a local area [110].

Wireless networks possess characteristics that make them significantly different from traditional wired LANs. In IEEE 802.11, the addressable unit is referred to as a station (STA). Such STAs can be subdivided into fixed STAs, portable STAs, and mobile STAs. A portable STA is one that is moved from location to location, but that is only used while at a fixed location. Mobile STAs actually access the LAN while in motion.

It is not sufficient to handle portable STAs only. Mobile STAs are often battery powered which means that power management is an important consideration in the IEEE 802.11 standard. For example, it cannot be presumed that a STA's receiver is always powered on.

As a result the PHYs used in the IEEE 802.11 standard are fundamentally different from wired media. Thus IEEE 802.11 PHYs:

1. Use a medium that has neither absolute nor readily observable boundaries.
2. Are unprotected from other signals that are sharing the medium.
3. Communicate over a medium significantly less reliable than wired PHYs.

4. Have dynamic topologies.
5. Lack full connectivity and therefore the assumption that every STA can normally "hear" every other STA is invalid.
6. Have time-varying and asymmetric propagation properties.
7. Might experience interference from logically disjoint IEEE 802.11 networks operating in overlapping areas.

The fundamental topological component of the IEEE 802.11 standard is the Basic Service Set (BSS). A BSS is formed by two or more STAs linked in a way that allows the exchange of data between said STAs, following the mechanisms dictated by the IEEE 802.11 standard [108], [109], [110]. When the BSS is independent of any other network, the BSS is called an Independent Basic Service Set (IBSS). In an IBSS, also known as an Ad Hoc Network, STAs communicate directly with each other.

BSSs can exchange data with each other and with other networks by means of a Distribution System (DS). To make use of the Distribution System a dedicated STA in the BSS must be tasked with transferring frames to/from the DS and the STAs. Such a STA is termed an Access Point (AP) in the IEEE 802.11 standard and a BSS which includes an AP is called an infrastructure BSS. In an infrastructure Basic Service Set all data exchange, including the exchange of frames between STAs belonging to the infrastructure BSS, is carried out through the AP. To access all the services of an infrastructure BSS a STA must first become "associated". Such associations are dynamic and involve the use of a Distribution System Service (DSS) described in the IEEE 802.11 standard.

By joining infrastructure Basic Service Sets through a Distribution System the IEEE 802.11 standard provides for the creation of wireless networks of arbitrary size and complexity. The service set resulting from joining two or more infrastructure BSSs is called an Extended Service Set (ESS).

The final topology supported in the IEEE 802.11 standard is the Mesh BSS (MBSS). An MBSS is an IEEE 802.11 LAN consisting of autonomous STAs. Inside the MBSS, all STAs establish wireless links with neighbor STAs to mutually exchange messages. Messages can be transferred between STAs that are not in direct communication with each other by using a multi-hop mechanism. From the data delivery point of view, it appears as if all STAs in an MBSS are directly connected at the MAC layer even if the STAs are not within range of each other. The multi-hop capability enhances the range of the STAs and benefits wireless LAN deployments. STAs in a mesh BSS might be sources, sinks, or propagators of traffic.

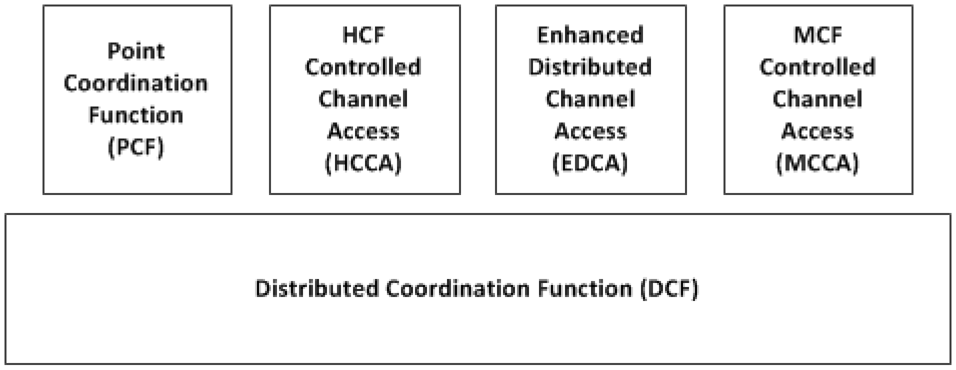

Point Coordination Function (PCF)
HCF Controlled Channel Access (HCCA)
Enhanced Distributed Channel Access (EDCA)
MCF Controlled Channel Access (MCCA)
Distributed Coordination Function (DCF)

Figure 3-2. IEEE 802.11 MAC Architecture.

Due to the distinct characteristics of the wireless medium (WM) and the diverse topologies defined in the standard, IEEE 802.11 offers several methods of medium access control:

1. *DCF.* Distributed Coordination Function (DCF) is the fundamental access method of the IEEE 802.11 MAC [110] (Figure 3-2). DCF is a contention based medium access control method which implements a Carrier Sense, Multiple Access, Collision Avoidance mechanism. In DCF, STAs monitor the WM to determine when said medium is available for transmission. When the WM is determined to be available, the STA waits a random back off period during which it continues to monitor the WM. If at the end of the back off period the medium is still available the STA then transmits its frame. Since collisions cannot be detected in a WM the STA expects to receive an Acknowledge (ACK) frame to determine that the transmission was successful. If such a frame is not

received the STA assumes that a collision has occurred and the process is repeated.

Depending upon the placement of the STAs in a BSS and the physical obstacles located between them, it is possible that a STA might be unable to detect the transmissions of another STA. As a result, a STA may begin transmission assuming the WM is available when in fact the medium is being used by a another “hidden” STA.

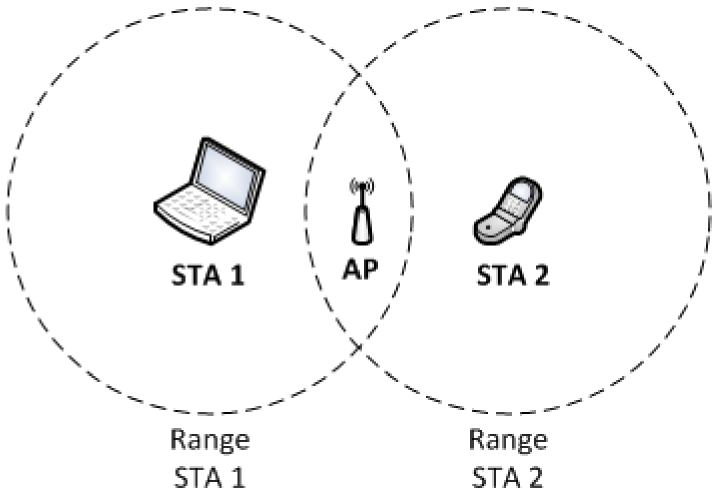


Figure 3-3. Hidden Terminal Problem.

IEEE 802.11 offers a refinement of the CSMA/CA method, the Request To Send/ Clear To Send (RTS/CTS) mechanism, which can be used under these and other circumstances.

The RTS/CTS medium access method requires that a sender STA begin by transmitting an RTS frame to the receiver STA. The receiver STA must then send a CTS frame back to the sender to indicate that the handshake has been successful and ensure that the medium has been reserved for the particular sender and receiver for the transmission. The RTS and CTS frames include information about the time period that the medium will be occupied. This information is used by the rest of the STAs to determine that the medium is busy even if they are unable to detect the transmissions from the sender STA.

2. *PCF*. The Point Coordination Function (PCF) is an optional medium access method that uses an alternating cycle of Contention Periods and Contention Free Periods [110]. During the CPs STAs contend for use of the WM using the DCF method described previously. During the CFP the AP polls each STA allowing it to make use of the medium. PCF can only operate in infrastructure BSSs. Although PCF has been implemented in a few devices it has not enjoyed widespread acceptance, hence some authors consider this medium access mechanism dead.
3. *HCF*. To support quality of service (QoS) IEEE 802.11 provides an additional coordination function called Hybrid Coordination Function [110]. The HCF combines functions from the DCF and PCF with QoS-specific enhancements that allow QoS data transfers during both the CP and CFP. A STA that provides QoS services is referred to as a QSTA in the IEEE 802.11 standard, while an AP that provides QoS is termed a QAP. The BSS where a QAP and QSTAs operate is referred to as a QBSS.

   Use of the wireless medium in a QBSS is regulated through the use of a time unit referred to as a Transmission Opportunity (TXOP). When a QSTA gets access to the medium it is said to be granted a TXOP. TXOPs are characterized by a starting time and a maximum duration, called TXOP Limit.

   HCF uses both a contention-based channel access method, called the Enhanced Distributed Channel Access (EDCA) mechanism, for contention-based transfer and a controlled channel access, referred to as the HCF Controlled Channel Access (HCCA) mechanism, for contention-free transfer:

   a. *EDCA*. To differentiate traffic types, the EDCA mechanism defines four different Access Categories (AC):

      i. *AC_BK*. Background access category. Traffic assigned to this access category has the lowest priority.

      ii. *AC_BE*. Best effort access category.

      iii. *AC_VI*. Video access category.

      iv. *AC_VO*. Voice access category. Traffic assigned to this access category has the highest priority.

Every QSTA in a QBSS maintains four transmit queues, one for each AC. Each queue contends for the channel using a method called Enhanced Distributed Channel Access Function (EDCAF). EDCAF is an enhanced version of DCF based on the same principles of CSMA/CA and back off periods, however the parameters used by the EDCAF of each queue to contend for a TXOP are different. This parameter difference ensures that the highest priority queues will have a higher opportunity of obtaining TXOPs and hence access to the WM.

b. *HCCA*. The HCCA mechanism uses a QoS-aware centralized coordinator, called a Hybrid Coordinator (HC), which is collocated with the QAP of the QBSS. The HC has a higher access priority to the WM than the rest of the devices in the QBSS. This allows the HC to allocate TXOPs to itself and the other STAs in order to provide limited-duration Controlled Access Phases (CAPs) for contention-free transfer of QoS data. TXOP allocations and contention-free transfers of QoS traffic are based on the HC's QBSS-wide knowledge of the amounts of pending traffic belonging to different Traffic Streams (TS) and/or Traffic Categories (TC).

4. *MCF*. Mesh Coordination Function (MCF) is an additional coordination function usable only in an MBSS [110]. MCF has both a contention-based channel access and a contention free channel access mechanism. The contention-based mechanism is EDCA while the contention free mechanism is called the MCF Controlled Channel Access (MCCA). MCCA is an optional access method that allows mesh STAs to access the wireless medium with lower contention.

   As in HCF, MCF uses TXOPs to allocate the right to transmit onto the wireless medium. There are two types of TXOP in MCF: EDCA TXOPs and MCCA TXOPs. The EDCA TXOP is obtained by a mesh STA winning an instance of EDCA contention while an MCCA TXOP is obtained by gaining control of the WM during an MCCAOP. The MCCAOP is a time interval for frame transmissions that has been reserved by the exchange of management frames. To initiate reservation of an MCCAOP a mesh STA transmits an MCCA Setup

Request frame. Said mesh STA becomes the owner of the MCCAOP reservation. The receivers of the MCCA Setup Request frame are the MCCAOP responders. The MCCAOP owner and the MCCAOP responders advertise this MCCAOP reservation to their neighbors through an MCCAOP advertisement. MCCA enabled neighbor mesh STAs that could cause interference during these reserved time periods do not initiate a transmission during these reserved periods. During its MCCAOP, the MCCAOP owner obtains a TXOP by winning an instance of EDCA contention. Due to its reservation, the MCCAOP owner experiences no competition from other MCCA enabled neighbor mesh STAs.

3.5.3.4 IEEE 802.15.4

IEEE 802.15.4 wireless technology is a short-range communication system intended for Low-Rate Wireless Personal Areas Networks (WPANs) [111]. The key features of the IEEE 802.15.4 wireless technology are low complexity, low cost, low power consumption and low data rate transmissions to be supported by cheap, fixed or moving devices [112]. The IEEE 802.15.4 standard provides specifications for the MAC and PHY layers. For a definition of the upper layers other standards, such as the ZigBee stack specified by the industrial consortia ZigBee Alliance [113] and the IPv6 over Low-power PAN (6LowPAN) [114], have been developed.

The IEEE 802.15.4 PHY operates using Direct Sequence Spread Spectrum in three different unlicensed bands according to the geographical area where the system is deployed:

1. The 868 MHz band in the European area with a raised-cosine-shaped Binary Phase Shift Keying (BPSK) modulation format. The ideal transmission range is approximately 1km.
2. The 915 MHz band in the North America and Pacific area with a raised-cosine-shaped BPSK modulation format. The ideal transmission range is approximately 1 km.
3. The 2.4 GHz ISM band with a half-sine-shaped Offset Quadrature Phase Shift Keying (O-QPSK) modulation format. The ideal transmission range is approximately 200m.

Transmission is organized in frames which are designated as a Physical Protocol Data Unit (PPDU). There are four types of PPDUs: a beacon frame, a data frame, an ACK frame and a MAC command frame. All are formed with a Synchronization Header (SHR), a Physical Header (PHR) and a Physical Service Data Unit (PSDU) which is composed of a MAC Payload Data Unit (MPDU). The MPDU is composed of a MAC Header (MHR), a MAC Service Data Unit (MSDU) and a MAC Footer (MFR), except for the ACK frame who's MPDU is composed only of an MHR and an MFR.

To improve the transmission range IEEE 802.15.4 devices can self-organize into either star or multi-hop peer-to-peer topologies. Star topologies are preferable for small area low-latency applications whereas peer-to-peer topologies are better suited when a large area has to be covered and latency is not an issue. To support these topologies IEEE 802.15.4 defines two types of devices:

1. *Full Function Device (FFD).* FFDs contain the complete set of MAC services and can operate either as a WPAN coordinator, or as a simple device. FFDs are the only nodes allowed to form links with other devices.
2. *Reduced Function Device (RFD).* RFDs contain a subset of MAC services and can only operate as a network device.

3.5.4 ZigBee

Following the same scheme employed in the OSI and IEEE 802 reference models the ZigBee architecture is made up of a set of layers, with each layer performing a specific set of services for the layer above. The IEEE 802.15.4-2004 standard defines the two lower layers: the PHY layer and the MAC sub-layer. On top of this the ZigBee standard defines the Network (NWK) layer and the application (APL) layer [115], [116]. The NWK layer performs the following operations:

1. Configuring a new device. A new device can begin operation as a ZigBee coordinator or try to join an existing network.
2. Starting a new network.
3. Joining and leaving a network.
4. NWK layer security.

5. Routing frames to their destination. Only ZigBee coordinators and routers can relay messages.
6. Discovering and maintaining routes.
7. Discovering one-hop neighbors and storing one-hop neighbor information.
8. Assigning addresses to devices joining the network. Only ZigBee coordinators and routers can assign addresses.

The APL layer consists of the application support sub-layer (APS), the ZigBee device objects (ZDO) and the manufacturer-defined application objects. Manufacturer-defined application objects use the application framework and share APS and security services with the ZDO. The APS provides an interface between the NWK layer and the rest of the APL layer components. The APS performs the following functions:

1. Maintain binding tables.
2. Forward messages between bound devices.
3. Manage group addresses.
4. Map 64-bit IEEE address to 16-bit network address, and vice versa.
5. Support reliable data transport.
6. While the ZDO performs the following functions:
7. Define the role of the device (ZigBee coordinator, router, or device).
8. Discover the devices on the network and their application. Initiate or respond to binding requests.
9. Perform security-related tasks.

3.6 Middleware

A middleware is a software layer which allows the components of a distributed system to interact [117]. Many different types of middleware have been developed, each based on a different paradigm, and targeted towards the solution of a certain class of problems.

3.6.1 Middleware Requirements

The proposed requirements for a middleware are the following [118], [119], [120]:

Network Communication. Distributed systems need a network for communication because the components are located on different hosts. This communication requires the transformation of the complex data structures into a suitable format which can be transmitted using transport protocols. Such transformations are called marshalling and unmarshalling. To enable automatic (un-) marshalling all data involved in a request have to be described. This can be done with the IDL.

Coordination. Coordination is required to control multiple communication points, which exist in distributed systems. There are several mechanisms, which can influence coordination of the components in a distributed system. These include synchronization and activation (deactivation) policies. Synchronization is required during communication of concurrent components. There are two ways to achieve synchronization. In synchronous communication a component is blocked until another component completes execution of a requested service. On the other hand when the component that requests some service from another component remains unblocked and can continue to perform its operations, then the communication is called asynchronous. Another coordination mechanism is activation and deactivation policies. Activation (deactivation) allows starting (ending) a component independently of the applications it executes.

Reliability. There are several types of reliability proposed in the literature: best effort, at-most-once, at least-once and exactly-once. Best effort service does not give any assurance about the successful execution of the request. At-most-once requests are guaranteed to execute only once or none at all. At-least-once service requests are guaranteed to be executed, possibly more than once. Exactly-once requests are guaranteed to be executed once and only once. Distributed system implementations need to include error detection and correction mechanisms to for reliability purposes. If some components in the system are not available then the reliability of the system suffers. The increase reliability in such scenarios component replication is employed. However increasing reliability decreases performance. It is a trade-of problem.

Scalability. Scalability defines the ability of a system to adapt to changes in demand. The limited scalability of centralized systems can be overcome by Distributed Systems. The main challenge here is to provide the flexibility of a distributed system without changing the architecture or design of the original system. In order to achieve this task middleware has to respect the different dimensions of transparency specified by the ISO Open Distributed Processing (ODP) reference model. For example, access transparency means that a component can access the services of a remote component as if it were local. Location transparency means that components are not aware of the physical location of the components with which they interact. Migration transparency allows components to change their location without affecting clients requesting services from these components. Replication transparency means that requesting components do not care about the location of the required service, it can either be the main component or a replica. Load balancing, where requests are forwarded to a replica in order to remove load from a server, can use replication mechanisms. A scalable system middleware needs to support access, location, migration, and replication transparency.

Heterogeneity. The components of distributed systems can be of different types (legacy or new components) and written in different programming languages. For example, legacy components tend to be written in imperative languages, such as COBOL, or C while newer components are often implemented using object-oriented programming languages. There are different dimensions of heterogeneity in DSs such as the hardware and operating system platforms, programming languages and the middleware itself. These differences need to be addressed by the middleware.

3.6.2 Middleware Classification

Middleware can be classified into four main types [121]:

3.6.2.1 Transactional Middleware (TM)

Transactional middleware (TM) or transaction processing (TP) monitors were designed to support distributed synchronous transactions. The main function of a TP monitor is the coordination of requests between clients and servers that can process these requests. A request is a message that asks the system to execute a transaction. TM uses clustering of the service requests into transactions. A transaction must support ACID properties, i.e. Atomic, Consistent, Isolated, and Durable. Atomic means that the transaction either completes entirely or is rejected. Consistency means that the system will remain in a coherent state regardless of the status of a transaction. Isolation is the ability of one transaction to be carried out independently of other transactions that are possibly run on the same TP monitor. Durability means that a transaction can survive system failures once the transaction is committed and complete.

Typical products: IBM's CICS, BEA's Tuxedo, Transarc's Encina.

Network Communication: Client and server components can reside on different hosts and therefore requests are transported via the network in a way that is transparent to client and server components.

Coordination: TP monitors can coordinate the distributed transactions through the two-phase commit protocol (2PC). This protocol is based on the "prepare to commit phase" and the "commit phase". Both synchronous and asynchronous communications are supported by TM. TM has support for various activation policies and services that can be activated and deactivated on demand if they haven't been used for some time. The server components can always reside in memory, hence enabling a persistent activation. Many TP monitors support failover and possess restart capabilities, thus increasing application up time.

Reliability: TM requires the 2PC to implement distributed transactions. The transaction can be committed, only if all processes involved in a transaction are ready to commit, otherwise the transaction is aborted. TP monitors use transactions logs, which can undo changes. The Failure/recovery service, which is supported by most TP monitors increases fault-tolerance and reliability consequently. Message queues are also supported by TM, thus enabling reliability when disk storage is used for queues. TM also supports database management systems (DBMS) which guarantee fault-tolerance.

Scalability: TP monitors are rather scalable, because they support load balancing and replication of server components. Load balancing is important, because TP monitors have to cope with a lot of transactions in a limited time. In order to sustain consistent response times, TP monitors are capable of starting additional process instances. This is an important feature for any enterprise environment that needs to have ensured scalability.

Heterogeneity: Heterogeneity support is realized on different levels. TM supports software and hardware heterogeneity, because the components can be located on different hardware and operating system platforms. TM, as mentioned above, has DBMS support. DBMS components can participate in transactions due to the Distributed Transaction Processing (DTP) Protocol, adopted by the Open Group. But TM doesn't support data heterogeneity very well, because it cannot express complex data structures and therefore can't marshall these structures.

Advantages:

- Components are kept in consistent states (due to ACID properties of transaction)
- TM is very reliable.
- TP monitors perform better than message-oriented and procedural middleware.
- TP monitors can dispatch, schedule and prioritize multiple application requests concurrently, thus reducing CPU overhead, response times and CPU cost for large applications.

Disadvantages:

- TM has often unnecessary or undesirable guarantees according to ACID. If a client is performing long-lived activities, then transactions could prevent other clients from being able to continue.
- Marshalling and unmarshalling have to be done manually in many products.
- The lack of a common standard for defining the services that server components offer reduces the portability of a DS between different TP monitors.
- TM runs on a smaller variety of platforms (UNIX and NT server only) compared to other types of middleware.

3.6.2.2 Message Oriented Middleware (MOM)

Message Oriented Middleware (MOM) is a type of middleware in which communication is carried out through messages. There are two different types of MOM: message queuing and message passing. Message queuing is an indirect communication model where communication is performed via a queue. Messages from one program are sent to a specific queue, identified by name, where they are stored and later sent to a receiver. Message queues require a Queue Manager and can be divided into two categories:

Persistent Message Queues. In persistent message queues messages are stored in persistent storage, such as an underlying database, until they can be delivered. In case of a server failure information will be restored after the server restarts. Persistent queues are preferred when reliability is more important than performance, such as in banking fund transfer systems.

Non-Persistent Message Queues. These types of queues do not make use of persistent storage and hence information is lost in the event of a server failure. The advantage of non-persistent queues is usually better performance.

In message passing, a direct communication model, information is sent directly to interested parties.

Typical products: IBM's MQSeries and Sun's Java Message Queue.

Network Communication: Network communication in MOM is based on messages. Messages are defined as strings of bytes that have meaning to the applications that exchange the data. Besides application related data, messages might include control data relevant to the message queuing system. This information is used to store, route, deliver, retrieve and track the payload data. After receiving a message from a component the server replies with a message which contains the results of the service execution. Most messaging products provide good support for several communication protocols.

Coordination: MOM supports both synchronous, via message passing, and asynchronous, via message queuing, communication. In asynchronous communications messages are sent to a server without blocking the client. The client does not need to wait for a reply and can proceed with other actions. Synchronous communication needs to be implemented manually in the client. MOM supports activation on demand. This is achieved through triggers where an application program is started whenever a request message or a reply message has arrived on a local queue, and the application program is not already active. The use of activation on demand decreases resource use.

Reliability: MOM can be seen as fault tolerant because it can use persistent queues which are stored in persistent media. To increase reliability message queuing supports different types of Quality of Service (QoS). These QoS types are defined as follows:

- Reliable message delivery. During exchange of messages no network packets are lost.
- Guaranteed message delivery. Messages are delivered to the destination node either immediately (with no latency - network is available), or eventually (with latency - when the network is unavailable). In the latter case, middleware guarantees that messages are delivered as long as the network becomes available within a specified time period.
- Assured, non-duplicate message delivery. If messages are delivered they are delivered only once.

Scalability: The publish-subscribe communications model provides location transparency, allowing a program to send the message with a subject as the destination property while the middleware routes the messages to all programs that have subscribed to that subject. Although location transparency is supported, MOMs have a limited support for access transparency. This is due to queues being used for remote and not for local communication.

Heterogeneity: The support of data heterogeneity is rather limited, because marshalling is not automatically generated, and needs to be implemented by programmers.

Advantages:

- MOM supports group communication, which is atomic. Either all clients receive a delivery or none. That's why a process doesn't have to worry about what to do, if some clients don't receive a message.
- The use of persistent queues increases reliability in MOM products.
- Support for transactional message queues is included in most MOM products, meaning that advanced delivery guarantees are supported.
- MOM supports more network protocols than RPC.
- MOM can send messages exactly-once due to QoS, thus increasing its reliability.

Disadvantages:

- There is bad portability support because MOM products do not support any standards. Applications that are made for one MOM product are not compatible with another MOM product.

3.6.2.3 Procedural Middleware (PM)

Remote Procedure Calls (RPCs) were developed by Sun Microsystems in the early 1980s. RPCs are present on different operating systems, including most UNIX and MS Windows systems. Windows NT, for example, supports lightweight RPCs across processes and, with DCOM, full RPCs.

Typical products: Open Software Foundation's Distributed Computing Environment DCE, Microsoft RPC Facility.

Network Communication: RPC defines server components as RPC programs. An RPC program contains parameterized procedures. Remote clients can invoke these procedures across the network using the network protocols. These protocols are low-level, such as TCP or UDP. Communication happens as follows; if a client wants to receive some service, then it makes a request to a server. This request consists of a message, which includes the marshalled parameters. On the other side, the server receives this message, unmarshalls the parameters, executes the requested service and sends the result back to the client. In order to connect the client and server components of a distributed application using RPC as the middleware link, it is required that every function that a client application can call should be represented by a stub, i.e. a placeholder, for the real function on the server. We can note that RPC is a category of middleware where marshalling and unmarshalling are implemented automatically by the IDL compiler in stubs.

Coordination: RPC-based communication is synchronous, i.e. an RPC client is blocked until the remote procedure has been executed or an error occurs. RPC does not support asynchronous communication. Procedural middleware provides different forms of activation policies. These include activation of RPC on demand and "RPC is always available".

Reliability: Procedural middleware possess at-most once reliability. If RPC fails then an exception is returned. RPC communication based on the TCP protocol can be seen as reliable since TCP provides reliability. An application that uses TCP knows that data it sends is received at the other end, and that it is received correctly.

Scalability: Scalability in RPC is rather limited because RPC lacks replication mechanisms. As mentioned previously, communication is based on stubs, which provides location transparency of the requested service to the client. This means that a client can invoke a remote procedure as if it were local. As mentioned in the requirements section location transparency is a prerequisite to scalability.

Heterogeneity: Heterogeneity in RPC is achieved through the IDL. It can define interfaces that represent relations between servers and clients. IDL is programming language independent, which means that a client does not need to know the language that the server supports as long as the IDL compiler can translate the client's request to a server.

Advantages:

- RPC has good heterogeneity support because RPC has bindings for multiple operating systems and programming languages.
- Marshalling and unmarshalling are automatically generated, thus simplifying the development of DSs.

Disadvantages:

- RPCs do not support group communication.
- They have no direct support for asynchronous communication, replication and load balancing, therefore leading to limited scalability.
- Fault tolerance is worse than for other middleware types, because many possible faults have to be caught and dealt with in the program.

3.6.2.4 Object-oriented Middleware (OOM)

Object middleware, evolved from RPC, extends it by adding object-oriented concepts such as inheritance, object references and exceptions. OOM allows referencing of remote objects and can call operations from these.

Typical products: OMG's CORBA, Microsoft COM, Java RMI and Enterprise Java Beans.

Network Communication: Object middleware supports distributed object requests. It is possible for a client to request an operation on a server object on another host. To achieve this, a client requires a reference to the server object. Marshalling of the parameters for the network request is made automatically in the client and server stubs.

Coordination: OOM generally supports synchronous communication. The client object remains blocked, waiting for the server object's response. This does not mean that other synchronization types are not supported. CORBA 3.0, for example, supports both deferred synchronous and asynchronous object requests. OOM supports different activation and threading policies. In OOM server objects can be active all the time or started on demand. CORBA Concurrency Service enables threading and coordinates concurrent access to shared resources. It also guarantees consistency of the object if it is accessed by multiple clients. CORBA also supports group communication through its Event and Notification services.

Reliability: At most once reliability is available by default in OOM. To handle failures during component requests OOM uses exceptions. Noticeable is the fact that due to CORBA messaging, exactly-once reliability can be achieved. FaultTolerant CORBA, for example, provides extra reliability in DS.

Scalability: Scalability remains limited. Some CORBA implementations support load-balancing. Enterprise Java Beans have replication support which increases scalability.

Heterogeneity: OOM provides wide support for heterogeneity. For example CORBA and COM both have multiple programming language bindings so that client and server objects do not need to be written in the same programming language. Java/RMI solves heterogeneity through its Java Virtual Machine.

Advantages:

- Marshalling and unmarshalling are generated automatically in client and server stubs.

- OOM supports both synchronous and asynchronous communication.
- Most OOM products support messaging and transactions. Hence, OOM can replace the other three types of middleware in certain applications.

Disadvantages:

- Lack of scalability.

3.7 Component Oriented Development

Component-oriented programming enables programs to be constructed from prebuilt software components, which are reusable, self-contained blocks of computer code. These components have to follow certain predefined standards including interface, connections, versioning, and deployment. While OOP emphasizes classes and objects, COP emphasizes interfaces and composition. In this sense, COP is an interface-based programming. Clients in COP do not need any knowledge of how a component implements its interfaces, as long as interfaces remain unchanged, clients are not affected by changes in interface implementations [122].

Building systems out of components is a natural part of engineering systems. The automotive industry, for instance, develops very complex cars using components of every size from a tiny screw to complex subsystems such as engines and transmissions. The modern automotive factory has become more of a system integrator than a manufacturer. It is easy to name many other industries and engineering disciplines making effective use of components by a rigorous set of standards that define interoperability. The main idea of the component-based approach is building systems from pre-existing components. As a result the development process of component-based systems is separated from the development process of the components themselves [123].

System development with components is focused on the identification of reusable entities and relations between them, based on the system requirements and the availability of already existing components.

The architecture of component-based systems is significantly more demanding than that of traditional monolithic integrated solutions. The process of building components can follow an arbitrary development process model. However any development model will require certain modification, since in addition to the demands on the component functionality, a component is built to be reused. Reusability implies generality and flexibility, and these requirements will significantly define the component characteristics. Generality often implies more functionality, requires more design and development effort, as well as qualified developers. Component development will also require more effort in the testing and specification stages. Components should be tested in isolation, but also in different configurations. Finally, documentation and delivery will require more efforts since detailed documentation is very important for an adequate understanding of the component.

Despite the initial higher development effort required by component development, generality and flexibility provide definite advantages. Change is inherent in software engineering. The user requirements change, specifications change, personnel change, budget changes, technology changes, and so on. One of the fundamental software engineering principles is to emphasize the importance of managing change. It is important to place primary emphasis during architecture and design on the dependencies between the components, and on the management of those dependencies. COD provides an effective way to follow the software engineering principle of dealing with change: planning for change, design for change, and building for change. Components are easy to adapt to new and changing requirements.

Software reuse allows to design and implement something once and to use it over and over again in different contexts [124]. There are different levels of software reuse. COD supports the highest level of software reuse because it allows various kinds of reuse including white-box reuse, gray-box reuse, and black-box reuse. White-box reuse means that the source of a software component is made available and can be studied, reused, adapted, or modified. Black-box reuse is based on the principle of information hiding. The interface specifies the services a client may request from a component. The component provides the implementation of the interface that the clients rely on. As long as the interfaces remain unchanged, components can be changed internally without affecting clients. Gray-box reuse is somewhere in between white-box reuse and black-box reuse.

As the size and complexity of software systems grows, the identification and proper management of interconnections among the pieces of the system becomes a central concern. COD provides a manageable solution to deal with the complexity of software, the constant change of systems, and the problems of software reuse.

A software component is a piece of self-contained, self-deployable computer code with well-defined functionality and can be assembled with other components through its interface.

From this definition, a component is a program or a collection of programs that can be compiled and made executable. It is self-contained; thus, it provides coherent functionality. It is self-deployable so that it can be installed and executed in an end user's environment. It can be assembled with other components so that it can be reused as a unit in various contexts.

Interfaces are the means by which components connect and exchange messages. An interface is a set of named operations that can be invoked by clients. Each operation's semantics is specified, and this specification plays a dual role as it serves both providers implementing the interface and clients using the interface. Providers and clients are ignorant of each other, the specification of the interface becomes the mediating middle that lets the two parties work together. It is therefore important to view interfaces and their specifications in isolation of any specific component that may implement or use such interfaces. Interface specifications can be considered as contracts between a client of an interface and a provider of an implementation of the interface. The contract states what the client needs to do to use the interface. It also states what the provider has to implement to meet the services promised by the interface.

While interfaces reside at the endpoints of interactions, message schemas reside on the logical line between interacting parties. Messages are directional, traveling from a sender to a receiver. Such logical lines can connect multiple senders and receivers. While the communication line connecting senders and receivers is an abstraction that refers to endpoints, the communicated messages are not. A message schema describes a set of valid messages, usually with no constraints on particular senders or receivers. A particular message schema may require information in a message that identifies the sender and/or the receiver, but such identification is not a basic requirement for all messages.

3.8 Design Patterns

Design patterns are solutions to software design problems you find again and again in real-world application development [125], [126], [127]. Patterns are formalized best practices that the programmer himself must implement in the application. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

The 23 Gang of Four (GoF) patterns are generally considered the foundation for all other patterns. They are categorized in three groups: Creational, Structural, and Behavioral

- *Creational Patterns.* Deal with object creation mechanisms:
  - *Abstract Factory*. Creates an instance of several families of classes.
  - *Builder*. Separates object construction from its representation.
  - *Factory*. Method Creates an instance of several derived classes.
  - *Prototype.* A fully initialized instance to be copied or cloned.
  - *Singleton.* A class of which only a single instance can exist.
- *Structural Patterns.* Ease the design by identifying a simple way to realize relationships between entities:
  - *Adapter.* Match interfaces of different classes.
  - *Bridge.* Separates an object’s interface from its implementation.
  - *Composite.* A tree structure of simple and composite objects.
  - *Decorator.* Add responsibilities to objects dynamically.
  - *Façade.* A single class that represents an entire subsystem.
  - *Flyweight.* A fine-grained instance used for efficient sharing.
  - *Proxy.* An object representing another object.
- *Behavioral Patterns.* Identify common communication patterns between objects and realize these patterns.

- *Chain of Responsibility.* A way of passing a request between a chain of objects.
- *Command.* Encapsulate a command request as an object.
- *Interpreter.* A way to include language elements in a program.
- *Iterator.* Sequentially access the elements of a collection.
- *Mediator.* Defines simplified communication between classes.
- *Memento.* Capture and restore an object's internal state.
- *Observer.* A way of notifying change to a number of classes.
- *State.* Alter an object's behavior when its state changes.
- *Strategy.* Encapsulates an algorithm inside a class.
- *Template.* Method Defer the exact steps of an algorithm to a subclass.
- *Visitor.* Defines a new operation to a class without change.

3.9 IEC 61499

A control system is a device or set of devices to manage, command, direct or regulate the behavior of other devices or systems. Although control systems of various types date back to antiquity, a more formal analysis of the field began with a dynamics analysis of the centrifugal governor, conducted by Maxwell in 1868 entitled “On Governors” [128].

Many industrial control systems fall into one of two categories, either based on traditional distributed control systems (DCSs) or on programmable logic controllers (PLCs). DCSs like the ones commonly found in petrochemical plants and refineries are structured around a few large processors that provide supervisory control and data acquisition communicating via local networks with controllers, instruments, sensors and actuators located out in the plant. These types of systems may have both discrete instruments and out-stations with clusters of instruments with local controllers. In a DCS the main supervisory control comes from one or more of the central processors while instruments positioned out in the plant typically provide local closed loop control such as PID control.

On the other hand, many machine control and production processes such as those found in automotive production lines have generally been designed using PLCs. A large PLC system will generally have a number of PLCs communicating via one or more proprietary high-speed networks. PLCs will generally be connected to a large number of input and output (I/O) signals for handling sensors and actuators. In some cases discrete instruments, for example for temperature and pressure control, are also connected to PLCs.

With both design approaches systems have tended to be developed by writing large monolithic software packages which are generally difficult to re-use in new applications and are notably difficult to integrate with each other. The data and functions of one application are not easily available to other applications even if they are written in the same programming language and are running in the same machine.

In order to promote the development of flexible solutions the concept of function blocks was introduced in industrial systems. A function block is a robust, re-usable software component. The function block is an abstract model representing a function that can be implemented by software and/ or hardware. A function block can provide a software solution to a small problem such as the control of a valve, or control a major unit of a plant, such as a complete production line. Function blocks allow industrial algorithms to be encapsulated in a form that can be readily understood and applied by people who are not software specialists. Each block has a defined set of input parameters, which are read by the internal algorithm when it executes. The results from the algorithm are written to the block’s outputs. Complete applications can be built from networks of function blocks formed by interconnecting block inputs and outputs [129], [133].

The function block concept was standardized for programmable logic controllers in the International Electrotechnical Commission’s IEC 61131 standard [130]. To extend the concept of function block outside the realm of PLCs the IEC 61499 standard was developed. The IEC 61499 built upon the function block concept defined in IEC 61131-3 to develop a generic standard that can also be applied in other industrial sectors; in fact wherever there is a requirement for software components that behave as function blocks, such as in building management systems [131].

IEC 61499 defines a general model and methodology for describing function blocks in a format that is independent of implementation. It allows a system to be defined in terms of logically connected function blocks that run on different processing resources. IEC 61499 provides terminology, models and concepts to allow the implementation of a function block oriented distributed control system to be described in an unambiguous and formal manner. Having a formal and standard approach to describing systems allows such systems to be validated, compared and understood.

While previous standards employ a data or signal based communication among the constructs and assume a cyclic execution, IEC 61499 introduces an event driven approach of interaction among the function blocks. This means that algorithms are executed only if an input event activates the block. As a result functions are separated from the execution control [132], [133]. Writing a program with function blocks involves drawing a network of function blocks and allocating them to devices for execution.

As the trend to use component based software continues, it is foreseen that industrial controllers and instruments will either provide function blocks as part of the device firmware or provide function block libraries from which blocks can be selected and downloaded. System design will become the process of software component selection, configuration and interconnection, just as much of electronic hardware design is now primarily concerned with the selection and interconnection of IC chips. In a function block world, the system designer’s main focus is to take standard proven encapsulated functionality and link it together in the quickest and most intuitive way possible. The use of function blocks is nearer to the mind-set of the industrial system designer who is familiar with connecting physical devices together in different ways to provide a particular system solution.

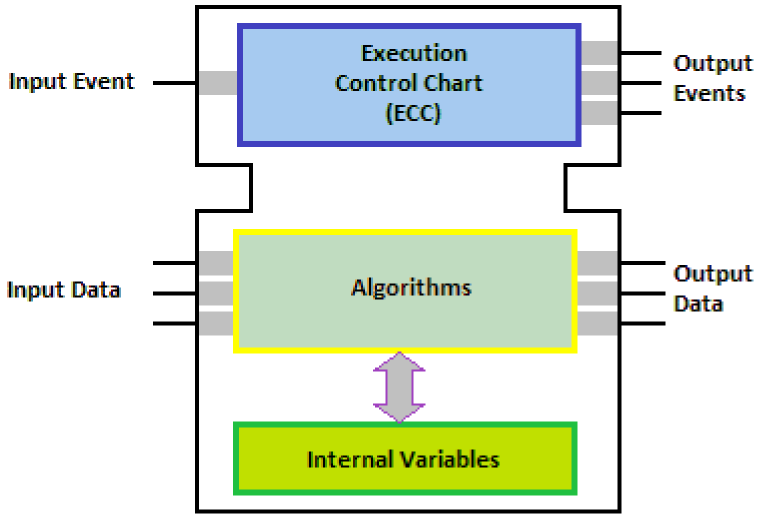


Figure 3-4. IEC 61499 Function Block.

The Function Block (FB), the basic construct of IEC61499, consists of a head and a body as shown in Figure 3-4. The head is connected to the event flows and the body to the data flows, while the functionality of the function block is provided by means of algorithms, which process inputs and internal data and generate output data. The sequencing of algorithm invocations is defined in the FB type specification using a variant of state charts called Execution Control Chart (ECC).

An ECC consists of EC states, EC transitions and EC actions. An EC state may have zero or more associated EC actions, except from the initial state that shall have no associated EC actions. An EC action may have an associated algorithm and an event that will be issued after the execution of the algorithm. EC transitions are directed links that represent the transition of the FB instance from one state to another. An EC transition has an associated Boolean expression that may contain event inputs, data inputs, and internal variables. As soon as this expression becomes true the EC transition fires [134].

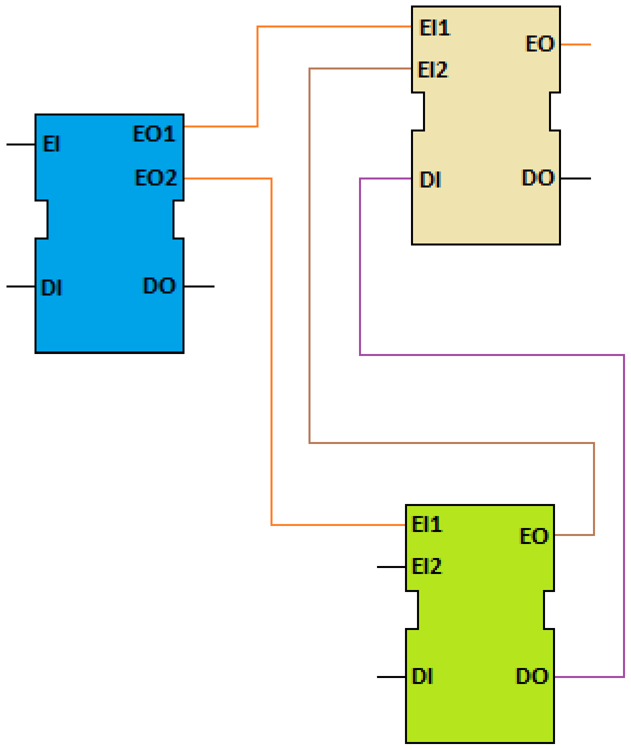


Figure 3-5. Function Block Network.

FB instances are interconnected to form a Function Block Network (FBN), as shown in Figure 3-5. An FBN may be executed on a single device or on a network of interconnected devices. A device may contain zero or more resources, where a resource is considered to be a functional unit, contained in a device which has independent control of its operation and may be created, configured, parameterized, started-up, deleted, etc., without affecting other resources within a device. The event connections and behavior of every single block completely determines the behavior of the network. An application in IEC 61499 is composed by one or more FBNs.

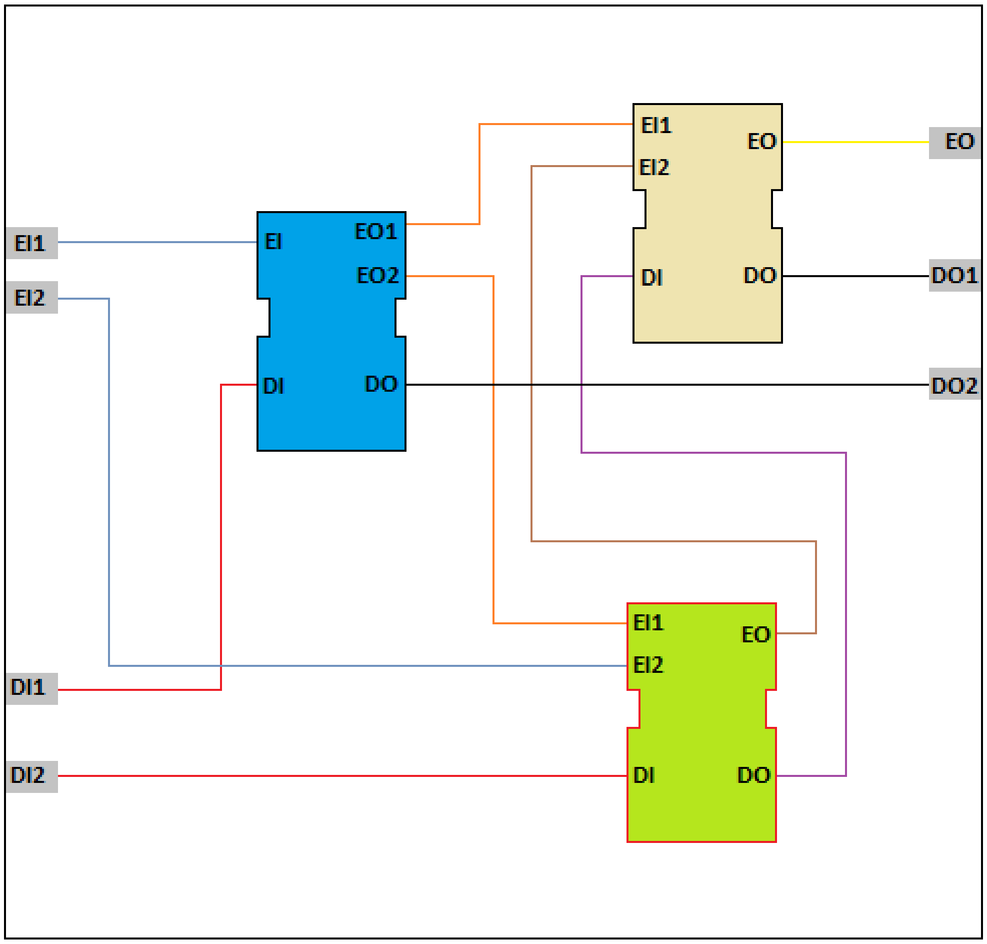


Figure 3-6. Composite Function Block.

Composite function blocks (Figure 3-6) provide a means for building up more complex blocks from basic and other smaller composite blocks in a hierarchical fashion. The type definition for a composite function block contains declarations of function block instances of selected types that are linked by data and event connections. The standard regards blocks that are used within a composite block as component function blocks. The data connections between component blocks define the transfer of data values between block outputs to inputs while the event connections define the order of execution of algorithms within the blocks. A composite function block is just a container for a network of other function blocks. The container as such performs no specific actions except for setting input and output variables and for the activities of its components. The network can include basic and composite function block types.

4 Modular Framework for Ambient Intelligence Systems

Our framework is an event-driven, minimalist, modular framework for the development of Ambient Intelligence systems. Contrary to other AmI frameworks, our modular framework for Ambient Intelligence systems approaches the development of AmI systems from the point of view of distributed control systems using the function block abstraction described in the IEC 61499 standard. This framework also makes use of a publish/subscribe, store and forward interaction scheme which is a well-established mechanism for the development of loosely coupled event-based distributed systems.

Due to its modular design and the use of the function module abstraction we call our modular framework for Ambient Intelligence systems Funblocks. Developing an AmI system using Funblocks consists of developing or reusing function blocks which provide the desired functionality and afterwards joining these blocks through the services provided by the framework.

4.1 Framework Design Principles

A static environment, where changes do not occur, does not require any type of adjustment, everything is always the same. In such an environment, since no activities or changes happen, there is no need to make any type of adaptation. We require Ambient Intelligence because our environments are dynamic and in constant change. But, from the point of view of Ambient Intelligence, our environments do not require constant adjustment; rather, it is the activities of the inhabitants that trigger the need to adapt an environment. Each activity performed by a user has a defined start and a defined stop. In other words, each activity performed by a person can be seen as an event. As a result, an event-driven processing system is very well suited as a basis for an Ambient Intelligence system.

Ambient intelligence (AmI) is a fast-growing multidisciplinary field that is expected to have a significant impact on society. Its foundation is the enrichment of the environment, through sensing and processing technologies, to understand, analyze, anticipate, and adapt to events and to users and their activities, preferences, intentions, and behaviors. Basically, AmI gathers real-time information from the environment along with user activities, and combines it with historical data accumulated over time to provide user services. Interfacing with the user is a major aspect of any AmI application. While interfaces may employ different technologies, the underlying notion of user centricity dominates their design. In AmI, the traditional paradigm of human–computer interaction, in which users must adapt to computers by learning how to use them, is replaced by a new one, in which computers adapt to users and learn how to interact with them in the most natural way.

4.2 Framework Components

To provide an overall view of Funblocks, in this section we give a brief description of the components that constitute the framework. In later sections we will describe with greater detail some of these components.

Our framework is formed by the following components (Figure 4-1):

1. *Controller (CTRL).*
2. *Middleware (MDLW).*
3. *Middleware Communications Interface (MCI).*
4. *Sensors/ Actuators (SAs).*
5. *Human/ Computer Interface (HCI).*
6. *Module and Sensor/ Actuator Catalog (MSAC).*
7. *Function Modules (FM).*
8. *Function Module Repositories (FMRs).*
9. *External Communications Modules (ECM).*

A fundamental component of Funblocks is the event-driven store and forward middleware. This middleware, which we call Midblocks due to its use in the Funblocks framework, provides the communication mechanism through which the components of the system interact. An innovative feature of Midblocks is its capability to monitor the operation of the components of the system through a component supervision mechanism. As a result, Midblocks can be employed in other systems which require keeping track of component operation such as systems with reliability requirements or systems with self-healing capabilities. To encourage the use of Midblocks in other systems besides the Funblocks framework careful attention was paid to the design of the middleware to avoid any dependency issues with the modular framework for AmI systems. Due to its independent nature from the rest of Funblocks, a detailed description of Midblocks is provided in the next chapter.

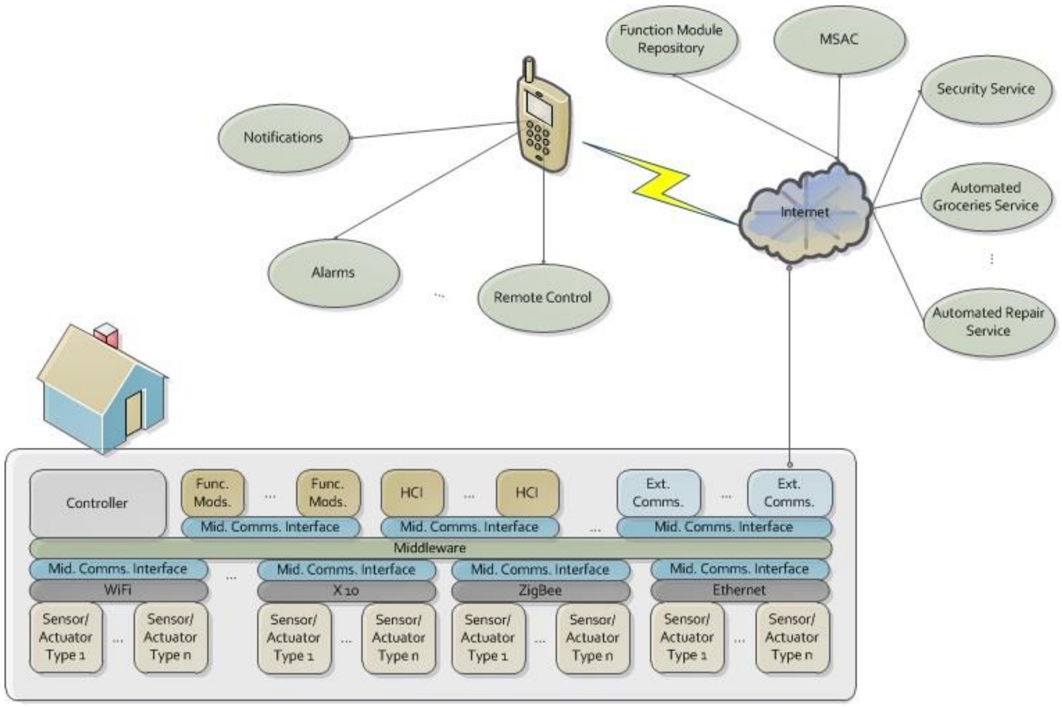


Figure 4-1. Framework Diagram.

4.3 Controller

The purpose of the Funblocks Controller module is to supervise component interaction during the System Adaptation and System Failure stages of operation. The main functions performed by the Funblocks Controller (CTRL) are the following:

1. Performs configuration of new modules.

2. Maintains information of the SAs and modules installed in the system and makes this information available to new modules.
3. Notifies modules when new SAs are installed.
4. Performs communication with external components, such as Automated Repair Services.
5. Prevents the installation of modules which could generate conflicts, such as multiple gesture recognition modules in the same area.

During "normal" operation, i.e. in the absence of system architecture changes and malfunctions, all component interaction is carried out through the Midblocks middleware. However, since ease of use is one of the objectives of the Funblocks framework, it becomes necessary to consider the operation of the system during the setup, adaptation and failure phases of the system's lifecycle. The tasks that have to be performed during system setup and adaptation are analogous and as a result system setup can be considered as the first adaptation of the system to the user's requirements.

4.3.1 System Adaptation and System Failure Operation

There are two phases of a Funblocks based system's lifecycle where interaction is not covered by MidBbocks:

- *System Adaptation.* Two types of system adaptation tasks are considered in the Funblocks framework:
    - *Addition or removal of system components.* New components can be added or removed during the lifetime of a Funblocks based system in order to adjust the system to the user's needs.
    - *Changes in component grouping.* In Funblocks SAs are grouped by means of an area identifier. During the lifecycle of a Funblocks based system users might need to modify the grouping scheme of their SAs to fine-tune the operation of their system.

  Currently, in scenarios such as Home Automation, these tasks are performed by specialized personnel. Due to the inconvenience and cost associated with such personnel HA systems may not be adequately tailored to user's needs. This leads to systems that instead of being a benefit to users sometimes operate “against” them [135].

- *System Failure.* Real world experience shows that complex systems can and do fail. Therefore it becomes necessary to consider how the system should respond in the event of a failure. This is particularly important in cases where the user’s wellbeing depends on the correct operation of the system. In these cases the system must notify appropriate parties of the failure to ensure that corrective measures are performed as soon as possible.

4.3.2 System Adaptation and System Failure Tasks

The CTRL aids users in performing system adaptation and notifies appropriate parties in the event of a system failure. The main functions performed by the CTRL during these stages of operation are the following:

- *System Adaptation.*
    - *Performs configuration of new modules.* Whenever a new module or SA is installed, the CTRL assigns the module the parameters required for it to operate. For example, if a new sensor is installed the CTRL must provide the sensor with the Area ID of the group to which this sensor will be assigned. If necessary the CTRL must obtain the parameters from users through an HCI device.
    - *Maintains information of the SAs and modules installed in the system and makes this information available to new modules.* The operation of some modules may depend on information provided by other modules of the system. In these cases modules may adapt their operation depending on what other modules are available in the system. To prevent users from having to manually tune such modules the CTRL can, upon demand, provide information of the modules currently installed in the system to a newly installed module. Since not all components are expected to be able to adjust their operation, this function is only performed if the newly installed module requests the information during its installation process.

    - *Notifies modules when new SAs are installed.* As mentioned above, the operation of a module may depend on the availability of other modules or SAs in a system. To allow installed modules to adjust their operation when new components become available, the CTLR notifies existing modules whenever a new module or SA is installed. As before, since not all modules can adapt their behaviour when new components are introduced, this notification is only sent to modules that have registered to be notified when changes in the system occur.
    - *Prevents the installation of modules which could generate conflicts, such as multiple Gesture Recognition modules in the same area.* Certain systems may contain modules which allow to customize the system for different applications but which cannot be simultaneously installed in the system. In other words, modules which could generate conflicts if installed at the same time on a given system. If the MSAD record of the component lists these failures then the CTRL will prevent installation, or at the very least operation, of one of the modules. In the event that the installation of the modules has been attempted the CTRL should notify the users through an appropriate HCI device.
- *System Failure*
    - *Notifies users in case of a failure.* Midblocks supervises the communication between the components of the system and, in case a component has failed to generate events after its MEI has expired and after attempts to obtain a response from the component have failed, generates a failure event. In case of such a failure the CTRL notifies users through one or more HCIs.
    - *Notifies external components, such as Automated Repair Services.* In case of a component failure the CTRL notifies adequate External Services. This is particularly important for systems in which the wellbeing of the users depends on the system and where external parties, such as caretakers, must immediately perform corrective actions in case of a system failure.

4.4 Sensors/ Actuators

A sensor is a device that receives a stimulus and responds with an electrical signal that is compatible with electronic circuits [136]. A smart sensor is obtained by combining a sensor, an analog interface circuit, an analog to digital converter, a microcontroller, and a communications interface [136]. Certain types of smart sensors, such as many wireless sensors, will also include a power supply in form of a battery (see Figure 4-2).

Sensors and actuators (SAs) are two of the fundamental components of an AmI system. Through sensors AmI systems gather information about physical environmental parameters such as heat, humidity, temperature, ambient light intensity among others. Sensors also allow AmI systems to collect information about the activities being performed by the inhabitants of an environment. Through actuators AmI systems can adjust the environment to the needs of inhabitants. Due to their digital and distributed nature, it is reasonable, and beneficial, to expect for Ambient Intelligence systems to employ only smart sensors.

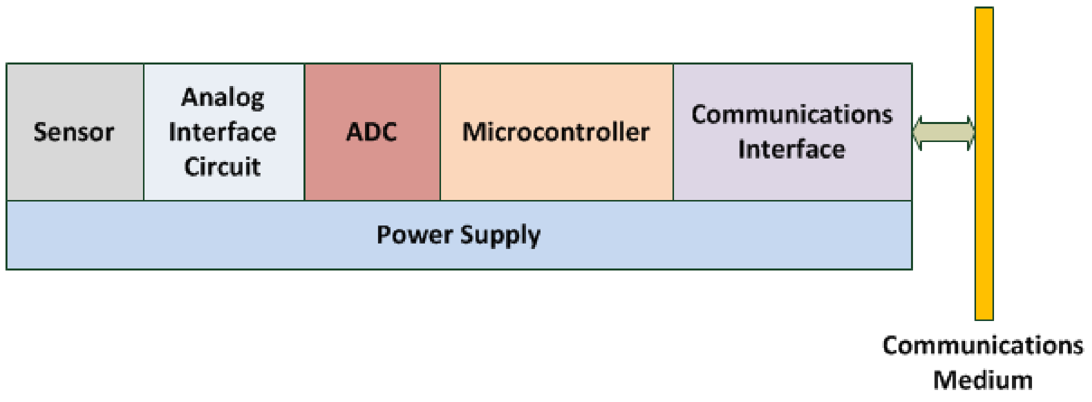


Figure 4-2. Smart Sensor Block Diagram.

There is a vast array of communications protocols available for sensors. There are currently more than fifty communication protocols for home automation and many more available for industrial environments. To enable the development of modular Ambient Intelligence systems sensors must be handled in a uniform way. Towards this end, in Funblocks we broadly classify sensors and actuators into 3 categories:

1. *Binary Sensors/Actuators (BSAs):* As the name implies binary sensors and actuators, such as smoke detectors or garage door open-close actuators, have two states.
2. *Multi-valued Sensors/Actuators (MVSAs):* Multi-valued sensors and actuators can provide or use more than two values. Examples of these types of sensors are temperature sensors and shade positioning actuators.
3. *Special Function Sensors/Actuators (SFSAs):* Sensors and actuators in this category provide a richer means of interaction with the environment, for example voice command recognition systems, video-based activity recognition systems, multimedia systems, etc.

This classification allows systems developed with Funblocks to handle a vast array of different sensors and actuators in a straightforward manner:

1. BSAs generate an event when a change of state occurs, for example, when a door is opened or when a person enters an area. In the absence of environmental changes, BSAs generate periodic "heartbeat" events for component monitoring purposes.
2. MVSAs generate current reading events periodically.
3. SFSAs can generate events when a state of change occurs, for example when a person start a certain activity, or generate events periodically, such as health monitoring devices.

For all three types of SAs, the exact time interval between events is dependent upon the type of sensor, the sensor's intended use and the system's configuration.

Handling systems such as voice recognition systems and video-based activity recognition systems as a form sensor actuator has several advantages:

- *Hardware/software specialization.* By treating these types of systems as independent components, researchers, designers and developers can focus on developing specialized hardware and software without having to take into consideration the rest of the hardware and software. This allows the use of specialized tools and languages which might not be adequate for the development of other parts of the AmI system.

- *Data processing distribution.* Incoming data from devices such as cameras or microphones is processed *in situ* by specialized hardware and software without having to transfer such data to other parts of the system. Such transfer of data would place additional load on the network.
- *Enhanced privacy and security.* Collecting data from user activities poses a threat to the privacy and security of the people inhabiting an Intelligent Environment (IE). For example, placing cameras or microphones inside a bedroom would be a serious privacy and security issue if the images and sounds obtained by the devices were accessible to other parties. By processing the images and sounds *in situ* and avoiding designs that would allow this type of data to be extracted from the components such privacy and security risks can be avoided.

In order to uniformly handle different sensor types we must establish a minimum set of parameters which can be obtained from all sensors regardless of the type of sensor used. This minimum set of parameters should be enough to allow an Ambient Intelligence system perform the following operations:

- *Presence.* An Ambient Intelligence system must be able to detect the insertion and removal of a sensor. This allows other components of the system whose operation relies on the data from the sensor to take adequate action.
- *Failure.* Sensors with self-diagnosing capabilities must be able to notify the AmI system in the event of a failure.
- *Grouping.* Sensors and actuators require some form of grouping scheme. This is required in order to assure proper evaluation of stimulus by the AmI system and that a correspondence can be established between stimulus and action.

- *Interpretation.* An AmI system must be able to adequately interpret the data coming from sensors. In particular there must be and unequivocal way to relate sensor readings with ambient parameters.

To provide AmI systems with the capability to perform the previously described functions we propose the use of a simple 3 field message format:

SENSOR ID – AREA – TYPE – DATA

Where:

- *Sensor ID.* This is a unique numerical identifier assigned to each sensor by the system when the device is installed.
- *AREA.* A numerical identifier assigned to each sensor by the system when the device is installed. This identifier need not be unique.
- *DATA.* This is the data obtained from this sensor.

With this simple message format an AmI system can perform the operations described previously:

- *Presence.* Presence is detected by message events generated by the sensors. Since each sensor is assigned a unique SENSOR ID, if a sensor has failed to generate an event after a certain time interval then the AmI system can assume that the sensor has either been removed or has failed. For sensors that are either critical or unable to report failures, absence of a sensor should be treated in the same manner. This is particularly important for critical sensors since the inability of the AmI system to receive sensor data can lead to potentially hazardous situations.
- *Failure.* Sensor failure is detected either through absence of sensor messages or by an explicit message from sensors with self-diagnosing capabilities.
- *Grouping.* Grouping is achieved by means of the AREA field. All sensors which have the same AREA identifier belong to the same group.
- *Interpretation.* Interpretation of sensor data is achieved through a TYPE identifier assigned to a class of sensors and read during system setup. For

example all temperature sensors which handle the same temperature range should have the same TYPE identifier.

Care should be exercised when implementing the TYPE identifier and DATA field for a particular system, or family of systems, developed with the Funblocks framework. If sensors made by different manufacturers share the same TYPE identifier then said sensors should be interchangeable and their DATA field should have the same format. However, if a high degree of customization is allowed for a particular type of sensor then the use of very specific Function Modules will be required. This results in a loss of flexibility in the system. The use of this simple message format allows devices with very limited resources, such as 8-bit microcontrollers, to be employed in Funblocks.

4.5 Human Computer Interfaces

There are two types of interaction users will perform with an Ambient Intelligence system: implicit or natural interaction, and explicit interaction. Implicit interaction is carried out by the users through their Daily Life Activities (DLAs) and, as described previously, is detected by a Funblocks based system through SFSAs.

On the other hand, to perform activities such as system configuration or system maintenance the users must interact explicitly with the system. HCI components allow users to perform such explicit interactions with the system. HCIs include touch panel interfaces and computing devices such as laptops or smartphones.

4.6 Function Modules

The intelligence, and in fact the entire useful features, of a Funblocks based system reside in the Function Modules (FMs). FMs receive events from SAs or from other FMs; reception of an event triggers the execution of the function block. The data required by the FM for its execution is contained in the event. The execution of a function block can result in events sent to actuators to perform adjustments in the environment, or in events sent to other function blocks to trigger their actions. This allows the creation of function block networks as described in the IEC61499 standard.

4.7 Module and Sensor/Actuator Catalog

In the early days of computers much of the software available to a computer came in the form of monolithic programs which included all of the functions necessary for the program to operate. As programs and computers became increasingly larger and more complex the development of hardware and software as single monolithic units was no longer feasible and the concept of modular development was introduced.

As more and more modules became available for the integration of systems, it became an increasingly difficult task to keep track of both the modules installed, and available for installation, in a given system. To aid users and developers in the task of customizing and managing their systems, software and device managers were introduced. These software and device management programs keep track of the devices and software installed in a given system. They also ensure that any software required by a new device or program is installed previous to the new components installation.

Furthermore, some manufacturers, such as Ubuntu Linux, keep a repository of available modules on the Internet. As part of the installation process of a new program any software required by the new program to operate is installed previous to the installation of the new program. In other words, any dependencies are automatically satisfied as part of the process of installing a new package. The advantage of this process is that system management is tremendously simplified. The user simply needs to select a new package of installation, wait for the installation process to complete, and the new software becomes immediately available.

This concept is brought into the Funblocks framework through the Module and Sensor/ Actuator Catalog (MSAC) and the Function Module Repository (FMR). The Module and Sensor/ Actuator Catalog stores MSAD records which describe the components available to a specific Funblocks based system. The MSAD lists the functions provided by a specific module or SA, the dependencies required by the component and the conflicts this component may have with other components. The MSAD also provides a default configuration for the components of an AmI system. This assists end-users and system integrators in the configuration of an AmI system.

4.8 Function Module Repository

The Function Module Repository contains modules which can be directly installed from the Internet onto a system to introduce new functionality and to satisfy dependencies. The FMR also stores Function Modules targeted at different hardware. This allows manufacturers to target multiple platforms, such as Intel or ARM architectures, while at the same time it allows end users to choose the architecture best suited for their expected needs and budget.

4.9 External Communications Modules

External Communications Modules (ECMs) provide a link to external communications networks such as the Internet or telephone services. The use of ECMs increases the security of the system by avoiding direct interaction of an AmI system's components with external entities. Currently ECMs are a planned feature of the Modular Framework for Ambient System development which will be detailed at a later stage in the development of the project.

5 Component Supervising Middleware

A solution to improve the reliability of a system is to perform supervising functions through the communication mechanism used for component interaction. Examples of this type of solution are the pneumatic control systems used in early building automation and the 4-20mA current-loop popular in industrial measurement and control systems.

In early building automation systems compressed air was used to transmit information between the components of the system and simultaneously supervise the system. To achieve this, components were linked with hoses that carried compressed air which varied in the range of 3-15psi. 3psi represents a live zero while 15psi represents 100%. Any pressure below 3psi is a dead zero and constitutes an alarm condition [137].

The same principle is used in the 4-20mA current-loop. In this scheme a current flowing through a wire to the sensor or actuator is varied between 4 and 20mA, with 4mA representing a live zero and 20mA representing 100%. If the current drops below 4mA this signals either a device failure or an open link while a current above 20mA signals a device failure or a short circuit [137], [138].

To supervise components in distributed systems, a frequent solution is to use heartbeat messages. In this type of scheme a special kind of message, called a heartbeat message, is periodically sent by the component to other components in the system. The main disadvantage of this type of scheme is that depending on the number of system components, the frequency with which heartbeat messages are sent and the bandwidth of the communications links, the load on the communications infrastructure can be significant.

The event based store and forward middleware is particularly well suited to provide a supervising mechanism without the need of a separate heartbeat message. Events generated by the system components can be considered as a type of heartbeat message. This coupled with the need of an event management entity for the store and forward mechanism, which can simultaneously be used to supervise the interval between events, provides the basis for a complete component supervision solution. This is the principle we use in Midblocks, an event based, store and forward, component supervising middleware.

5.1 Middleware Description

Midblocks is an event based, store and forward type of middleware in which the store and forward entity additionally performs component supervision tasks [139], [140], [141]. Every component of a Midblocks based system specifies a maximum time interval between the events that it generates. If a component exceeds this maximum time interval, called the Maximum Event Interval (MEI), Midblocks will generate a failure event notifying the component's failure to those components that have subscribed to failure events.

A useful feature of Midblocks is that it does not assume any particular type of communication mechanism or underlying hardware. To achieve this Midblocks makes use of a layered design in which events are formed independent of any communications mechanism or hardware assumptions and as the events descend through the layers information required by the specific communication mechanism and hardware chosen is added. As a result Midblocks can be used to provide component supervision in a large array of different systems.

In order to use Midblocks in a particular system only two elemental requirements have to be satisfied:

- Each component of the system must be uniquely identifiable.
- A communication link must be present, capable of sending and receiving discrete data packets between the components and the store and forward/component supervising module.

5.2 Middleware Events

All components in Midblocks must be assigned a unique identifier. This way event messages can be related to the component from which the message originated. Component identifiers in Midblocks are 48 bits long analogous to the MAC address used in IEEE 802 networks. These identifiers can either be assigned statically during system installation or dynamically, the only requirement is that the identifiers are system wide unique.

All Midblocks events have the same basic structure which is the 48 bit component id, followed by an 8 bit event type id and, depending on the type of event, additional event data (figure 5.2.1).

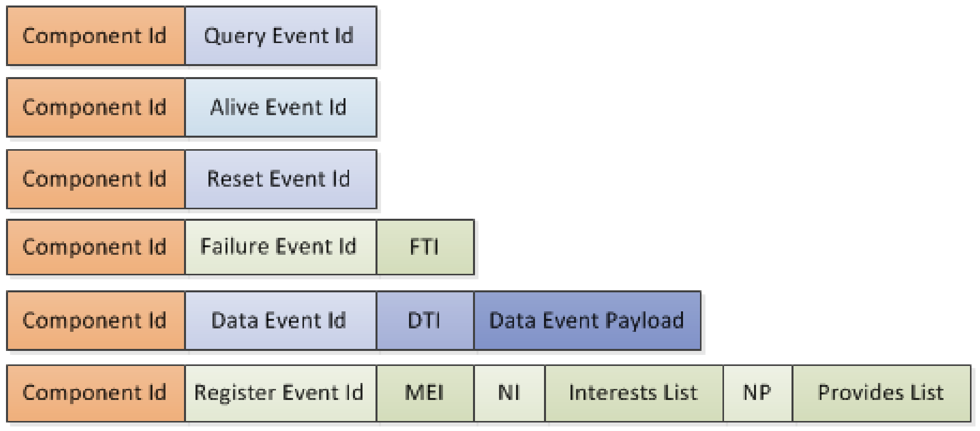


Figure 5-1. Midblocks Event Structure.

Midblocks uses the following six types of events:

- *Query events.* When a component has exceeded its maximum MEI the Component Supervisor (CS) module sends a query event to the component to determine if it is responsive. If the component does not respond within an MEI then a critical failure event is sent to those components subscribed to failure events.
- *Alive events.* There are two types of Alive events with different event type id which can be issued by components. The first one is issued in response to a query event. While the second one is issued when the component has not generated any events and its MEI is close to expiring.
- *Reset events.* When an event is received by a component that is not registered with the Message Receiver then a Reset event is issued immediately back to the component. Upon reception of a Reset event the component should clear any pending outbound events and perform its registration procedure.

- *Failure events.* Failure events are generated when messages are not received or cannot be delivered to components. Failure events have an 8 bit Failure Type Id (FTI) which indicates the type of failure detected.
- *Data events.* Data events are the normal events exchanged between components and the store and forward entity in Midblocks. After the event type id, Data events have a 64 bit Data Type Id field (DTI). The possible values of the DTI are application specific and are assigned by the system developers using Midblocks. After the DTI comes the actual event data. Midblocks does not assume any particular structure or size for this data.
- Register events. Register events are used to register components with the store and forward/component supervision entity. Midblocks handles two values for the Register Event Id field (REI). The first value is used for components that have interest in receiving failure events, while the second value is used for components that do not need to be notified of failure events. Following the event type id Register events have the following fields:
  - MEI. This is a 32 bit field containing the Maximum Event Interval for this particular component expressed in milliseconds. This value is sent using network byte order.
  - NI. The Number of Interests field (NI) is a 64 bit field indicating the number of data event types that this component has interest in. This value is sent using network byte order.
  - Interest List field. This field lists the DTIs of all the data types that this component is interested in receiving.
  - NP. The Number of Provides field (NP) is a 64 bit field indicating the number of data event types that this component provides. This value is sent using network byte order.
  - Provides List field. This field lists the DTIs of all the data types that this component provides.

5.3 Middleware Architecture

Midblocks can be broadly separated into two distinct parts (see Figure 5-2): A component side part (CMPS) and a Store and Forward/Component Supervising entity (SFCS).

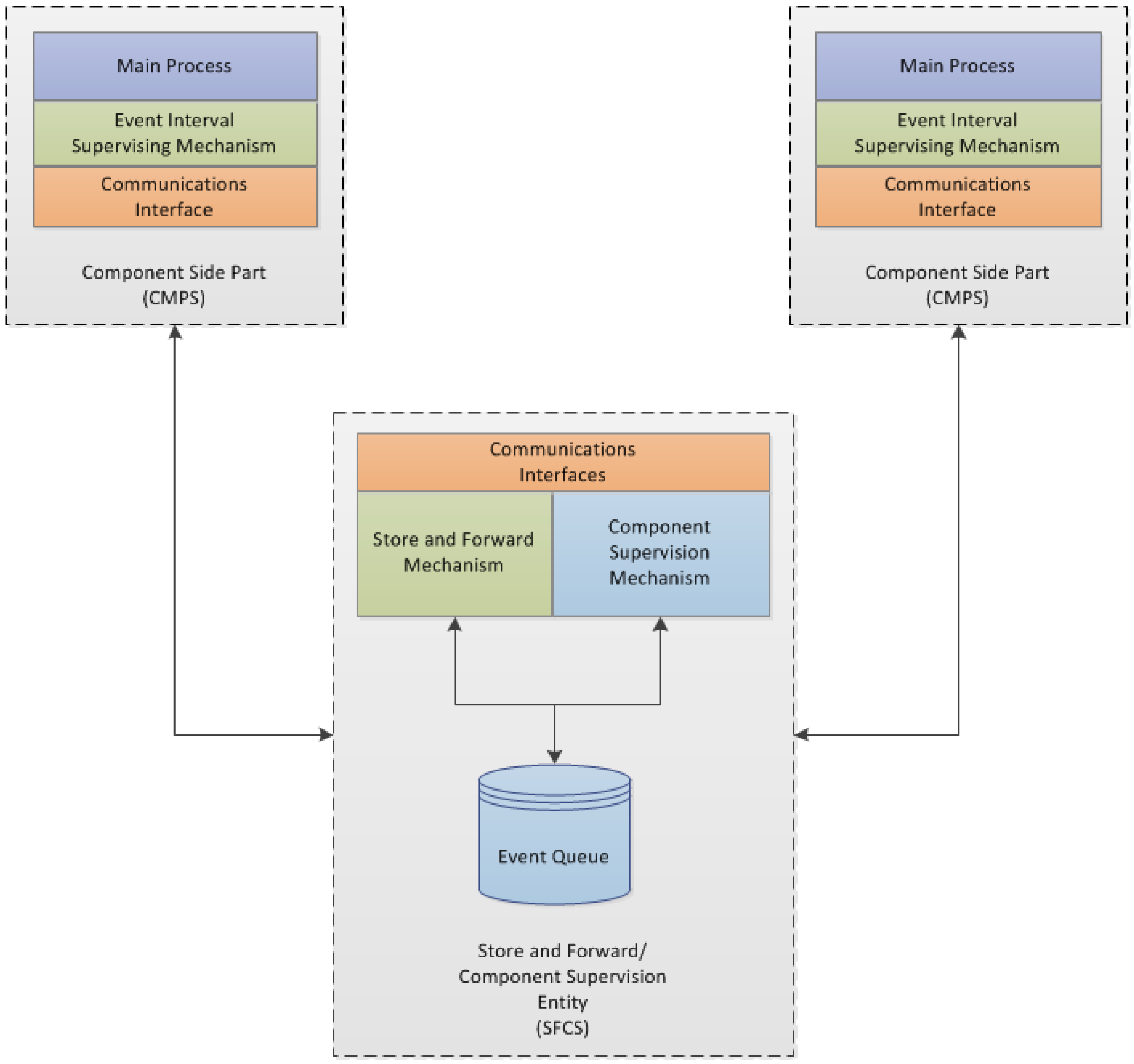

Main Process
Event Interval Supervising Mechanism
Communications Interface
Component Side Part (CMPS)
Main Process
Event Interval Supervising Mechanism
Communications Interface
Component Side Part (CMPS)
Communications Interfaces
Store and Forward Mechanism
Component Supervision Mechanism
Event Queue
Store and Forward/ Component Supervision Entity (SFCS)

Figure 5-2. Middleware Architecture

As with any other type of store and forward event-based middleware the primary task performed by Midblocks is to deliver events from producers to consumers. However, unlike other types of middleware Midblocks supervises the components to ensure that at least one event is sent during a components MEI and that events are being accepted by the components. To achieve this Midblocks requires event message supervising functions which constantly monitor the events being accepted from components by the SFCP and the events delivered to components from the SFCP. In order to perform adequate supervision Midblocks does not support direct component to component interaction.

In the following sections we will begin by describing the operation of the CMPS part of Midblocks and afterwards describe the SFCS entity.

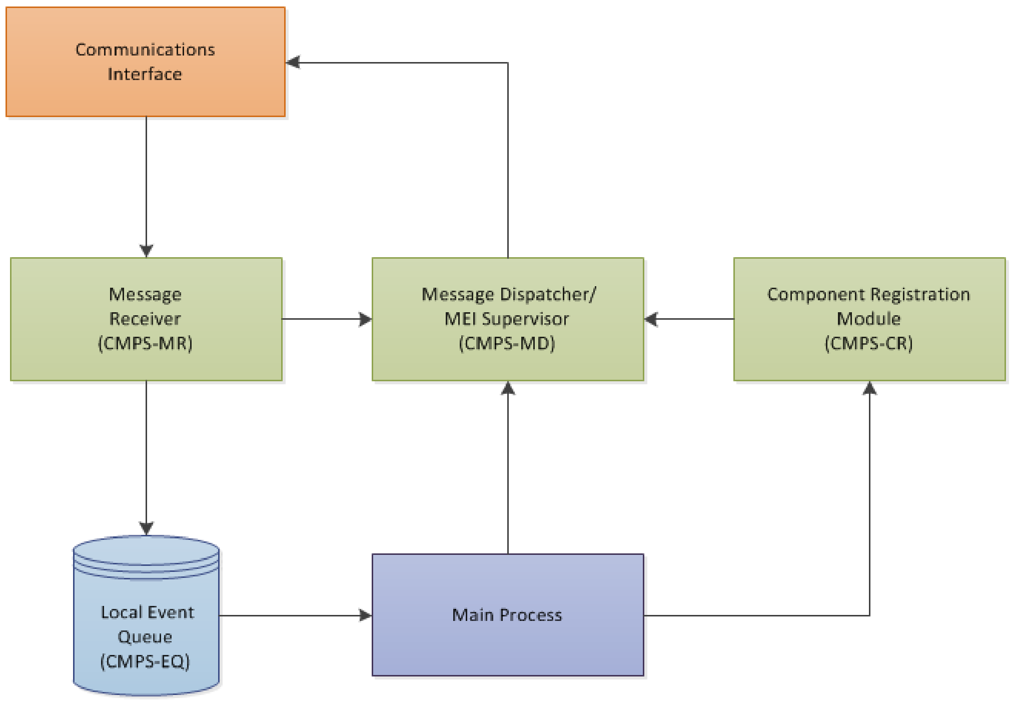

Communications Interface
Message Receiver (CMPS-MR)
Message Dispatcher/ MEI Supervisor (CMPS-MD)
Component Registration Module (CMPS-CR)
Local Event Queue (CMPS-EQ)
Main Process

Figure 5-3. Midblocks Component Side (CMPS) Block Diagram.

5.3.1 Component Side Mechanism

When a component is first introduced into the system it must perform a registration procedure. The purpose of the registration procedure is to allow the SFCS to configure the queuing and component supervision mechanisms adequately to handle the events of this component and to supervise the component. Registration of the component with the SFCS is performed by the CMPS Component Registration Module (CMPS-CR) upon request of the component's main process (Figure 5-3). The registration sequence has to be initiated by the main process in order to ensure that the communications hardware and software have been properly initialized and are able to send and receive messages. Components must use one of two Register Event Ids to indicate whether or not they are interested in receiving Failure events.

Inbound event messages arrive at the CMPS from the SFCS through a communications interface and are received by the CMPS Message Receiver (CMPS-MR). Four types of events can be received by a component: Query, Reset, Data, and Failure events. Failure events are received only by components that have expressed interest in receiving this type of events by using the corresponding REI during their registration procedure. Depending upon the type of event received by the CMPS-MR the following tasks are performed:

- *Query event.* When a Query event is received, the CMPS-MR immediately notifies the CMPS Message Dispatcher/MEI Supervisor (CMPS-MD) to send an appropriate Alive event. The Query event is then discarded and no further processing of the event occurs.
- *Reset event.* Upon reception of a Reset event the following sequence of actions occurs:
    - CMPS-MR signals the CMPS-MD to block any further event message sending, with the exception of Register events.
    - The CMPS-MR then introduces the event into the CMPS Local Event Queue (CMPS-EQ) and will discard any further Reset events until it is signaled by the CMPS-MD that a Register event has been sent.
    - The CMPS-MD will not accept any events from the component's Main Process until it receives a Register event from the CMPS-CR.
    - When the component's Main Process extracts the event from the queue it must signal the CMPS-CR to perform the registration procedure to register the component with the SFCS.
- Data and Failure events. Data and Failure events are placed directly in the CMPS-EQ for processing by the component's Main Process.

Outbound data events are sent through the CMPS-MD and the communications interface. The CMPS-MD monitors the time interval since the last event was sent and, in case it approaches the component's MEI, it issues an Alive event to signal that the component is operating correctly.

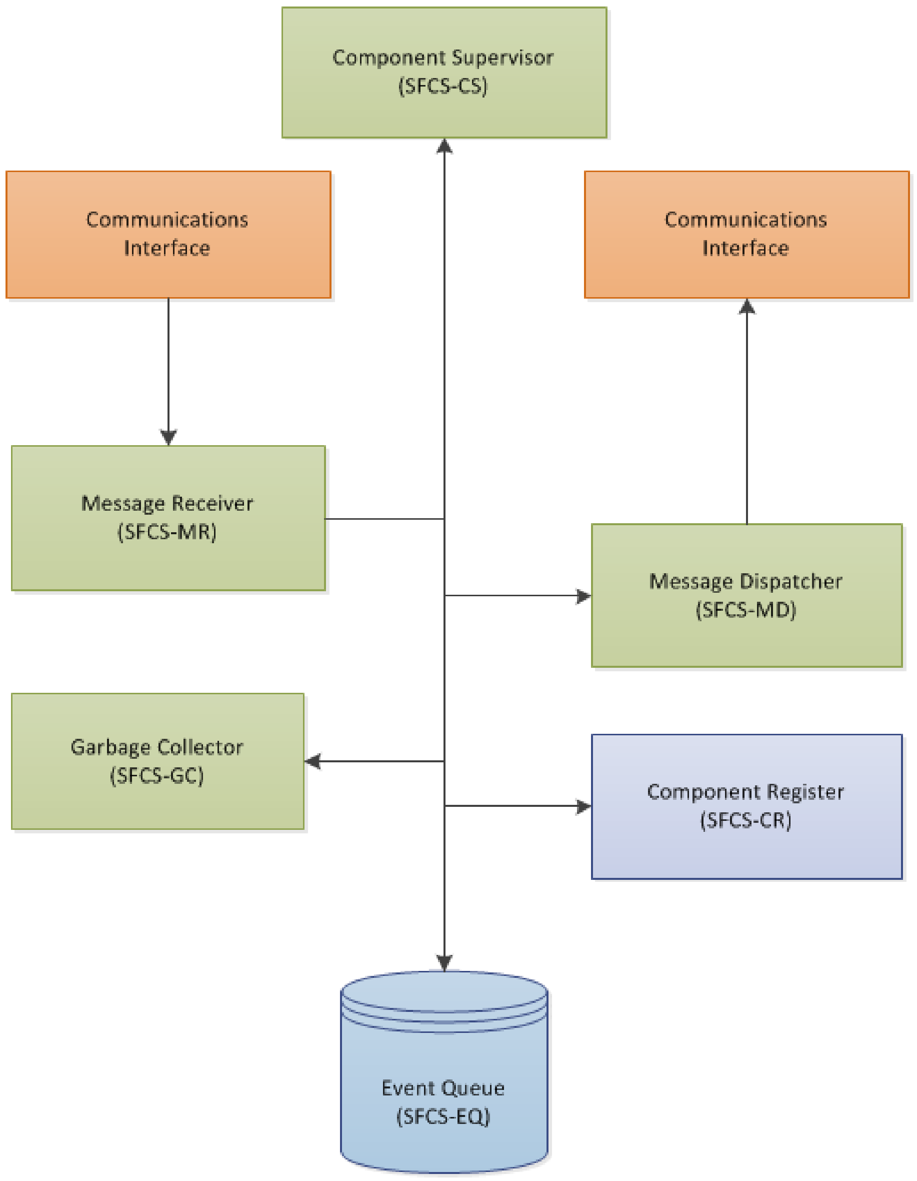

Component Supervisor (SFCS-CS)
Communications Interface
Communications Interface
Message Receiver (SFCS-MR)
Message Dispatcher (SFCS-MD)
Garbage Collector (SFCS-GC)
Component Register (SFCS-CR)
Event Queue (SFCS-EQ)

Figure 5-4. Middleware SFCS Block Diagram.

5.3.2 Store and Forward with Component Supervision

The Store and Forward with Component Supervision mechanism of the middleware is comprised of the event queue and the logic necessary to supervise the transmission and recollection of messages from the components.

5.3.2.1 Midblocks SFCS Event Reception

On the SFCS side (see Figure 5-4) event messages are received by the SFCS Message Receiver (SFCS-MR) from the communications interfaces. Three types of messages can be received by the SFCS: Alive, Register and Data events. The first task performed by the SFCS-MR is to verify the event type and the Component Id. If the event is not a Register event and the component's data is not registered in the SFCS Component Register (SFCS-CR) then a Reset event for the component is issued through the same communications interface from which the message arrived. Since Reset events are not entered into the SFCS Event Queue (SFCS-EQ) and are not sent by the SFCS Message Dispatcher (SFCS-MD) the data in the Component Id field of the Reset event is not important.

After verifying that the event is valid one of the following actions is performed depending on the event type:

- *Alive and Register events.* Alive and Register events are directly placed in the message queue without any further processing from the SFCS-MR.
- *Data event.* When the event received is a Data event the SFCS-MR scans the SFCS-CR to determine which components are subscribed to the type of data event received. Next the SFCS-MR attaches a Time-To-Live (TTL) field to the event, and inserts one copy of the event into the SFCS-EQ for each recipient. The inserted events are tagged as New events prior to insertion as explained in the following section.

5.3.2.2 Midblocks SFCS Event Processing

Events in the SFCS-EQ can be tagged with four states:

- New event.
- Ready for Delivery.
- Unable to Deliver.

- Processed.

Any event in the SFCS-EQ can only be in one of these states at any given time.

The SFCS Component Supervisor (SFCS-CS) constantly scans the SFCS-EQ for events that require processing. Depending on the event type and the state of the event one of the following actions is performed:

- *New event tag - Alive event.* When the SFCS-CS encounters an Alive event tagged as a New event, it resets the timer associated with the producer component and tags the event as Processed. If the Alive event was issued in response to a Query event then the SFCS-CS inserts a Failure-Non critical event into the SFCS-EQ for each Failure event subscriber. Failure events have the same structure as Data events in the SFCS-EQ meaning that the event has a TTL field and a state tag. When the Failure event is created it is tagged as *Ready for Delivery*.
- *New event tag - Data event.* Data events are handled in much the same way as Alive events. The SFCS-CS resets the timer associated with the producer component and tags the event as *Ready for Delivery*.
- *New event tag - Register event.* On encountering a Register event tagged as a *New* event, the SFCS-CS creates and initializes a new timer for the component and inserts the component's data into the SFCS-CR. The Register event is then tagged as *Processed*.
- *Ready for Delivery*. Events tagged as Ready for Delivery are not handled by the SFCS-CS but are handled by the SFCS-MD. How the SFCS-MD handles Ready for Delivery events is described later on in this section.
- *Unable to Deliver*. This tag indicates that the SFCS-MD was unable to deliver the event after TTL attempts. When encountering a message tagged as *Unable to Deliver* the SFCS-CS performs the following actions:
    - The destination component's data is removed from the SFCS-CR.
    - Any events in the SFCS-EQ destined for the component, including the current event, are tagged as Processed.
    - Failure events of type Critical are inserted into the SFCS-EQ indicating that the component has failed.

- *Processed.* Processed are handled by the SFCS Garbage Collector (SFCS-GC). The SFCS-GC scans the SFCS-EQ for events tagged as Processed and removes the events from the SFCS-EQ.

The SFCS-CS also keeps track of the time elapsed since component's last event was received. If a component exceeds its MEI the SFCS-CS inserts a Query event destined for the component into the SFCS-EQ. Additionally the SFCS-CS inserts a non-critical failure event into SFCS-EQ and resets the timer associated with the component.

5.3.2.3 Midblocks SFCS Event Delivery

Data events marked as Ready for Delivery are read from the SFCS-EQ by the SFCS-MD on a FIFO basis and delivered to consumers via the appropriate communications interface. If the event is delivered successfully then it is tagged as Processed, otherwise the event's TTL field is decremented by one. If the event's TTL field reaches zero then the event is tagged as Unable to Deliver and no further attempts will be made to deliver the event.

5.4 Middleware Communications Interfaces

To allow use with a wide variety of devices and communications technologies, Midblocks employs a 3 layer design for its communication interfaces as shown in figure Figure 5-5.

In the first layer event messages are formed without using communication specific data such as IP addresses or node ids. Once the core event messages are formed, communication specific parameters such as IP addresses are added to the message. Finally, in the last layer, platform specific routines are invoked to deliver the message.

A detailed description of the message composition process can be found in section 6.4.1.

Currently, the CMPS can only use a single communications interface in each component, however the SFCS can use multiple communications interfaces both for inbound as well as for outbound messages. This allows the SFCS to function as a gateway which permits components with different types of communication mechanisms to exchange events.

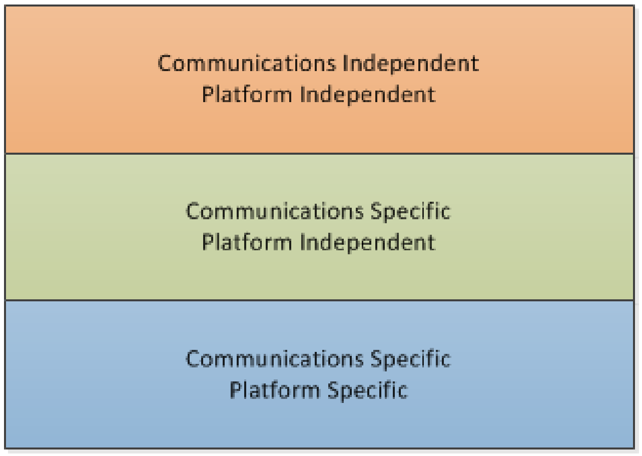


Figure 5-5. Middleware Communications Interface Layers.

6 Implementation of the Modular Framework for Ambient Intelligence Systems

For testing purposes we developed a simple temperature measurement demo application. The demo tested the main features of the framework including registration, removal and failure of a component. A video of the demo can be found at:

http://www.youtube.com/watch?v=y5LxC44FWwA

Two basic criteria were used during the selection process of the development tools that were employed to develop the test implementation:

1. Keep the number of development languages at a minimum. This criterion allowed us to focus our efforts in developing the API and the test implementation instead of using development time for dealing with differences in programming languages.
2. Use easily available development hardware. This criterion insured we had all the tools required to develop the test implementation.

Based on these criteria the following tools were selected for a first implementation of Funblocks and Midblocks:

1. *Laptop PCs.* Laptop PCs were chosen because it is a very easily available and highly customizable hardware. Laptops can be customized through the use of different operating systems, add-on peripherals, and a wide array of development tools is available for these devices.
2. *Netduino Embedded Systems.* Netduino embedded development boards are easily available on the local market and represent an easily extendable development platform.
3. *Visual C# Programming Language.* This language was selected due to it being available both for PCs and Netduino embedded systems. Although the PC and embedded versions are not identical, they share the same syntax and most of the libraries required for the test implementation.

6.1 Hardware

Our Modular Framework for Ambient Intelligence Systems has been implemented on the following hardware:

Controller, SFCS and Component:

- HP Mini 110-3121 LA with Windows 7 32-bit OS and WiFi communications link.
- Asus K52F with Windows 7 64-bit OS and WiFi communications link.
- Acer 7520 with Windows 7 32-bit OS and WiFi communications link.

Component:

- Netduino Plus and Ethernet communications link

6.2 Development Platforms

Funblocks has been implemented in the following programming languages:

- Net Framework/Visual C#
- Net Micro Framework/Visual C#

6.3 Communication Schemes

The test implementation employed IP (Wi-Fi and Ethernet) links.

6.4 Application Programming Interface

An Application Programming Interface (API) is the set of symbols that are exported and available to the users of a library to write their applications. The main characteristics that an API should possess are [142]:

1. *Easy to learn and memorize.* An easy-to-learn API features consistent naming conventions and patterns, economy of concepts, and predictability. It uses the same name for the same concept and different names for different concepts. An API must be minimal and consistent. A consistent API is easy to memorize because the user can reapply what they learned in one part of the API when using another part. A minimal API is easy to memorize because there is little to

remember. An API is not only the names of the classes and methods that compose it, but also their intended semantics.

2. *Leads to readable code.* Readable code is less likely to contain errors because errors are made more visible. Readable code is also easier to document and maintain.
3. *Hard to misuse.* A well-designed API makes it easier to write correct code than incorrect code and encourages good programming practices. It does not needlessly force the user to call methods in a strict order or to be aware of implicit side effects or semantic oddities.
4. *Easy to extend.* APIs should be designed with future growth in mind.
5. *Complete.* Ideally, an API should be complete and let users do everything they want. Completeness is also something that can appear over time, by incrementally adding functionality to existing APIs. It is important to have a clear idea of the direction of for future development so that each step is a step in the right direction.

To achieve these characteristics, the Funblocks and Midblocks APIs are being developed using an iterative approach.

The development of a project such as Funblocks and Midblocks is an effort that vastly exceeds the scope of a single PhD thesis. Other AmI projects have been developed by teams of individuals from many different companies and research institutions. As a result, although a first stage design of the framework has been carried out and shown in this thesis, the details of many of the components are still an ongoing research effort which will be carried out in future stages of the project. Because of this the Funblocks and Midblocks APIs are in a constant state of evolution.

As mentioned previously, the components of Funblocks that are currently being developed are the following:

1. Midblocks
2. Controllers.
3. Components.

The following sections describe the current development of the API.

6.4.1 Midblocks

The Midblocks middleware is the part of the Modular Framework for Ambient Intelligence Systems that has been most developed. The reason for starting the development of the project with the middleware is that the other parts of the framework rely on the middleware for their interaction. Therefore, the middleware is required to perform any type of tests on the system.

The Midblocks API is divided into the following parts:

1. Event Handling.
2. Base Communications.
3. Messages.
4. Network (IEEE 802.3) Communications.
5. Event Processing.
6. Component.
7. Controller.
8. SFCS.

These API parts are described in the following sections. It is important to note that this is still an ongoing project and therefore subject to rapid changes in the future.

6.4.1.1 Midblocks Event Handling

All Midblocks events have the same basic structure which is the 48-bit component id, followed by an 8-bit event type id and, depending on the type of event, additional event data (see 5.2. Middleware Events). The advantage of this event architecture scheme is the flexibility it provides by allowing new types of events to be introduced and handled in a uniform manner.

Midblocks events stem from the abstract base class *MB_Event* (see Figure 6-1, Figure 6-2 and Figure 6-3), which provides the basic structure of an event. Midblocks events derive from this base class and must override the methods *SerializeEvent* and *DeserializeEvent* which are methods specific to the characteristics of each event type.

Derived classes must also modify the *_eventTypeId* member variable which allows distinguishing between different types of events.

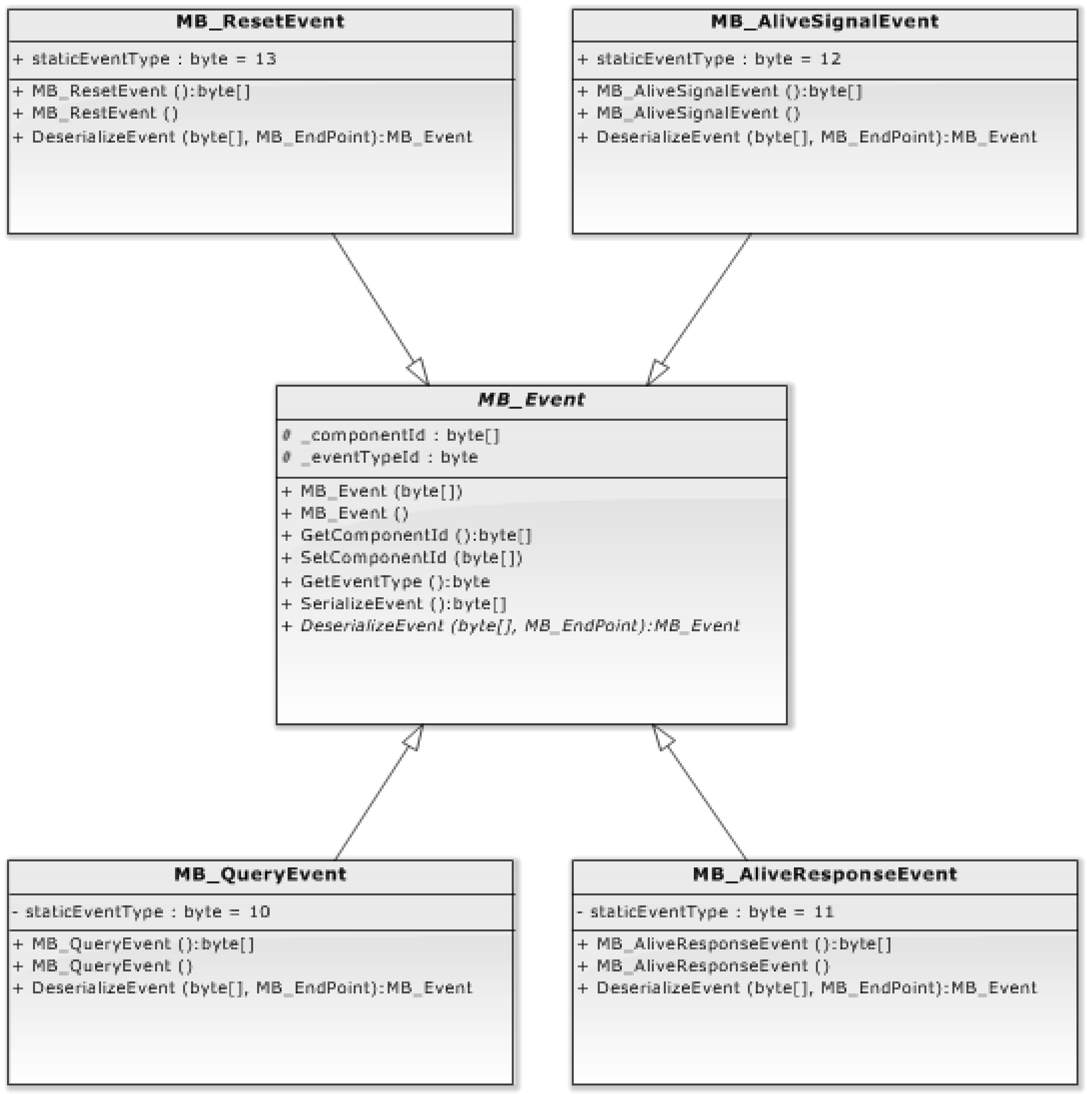

MB_ResetEvent
+ staticEventType : byte = 13
+ MB_ResetEvent ():byte[]
+ MB_RestEvent ()
+ DeserializeEvent (byte[], MB_EndPoint):MB_Event
MB_AliveSignalEvent
+ staticEventType : byte = 12
+ MB_AliveSignalEvent ():byte[]
+ MB_AliveSignalEvent ()
+ DeserializeEvent (byte[], MB_EndPoint):MB_Event
MB_Event
# _componentId : byte[]
# _eventTypeId : byte
+ MB_Event (byte[])
+ MB_Event ()
+ GetComponentId ():byte[]
+ SetComponentId (byte[])
+ GetEventType ():byte
+ SerializeEvent ():byte[]
+ DeserializeEvent (byte[], MB_EndPoint):MB_Event
MB_QueryEvent
- staticEventType : byte = 10
+ MB_QueryEvent ():byte[]
+ MB_QueryEvent ()
+ DeserializeEvent (byte[], MB_EndPoint):MB_Event
MB_AliveResponseEvent
- staticEventType : byte = 11
+ MB_AliveResponseEvent ():byte[]
+ MB_AliveResponseEvent ()
+ DeserializeEvent (byte[], MB_EndPoint):MB_Event

Figure 6-1. Midblocks events. Part 1.

The currently assigned values for the *_eventTypeId* are the following:

1. *MB_QueryEvent:* Type id 10.
2. *MB_AliveResponseEvent:* Type id 11.
3. *MB_AliveSignalEvent:* Type id 12.
4. *MB_ResetEvent:* Type id 13.
5. *MB_ControllerDataEvent:* Type id 14.
6. *MB_ComponentDataEvent:* Type id 15.
7. *MB_ComponentRegisterEvent:* Type id 16.
8. *MB_ControllerRegisterEvent:* Type id 17.

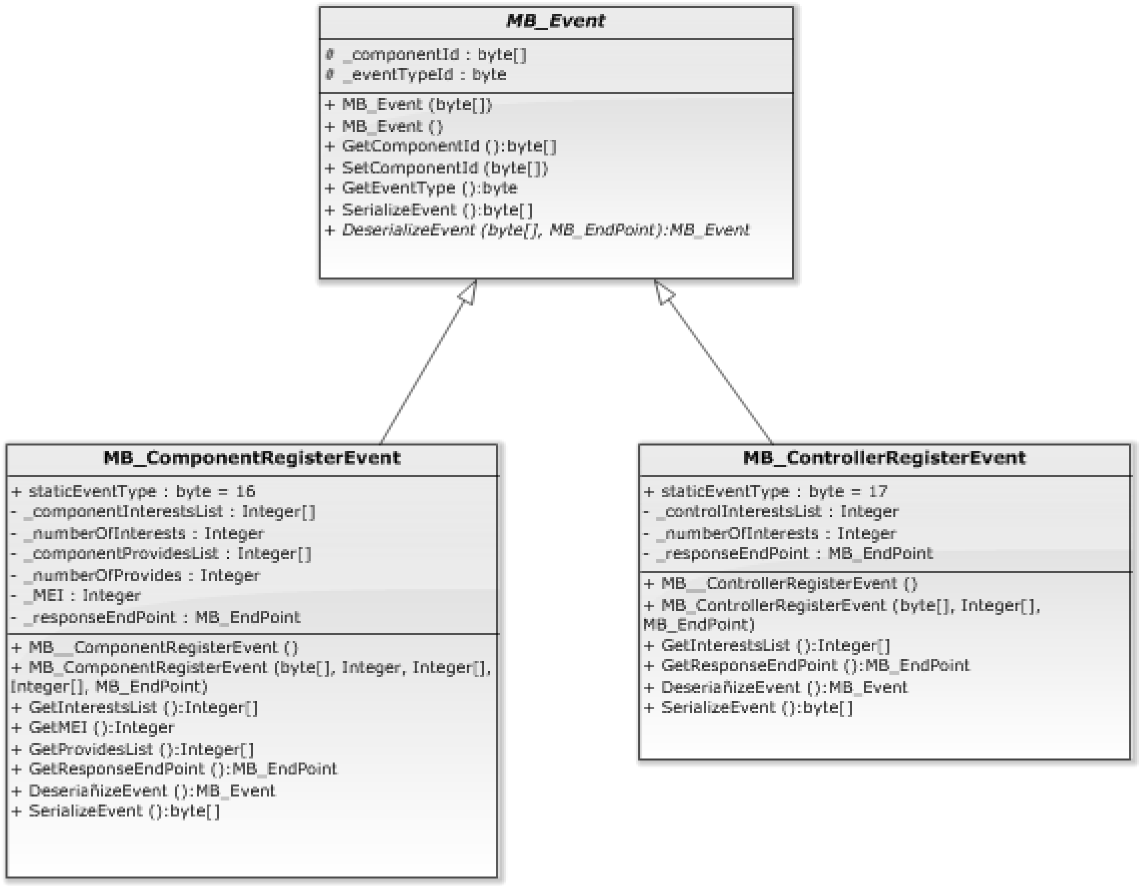

MB_Event
# _componentId : byte[]
# _eventTypeId : byte
+ MB_Event (byte[])
+ MB_Event ()
+ GetComponentId ():byte[]
+ SetComponentId (byte[])
+ GetEventType ():byte
+ SerializeEvent ():byte[]
+ DeserializeEvent (byte[], MB_EndPoint):MB_Event
MB_ComponentRegisterEvent
+ staticEventType : byte = 16
- _componentInterestsList : Integer[]
- _numberOfInterests : Integer
- _componentProvidesList : Integer[]
- _numberOfProvides : Integer
- _MEI : Integer
- _responseEndPoint : MB_EndPoint
+ MB__ComponentRegisterEvent ()
+ MB_ComponentRegisterEvent (byte[], Integer, Integer[], Integer[], MB_EndPoint)
+ GetInterestsList ():Integer[]
+ GetMEI ():Integer
+ GetProvidesList ():Integer[]
+ GetResponseEndPoint ():MB_EndPoint
+ DeseriañizeEvent ():MB_Event
+ SerializeEvent ():byte[]
MB_ControllerRegisterEvent
+ staticEventType : byte = 17
- _controlInterestsList : Integer
- _numberOfInterests : Integer
- _responseEndPoint : MB_EndPoint
+ MB__ControllerRegisterEvent ()
+ MB_ControllerRegisterEvent (byte[], Integer[], MB_EndPoint)
+ GetInterestsList ():Integer[]
+ GetResponseEndPoint ():MB_EndPoint
+ DeseriañizeEvent ():MB_Event
+ SerializeEvent ():byte[]

Figure 6-2. Midblocks events. Part 2.

As mentioned in section 5.2. Middleware Events, two types of alive events are required. The first type is an alive event which is issued to avoid a component from exceeding its Maximum Event Interval (MEI), which is the maximum time period between events. This type of event is represented by the *MB_AliveSignalEvent* class. The second type of event, represented by the *MB_AliveResponseEvent,* is issued in response to a query event generated by the Midblocks SFCS.

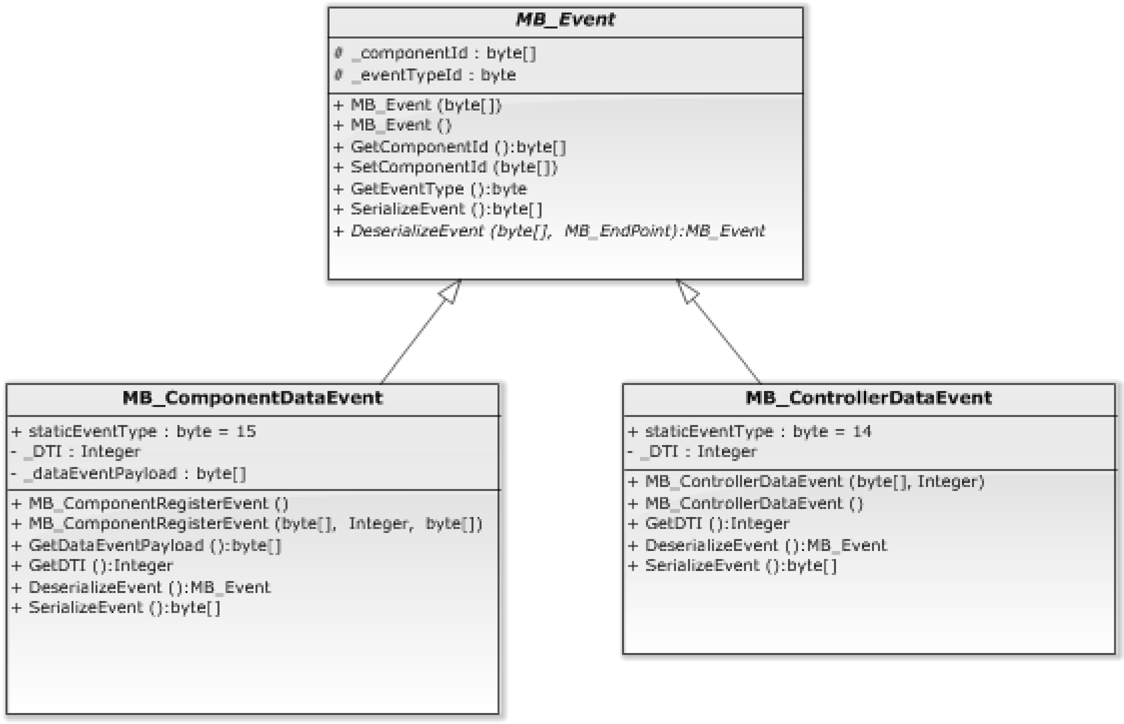

MB_Event
# _componentId : byte[]
# _eventTypeId : byte
+ MB_Event (byte[])
+ MB_Event ()
+ GetComponentId ():byte[]
+ SetComponentId (byte[])
+ GetEventType ():byte
+ SerializeEvent ():byte[]
+ DeserializeEvent (byte[], MB_EndPoint):MB_Event
MB_ComponentDataEvent
+ staticEventType : byte = 15
- _DTI : Integer
- _dataEventPayload : byte[]
+ MB_ComponentRegisterEvent ()
+ MB_ComponentRegisterEvent (byte[], Integer, byte[])
+ GetDataEventPayload ():byte[]
+ GetDTI ():Integer
+ DeserializeEvent ():MB_Event
+ SerializeEvent ():byte[]
MB_ControllerDataEvent
+ staticEventType : byte = 14
- _DTI : Integer
+ MB_ControllerDataEvent (byte[], Integer)
+ MB_ControllerDataEvent ()
+ GetDTI ():Integer
+ DeserializeEvent ():MB_Event
+ SerializeEvent ():byte[]

Figure 6-3. Midblocks events. Part 3.

6.4.1.2 Midblocks Communications

The Midblocks communications layer is based on three main concepts:

1. *Communications Client.* Communications clients are tasked with sending data through a specific type of communications interface, e.g. IEEE 802.3 type interface, RS-485, Bluetooth, etc. Communications clients are a well-known

concept in networking software. Midblocks clients are derived from the *MB_Client* class (see Figure 6-4).

2. *Communications Server.* Communications servers are the counterpart for communications clients, i.e. servers are tasked with sending data through a specific type of communications interface. Midblocks servers are derived from the *MB_Server* class (Figure 6-5).

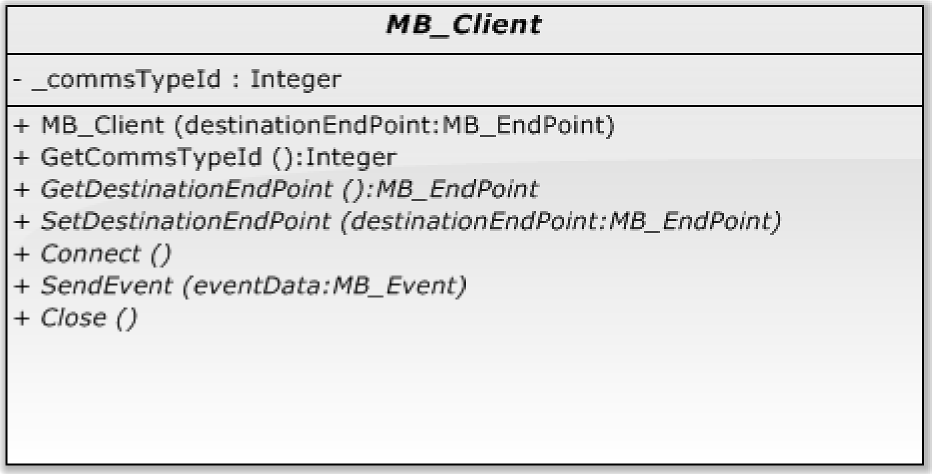

MB_Client
- _commsTypeId : Integer
+ MB_Client (destinationEndPoint:MB_EndPoint)
+ GetCommsTypeId ():Integer
+ GetDestinationEndPoint ():MB_EndPoint
+ SetDestinationEndPoint (destinationEndPoint:MB_EndPoint)
+ Connect ()
+ SendEvent (eventData:MB_Event)
+ Close ()

Figure 6-4. Midblocks Client Base Class

3. *End Point.* A Midblocks end point represents the type of communication interface being used, e.g. IEEE 802.3, RS-485, etc., plus all necessary information for data reaching its destination. For example, the end point for IEEE 802.3 type interfaces contains the interface type id, the ip address and the port number (see Figure 6-6).

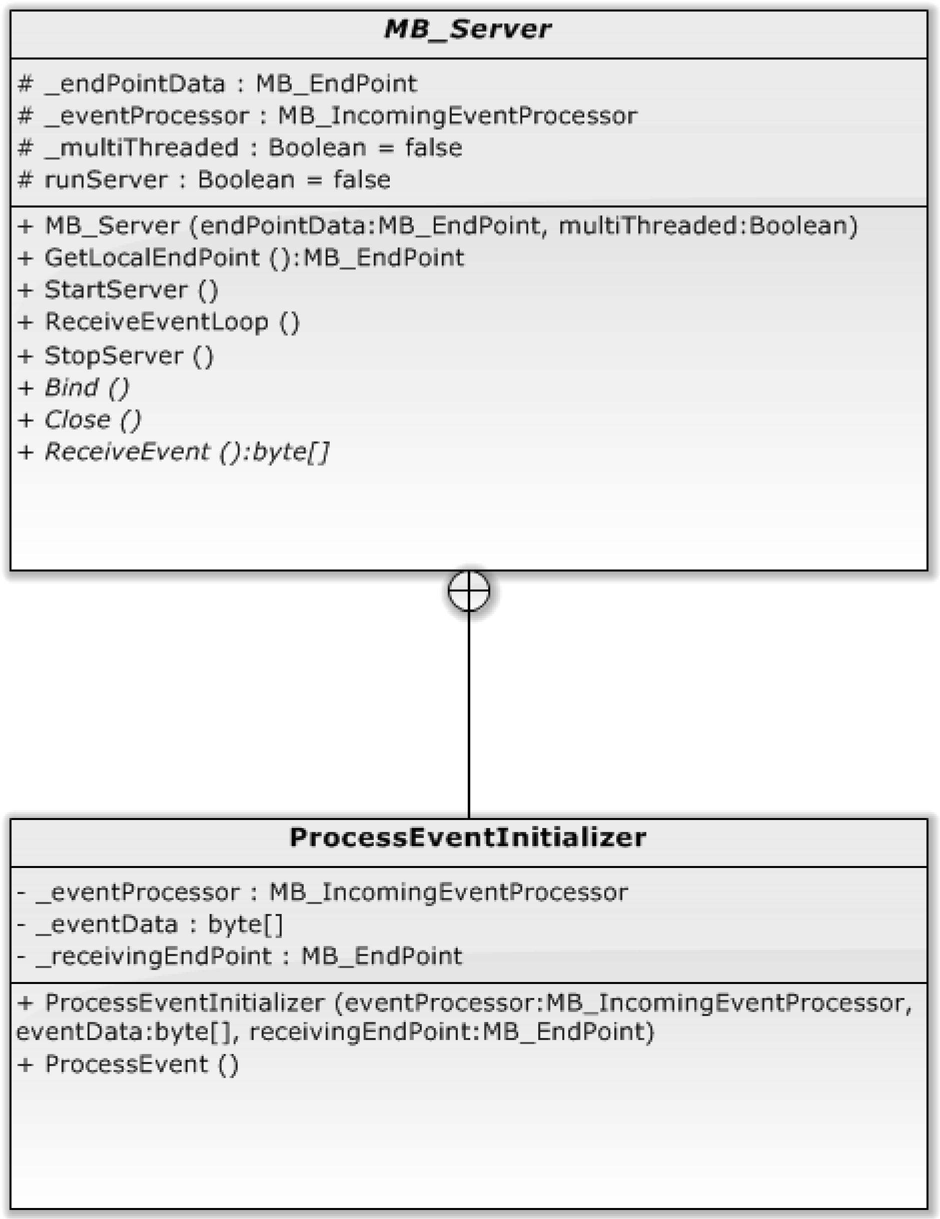
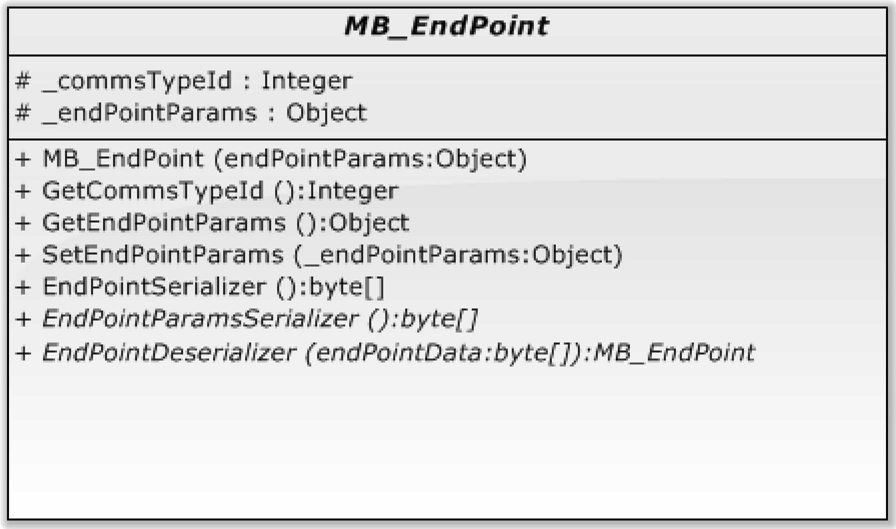

MB_Server
# _endPointData : MB_EndPoint
# _eventProcessor : MB_IncomingEventProcessor
# _multiThreaded : Boolean = false
# runServer : Boolean = false
+ MB_Server (endPointData:MB_EndPoint, multiThreaded:Boolean)
+ GetLocalEndPoint ():MB_EndPoint
+ StartServer ()
+ ReceiveEventLoop ()
+ StopServer ()
+ Bind ()
+ Close ()
+ ReceiveEvent ():byte[]
ProcessEventInitializer
- _eventProcessor : MB_IncomingEventProcessor
- _eventData : byte[]
- _receivingEndPoint : MB_EndPoint
+ ProcessEventInitializer (eventProcessor:MB_IncomingEventProcessor, eventData:byte[], receivingEndPoint:MB_EndPoint)
+ ProcessEvent ()

Figure 6-5. Midblocks Server Base Class

With these three concepts Midblocks can handle a large array of different types of communication interfaces.


MB_EndPoint
# _commsTypeId : Integer
# _endPointParams : Object
+ MB_EndPoint (endPointParams:Object)
+ GetCommsTypeId ():Integer
+ GetEndPointParams ():Object
+ SetEndPointParams (_endPointParams:Object)
+ EndPointSerializer ():byte[]
+ *EndPointParamsSerializer ():byte[]*
+ *EndPointDeserializer (endPointData:byte[]):MB_EndPoint*

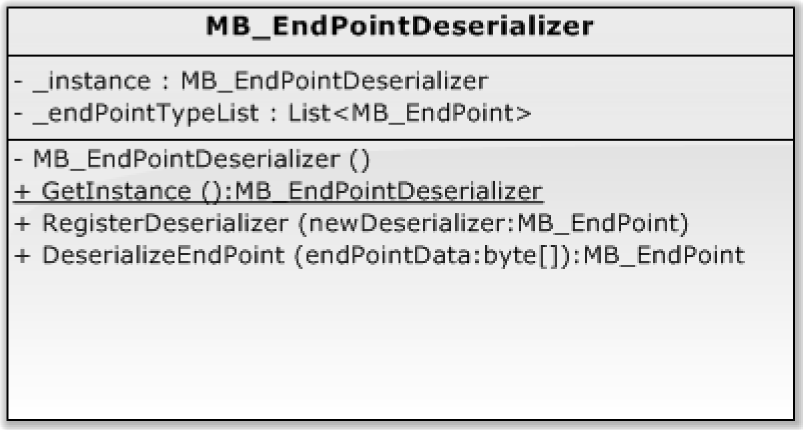

MB_EndPointDeserializer
- _instance : MB_EndPointDeserializer
- _endPointTypeList : List<MB_EndPoint>
- MB_EndPointDeserializer ()
+ GetInstance ():MB_EndPointDeserializer
+ RegisterDeserializer (newDeserializer:MB_EndPoint)
+ DeserializeEndPoint (endPointData:byte[]):MB_EndPoint

Figure 6-6. Midblocks Endpoint Base Class and Endpoint Deserializer Class

Midblocks communications servers are designed to operate both in single-threaded and multi-threaded forms. Some languages, like Visual C#, cannot pass parameters to methods that are launched in a new thread. As a result the *MB_Server* base class makes use of a *ProcessEventInitializer* inner class as a way to pass parameters to a method that is launched as a new thread.

Since no assumption is made about end points, it is necessary to have a type-specific method which will obtain end point data from received messages. To provide support for various types of end points Midblocks makes use of an end point deserializer class, called *MB_EndPointDeserializer* (see Figure 6-6). Each end point type must implement the *EndPointDeserializer* method which provides the means to obtain the end point data from Midblocks messages. Each Midblocks server registers this deserializer method with an instance of the *MB_EndPointDeserializer* class. To avoid inconsistencies, there should only be one instance of the *MB_EndPointDeserializer* class per component, therefore this class makes use of a singleton pattern.

6.4.1.3 Midblocks Messages

Midblocks messages are the combination of an event and an end point. Messages contain the data of an event and the information necessary to route the event to its intended destination. Midblocks messages are represented by the *MB_Message* class.

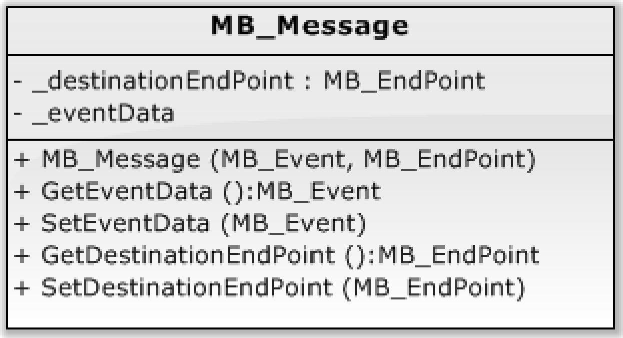

MB_Message
- _destinationEndPoint : MB_EndPoint
- _eventData
+ MB_Message (MB_Event, MB_EndPoint)
+ GetEventData ():MB_Event
+ SetEventData (MB_Event)
+ GetDestinationEndPoint ():MB_EndPoint
+ SetDestinationEndPoint (MB_EndPoint)

Figure 6-7. Midblocks Message Class

6.4.1.4 Midblocks Network (IEEE 802.3) Communications

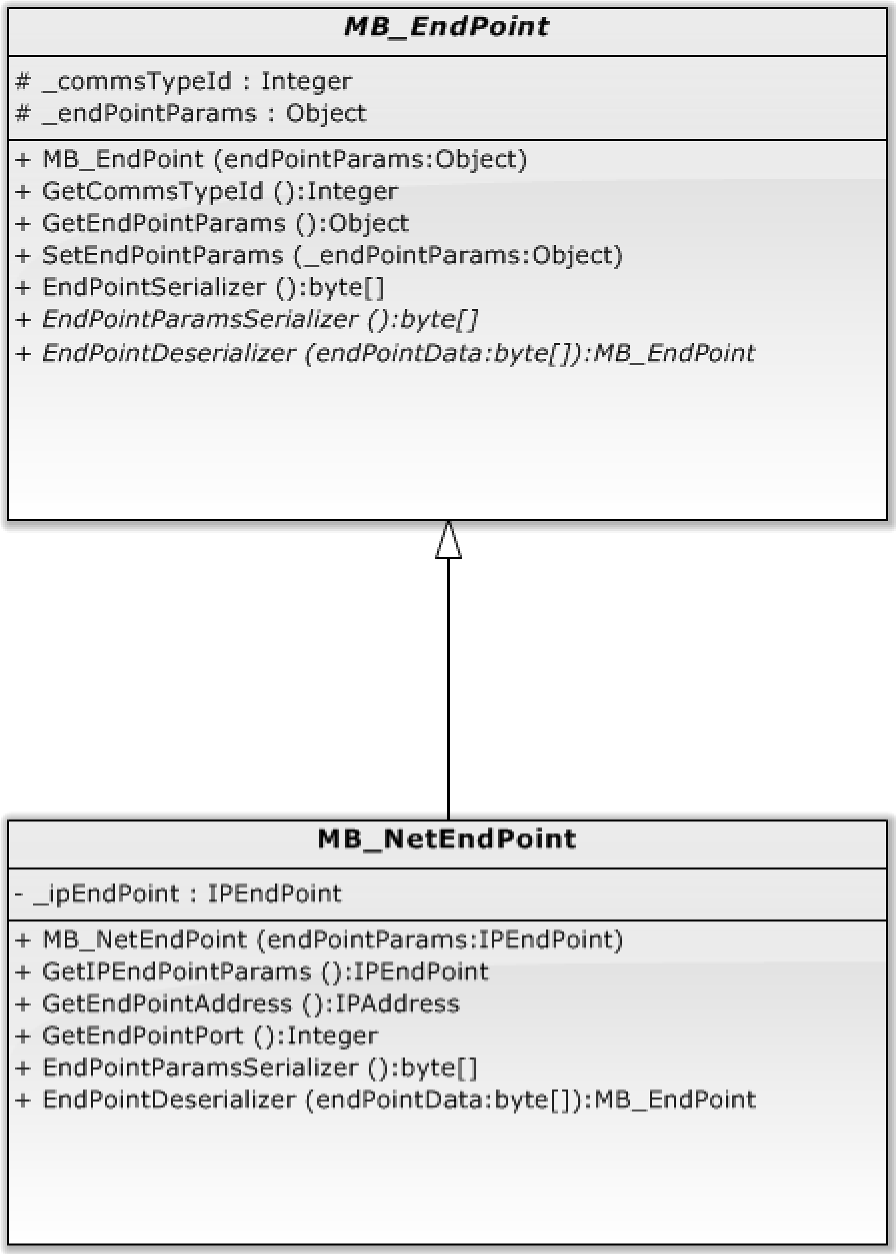

MB_EndPoint
# _commsTypeId : Integer
# _endPointParams : Object
+ MB_EndPoint (endPointParams:Object)
+ GetCommsTypeId ():Integer
+ GetEndPointParams ():Object
+ SetEndPointParams (_endPointParams:Object)
+ EndPointSerializer ():byte[]
+ *EndPointParamsSerializer ():byte[]*
+ *EndPointDeserializer (endPointData:byte[]):MB_EndPoint*
MB_NetEndPoint
- _ipEndPoint : IPEndPoint
+ MB_NetEndPoint (endPointParams:IPEndPoint)
+ GetIPEndPointParams ():IPEndPoint
+ GetEndPointAddress ():IPAddress
+ GetEndPointPort ():Integer
+ EndPointParamsSerializer ():byte[]
+ EndPointDeserializer (endPointData:byte[]):MB_EndPoint

Figure 6-8. Midblocks IEEE 802.3 networking endpoint class

This first implementation of Midblocks has support for IEEE 802.3 (Ethernet) and IEEE 802.11 (Wi-Fi) based communications. The implementation of IEEE networking communications serves both to provide support for networking and as a template for the implementation of other communication types.

To provide support for IEEE networks, we first derive a new class from the *MB_EndPoint* class. This new class, called *MB_NetEndPoint*, implements the *EndPointParamsSerializer* and *EndPointParamsDeserializer* methods (Figure 6-8).

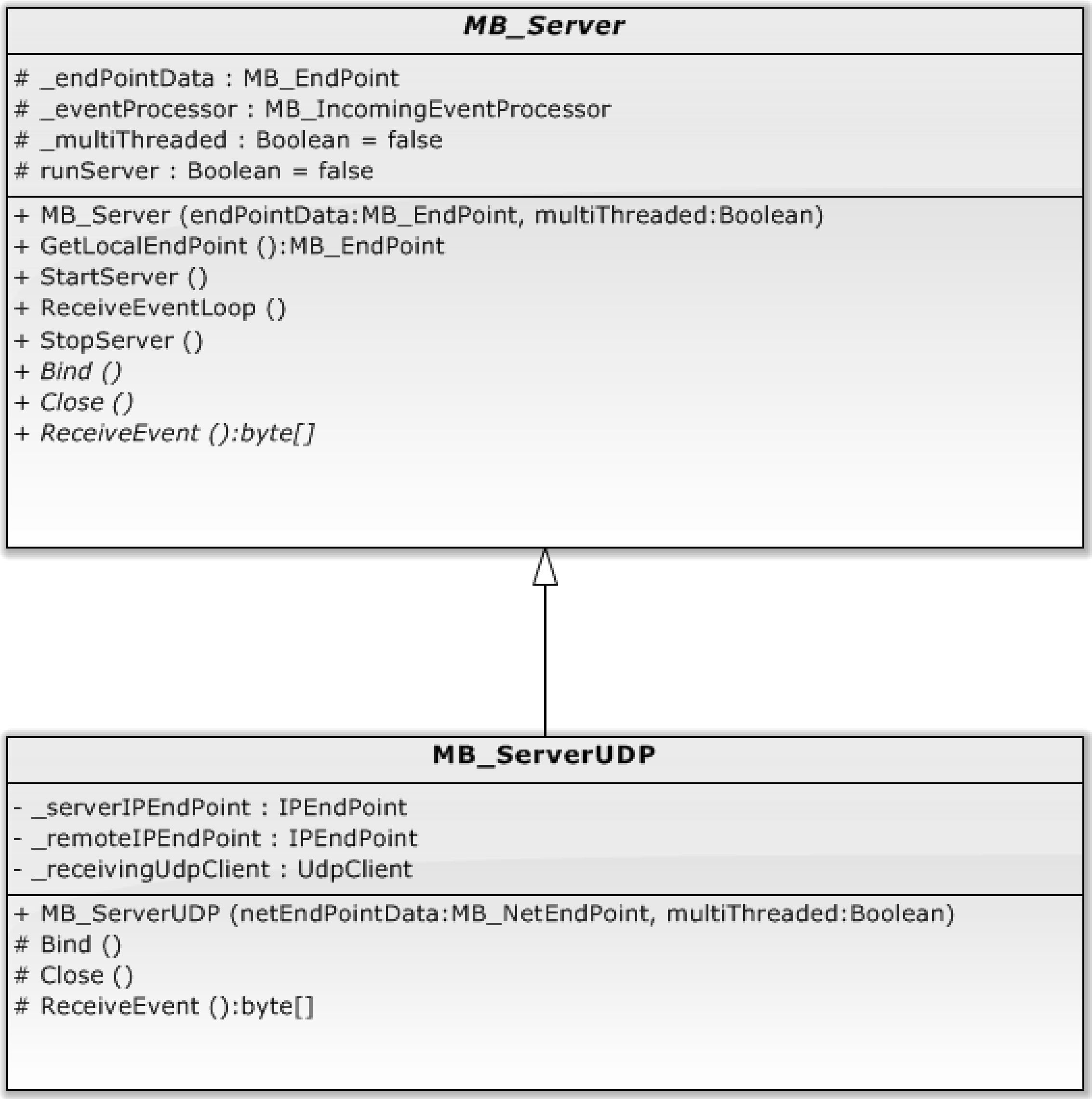

MB_Server
# _endPointData : MB_EndPoint
# _eventProcessor : MB_IncomingEventProcessor
# _multiThreaded : Boolean = false
# runServer : Boolean = false
+ MB_Server (endPointData:MB_EndPoint, multiThreaded:Boolean)
+ GetLocalEndPoint ():MB_EndPoint
+ StartServer ()
+ ReceiveEventLoop ()
+ StopServer ()
+ Bind ()
+ Close ()
+ ReceiveEvent ():byte[]
MB_ServerUDP
- _serverIPEndPoint : IPEndPoint
- _remoteIPEndPoint : IPEndPoint
- _receivingUdpClient : UdpClient
+ MB_ServerUDP (netEndPointData:MB_NetEndPoint, multiThreaded:Boolean)
# Bind ()
# Close ()
# ReceiveEvent ():byte[]

Figure 6-9. Midblocks IEEE 802.3 server class

To send data over the network and to handle incoming data we derived classes from the *MB_Server* and *MB_Client* classes. The class derived from *MB_Server*, called *MB_ServerUDP*, implements the *Bind*, *Close* and *ReceiveEvent* methods (Figure 6-9). While the class derived from *MB_Client*, called *MB_ClientUDP*, implements the *Close, Connect*, and *SendEvent* methods.

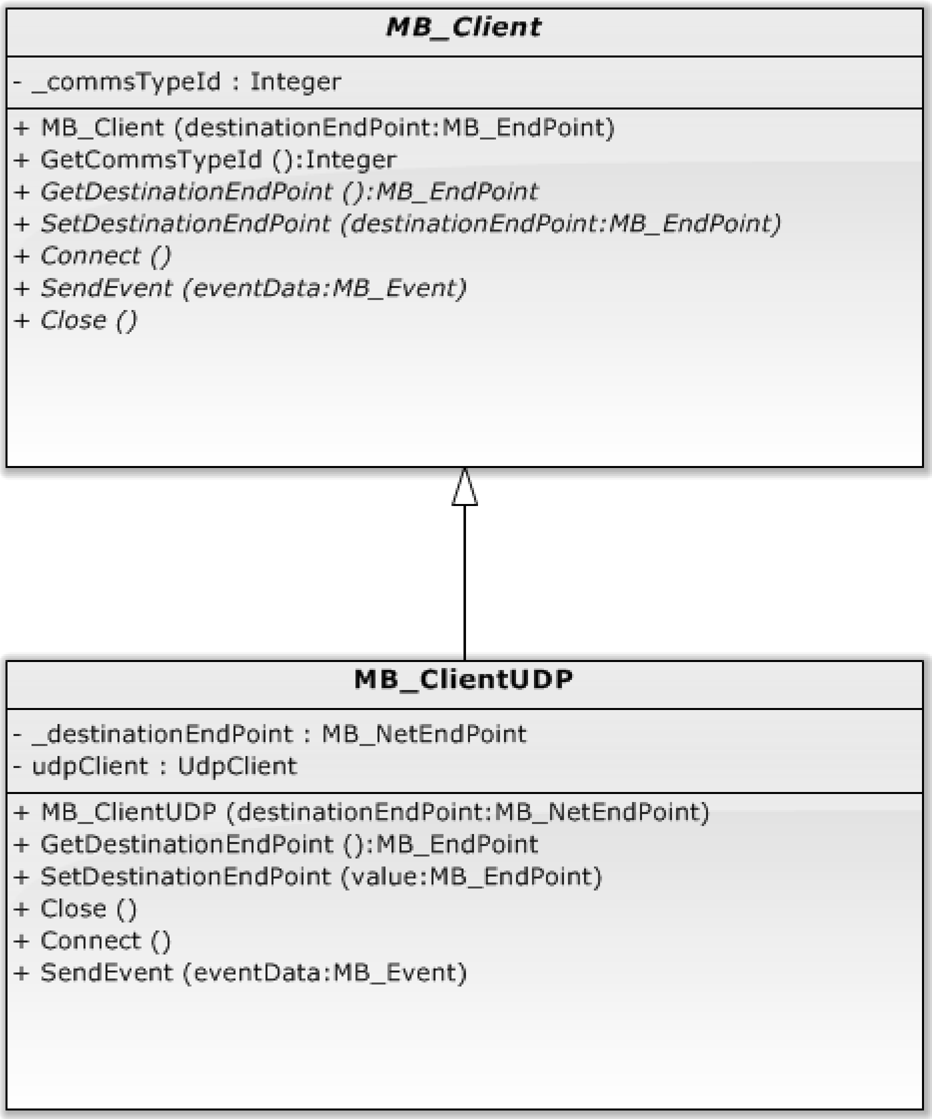

MB_Client
- _commsTypeId : Integer
+ MB_Client (destinationEndPoint:MB_EndPoint)
+ GetCommsTypeId ():Integer
+ GetDestinationEndPoint ():MB_EndPoint
+ SetDestinationEndPoint (destinationEndPoint:MB_EndPoint)
+ Connect ()
+ SendEvent (eventData:MB_Event)
+ Close ()
MB_ClientUDP
- _destinationEndPoint : MB_NetEndPoint
- udpClient : UdpClient
+ MB_ClientUDP (destinationEndPoint:MB_NetEndPoint)
+ GetDestinationEndPoint ():MB_EndPoint
+ SetDestinationEndPoint (value:MB_EndPoint)
+ Close ()
+ Connect ()
+ SendEvent (eventData:MB_Event)

Figure 6-10. Midblocks IEEE 802.3 client class

6.4.1.5 Midblocks Event Processing

To allow future introduction of new events, an expandable event processing scheme is used in Midblocks.

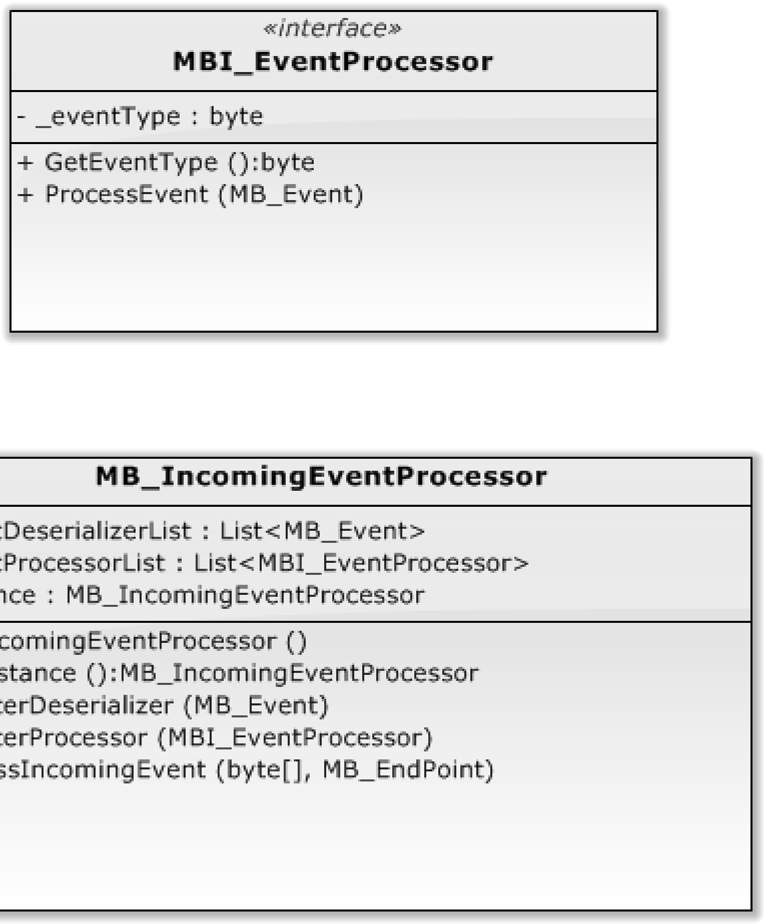

«interface»
MBI_EventProcessor
- _eventType : byte
+ GetEventType ():byte
+ ProcessEvent (MB_Event)
MB_IncomingEventProcessor
- _eventDeserializerList : List<MB_Event>
- _eventProcessorList : List<MBI_EventProcessor>
- _instance : MB_IncomingEventProcessor
- MB_IncomingEventProcessor ()
+ GetInstance ():MB_IncomingEventProcessor
+ RegisterDeserializer (MB_Event)
+ RegisterProcessor (MBI_EventProcessor)
+ ProcessIncomingEvent (byte[], MB_EndPoint)

Figure 6-11. Midblocks event processing class and interface

Event processing is done through classes that implement the *MBI_EventProcessor* interface. Instances of said classes are the registered with an instance of the *MB_IncomingEventProcessor* class (see Figure 6-11). *MB_IncomingEventProcessor* serves as a form of switchboard that selects the appropriate processor based on the event type. *MB_IncomingEventProcessor* is used by *MB_Server* to select the correct processor to process each incoming event. To ensure that each event is processed adequately, there must be only one event processor per component type. Therefore *MB_IncomingEventProcessor* makes use of a singleton pattern.

6.4.1.6 Midblocks Component

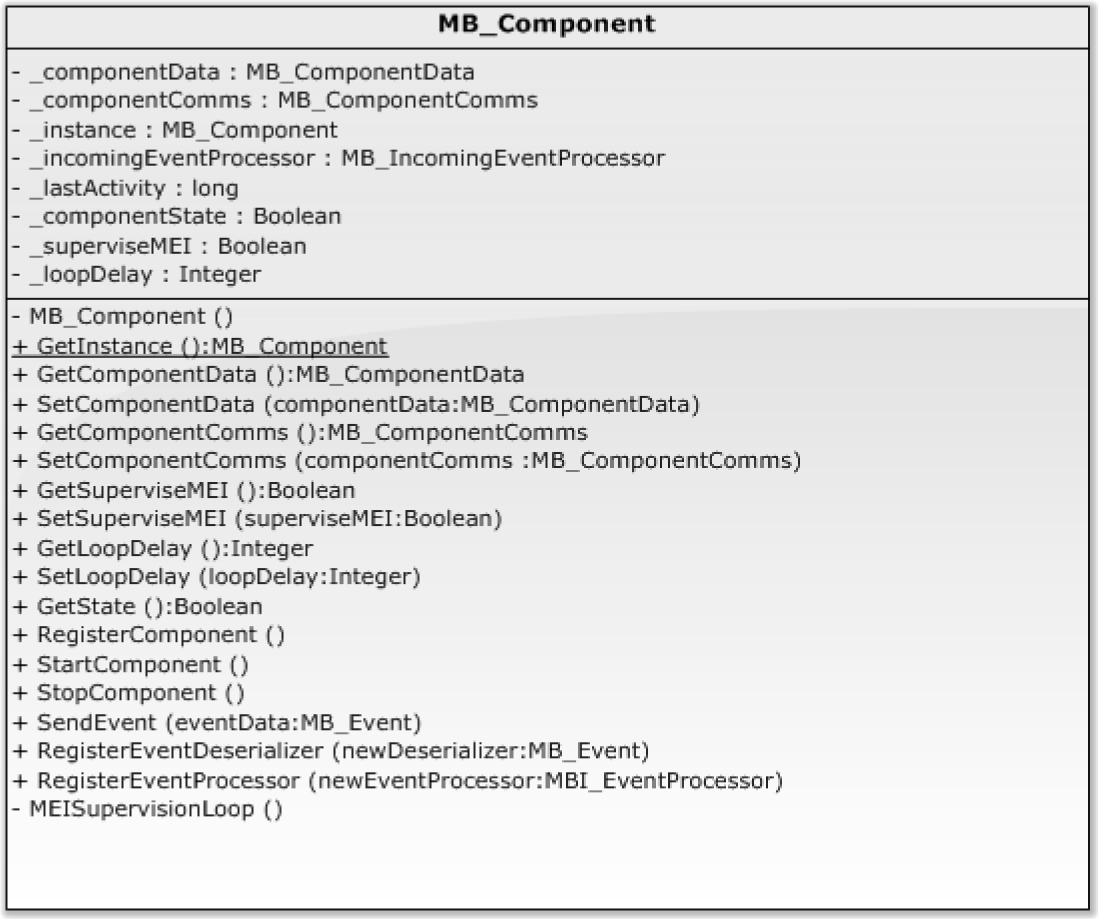

MB_Component
- _componentData : MB_ComponentData
- _componentComms : MB_ComponentComms
- _instance : MB_Component
- _incomingEventProcessor : MB_IncomingEventProcessor
- _lastActivity : long
- _componentState : Boolean
- _superviseMEI : Boolean
- _loopDelay : Integer
- MB_Component ()
+ GetInstance ():MB_Component
+ GetComponentData ():MB_ComponentData
+ SetComponentData (componentData:MB_ComponentData)
+ GetComponentComms ():MB_ComponentComms
+ SetComponentComms (componentComms :MB_ComponentComms)
+ GetSuperviseMEI ():Boolean
+ SetSuperviseMEI (superviseMEI:Boolean)
+ GetLoopDelay ():Integer
+ SetLoopDelay (loopDelay:Integer)
+ GetState ():Boolean
+ RegisterComponent ()
+ StartComponent ()
+ StopComponent ()
+ SendEvent (eventData:MB_Event)
+ RegisterEventDeserializer (newDeserializer:MB_Event)
+ RegisterEventProcessor (newEventProcessor:MBI_EventProcessor)
- MEISupervisionLoop ()

Figure 6-12. Midblocks component class

Components in Midblocks are represented by the *MB_Component* class (). Instances of this class have two modes of operation, dependent on the value of the supervise MEI member variable:

1. *Independently Supervised MEI.* In this mode of operation a separate thread is launched to supervise that the component does not exceed its Maximum Event Interval.
2. *In-line Supervised MEI.* In this mode of operation MEI supervision is performed in the component's main event loop or in an independent user-provided thread.

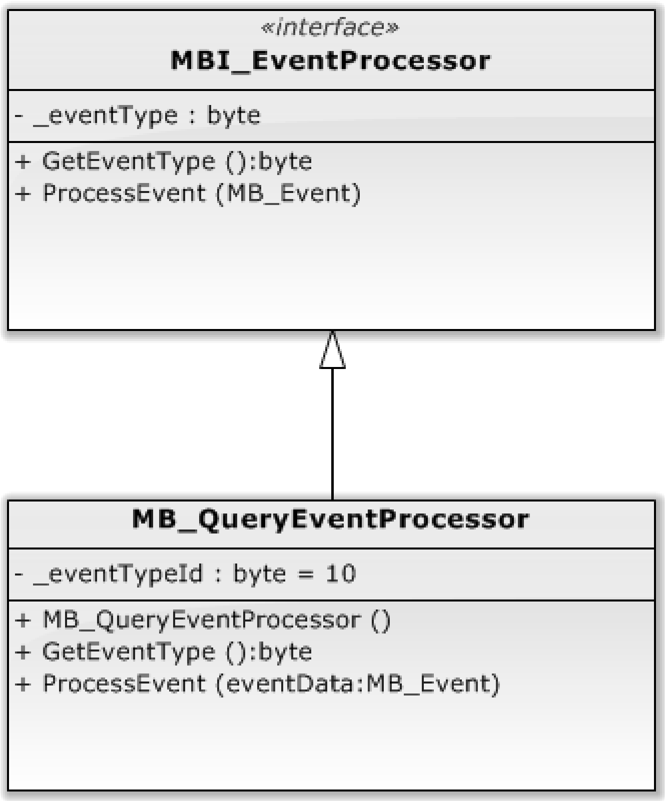


Figure 6-13. Midblocks component default query event processing

In the *Independently Supervised MEI* mode Midblocks provides a default query event handling mechanism by implementing *MBI_EventProcessor* interface in a class called *MB_QueryEventProcessor* (Figure 6-13). The default behaviour is simply to respond to query events with an alive signal event. If a component requires further processing upon reception of a query event the *In-line Supervised MEI* mode of operation must be employed and an independent MEI supervision thread must be employed by the user or query event supervision must be performed in the component's main loop.

Each component employs an instance of the *MB_IncomingEventProcessor* class to handle incoming events. Regardless of the MEI supervision method employed in a particular component, if the component must process other events, such as data events, the classes implementing the *MBI_EventProcessor* interface must be defined to handle said events, and instances of these classes must be registered in the components *MB_IncomingEventProcessor* object at component startup.

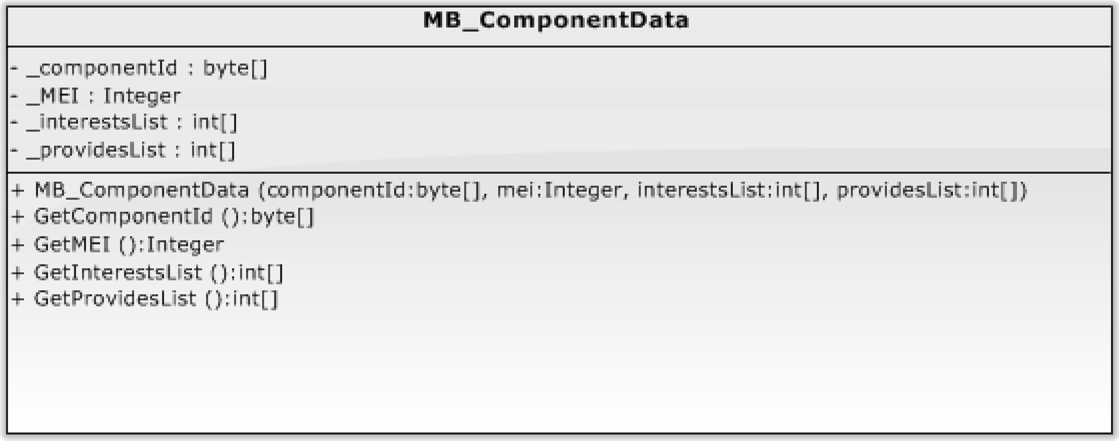

MB_ComponentData
- _componentId : byte[]
- _MEI : Integer
- _interestsList : int[]
- _providesList : int[]
+ MB_ComponentData (componentId:byte[], mei:Integer, interestsList:int[], providesList:int[])
+ GetComponentId ():byte[]
+ GetMEI ():Integer
+ GetInterestsList ():int[]
+ GetProvidesList ():int[]

Figure 6-14. Midblocks component data class

To handle component parameters in an efficient manner, Midblocks provides the *MB_ComponentData* class which encapsulates the following parameters (see Figure 6-14):

1. Component Id.
2. Maximum Event Interval
3. Interests List.
4. Provides List.

The interests list is an array of integer which specifies the types of events that a component should receive. The values, and meaning, of these integers are application specific. The provides list, on the other hand, is an array of integers that specifies the types of data that a component provides. The interests list and the provides list are used as a mechanism to connect data generators and data consumers.

Since Midblocks components are designed to be independent of a specific communications mechanism, it is necessary to encapsulate the details of communications handling independently from component behaviour. This is achieved by providing a class, called *MB_ComponentComms* (Figure 6-15), which encapsulates communications details.

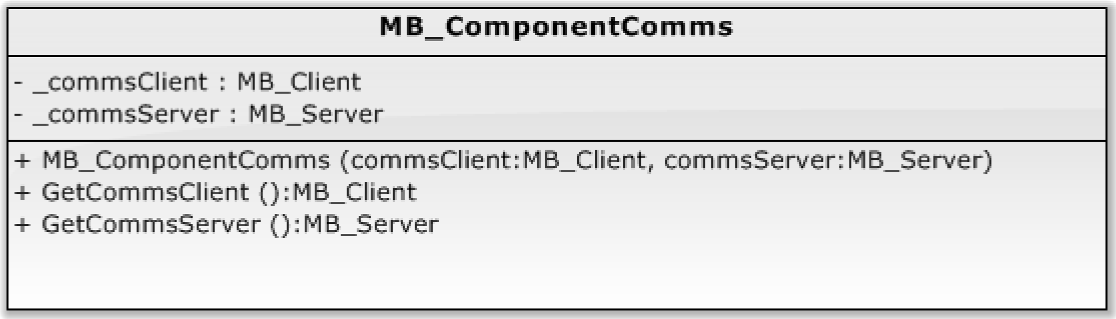

MB_ComponentComms
- _commsClient : MB_Client
- _commsServer : MB_Server
+ MB_ComponentComms (commsClient:MB_Client, commsServer:MB_Server)
+ GetCommsClient ():MB_Client
+ GetCommsServer ():MB_Server

Figure 6-15. Midblocks component communications class

Currently, Midblocks components can handle only one communications interface however, by separating component behaviour from communications interface interaction, we expect to introduce the capability of handling multiple communications interfaces in future versions of Midblocks components.

6.4.1.7 Midblocks Controller

Controller implementation is very similar to component implementation. Controllers are represented by the *MB_Controller* class (see Figure 6-16)

Currently controllers receive notifications only when a component exceeds its MEI. However future versions of Midblocks could use more than one notification for controller, for example, one type of notification when a component has exceeded its MEI and a second type of notification when it has been necessary to send a query event to a component.

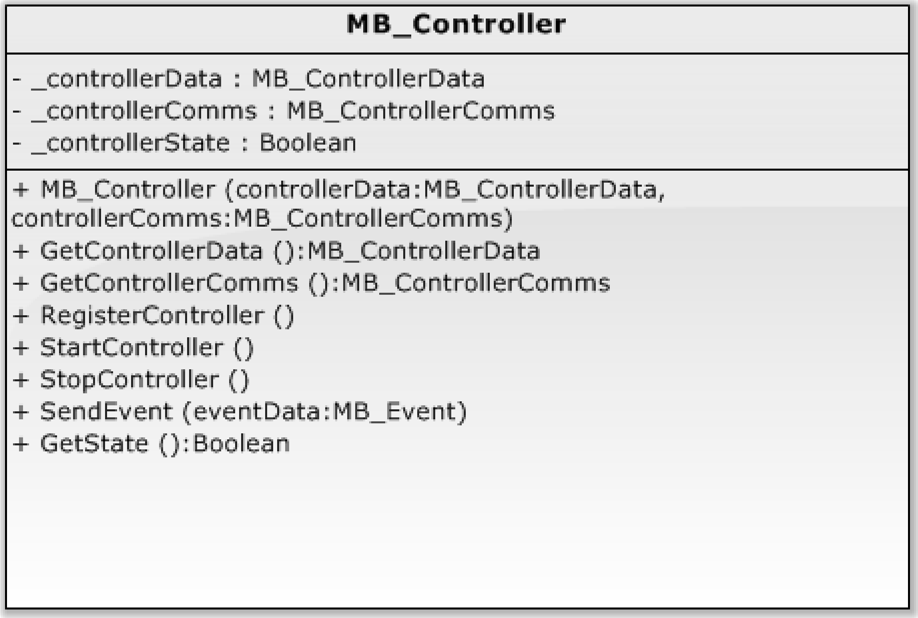

MB_Controller
- _controllerData : MB_ControllerData
- _controllerComms : MB_ControllerComms
- _controllerState : Boolean
+ MB_Controller (controllerData:MB_ControllerData, controllerComms:MB_ControllerComms)
+ GetControllerData ():MB_ControllerData
+ GetControllerComms ():MB_ControllerComms
+ RegisterController ()
+ StartController ()
+ StopController ()
+ SendEvent (eventData:MB_Event)
+ GetState ():Boolean

Figure 6-16. Midblocks controller class

The use of more than one notification type could be useful in scenarios such as diagnosing component or communications links which have intermittent failures.

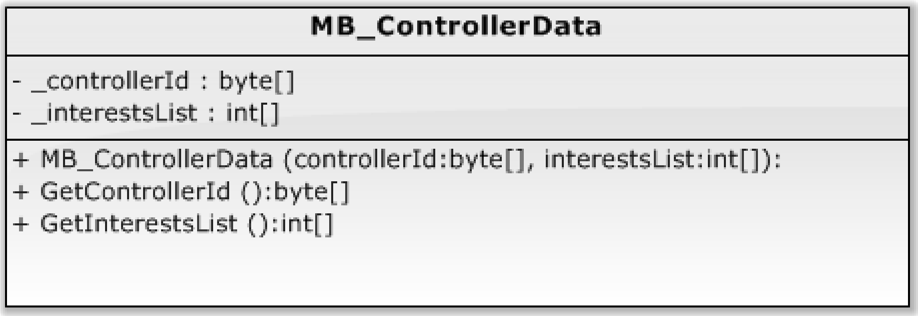

MB_ControllerData
- _controllerId : byte[]
- _interestsList : int[]
+ MB_ControllerData (controllerId:byte[], interestsList:int[]):
+ GetControllerId ():byte[]
+ GetInterestsList ():int[]

Figure 6-17. Midblocks controller data class

To provide this type of flexibility Midblocks makes use of an interests list for controllers analogous to the interests list used in components. It is important to note, however, that the set of interests identifiers used for component data and the set of interests identifiers used for controller notifications are independent. To facilitate the handling of controller parameters Midblocks provides a *MB_ControllerData* class that is analogous to the *MB_ComponentData* class used for component data handling (Figure 6-17).

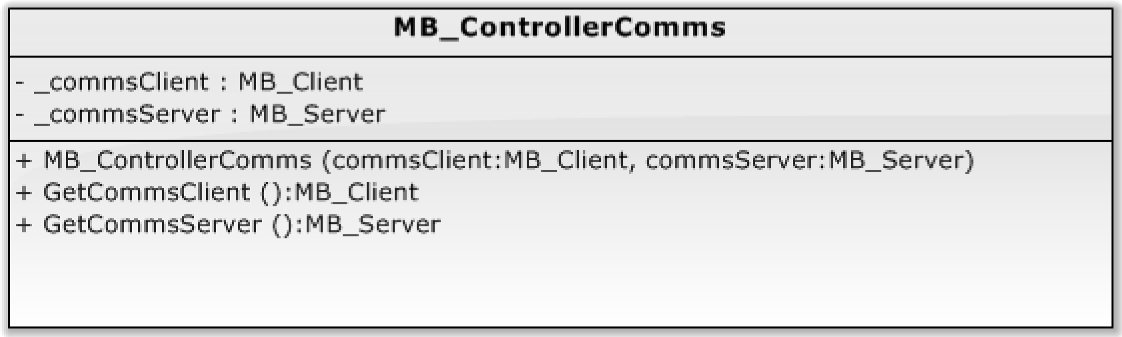

MB_ControllerComms
- _commsClient : MB_Client
- _commsServer : MB_Server
+ MB_ControllerComms (commsClient:MB_Client, commsServer:MB_Server)
+ GetCommsClient ():MB_Client
+ GetCommsServer ():MB_Server

Figure 6-18. Midblocks controller communications class

As in the case of components, to allow use of diverse types of communication mechanisms it is necessary to encapsulate the details of communications handling in the controller by means of a class called *MB_ControllerComms* (Figure 6-18).

6.4.1.8 Midblocks Store and Forward with Component Supervision

The core of the Midblocks Store and Forward with Component Supervision handling is encapsulated in the *MB_SFCS* class (see Figure 6-19). This class provides the tools to start, stop, and obtain the running status of a Midblocks SFCS part. To perform the store and forward, and component supervision tasks, the Midblocks SFCS class uses three main members:

1. A message receiver object.
2. A message dispatcher object.
3. A component supervisor object.

These three objects have to be started and stopped in a specific order to avoid generating false component failure events.

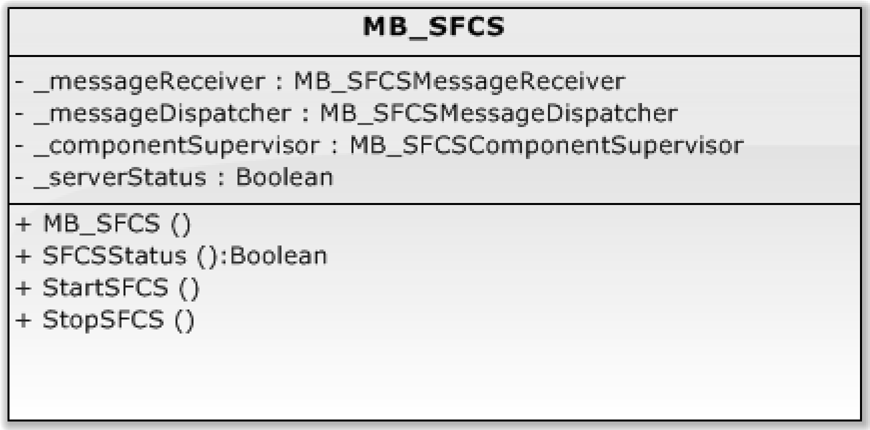

MB_SFCS
- _messageReceiver : MB_SFCSMessageReceiver
- _messageDispatcher : MB_SFCSMessageDispatcher
- _componentSupervisor : MB_SFCSComponentSupervisor
- _serverStatus : Boolean
+ MB_SFCS ()
+ SFCSStatus ():Boolean
+ StartSFCS ()
+ StopSFCS ()

Figure 6-19. Midblocks Store and Forward with Component Supervision class

The message dispatcher, represented by the *MB_SFCSMessageDispatcher* class (Figure 6-20), is used to send events to components.

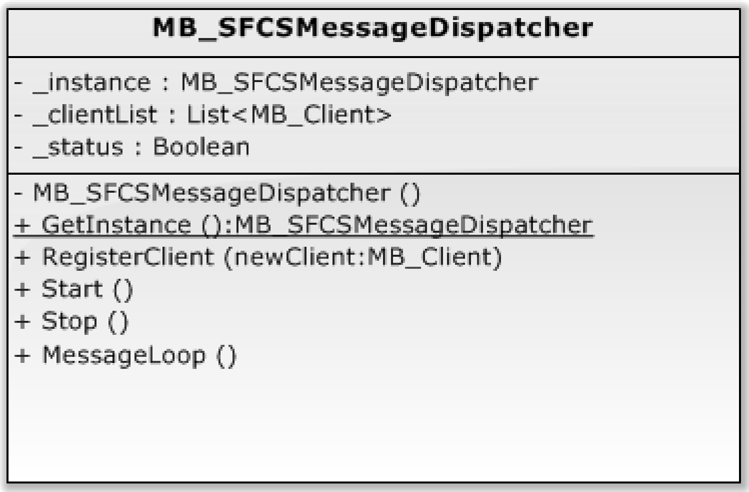

MB_SFCSMessageDispatcher
- _instance : MB_SFCSMessageDispatcher
- _clientList : List<MB_Client>
- _status : Boolean
- MB_SFCSMessageDispatcher ()
+ GetInstance ():MB_SFCSMessageDispatcher
+ RegisterClient (newClient:MB_Client)
+ Start ()
+ Stop ()
+ MessageLoop ()

Figure 6-20. Midblocks SFCS message dispatcher class

Since there must be only one message dispatcher in every instance of a Midblocks SFCS a singleton pattern is used in the *MB_SFCSMessageDispatcher* class.

The message dispatcher can handle several different types of communication interfaces simultaneously. To provide this functionality, the message dispatcher contains an array of references to the communications interfaces being employed. Registration of a communications interface is performed by means of the *RegisterClient* method. All communications interfaces that will be used in the message dispatcher must be registered before the starting the Midblocks SFCS.

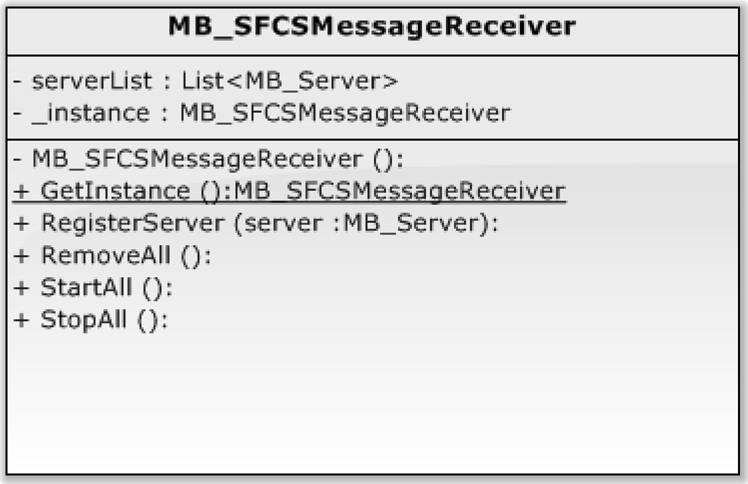

MB_SFCSMessageReceiver
- serverList : List<MB_Server>
- _instance : MB_SFCSMessageReceiver
- MB_SFCSMessageReceiver ():
+ GetInstance ():MB_SFCSMessageReceiver
+ RegisterServer (server :MB_Server):
+ RemoveAll ():
+ StartAll ():
+ StopAll ():

Figure 6-21. Midblocks SFCS message receiver class

The counterpart of the message dispatcher is the message receiver, which is represented by the *MB_SFCSMessageReceiver* class (see Figure 6-21). The message receiver is tasked with receiving event messages from components and triggering the processing of said messages. As with the message dispatcher, the message receiver can make use of multiple communications interfaces. To register the communications interfaces that will be used for message reception the *MB_SFCSMessageReceiver* class has a *RegisterServer* method. All communications interfaces that will be used by the message receiver must be registered before starting the Midblocks SFCS. There should also be only one instance of the message receiver per SFCS, therefore a singleton pattern is used in the *MB_SFCSMessageReceiver* class.

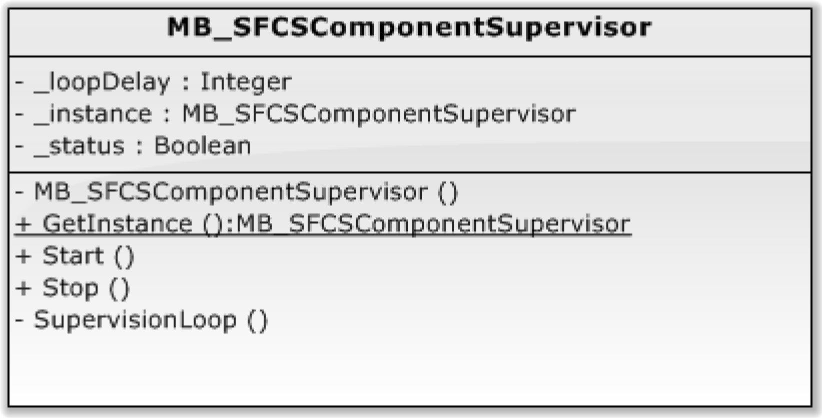

MB_SFCSComponentSupervisor
- _loopDelay : Integer
- _instance : MB_SFCSComponentSupervisor
- _status : Boolean
- MB_SFCSComponentSupervisor ()
+ GetInstance ():MB_SFCSComponentSupervisor
+ Start ()
+ Stop ()
- SupervisionLoop ()

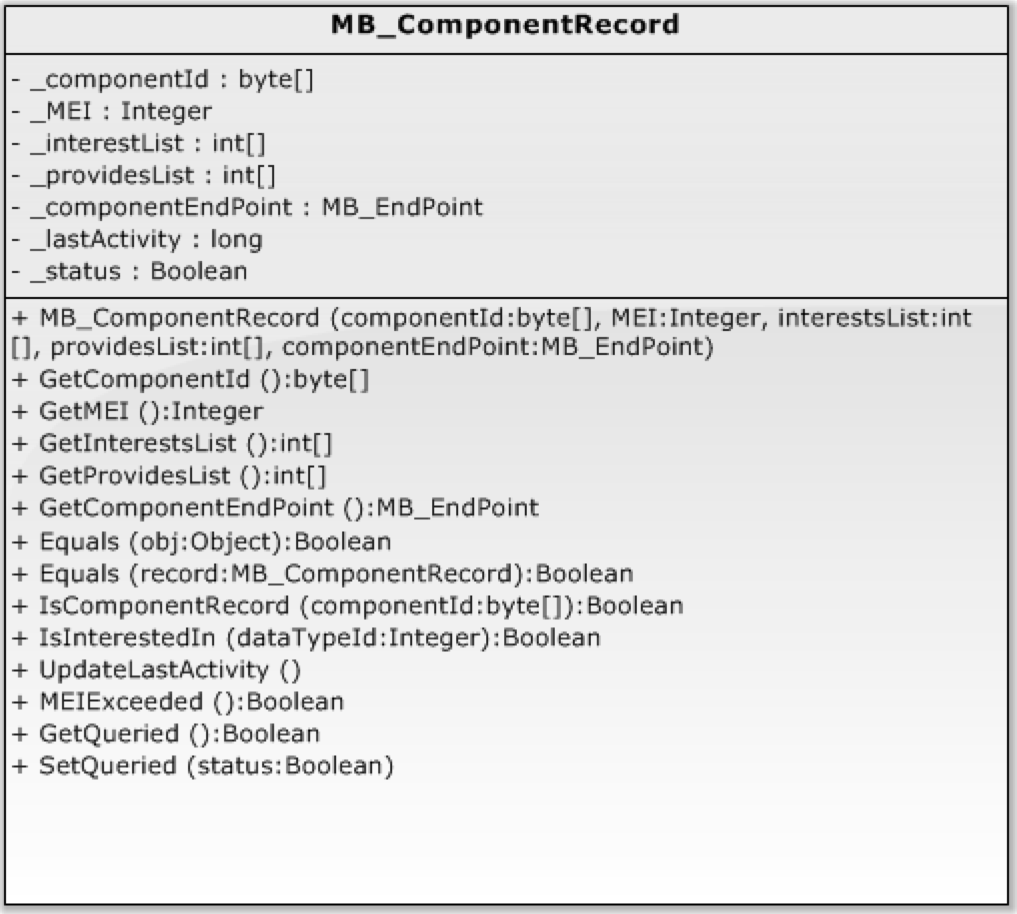
Figure 6-22. Midblocks SFCS component supervisor class

Component supervision is done by means of the component supervisor, whose behavior is encapsulated in the *MB_SFCSComponentSupervisor* class. The component supervisor performs a loop verifying the time elapsed since each component sent its last event. If the time elapsed is larger than the component's MEI then one of the following two tasks is performed by the component supervisor:

1. If the component has not been sent a query event, the component supervisor sends a query event to the component and updates the component's record last activity field.
2. If the component has been sent a query event previously, and has not responded to the query event, the component is eliminated from the SFCS and controllers are notified of the component's failure.

The parameters associated with each component are handled through instances of the *MB_ComponentRecord* class (see Figure 6-23). When a component registers with the Midblocks SFCS the following sequence of events happens:

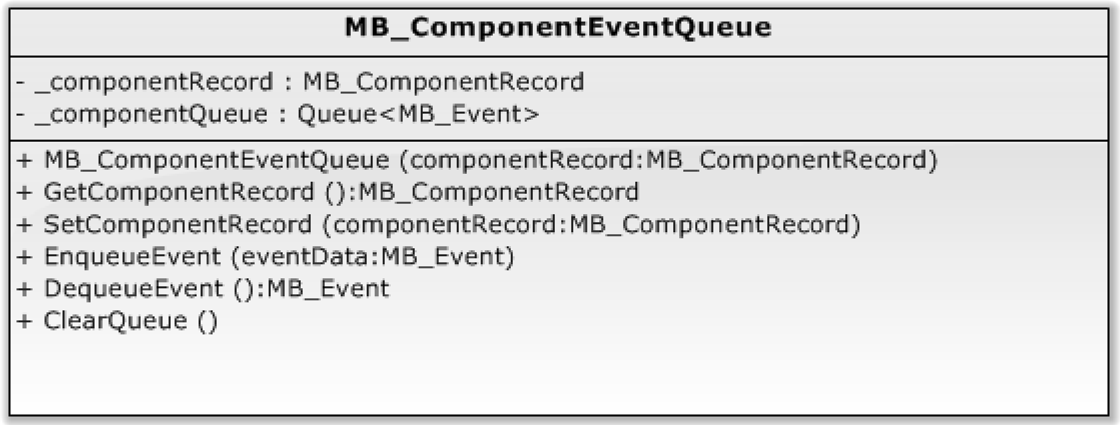
1. An instance of the *MB_ComponentRecord* class is created to handle the component's parameters.
2. An instance of the *MB_ComponentEventQueue* class (Figure 6-24) is created by the Midblocks component event queue handler to store the component's outgoing messages.
3. The component record's last activity field is updated to begin supervising the component.
4. Controllers subscribed to component registration notification events are notified of the components registration.



```
MB_ComponentRecord
- _componentId : byte[]
- _MEI : Integer
- _interestList : int[]
- _providesList : int[]
- _componentEndPoint : MB_EndPoint
- _lastActivity : long
- _status : Boolean
+ MB_ComponentRecord (componentId:byte[], MEI:Integer, interestsList:int
[], providesList:int[], componentEndPoint:MB_EndPoint)
+ GetComponentId ():byte[]
+ GetMEI ():Integer
+ GetInterestsList ():int[]
+ GetProvidesList ():int[]
+ GetComponentEndPoint ():MB_EndPoint
+ Equals (obj:Object):Boolean
+ Equals (record:MB_ComponentRecord):Boolean
+ IsComponentRecord (componentId:byte[]):Boolean
+ IsInterestedIn (dataTypeId:Integer):Boolean
+ UpdateLastActivity ()
+ MEIExceeded ():Boolean
+ GetQueried ():Boolean
+ SetQueried (status:Boolean)
```

Figure 6-23. Midblocks component record class

The component's event queue serves as a temporary buffer for events that are destined for the component. These messages are dequeued by the message dispatcher and sent to the component. If a component exceeds its MEI and fails to respond to a query event, then the component's event queue, along with the component's record, is destroyed and notifications are sent to the controllers.



```
MB_ComponentEventQueue
- _componentRecord : MB_ComponentRecord
- _componentQueue : Queue<MB_Event>
+ MB_ComponentEventQueue (componentRecord:MB_ComponentRecord)
+ GetComponentRecord ():MB_ComponentRecord
+ SetComponentRecord (componentRecord:MB_ComponentRecord)
+ EnqueueEvent (eventData:MB_Event)
+ DequeueEvent ():MB_Event
+ ClearQueue ()
```

Figure 6-24. Midblocks component event queue

Component queue management is performed through the component event queue handler. The component event queue handler is represented by the *MB_SFCSComponentEventQueue* class (see Figure 6-25). Since there must be only one component event queue handler in each Midblocs SFCS, the *MB_SFCSComponentEventQueue* class makes use of a singleton pattern.

Creation and deletion of component event queues is performed as part of the registration and removal process of a component using the *RegisterComponent* and *RemoveComponent* methods of the component event queue handler. To avoid any potential inconsistencies, other parts of the Midblocks SFCS do not have direct access to the component event queues.

Insertion and extraction of events into the event queues is performed by means of the *EnqueueForComponent*, *EnqueueForInterested*, and *Dequeue* methods of the component event queue handler. *EnqueueForComponent* is used to send an event to a specific component. This method is used to send query events to components that have exceeded their MEI. The *EnqueueForInterested* method is used to send events to all components that have subscribed to a certain type of event. The message dispatcher uses the *Dequeue* method to extract the next message destined for a component. The *Dequeue* method returns the next message destined for a component and simultaneously deletes the message from the queue.

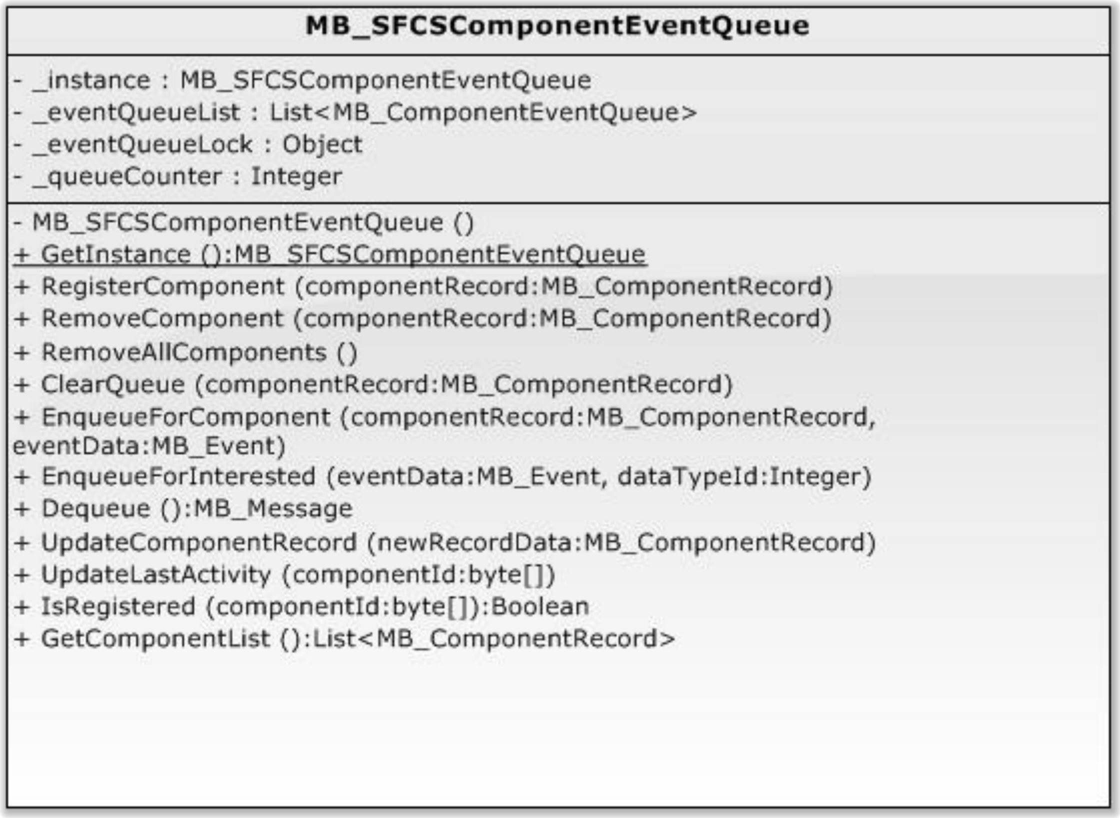

MB_SFCSComponentEventQueue
- _instance : MB_SFCSComponentEventQueue
- _eventQueueList : List<MB_ComponentEventQueue>
- _eventQueueLock : Object
- _queueCounter : Integer
- MB_SFCSComponentEventQueue ()
+ GetInstance ():MB_SFCSComponentEventQueue
+ RegisterComponent (componentRecord:MB_ComponentRecord)
+ RemoveComponent (componentRecord:MB_ComponentRecord)
+ RemoveAllComponents ()
+ ClearQueue (componentRecord:MB_ComponentRecord)
+ EnqueueForComponent (componentRecord:MB_ComponentRecord, eventData:MB_Event)
+ EnqueueForInterested (eventData:MB_Event, dataTypeId:Integer)
+ Dequeue ():MB_Message
+ UpdateComponentRecord (newRecordData:MB_ComponentRecord)
+ UpdateLastActivity (componentId:byte[])
+ IsRegistered (componentId:byte[]):Boolean
+ GetComponentList ():List<MB_ComponentRecord>

Figure 6-25. Midblocks SFCS component event queue handler

Controller handling in the Midblocks SFCS is analogous to component handling. Parameters associated with a controller are handled by means of an instance of the *MB_ControllerRecord* class (see Figure 6-26). This class contains the information necessary to send failure events to controllers. Contrary to components, controllers are not expected to send events periodically and therefore no supervision is performed on controllers.

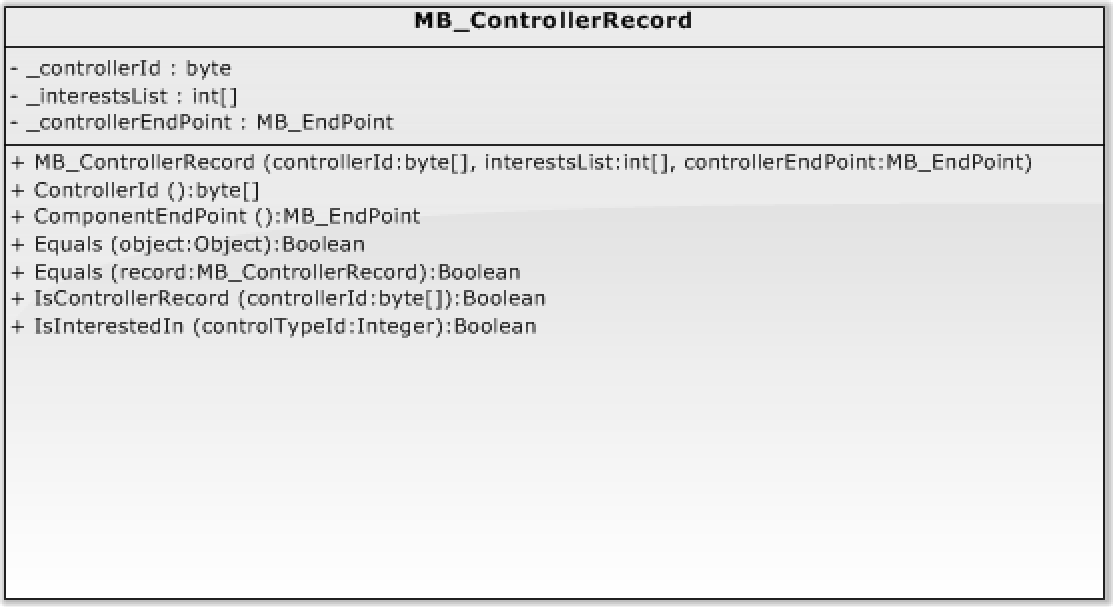

MB_ControllerRecord
- _controllerId : byte
- _interestsList : int[]
- _controllerEndPoint : MB_EndPoint
+ MB_ControllerRecord (controllerId:byte[], interestsList:int[], controllerEndPoint:MB_EndPoint)
+ ControllerId ():byte[]
+ ComponentEndPoint ():MB_EndPoint
+ Equals (object:Object):Boolean
+ Equals (record:MB_ControllerRecord):Boolean
+ IsControllerRecord (controllerId:byte[]):Boolean
+ IsInterestedIn (controlTypeId:Integer):Boolean

Figure 6-26. Midblocks controller record class

When a controller registers with the Midblocks SFCS the following sequence of events happens:

1. An instance of the *MB_ControllerRecord* class is created to handle the component's parameters.
2. An instance of the *MB_ControllerEventQueue* class (Figure 6-27) is created by the Midblocks component event queue handler to store the component's outgoing messages.

3. Controllers subscribed to controller registration notification events are notified of the components registration.

Since controllers are not supervised there is no last activity field associated to a controller.

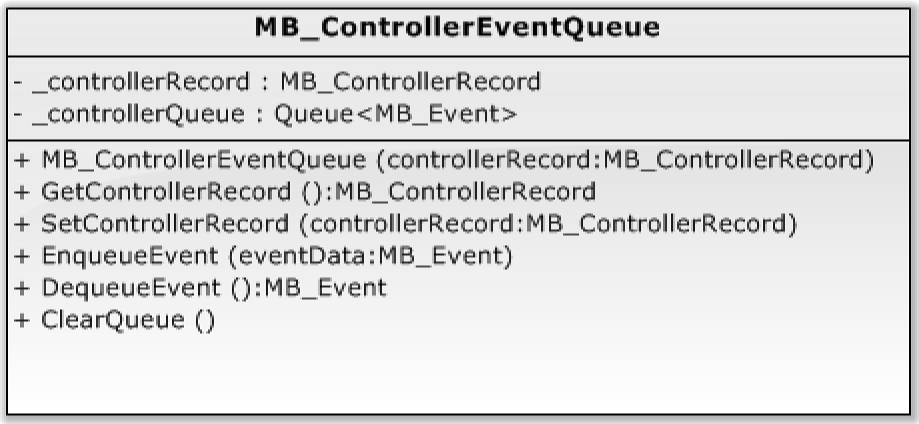

MB_ControllerEventQueue
- _controllerRecord : MB_ControllerRecord
- _controllerQueue : Queue<MB_Event>
+ MB_ControllerEventQueue (controllerRecord:MB_ControllerRecord)
+ GetControllerRecord ():MB_ControllerRecord
+ SetControllerRecord (controllerRecord:MB_ControllerRecord)
+ EnqueueEvent (eventData:MB_Event)
+ DequeueEvent ():MB_Event
+ ClearQueue ()

Figure 6-27. Midblocks controller event queue

Controller event queue behavior is encapsulated in the *MB_ControllerEventQueue* class (Figure 6-27). Controller event queues are managed by means of the controller event queue handler which is represented by the *MB_SFCSControllerEventQueue* class (Figure 6-28). As with component queues, other parts of the Midblocks SFCS do not have direct access to component queues.

Creation and deletion of controller event queues is performed as part of the registration and removal process of a controller using the *RegisterController* and *RemoveController* methods of the component event queue handler.

Insertion and extraction of events into the event queues is performed by means of the *EnqueueForController*, *EnqueueForInterested*, and *Dequeue* methods of the controller event queue handler. *EnqueueForController* is used to send an event to a specific controller. This method is intended for future applications of Midblocks such as self-organizing systems and is currently unused by Funblocks. The *EnqueueForInterested* method is used to send events to all controllers that have subscribed to a certain type of event. The message dispatcher uses the *Dequeue* method to extract the next message destined for a controller. The *Dequeue* method returns the next message destined for a controller and simultaneously deletes the message from the queue.

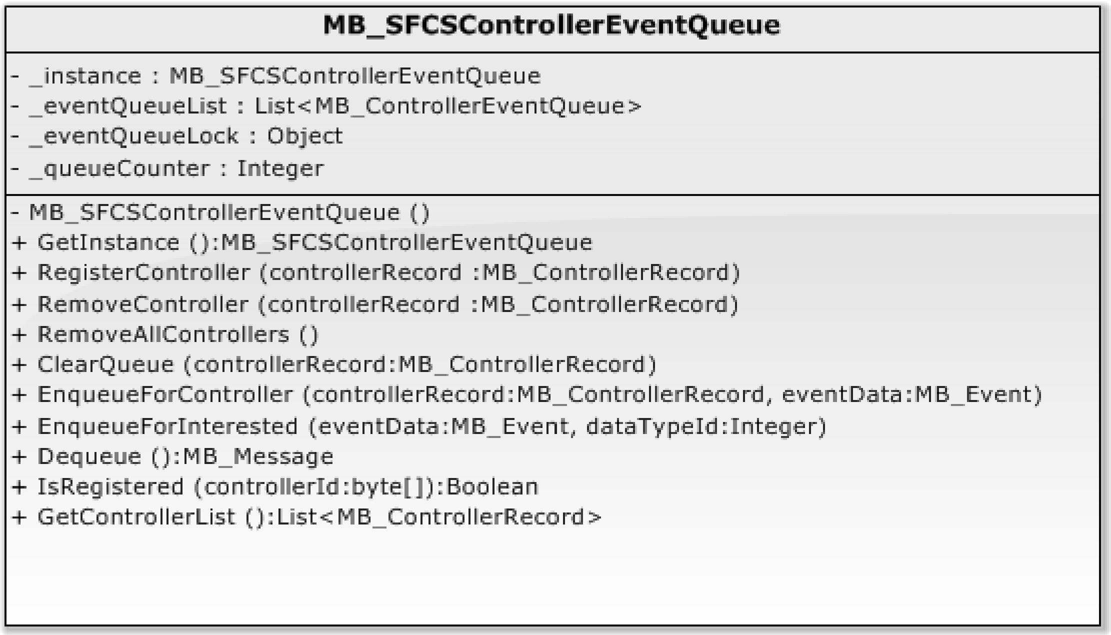

MB_SFCSControllerEventQueue
- _instance : MB_SFCSControllerEventQueue
- _eventQueueList : List<MB_ControllerEventQueue>
- _eventQueueLock : Object
- _queueCounter : Integer
- MB_SFCSControllerEventQueue ()
+ GetInstance ():MB_SFCSControllerEventQueue
+ RegisterController (controllerRecord :MB_ControllerRecord)
+ RemoveController (controllerRecord :MB_ControllerRecord)
+ RemoveAllControllers ()
+ ClearQueue (controllerRecord:MB_ControllerRecord)
+ EnqueueForController (controllerRecord:MB_ControllerRecord, eventData:MB_Event)
+ EnqueueForInterested (eventData:MB_Event, dataTypeId:Integer)
+ Dequeue ():MB_Message
+ IsRegistered (controllerId:byte[]):Boolean
+ GetControllerList ():List<MB_ControllerRecord>

Figure 6-28. Midblocks SFCS controller event queue handler

Incoming events are processed in the Midblocks SFCS by means of event processors, i.e. classes that implement the *MBI_EventProcessor* interface. This scheme allows future versions or particular applications of Midblocks to handle new events easily. If new events are required for some application, or in future versions of Midblocks, it is only necessary to implement the event handling logic in new classes that implement the *MBI_EventProcessor* interface.

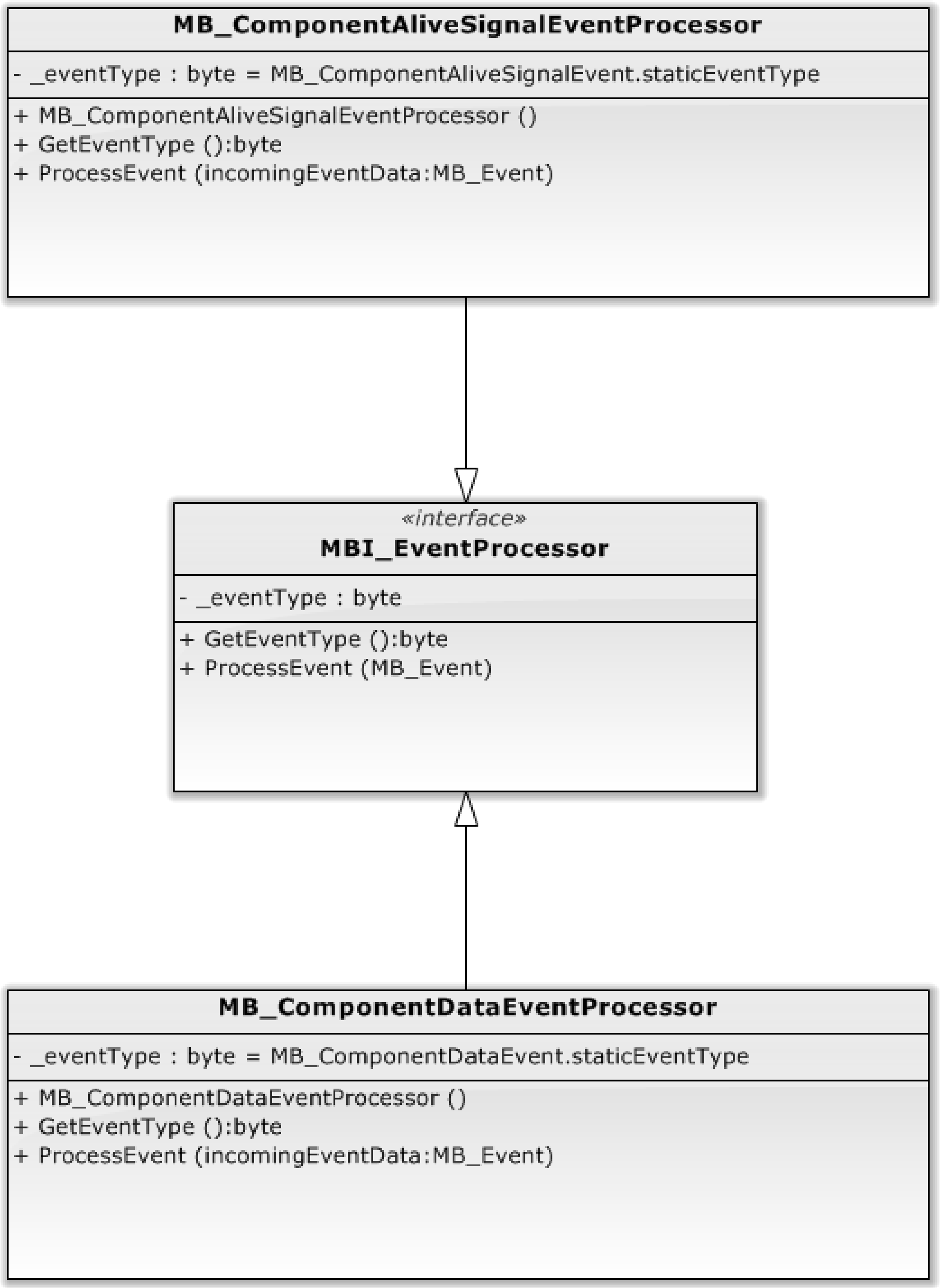

MB_ComponentAliveSignalEventProcessor
- _eventType : byte = MB_ComponentAliveSignalEvent.staticEventType
+ MB_ComponentAliveSignalEventProcessor ()
+ GetEventType ():byte
+ ProcessEvent (incomingEventData:MB_Event)
«interface»
MBI_EventProcessor
- _eventType : byte
+ GetEventType ():byte
+ ProcessEvent (MB_Event)
MB_ComponentDataEventProcessor
- _eventType : byte = MB_ComponentDataEvent.staticEventType
+ MB_ComponentDataEventProcessor ()
+ GetEventType ():byte
+ ProcessEvent (incomingEventData:MB_Event)

Figure 6-29. Midblocks SFCS event processors. Part 1

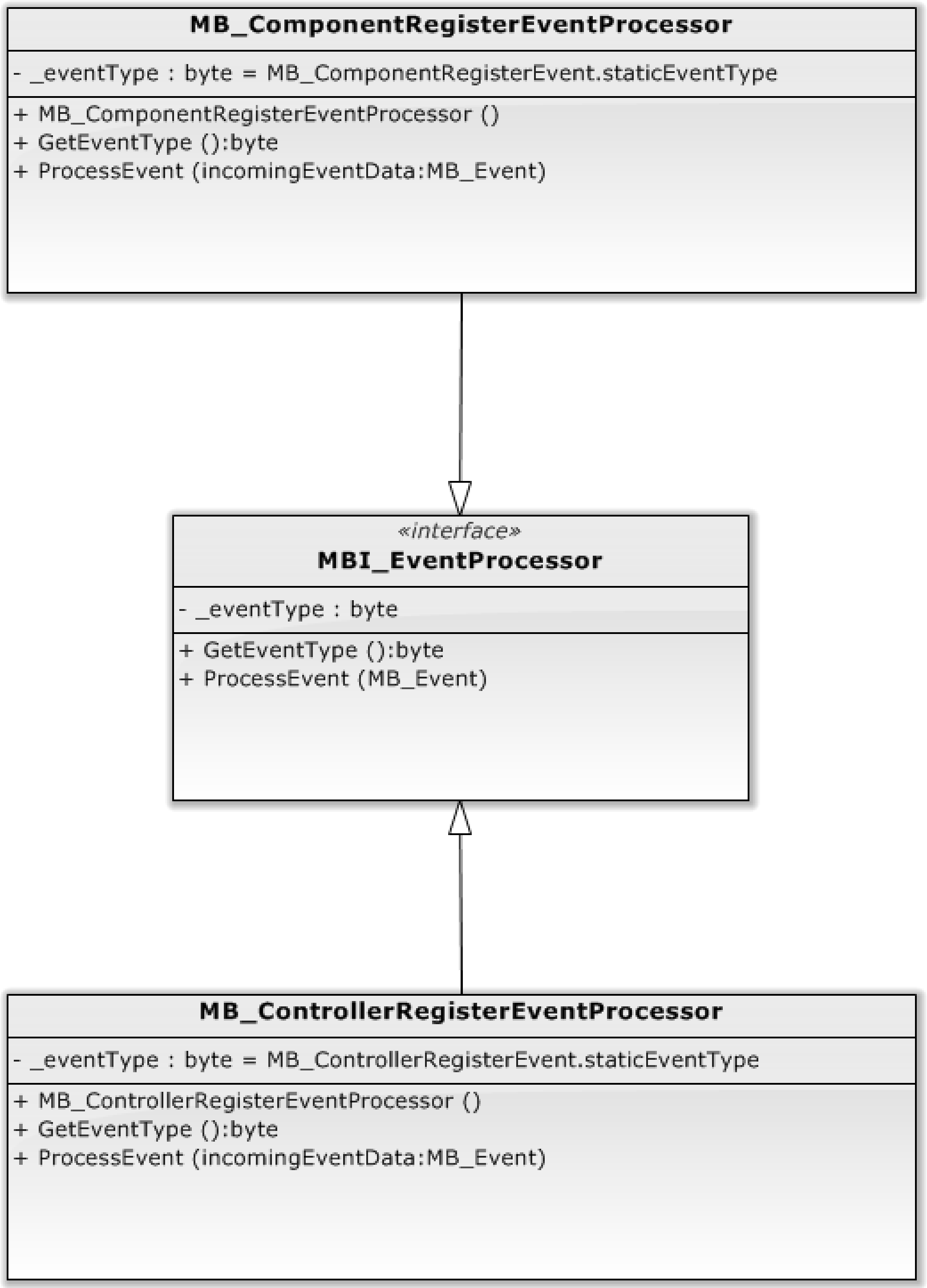

MB_ComponentRegisterEventProcessor
- _eventType : byte = MB_ComponentRegisterEvent.staticEventType
+ MB_ComponentRegisterEventProcessor ()
+ GetEventType ():byte
+ ProcessEvent (incomingEventData:MB_Event)
«interface»
MBI_EventProcessor
- _eventType : byte
+ GetEventType ():byte
+ ProcessEvent (MB_Event)
MB_ControllerRegisterEventProcessor
- _eventType : byte = MB_ControllerRegisterEvent.staticEventType
+ MB_ControllerRegisterEventProcessor ()
+ GetEventType ():byte
+ ProcessEvent (incomingEventData:MB_Event)

Figure 6-30. Midblocks SFCS event processors. Part 2

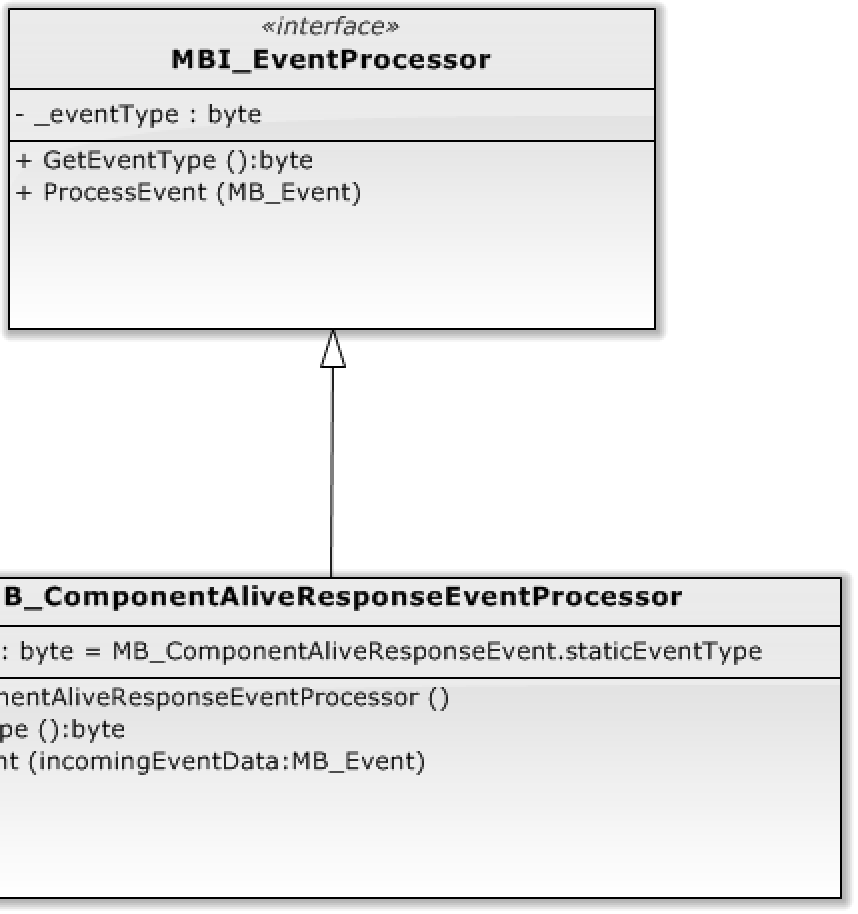

«interface»
MBI_EventProcessor
- _eventType : byte
+ GetEventType ():byte
+ ProcessEvent (MB_Event)
MB_ComponentAliveResponseEventProcessor
- _eventType : byte = MB_ComponentAliveResponseEvent.staticEventType
+ MB_ComponentAliveResponseEventProcessor ()
+ GetEventType ():byte
+ ProcessEvent (incomingEventData:MB_Event)

Figure 6-31. Midblocks SFCS event processors. Part 3

Currently, the following event processors have been implemented (see Figure 6-29, Figure 6-30, and Figure 6-31):

1. *Component Alive Signal Event Processor*. Upon reception of an alive signal event, the component alive signal event processor, represented by the class *MB_ComponentAliveSignalEventProcessor*, updates the component's last activity field.
2. *Component Alive Response Event Processor*. When an alive response event arrives, the alive response event processor, represented by the *MB_ComponentAliveResponseEventProcessor* class, updates the component's last activity field and resets the component's queried field.

3. *Component Data Event Processor.* When a data event arrives, the data event processor, represented by the *MB_ComponentDataEventProcessor*, updates the component's last activity field and places the data event in the event queue of the component's subscribed for this type of data event using the *EnqueueForInterested* method of the component event queue handler.
4. *Component Register Event Processor.* Upon reception of a component register event, the component register event processor performs the component registration procedure described previously. Currently if a component has registered previously, its event queue is cleared, its record is deleted and the component is registered again with the new parameters. This behavior may change in future versions of Midblocks so that when a component re-registers controllers are notified.
5. *Controller Register Event Processor.* When a controller register event is received, the controller register event processor performs the controller registration procedure described previously. As with components, if a controller has registered previously the current procedure is to delete the controller's event queue, delete its record, and perform the registration procedure with the new parameters. This behavior can also change in future versions of Midblocks to notify controllers in case of a re-registration.

6.4.2 Controllers

As mentioned previously, the purpose of the Funblocks Controller module is to supervise component interaction during the System Adaptation and System Failure stages of operation. The main functions performed by the Funblocks Controller (CTRL) are the following:

1. Performs configuration of new modules.
2. Maintains information of the SAs and modules installed in the system and makes this information available to new modules.
3. Notifies modules when new SAs are installed.
4. Performs communication with external components, such as Automated Repair Services.
5. Prevents the installation of modules which could generate conflicts, such as multiple gesture recognition modules in the same area.

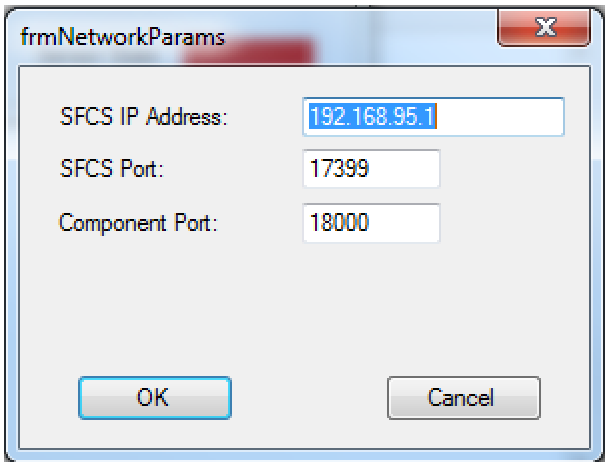
Field	Description
Component Id	A unique local identifier for this component.
Type	1 – SA, 2 – FM, 3 – HCI, 4 – Controller.
Provides	A comma separated integer list of the type of information that the component provides or the type of function that it performs. For example temperature sensor, or ambient light sensor. This identifier is the same one used in the MSAC.
Conflicts	A comma separated list of the types with which this component conflicts.
Area	Area that the component is assigned to.
Additional	Text string that provides additional information such as the units used for the component (if any) or the measurement range of the component (in case of a sensor).

Table 6-1. Controller Catalog Schema

To maintain a detailed description of the current state of the system Funblocks Controllers must maintain a catalog of currently registered components. This is achieved by maintaining an array of component descriptors which have the schema described in Table 6-1.

6.4.3 Components

Currently Funblocks Controllers operate with a fixed IP address when using IEEE 802.3 communications interfaces. This may create conflicts in networks where the address required by the controller is in use by another device or in networks that assign addresses dynamically to devices. As a result Funblocks Components currently require a Network Parameter configuration form (Figure 6-32) to obtain the information necessary to contact the Funblocks controller.

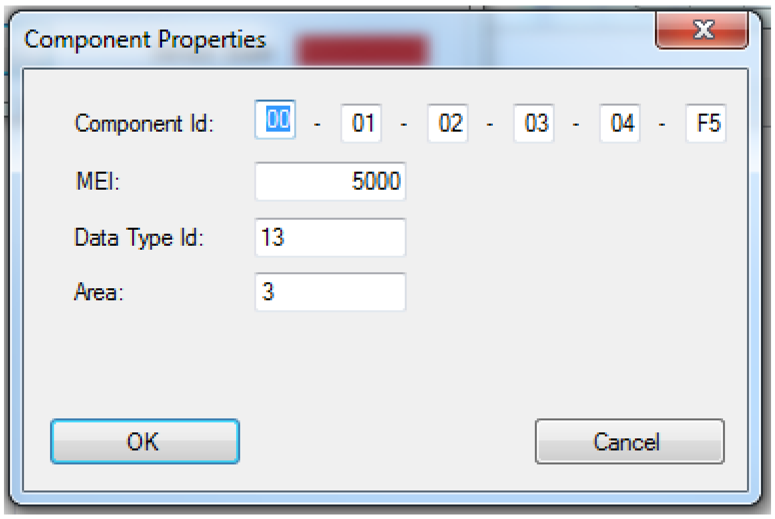


SFCS IP Address: 192.168.95.1
SFCS Port: 17399
Component Port: 18000

Figure 6-32. Funblocks network parameters form

Although this method is adequate for testing purposes it defeats the ease-of-use objective of Funblocks. As a result, a mechanism which allows autonomous discovery of controllers is required in Funblocks. Several service discovery protocols, such as the Simple Service Discovery Protocol (SSDP) or the Service Location Protocol (SLP), have been proposed. However these service discovery protocols have not been developed with resource constrained micro-controllers in mind. Since many of the devices that will be used in Funblocks, particularly sensors, are based on resource constrained micro-controllers it is necessary to evaluate, and probably adapt, these service discovery protocols to insure that they will perform adequately in Funblocks.

Current Funblocks Components employ a simple Component Properties form to configure the component prior to its introduction into a Funblocks-based system (see Figure 6-33).


Component Properties [redacted]

Component Id:	00 - 01 - 02 - 03 - 04 - F5
MEI:	5000
Data Type Id:	13
Area:	3

Figure 6-33. Component Properties form

There is no algorithm used at this moment for the assignment of component identification numbers, the component id is assigned sequentially/randomly. The benefit of encoding information, such as the assigned area, should be evaluated and compared with the simplicity of assigning component ids sequentially. With the current design of Funblocks the use of a component id assignment algorithm vs. a sequential component assignment scheme does not provide a major benefit. This is due to the single controller architecture currently being employed in Funblocks. However this single controller architecture is expected to change in future version of Funblocks to improve the robustness of a Funblocks-based system against controller or communications link failures. In a multiple-controller scenario encoding additional information in the component id may be useful to provide self-organizing and self-healing capabilities to Funblocks-based systems.

Data type ids provide a logical link between the components of a Funblocks-based system. To promote component reuse between Funblocks-based research projects it is important to maintain consistent data type ids between applications.

6.4.4 Module and Sensor Actuator Catalog (MSAC)

As described in section 4.7 Funblocks makes use of a Module and Sensor/ Actuator Catalog (MSAC) and the Function Module Repository (FMR). The Module and Sensor/ Actuator Catalog stores MSAD records which describe the components available to a specific Funblocks based system. The MSAD lists the functions provided by a specific module or SA, the dependencies required by the component and the conflicts this component may have with other components. The MSAD also provides a default configuration for the components of an AmI system. This assists end-users and system integrators in the configuration of an AmI system.

The current test implementation of Funblocks uses the schema described in Table 6-2 for the MSAD records.

Presently, in the test implementation of Funblocks, if an FM which requires data sources not available in the system is inserted, the MSAC is searched to find the first MSAD that describes a component which can provide the required data. This allows for a very rudimentary form of service composition. However, for Funblocks to be an effective tool in the integration of Ambient Intelligence systems, it is necessary to develop algorithms which can provide goal-based system composition. A first step in the development of said algorithms will be to develop an AmI system description language which can be used to describe the services required from the system by an end user.

Field	Description
Record id	A unique identifier for this MSAD record.
Name	Name (human readable text string) of the component.
Type	An integer describing the type of component: 0-SA, 1-FM, 2-Ctlr, 3-HCI.
Provides	Comma separated list of data type ids provided by this component.
Required SAs	Comma separated list of SA data type ids required by this component.
Required FMs	Comma separated list of FM data type ids required by this component.
Conflicts SAs	Comma separated list of SA data type ids that the data provided by this component conflicts with.
Conflicts FMs	Comma separated list of FM data type ids that the data provided by this component conflicts with.
Latest version	Latest version of the component (applies to FM only).
Platforms	Comma separated list of the platforms for which this component is available (applies to FM only).
MEI	Default MEI for this component (applies to SAs and FMs only).
Description	A human readable text description of the component.

Table 6-2. MSAD Schema

To provide support for multiple platforms, a second set of records is used to obtain an URI to the correct package from a FMR. The schema for this set of records is described in Table 6-3.

This is a description of the current state of the development of the Modular Framework for Ambient Intelligence systems and these are some of the challenges currently being worked on in the development of the Funblocks framework. It is expected that the solutions to some of these problems will give rise to further research articles.

Field	**Description**
Id	Unique identifier.
Platform	Platform for which this FM is destined.
Version	Version of the FM.
Type Id	FM type id.
URI	URI from which the Function Module can be downloaded.

Table 6-3. FMR Schema

7 Conclusions and Future Work

Funblocks provides an extremely flexible framework for the development of Ambient Intelligence systems. Contrary to other AmI frameworks, Funblocks makes no assumption about the types of environments or devices that can be handled in an AmI system. As a result Funblocks can be used transparently with existing systems such as home/building automation systems, security systems, etc.

By considering AmI systems as a type of distributed control systems, and by making use of the IEC 61499 function block abstraction, Funblocks can be used to develop AmI systems which can be customized to a wide range of different usage scenarios. Additionally the use of the function block abstraction allows AmI systems developed with the Funblocks framework to be easily modified by users with little or no technical experience. Funblocks is also language and hardware independent, which means that the framework can be used with a vast variety of devices ranging from 8 bit microcontrollers to servers.

The use of a middleware with supervising capabilities allows AmI systems developed with Funblocks to monitor the operational state of their components and, whenever possible, to take corrective actions in case of component failure. Since the middleware makes use of a publish/subscribe paradigm with a store and forward entity for component interaction it can easily be reused in other event-based systems which require component monitoring.

To provide an idea of the benefits of using Funblocks in the development of an AmI system in the following section we provide a brief comparison with other frameworks.

7.1 Framework Features Comparison

Ambient Intelligence is a currently a field of intense research but as of yet there is not an AmI systems development framework that enjoys widespread use. A possible reason for the lack of a widely used framework may be that the majority of current proposals tend to be rather complex, which means that a research or development team has to invest significant effort in order to be able to use the framework. Ease of use has been one of our main goals during the development of Funblocks.

The problems that have been approached by the developers of different frameworks are varied, and as a result the characteristics of the frameworks are also diverse. In [143] Becker describes a set of system and service qualities that an Ambient Assisted Living system should possess. This set of system and service qualities are valid in general for any Ambient Intelligence system. The system and service qualities that should be achieved by AmI systems are the following:

- *Affordability:* The services and the required technical installation and resources must be affordable.
- *Usability and User Experience:* If the system requires some explicit interaction with the user, the system should perform the interaction accordingly with the user´s capabilities and the user should have a positive experience during such interaction.
- *Suitability.* The services provided by the system must meet the demands of the user and the user must benefit from the services, otherwise acceptance of the system will decline over time.
- *Dependability.* The system must be robust against misuse, errors, hardware component crashes and shortage of resources. Furthermore the system must guarantee a minimum of privacy and security for the users. Finally the system must be safe to use and not pose a risk to its users.
- *Adaptability.* Systems must be able to adapt themselves at runtime. Systems must monitor themselves, users and the environment and based on the information gathered perform one of the following:
- *Self-configuration.* Integrate dynamically new components and remove existing components not required any more.
- *Self-healing.* Denotes the ability of detecting component problems and taking appropriate measures.
- *Self-optimization.* The ability of the system to adapt its algorithmic behavior to application requirements changes.
- *Self-protection.* Denotes the ability of the system to protect itself against misuse.
- *Extensibility.* In order to adapt to new demands the system must be able to extend itself with new devices and services.

- *Resource Efficiency.* Available resources such as processors, memory, communications bandwidth and energy have to be used as efficiently as possible.
- *Heterogeneity.* The system will typically consist of several subsystems provided by different manufacturers.

	Amigo	**MPOWER**	**SOPRANO**	**universAAL**	**Funblocks**
Purpose	Dynamic Interoperability	AAL	AAL	AAL	General Purpose
Architecture	SOA	SOA	SOA	SOA	Function Block
Platform	.NET/OSGi	Web Services, HTTP, SOAP	OSGi	OSGi	Platform Independent
Component Supervision	No	No	Yes	No	Yes
Module Repository	No	No	No	Yes	Yes
Sensor Handling	Through wrappers	Network enabled	As OSGi bundles	Networked enabled	Native prot. support

Table 7-1. Framework Features.

In table 7.1 we provide a brief list of some of the features of the frameworks described previously along with the corresponding features of Funblocks. The criteria used for the comparison are the following:

- *Purpose.* As mentioned before frameworks tend to be targeted toward the solution of specific problems such as AAL.
- *Architecture.* Current frameworks are based on different paradigms being SOA the most popular. The architecture selected for the framework will have a strong impact on

its system and service qualities. Particularly affordability, adaptivity, extensibility, and heterogeneity are influenced by the choice of architecture.

- *Platform.* The platform selected for the implementation of the system will have a direct impact on the affordability and heterogeneity of the system and is therefore an important consideration in the selection of a framework.
- *Component Supervision.* Component supervision is an important feature which helps improve the dependability of a system.
- *Module Repository.* The use of a Module Repository allows end user customization of the system hence improving usability and user experience.
- *Sensor Handling.* AmI systems have to interact with a vast array of existing sensor technologies. The choice of the sensor handling scheme in the framework will have a direct impact on the affordability and heterogeneity of the system.

7.2 Future Work

There is still much work to do developing Funblocks. The current design of Funblocks does not allow the use of redundancy in components such as the controller and the message queue. As a result these components represent single points of failure. The use of redundant controllers and redundant message queues must be addressed in order to provide a higher degree of reliability. Once the issue of redundancy has been addressed self-healing and service composition capabilities must be introduced in Funblocks. This is particularly important for wide area systems where communications links could fail leading to a fragmented system. It would be desirable for each fragment to reconfigure itself into an independent AmI system perhaps with limited capabilities.

Midblocks is currently the most developed component of the Modular Framework for Ambient Intelligence systems. The Midblocks middleware has applications outside of the Funblocks framework. It will be important for the development of Midblocks to test it in applications other than Funblocks.

One such area in which Midblocks could be tested is in the field of mobile-based fall detection systems.

Everybody has experienced an unwanted fall, whether in childhood while training to walk, or occasionally in adulthood. The fall experience is thus well-known to everybody. However a fall is difficult to describe precisely which makes it hard to devise a means for its detection. Falls are commonly defined as "inadvertently coming to rest on the ground, floor or other lower level, excluding intentional change in position to rest in furniture, wall or other objects".

Globally, falls are a major public health problem. An estimated 424 000 fatal falls occur each year, making it the second leading cause of unintentional injury death, after road traffic injuries. Falls account for 40% of all injury deaths. Rates vary depending on the country and the studied population. Fall fatality rate for people aged 65 and older in the United States of America (USA) is 36.8 per 100 000 population (46.2 for men and 31.1 for women) whereas in Canada mortality rate for the same age group is 9.4 per 10 000 population. Mortality rate for people age 50 and older in Finland is 55.4 for men and 43.1 for women per 100 000 population. Over 80% of fall-related fatalities occur in low- and middle-income countries. In Mexico, falls account for 30% of deaths of people aged 65 and older.

According to the Centers for Disease Control and Prevention of the US (CDC) and the World Health Organization (WHO), unintentional falls are a common occurrence among older adults, affecting approximately 28 to 35% of adults over the age of 65. Older people who have suffered a fall are at increased risk of falling again. A study of 325 community-dwelling persons who had fallen found that 57% experienced at least one fall in a 12-month follow-up period and 31% had two or more falls. 20 to 30% of people who fall suffer moderate to severe injuries such as lacerations, hip fractures, or head traumas. Most fractures among older adults are caused by falls. The most common are fractures of the spine, hip, forearm, leg, ankle, pelvis, upper arm, and hand.

One of the serious consequences of a fall is the "long-lie" which is remaining on the ground for one hour or more. Half of those who lie on the floor for an hour or longer die within 6 months, even if there is no direct injury from the fall. Long-lies are not uncommon. More than 20% of the patients admitted to a hospital because of a fall had been on the ground for an hour or more. Up to 47% of non-injured fallers are unable to get up off the floor without assistance. Detection of a fall might reduce the consequences of a long lie by reducing the time between the fall and the arrival of medical attention. If an elderly person living alone experiences a fall at home, he or she may not be able to get to the phone or press an alarm button due to sustained injuries or loss of consciousness. Moreover, some elderly people do not activate their personal emergency response systems, even when they have the ability to do so.

Smartphones present a mature hardware and software platform for the development of fall detection systems. Smartphone based fall detection retains the advantages of the wearable fall detection methods and removes the main disadvantages of such methods. Since smartphones are commonly used devices which have multiple functions most users are used to carry their smartphones with them. As a result the odds of a person forgetting his fall detection device are reduced. For this same reason smartphones can also be considered as being less intrusive than dedicated wearable fall detection devices. Additionally smartphones are used by people of all ages and are therefore not associated with the fragility of the old person which can also promote the acceptance of the fall detection device.

An application of Midblocks would be in the development of a fall detection system which supervises a set of fall detection enabled smartphones. Since the welfare of the user is dependent on the correct operation of the smartphone, it is important that caregivers be notified in case of a failure of the fall detection enabled smartphone.

An early stage block diagram of such a supervised smartphone-based fall detection system, which we call Superfall is shown in Figure 7-1. Superfall system block diagram

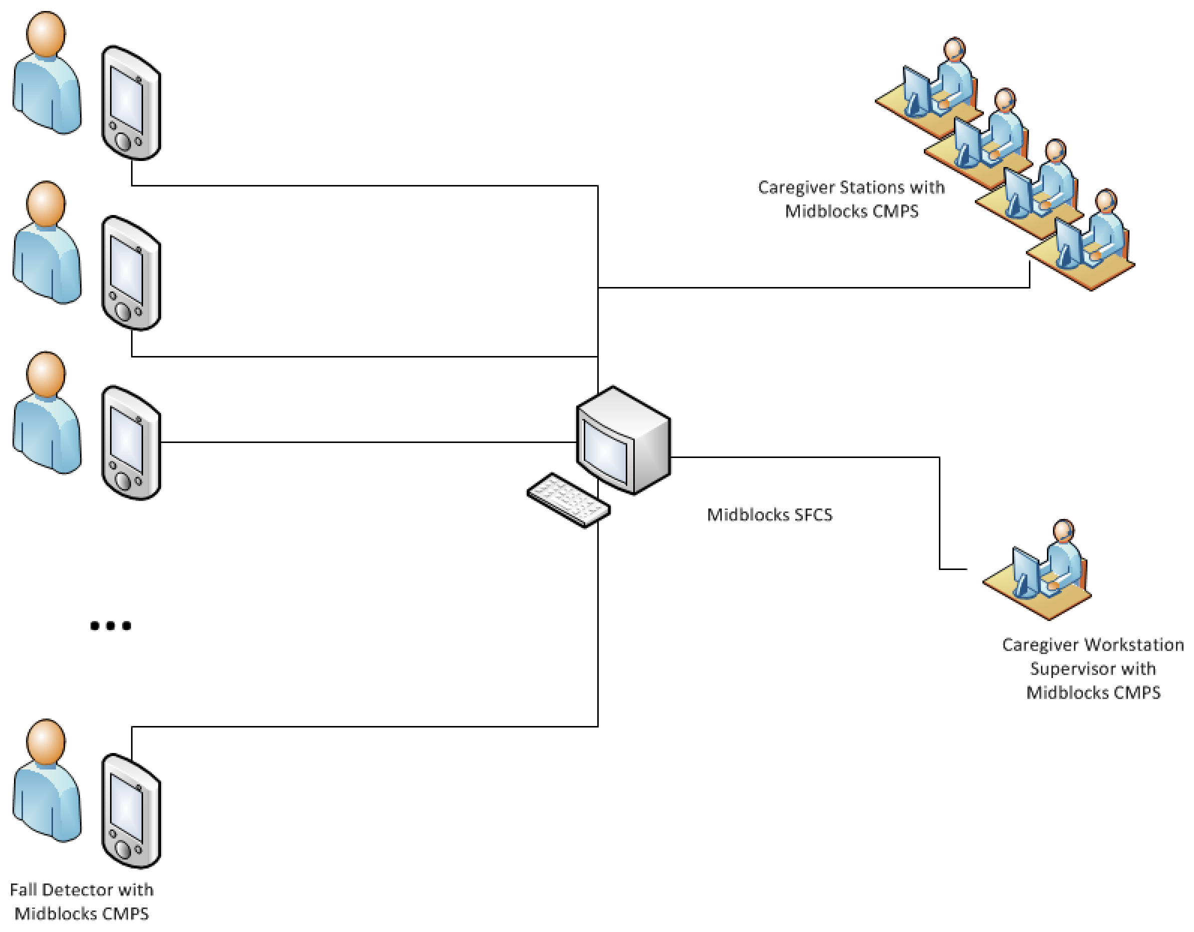

Caregiver Stations with Midblocks CMPS
Midblocks SFCS
Caregiver Workstation Supervisor with Midblocks CMPS
...
Fall Detector with Midblocks CMPS

Figure 7-1. Superfall system block diagram

8 References

[1] ISTAG, “Ambient Intelligence. From Vision to Reality,” ISTAG, 2003.

[2] ISTAG, “Strategic Orientations and Priorities for IST in FP6,” ISTAG, 2002.

[3] M. Lindwer, D. Marculescu, T. Basten, R. Zimmennann, R. Marculescu, S. Jung, and E. Cantatore, “Ambient Intelligence Visions and Achievements: Linking Abstract Ideas to Real-World Concepts,” in *Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 10–15.

[4] Y. Punie, “The Future of Ambient Intelligence in Europe: The Need for More Everyday Life,” *Media, Technology and Everyday Life in Europe: From Information to Communication. London*, pp. 159–177, 2005.

[5] J. C. Augusto, H. Nakashima, and H. Aghajan, “Ambient Intelligence and Smart Environments: A State of the Art,” in *Handbook of Ambient Intelligence and Smart Environments*, H. Nakashima, H. Aghajan, and J. C. Augusto, Eds. Springer, 2010, pp. 3–31.

[6] D. J. Cook and S. K. Das, “How Smart Are Our Environments? An Updated Look at the State of the Art,” *Pervasive and Mobile Computing*, vol. 3, no. 2, pp. 53–73, 2007.

[7] F. Sadri, “Ambient Intelligence: A Survey,” *ACM Computing Surveys*, vol. 43, no. 4, 2011.

[8] J. C. Augusto, “Past, Present and Future of Ambient Intelligence and Smart Environments,” in *Agents and Artificial Intelligence*, Springer, 2010, pp. 3–15.

[9] A. S. Taylor, R. Harper, L. Swan, S. Izadi, A. Sellen, and M. Perry, “Homes That Make Us Smart,” *Personal and Ubiquitous Computing*, vol. 11, no. 5, pp. 383–393, 2007.

[10] UNFPA, “The Day of 6 Billion. http://www.unfpa.org/6billion/index.htm (Accessed December 22, 2013).” .

[11] UNFPA, “Day of 7 Billion, http://www.unfpa.org/public/home/news/events/pid/8533 (Accessed December 22, 2013).” .

[12] “Instituto Nacional de Estadistica y Geografia, http://www.inegi.org.mx/default.aspx (Accessed December 22, 2013).” .

[13] “United States Census Bureau - World Population, http://www.census.gov/population/international/data/worldpop/table_population.php (Accessed December 22, 2013).” .

[14] UN, “World Population Prospects. The 2012 Revision,” United Nations Department of Economic and Social Affair. Population Division, 2013.

[15] “US Energy Information Administration - International Energy Outlook 2013, http://www.eia.gov/forecasts/ieo/world.cfm (Accessed December 22, 2014).” .

[16] M. Jahn, M. Jentsch, C. R. Prause, F. Pramudianto, A. Al-Akkad, and R. Reiners, “The Energy Aware Smart Home,” in *Future Information Technology (FutureTech), 2010 5th International Conference on*, 2010, pp. 1–8.

[17] R. Eddington, “The Eddington Transport Study,” 2006.

[18] “TTI’s 2012 Urban Transportation Report,” Texas A&M Transportation Institute, Texas A&M University, 2012.

[19] "WHO Fact Sheet No. 358. Road Traffic Injuries," World Health Organization, 2013.

[20] ISTAG, "Scenarios for Ambient Intelligence in 2010," ISTAG, 2001.

[21] "10 Facts on Ageing and the Life Course," World Health Organization, 2012.

[22] "A Grass-Roots Effort to Grow Old at Home, The New York Times." Aug-2007.

[23] D. Redfoot, L. Feinberg, and A. Houser, "The Aging of the Baby Boom and the Growing Care Gap. A Look at Future Declines in the Availability of Family Caregivers," AARP Public Policy Institute, 2013.

[24] L. Leahu, P. Sengers, and M. Mateas, "Interactionist AI and the Promise of Ubicomp, or, How to Put Your Box in the World Without Putting the World in Your Box," in *Proceedings of the 10th international conference on Ubiquitous computing*, 2008, pp. 134–143.

[25] P. Holroyd, P. Watten, and P. Newbury, "Why Is My Home Not Smart?," in *Aging Friendly Technology for Health and Independence*, Springer, 2010, pp. 53–59.

[26] S. Mennicken and E. M. Huang, "Hacking the Natural Habitat. An In-the-Wild Study of Smart Homes, Their Development, and the People Who Live in Them," in *Pervasive Computing*, Springer, 2012, pp. 143–160.

[27] R. José, H. Rodrigues, and N. Otero, "Ambient Intelligence: Beyond the Inspiring Vision.," *Journal of Universal Computer Science*, vol. 16, no. 12, pp. 1480–1499, 2010.

[28] D. Sampaio, L. P. Reis, and R. Rodrigues, "A Survey on Ambient Intelligence Projects," in *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*, 2012.

[29] J. Dooley, M. Ball, and M. R. Al-Mulla, "Beyond Four Walls: Towards Large-Scale Intelligent Environments," in *Workshop Proceedings of the 8th International Conference on Intelligent Environments*, 2012.

[30] M. Banzi, *Getting Started with Arduino, 2nd Ed.* O'Reilly, 2011.

[31] M. Evans, J. Noble, and J. Hochenbaum, *Arduino in Action*. Manning, 2013.

[32] S. Wang, *Intelligent Buildings and Building Automation*. Spon Press, 2010.

[33] H. Merz, T. Hansemann, and C. Hübner, *Building Automation: Communication Systems with EIB KNX, LON und BACnet*. Springer, 2009.

[34] W. Kastner, G. Neugschwandtner, S. Soucek, and H. Newmann, "Communication Systems for Building Automation and Control," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1178–1203, 2005.

[35] W. Kastner, F. Praus, G. Neugschwandtner, and W. Granzer, "KNX," in *Industrial Electronics Handbook, Industrial Communication Systems*, 2nd ed., vol. 4, B. M. Wilamowski and J. D. Irwin, Eds. CRC Press, 2011, pp. 42.1 – 42.14.

[36] M. Janse, P. Vink, and N. Georgantas, "Amigo Architecture: Service Oriented Architecture for Intelligent Future In-Home Networks," in *Constructing Ambient Intelligence*, Springer, 2008, pp. 371–378.

[37] C. Magerkurth, R. Etter, M. Janse, J. Kela, O. Kocsis, and F. Ramparany, "An Intelligent User Service Architecture for Networked Home Environments," in *Intelligent Environments, 2006. IE 06. 2nd IET International Conference on*, 2006, pp. 361–370.

[38] J. Schmalenstroeer, V. Leutnant, and R. Haeb-Umbach, "Amigo Context Management Service with Applications in Ambient Communication Scenarios," in *Constructing Ambient Intelligence*, Springer, 2008.

[39] G. Thomson, D. Sacchetti, Y.-D. Bromberg, J. Parra, N. Georgantas, and V. Issarny, “Amigo Interoperability Framework: Dynamically Integrating Heterogeneous Devices and Services,” in *Constructing Ambient Intelligence*, Springer, 2008, pp. 421–425.

[40] P. Vink, N. Georgantas, and C. Magerkurth, “Amigo Architecture. http://www.hitech-projects.com/euprojects/amigo/Seminar/amigo-handouts-openday-nov0/08Handout%20-%20Architecture-mj.pdf (Accessed: December 22, 2013).” .

[41] “Amigo. Ambient Intelligence for the Networked Home Environment. http://www.hitech-projects.com/euprojects/amigo/ (Accessed: December 22, 2013).” .

[42] A. Brush, E. Filippov, D. Huang, J. Jung, R. Mahajan, F. Martinez, K. Mazhar, A. Phanishayee, A. Samuel, J. Scott, and others, “Lab of Things: A Platform for Conducting Studies with Connected Devices in Multiple Homes,” in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, 2013.

[43] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and V. Bahl, “An Operating System for the Home,” in *Proceedings NSDI*, 2012.

[44] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and V. Bahl, “The Home Needs an Operating System (and an App Store),” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[45] W. K. Edwards, R. E. Grinter, R. Mahajan, and D. Wetherall, “Advancing the State of Home Networking,” *Communications of the ACM*, vol. 54, no. 6, pp. 62–71, 2011.

[46] T. Gupta, A. Phanishayee, J. Jung, and R. Mahajan, “Towards a Storage System for Connected Homes,” in *Workshop on Large-Scale Distributed Systems and Middleware*, 2013.

[47] J. Scott, A. J. B. Brush, and R. Mahajan, “Augmenting Homes with Custom Devices Using .NET Gadgeteer and HomeOS,” in *Proceedings of BuildSys 2012*, 2012.

[48] B. Ur, J. Jung, and S. Schechter, “The Current State of Access Control for Smart Devices in Homes,” in *Workshop on Home Usable Privacy and Security (HUPS)*, 2013.

[49] A. Grguric, S. Desic, M. Mosmondor, I. Benc, J. Krizanic, and P. Lazarevski, “Proof-of-Concept Applications for Validation of ICT Services for Elderly Care,” in *Proceedings of the 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2010)*, 2010, pp. 355–359.

[50] A. Grgurić, I. Benc, S. Dešić, M. Mošmondor, J. Križanić, and P. Lazarevski, “Designing User Interfaces for Elderly: A Case Study in Applicability of Thin vs. Fat Clients,” in *e-Health Networking Applications and Services (Healthcom), 2010 12th IEEE International Conference on*, 2010.

[51] M. Mikalsen, S. Hanke, T. Fuxreiter, S. Walderhaug, L. W. Wienhofen, and N. Trondheim, “Interoperability Services in the MPOWER Ambient Assisted Living Platform,” in *Medical Informatics Europe (MIE) Conference*, 2009.

[52] A. Pitsillides, E. Themistokleous, G. Samaras, S. Walderhaug, and O. Winnem, “Overview of MPOWER: Middleware Platform for the Cognitively Impaired and

Elderly," in *Proceedings of IST-Africa 2007 Conference & Exhibition, Maputo, Mosambique*, 2007.

[53] S. Walderhaug, E. Stav, and M. Mikalsen, "The MPOWER Tool Chain - Enabling Rapid Development of Standards-based and Interoperable Homecare Applications," in *Proceedings of Norsk Informatikk Konferanse (NIK 2007)*, 2007.

[54] M. Klein, A. Schmidt, and R. Lauer, "Ontology-Centred Design of an Ambient Middleware for Assisted Living: The Case of SOPRANO," in *Towards Ambient Intelligence: Methods for Cooperating Ensembles in Ubiquitous Environments (AIM-CU), 30th Annual German Conference on Artificial Intelligence (KI 2007), Osnabrück*, 2007.

[55] G. Virone and A. Sixsmith, "Monitoring Activity Patterns and Trends of Older Adults," in *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE*, 2008.

[56] P. Wolf, A. Schmidt, and M. Klein, "Applying Semantic Technologies for Context-Aware AAL Services: What we can learn from SOPRANO," in *Workshop on Applications of Semantic Technologies 09, Informatik 2009*, 2009.

[57] P. Wolf, A. Schmidt, and M. Klein, "SOPRANO - An Extensible, Open AAL Platform for Elderly People Based on Semantical Contracts," in *3rd Workshop on Artificial Intelligence Techniques for Ambient Intelligence (AITAmI'08), 18th European Conference on Artificial Intelligence (ECAI 08)*, 2008.

[58] S. Hanke, C. Mayer, O. Hoeftberger, H. Boos, R. Wichert, M.-R. Tazari, P. Wolf, and F. Furfari, "universAAL - An Open and Consolidated AAL Platform," in *Ambient Assisted Living*, Springer, 2011, pp. 127–140.

[59] M.-R. Tazari, F. Furfari, Á. Fides-Valero, S. Hanke, O. Höftberger, D. Kehagias, M. Mosmondor, R. Wichert, and P. Wolf, "The universAAL Reference Model for AAL," in *Handbook of Ambient Assisted Living*, J. C. Augusto, M. Huch, A. Kameas, J. Maitland, P. McCullagh, A. Sixsmith, and R. Wichert, Eds. IOS Press, 2012, pp. 610–625.

[60] "UNIVERsal open platform and reference Specification for Ambient Assisted Living. http://universaal.org/ (Accessed December 22, 2013)." .

[61] "D1.3. The universAAL Reference Architecture. http://universaal.org/images/stories/deliverables/D1.3-E.pdf (Accessed December 22, 2013)," 2013.

[62] "D2.4. universAAL Developers Handbook. http://universaal.org/images/stories/deliverables/D2.4-D.pdf (Accessed December 22, 2013)," 2013.

[63] J. Chong, S. See, L. L. Seah, S. Ling Koh, and Y. Theng, "Ubiquitous Computing. History, Development and Future Scenarios," in *Ubiquitous Computing: Design, Implementation, and Usability*, Y.-L. Theng and H. B. Duh, Eds. IGI Global, 2008, pp. 1–8.

[64] E. Loureiro, G. Ferreira, H. Almeida, and A. Perkusich, "Pervasive Computing: What is it Anyway?," in *Ubiquitous Computing: Design, Implementation, and Usability*, Y.-L. Theng and H. B. Duh, Eds. IGI Global, 2008, pp. 9–36.

[65] M. Weiser, "Some Computer Science Issues in Ubiquitous Computing," *Communications of the ACM*, vol. 36, no. 7, pp. 75–84, 1993.

[66] M. Weiser, "Hot Topics - Ubiquitous Computing," *Computer*, vol. 26, no. 10, pp. 71–72, 1993.

[67] M. Weiser, “The Computer for the 21st Century,” *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[68] F. Karray, M. Alemzadeh, J. A. Saleh, and M. N. Arab, “Human-Computer Interaction: Overview on State of the Art,” *International Journal on Smart Sensing and Intelligent Systems*, vol. 1, no. 1, pp. 137–159, 2008.

[69] M. Baldauf, S. Dustdar, and F. Rosenberg, “A Survey on Context-Aware Systems,” *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[70] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, “A Survey of Context Modelling and Reasoning Techniques,” *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.

[71] K. Henricksen and J. Indulska, “Developing Context-Aware Pervasive Computing Applications: Models and Approach,” *Pervasive and Mobile Computing*, vol. 2, no. 1, pp. 37–64, 2006.

[72] A. Greenfield, *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders, 2010.

[73] S. Poslad, *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Wiley, 2009.

[74] C. Ramos, “Ambient Intelligence - A State of the Art from Artificial Intelligence Perspective,” in *Progress in Artificial Intelligence*, 2007, pp. 285–295.

[75] L. Roalter, A. Moller, S. Diewald, and M. Kranz, “Developing Intelligent Environments: A Development Tool Chain for Creation, Testing and Simulation of Smart and Intelligent Environments,” in *Intelligent Environments (IE), 2011 7th International Conference on*, 2011.

[76] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a Better Understanding of Context and Context-Awareness,” in *Handheld and Ubiquitous Computing*, 1999, pp. 304–307.

[77] A. K. Dey, “Understanding and Using Context,” *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001.

[78] J. C. Augusto, “Ambient Intelligence: Basic Concepts and Applications,” in *Software and Data Technologies*, Springer, 2008.

[79] J. C. Augusto, “Ambient Intelligence: The Confluence of Ubiquitous/Pervasive Computing and Artificial Intelligence,” in *Intelligent Computing Everywhere*, Springer, 2007.

[80] C. Ramos, J. C. Augusto, and D. Shapiro, “Ambient Intelligence - The Next Step for Artificial Intelligence,” *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 15–18, 2008.

[81] R. Baquero, J. G. Rodriguez, S. Mendoza, and D. Dominique, “Towards a Modular Scheme for the Integration of Ambient Intelligence Systems,” in *5th International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI 2011)*, 2011.

[82] R. Baquero, J. Rodrıguez, S. Mendoza, D. Decouchant, and A. P. M. Papis, “FunBlocks. A Modular Framework for AmI System Development,” *Sensors*, vol. 12, no. 8, pp. 10259–10291, 2012.

[83] D. J. Cook, J. C. Augusto, and V. R. Jakkula, “Ambient Intelligence: Technologies, Applications, and Opportunities,” *Pervasive and Mobile Computing*, vol. 5, no. 4, pp. 277–298, 2009.

[84] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier, "The Internet of Things for Ambient Assisted Living," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, 2010, pp. 804–809.

[85] M. J. O'Grady, C. Muldoon, M. Dragone, R. Tynan, and G. M. O'Hare, "Towards Evolutionary Ambient Assisted Living Systems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 1, no. 1, pp. 15–29, 2010.

[86] H. Sun, V. De Florio, N. Gui, and C. Blondia, "The Missing Ones: Key Ingredients Towards Effective Ambient Assisted Living Systems," *Journal of Ambient Intelligence and Smart Environments*, vol. 2, no. 2, pp. 109–120, 2010.

[87] T. Kleinberger, M. Becker, E. Ras, A. Holzinger, and P. Müller, "Ambient Intelligence in Assisted Living: Enable Elderly People to Handle Future Interfaces," in *Universal Access in Human-Computer Interaction. Ambient Interaction*, Springer, 2007, pp. 103–112.

[88] H. Sun, V. De Florio, N. Gui, and C. Blondia, "Promises and Challenges of Ambient Assisted Living Systems," in *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, 2009, pp. 1201–1207.

[89] P. Rashidi and A. Mihailidis, "A Survey on Ambient-Assisted Living Tools for Older Adults," *IEEE Journal of Biomedical and Health Informatics*, vol. 17, no. 3, pp. 579–590, 2013.

[90] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2008.

[91] N. Santoro, *Design and Analysis of Distributed Algorithms*. Wiley, 2006.

[92] A. Puder, *Distributed Systems Architecture: A Middleware Approach*. Elsevier, 2006.

[93] J. Fraden, *Handbook of Modern Sensors*, 4th ed. Springer, 2010.

[94] J. H. Huijsing, "Smart Sensor Systems: Why? Where? How?," in *Smart Sensor Systems*, C. M. Meijer, Ed. Wiley, 2008, pp. 1–21.

[95] I. R. Sinclair, *Sensors and Transducers*. Newnes, 2000.

[96] J. S. Wilson, *Sensor Technology Handbook*. Elsevier, 2004.

[97] T. Xie and B. Wilamowski, "Sensors," in *The Industrial Electronics Handbook. Control and Mechatronics, 2nd Ed.*, 2nd Ed., B. Wilamowski and J. D. Irwin, Eds. CRC Press, 2011.

[98] J. Burroughs, "AN-236. X-10 Home Automation Using the PIC16F877A," *http://ww1.microchip.com/downloads/en/AppNotes/00236a.pdf (Accessed December 22, 2013)*, 2002.

[99] "Standard and Extended X-10 Code Protocol," *ftp://ftp.x10.com/pub/manuals/xtdcode.pdf*.

[100] "Understanding EIA-485 Networks," *http://www.ccontrols.com/pdf/ExtV1N1.pdf (Accessed December 22, 2013)*, 1999.

[101] T. Kugelstadt, "The RS-485 Design Guide," *http://www.ti.com/lit/an/slla272b/slla272b.pdf (Accessed December 22, 2013)*, 2008.

[102] *IEEE Std 802-2001. IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture*. IEEE Computer Society, 2002.

[103] *ISO/IEC 7498-1. Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. International Organization for Standarization (ISO)/ International Electrotechnical Comission, 1996.

[104] *IEEE Std 802.2, 1998 Edition(R2003). IEEE Standard for Information Technology. Telecommunications and Information Exchange Between Systems. Local and Metropolitan Area Networks. Specific Requirements Part 2: Logical Link Control*. LAN/MAN Standards Committee of the IEEE Computer Society, 1998.

[105] *IEEE Std 802.3-2012. IEEE Standard for Ethernet*. LAN/MAN Standards Committee of the IEEE Computer Society, 2012.

[106] CISCO, "Ethernet Technologies," *http://docwiki.cisco.com/wiki/Ethernet_Technologies (Accessed December 22, 2013)*, 2013.

[107] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, vol. 19, no. 7, pp. 395–404, 1976.

[108] T. Carpenter and J. Barrett, *CWNA Certified Wireless Network Administrator Official Study Guide (Exam PW0-100)*. McGraw-Hill, 2007.

[109] D. Coleman and D. Westcott, *Certified Wireless Network Administrator*. Wiley, 2006.

[110] *IEEE Std 802.11-2012. IEEE Standard for Information Technology. Telecommunications and Information Exchange Between Systems. Local and Metropolitan Area Networks. Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. LAN/MAN Standards Committee of the IEEE Computer Society, 2012.

[111] *IEEE Std 802.15.4-2011. IEEE Standard for Local and Metropolitan Area Networks. Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Computer Society, 2011.

[112] C. Buratti, M. Martalo, R. Verdone, and G. Ferrari, *Sensor Networks with IEEE 802.15.4 Systems. Distributed Processing, MAC, and Connectivity*. Springer, 2011.

[113] "ZigBee Alliance. http://www.zigbee.org/ (Accessed December 22, 2013)." .

[114] "The Internet Engineering Task Force. IPv6 over Low power WPAN Working Group (Concluded). https://datatracker.ietf.org/wg/6lowpan/charter/ (Accessed December 22, 2013)." .

[115] S. Farahani, *ZigBee Wireless Networks and Transceivers*. Newnes, 2008.

[116] *ZigBee Specification. ZigBee Document 05347r17*. ZigBee Alliance, 2007.

[117] P. A. Bernstein, "Middleware: A Model for Distributed System Services," *Communications of the ACM*, vol. 39, no. 2, pp. 86–98, 1996.

[118] W. Emmerich, "Software Engineering and Middleware: A Roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 117–119.

[119] M. Lerner, G. Vanecek, N. Vidovic, and D. Vrsalovic, *Middleware Networks: Concept, Design and Deployment of Internet Infrastructure*. Kluwer Academic Publishers, 2002.

[120] A. S. Tanenbaum and M. van Steen, *Distributed Systems. Principles and Paradigms, 2nd Ed.* Pearson, 2007.

[121] H. Pinus, "Middleware: Past and Present a Comparison," *http://userpages.umbc.edu/ dgorin1/451/middleware/middleware.pdf (Accessed December 22, 2013)*, 2004.

[122] A. J. A. Wang and K. Qian, *Component-Oriented Programming*. Wiley, 2005.

[123] I. Crnkovic, M. Chaudron, and S. Larsson, “Component-Based Development Process and Component Lifecycle,” in *Software Engineering Advances, International Conference on*, 2006.
[124] C. Szyperski, D. Gruntz, and S. Murer, *Component Software, 2nd Ed*. ACM Press, 2002.
[125] F. Bushmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley&Sons, 1996.
[126] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Design*. Addison-Wesley, 1995.
[127] C. G. Lasater, *Design Patterns*. Wordware Publishing, 2010.
[128] R. A. Gupta and M.-Y. Chow, “Overview of Networked Control Systems,” in *Networked Control Systems*, Springer, 2008, pp. 1–23.
[129] V. Viatkin, *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*. ISA, 2007.
[130] *IEC 61131-3. Programmable Controllers - Part 3: Programming Languages. Edition 2.0*. International Electrotechnical Commision, 2003.
[131] *IEC 61499-1. Function Blocks - Part 1: Architecture*. International Electrotechnical Commision, 2003.
[132] G. Frey and T. Hussain, “Modeling Techniques for Distributed Control Systems Based on the IEC 61499 Standard - Current Approaches and Open Problems,” in *Discrete Event Systems, 8th International Workshop on*, 2006.
[133] R. W. Lewis, *Modelling Distributed Control Systems Using IEC 61499: Applying Function Blocks to Distributed Systems*. The Institution of Engineering Technology, 2008.
[134] K. Thramboulidis and G. Doukas, “IEC61499 Execution Model Semantics,” in *Innovative Algorithms and Techniques in Automation, Industrial Electronics and Telecommunications*, Springer, 2007.
[135] B. A. J. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, “Home Automation in the Wild: Challenges and Opportunities,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 2115–2124.
[136] R. Baquero, J. G. Rodriguez, S. Mendoza, and D. Decouchant, “Towards a Uniform Sensor-Handling Scheme for Ambient Intelligence Systems,” in *Electrical Engineering Computing Science and Automatic Control (CCE), 2011 8th International Conference on*, 2011, pp. 1–6.
[137] BAPI, “Understanding 4-20 mA Current Loops,” *http://www.bapihvac.com/CatalogPDFs/I_App_Notes/Understanding_Current_Loops.pdf (Accessed December 22, 2013)*, 2006.
[138] TI, “AN-300. Simple Circuit Detects Loss of 4-20 mA Signal,” *http://www.ti.com/analog/docs/litabsmultiplefilelist.tsp?literatureNumber=snoa605b&docCategoryId=1&familyId=78 (Accessed December 22, 2013)*, 2013.
[139] R. Baquero, J. Rodriguez, S. Mendoza, and D. Decouchant, “MidBlocks: A Supervising Middleware for Reliable Intelligent Environments.,” in *Intelligent Environments (Workshops)*, 2012, pp. 389–400.
[140] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The Many Faces of Publish/Subscribe,” *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[141] S. Tarkoma and K. Raatikainen, “State of the Art Review of Distributed Event Systems,” Helsinki University Computer Science Department. Helsinki Institute for Information Technology, 2006.

[142] J. Blanchette, *The Little Manual of API Design*. Trolltech, 2008.

[143] M. Becker, “Software Architecture Trends and Promising Technology for Ambient Assisted Living Systems,” in *Assisted Living Systems - Models, Architectures and Engineering Approaches, Proc. Dagstuhl Seminar, no. 07462*, 2008.

9 Journal Articles

9.1 Sensors 2012

Baquero, R.; Rodríguez, J.; Mendoza, S.; Decouchant, D.; Papis, A.P.M. *"Funblocks. A Modular Framework for AmI System Development"*; Sensors 2012, 12, 10259-10291. Available at: http://www.mdpi.com/1424-8220/12/8/10259

10 Conference Articles

10.1 Intelligent Environments 2012

Rafael Baquero, José Rodriguez, Sonia Mendoza, Dominique Decouchant; *"Midblocks: A Supervising Middleware for Reliable Intelligent Environments"*; Workshop Proceedings of the 8th International Conference on Intelligent Environments, IOS Press, 2012. Available at: http://www.booksonline.iospress.nl/Content/View.aspx?piid=30714

10.2 UCAmI 2011

Rafael Baquero S., José G. Rodríguez G., Sonia Mendoza C., and Dominique Decouchant; *"Towards a Modular Scheme for the Integration of Ambient Intelligence Systems"*; 5th International Symposium on Ubiquitous Computing & Ambient Intelligence, UCAmI 2011, December 5-9 2011, Riviera Maya, México.

10.3 CCE 2011

Rafael Baquero S., José G. Rodríguez G., Sonia Mendoza C., and Dominique Decouchant; *"Towards a Uniform Sensor-Handling Scheme for Ambient Intelligence Systems"*; 8th International Conference on Electrical Engineering, Computer Science and Automatic Control, CCE 2011, October 26-28 2011, Mérida, Yucatán, México.