DIPLOMARBEIT

Development of a Plug and Play Concept for Industrial Automation

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs

unter der Leitung von

Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Markus Vincze
Dipl.-Ing. Ingo Hegny

E376
Institut für Automatisierungs- und Regelungstechnik

eingereicht an der Technischen Universität Wien
Fakultat für Elektrotechnik und Informationstechnik

von

Pengzhou Xie
Matr.-Nr.: 0327510
Obermüllnerstraße 2C/6/105, 1020 Wien

Wien, im März 2010

Pengzhou Xie

Abstract

Nowadays the industrial automation is facing a rising pressure from the progressing globalization of the market, which is demanding for a higher efficiency and flexible reconfiguration ability in manufacturing systems. The requirements of increasing integration and interoperability for a dynamic network and communication configuration are becoming the key to ensure the configurability in distributed control and large complex systems.

From the aspect of device governance among autonomous sub-systems, the overall process of exchanging data among the systems based on different engineering tools requires established channels among the devices. The process of discovering new devices and managing communication configurations requires a concept, which makes assigning and managing channels automatically for the industrial automation domain possible.

The technology of Plug and Play provides a basic approach from the aspect of discovering communications among the devices. Currently, the Plug and Play standards available are developed mainly focusing on the consumer market and are based on Internet based protocols. Such an orientation makes them complicated to be directly integrated into the domain of industrial automation.

The focus of this work is to develop a Plug and Play concept for the industrial automation domain. Such a concept is able to distribute and manage channels automatically based on the communication medium and network protocols of the field devices. The framework introduced in this work for assigning and managing channels supports current and possible future industrial automation environments (e.g., IEC 61131-3, IEC 61499, Multi-Agent Systems).

By applying such a framework with its components, it is possible to distribute and administrate request-based channels automatically. The integration into different devices from various vendors is feasible. This concept is designed as an open and flexible structure that can be easily applied for industrial automation systems.

Kurzfassung

Die fortschreitende Globalisierung verstärkt den Leistungsdruck auf die Industrielle Automation, indem sie unter anderem höhere Effizienz und flexible Rekonfiguration in Fertigungssystemen fordert.

Steigende Integrität und Interoperabilität sind Voraussetzungen in dynamischen Netzwerken sowie für die Konfiguration der Kommunikation. Diese Erfordernisse bilden den Kernpunkt, um Konfiguration in verteilten und komplexen Systemen zu gewährleisten.

Für die Gerätesteuerung in autonomen Subsystemen werden Kommunikationskanäle zwischen den einzelnen Geräten benötigt, welche einen Datenaustausch zwischen den verschiedensten Entwicklungswerkzeugen ermöglichen.

Der Prozess, welcher neue Geräte entdeckt und die Konfiguration der Kommunikation organisiert, verlangt nach einem Konzept, welches die automatische Zuordnung und Organisation von Kanälen ermöglicht.

Die Technik von "Plug and Play" bietet einen Ansatz in Hinsicht auf die Errichtung von Kommunikation zwischen Geräten. Die gegenwärtigen "Plug and Play" Standards sind hauptsächlich auf den Massenmarkt ausgerichtet und basieren auf Internetprotokollen. Dieser Umstand macht es schwierig, sie direkt in das Gebiet der Industriellen Automation zu übernehmen.

Der Fokus dieser Arbeit liegt darin, ein "Plug and Play" Konzept für die Industrielle Automation zu entwickeln. Dieses Konzept kann Kanäle automatisch zuteilen und verwalten unabhängig davon, welches Kommunikationsmedium eingesetzt wird und welche Netzwerkprotokolle verwendet werden. Die Grundstruktur, die in dieser Arbeit vorgestellt wird, unterstützt aktuelle und zukünftige Standards der Industriellen Automation (z. B. IEC 61131-3, IEC 61499, Multi-Agenten Systeme).

Die Anwendungen eines solchen Grundstruktur mit seinen Komponenten ermöglicht die Verwaltung und die automatische Verteilung von Kommunikationskanälen. Die Integration in Geräte unterschiedlicher Hersteller erscheint praktikabel, weil die offene und flexible Struktur einfach für industrielle Automatisierungssystem angewandt werden kann.

Acknowledgment

Firstly I would like to give my sincere thanks to Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze for supervising my diploma-thesis. I would also like to thank Dipl.-Ing. Ingo Hegny for his great support and suggestions during the whole time of this work. His patience and ongoing support meant a lot to me. I also learned so much from him about how to achieve a higher level in the scientific research, which is much more than what a diploma-thesis can cover. Such an experience is a lifetime benefit for me.

Furthermore, I would also like to thank Dipl.-Ing. Martin Melik-Merkumians, all colleagues at the Odo Strugger Laboratory, and Yuzhang Han for their help and suggestions.

I would like to give my special thanks to my girlfriend Hanhui for always being there for these years. It is such a beautiful thing that I have her support in all kinds of ways.

My thanks also go to my friends Thomas Robatscher and Dipl. -Ing. Rupert Langegger. Without their heart-warming friendship, the life while studying in a foreign country wouldn't be such a joyful experience for me at all.

Above all, I want to give the most important thanks to my parents Donggang Xie and Min Zhao. Without them, I wouldn't have been able to come this far to the place where I am today. Their dedication, support and encouragement along the whole way are the most valuable things I could ever get in my entire life.

I dedicate my diploma-thesis to my grandfather Hongkui Xie, who passed away during the time that I work on this paper. It was such a hard thing that I never had the chance to say goodbye to him, but he stays forever deep inside my heart.

Pengzhou Xie

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The trend of the automation systems is developing from the decentralized control to the distributed control based on intelligent devices. The reconfiguration ability is one of the most important characteristics required for the industrial automation environment. Based on the requirements of a better reconfiguration ability for the manufacturing process, Multi-Agent Systems are introduced. Considering the requirements of real time execution in large complex systems, an approach that splits the Multi-Agent System into the High Level Control and Low Level Control is applied. The High Level Control is controlled by an agent for coordinating the manufacturing resources and overall execution, and the Low Level Control governs the actions from the physical system with field devices. The information from the field devices is collected and processed by the Low Level Control. From the overall aspect, process data are exchanged via the established channels among the communication interfaces at the Low Level Control and the High Level Control. The channels must be previously defined, which brings a few facts that need to be concerned.

When new devices are attached into the system, extra effort has to be made on configuring channels for the devices, which brings complexity for reconfiguration. Especially for the sub-systems developed with different engineering tools, the channel configuration developed with one engineering tool must be able to be recognized and integrated by other engineering tools. As for devices distributed into autonomous sub-systems, the channels must be administrated so that the field devices can coordinate with each other not only inside one system, but also among different systems. In order to exchange process data among devices, the devices should be aware of the presence of other devices without causing conflicts, in other words, make them ready for plug and play. From the aspect of various network and communication systems, different network protocols and parameter configurations can be available, which also leads to the need of the ability of an automatic and dynamic channel configuration.

Based on such facts, there is a need to develop a Plug and Play concept, which can assign and manage channels automatically. The channels among

the field devices should be automatically configured based on various system and communication specifications. The architecture should be able to be integrated into the industrial automation systems. By applying such a concept, the discovery of field devices, automated configuration and management of the communication among devices should be realized.

1.2 Guideline Through the Work

This work comes with following chapters after this brief introduction:

Chapter 2: The introduction of the current technologies in industrial automation covers the areas of the industrial automation systems and their important characteristics. The network and communication systems in the industrial automation domain are also discussed, in order to give a conceptual understanding for the communication systems and technologies. As for Plug and Play standards, a few current implementations are introduced. Their limitations are also analyzed.

Chapter 3: As the main focus of this work, which is the Plug and Play concept itself, the development of such a concept shows the general framework that this concept based on. The requirements of the concept will be discussed. The three-layer framework with its components will be introduced. Each component will be analyzed and a mechanism of distributing channels for such a framework is provided.

Chapter 4: A prototypical implementation of the framework within IEC 61499 according to the concept is discussed. The configuration of each module will be analyzed. The results of the implementation are discussed.

Chapter 5: An outlook for future steps of the development of this concept is described.

Chapter 6: A conclusion is presented as a summary of this work.

2 State of the Art

This chapter provides a summarized overview of the current systems and architectures in the field of industrial automation. It will provide some principles and background knowledge, which are related to this work. Most of the terms and definitions used in this work will be clarified.

As for the detailed content of this chapter, two parts will be introduced, which are the overview of industrial automation systems, and industrial network and communication systems. The first part will start with an introduction of development in industrial automation, followed by introducing IEC 61499 and Multi-Agent Systems. Industrial network and communication systems will be taken as the focal point of this chapter. Some available network structures and implementations are explained. The emphasized points will be on the Dynamic Host Configuration Protocol and the Plug and Play concept.

2.1 Industrial Automation Systems

The development of industrial automation system has a goal of chasing a better capability, which is more self-automated, reconfigurable, and flexible. In order to achieve this goal, the automation system is evolved from a components-based, centralized system to a systems-based, distributed system.

2.1.1 Decentralized and Distributed Systems

The more intelligent the system is, the more intelligent the components are. But this standard has not come since the very beginning of industrial automation. As for the "classic" automation system, it is typical to find that a central process computer is surrounded by all the sensors and drivers, etc. The conventional architecture is built on all these components, which are controlled by using big software blocks. But as the automation system develops, such a centralized automation architecture with huge software modules no longer meets the growing needs of a more adaptive system. Factors such as cost and complexity are becoming issues that customers would not ignore. The trend is going from a monolithic software module with field devices to distributed

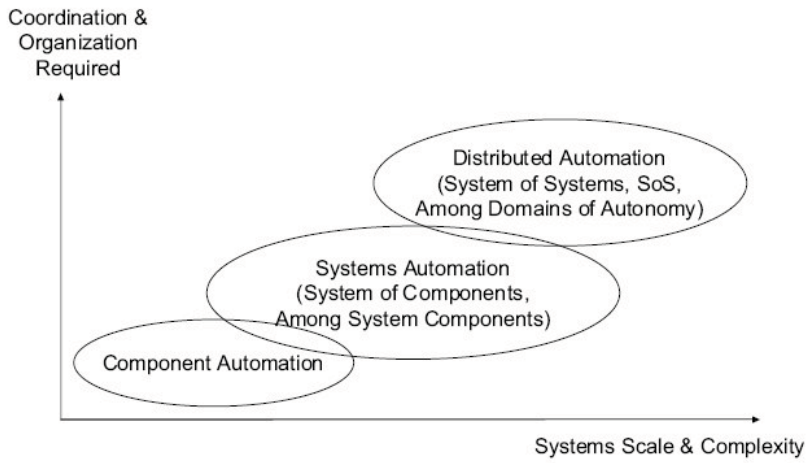

Figure 2.1: Levels of Automation [TQ08]

intelligent components. The different paradigms of automation are shown in Figure 2.1.

Between 1960s and 1970s, automation architecture based on a central common control processor was the heart of intelligence. Any change of main function has to be made on this part for such automation architecture. In the following 20 years there were more intelligent components developed, but they are only physically and spatially distributed, which means they are still controlled by monolithic software blocks. Such architecture is not able to run over more distributed equipments. Only since 2000, there are distributed systems implemented and used [FB04].

From the aspect of decentralized control and distributed control, the development between different generations can be also seen. The main part of the decentralized control is PLC (Programmable Logic Controller). The execution of tasks is carried out by a very powerful CPU, and all field devices are connected via fieldbuses. In such a structure, all input values would be first stored in order, then instructions are calculated one-by-one in a fixed order and the results are written to the outputs, afterwards the cycle starts again. There is however no cooperating or interactive communications among devices themselves. And there is no application crossing over several devices and carried out via integration at the system level.

There might be more than one PLC available in the whole automation system, and it is possible that there is various communication interfaces for each specified application. The whole system for decentralized control is divided into different fieldbus systems. But each PLC works on its own without much

interaction to other PLCs. As for the modern trend of automation system, with the appearance of more intelligent devices, the distributed control requires no more PLCs; the whole control program and tasks are split up into small parts and executed by those "smart" devices, which have more control intelligence and are connected over standardized software interfaces to the system. For example, now a pressure sensor can directly cooperate with a valve and controller module then executes the task of pressure control and furthermore to adjust the speed of pump's motor. Distributed automation system is the system that is based on autonomous intelligent units, which are connected with each other using a common communication system. In order to execute a global task in the whole system, the units coordinate their activities by exchanging information over the communication system [Sün04] [FB04].

At this point, it is clear that the control task is modularly unitized in distributed control. A distributed automation system provides better possibilities for construction, maintaining, configuration and installation. Above all, a cost reducing over the whole product cycle is also noticeable [Fur00].

In order to make the data and information exchange possible, the network and communication mechanism is playing an important role in the distributed automation system. The definition of distributed automation system also brings us the goal of the communication system: data (measured values, variables, status information) and signals exchange. A successful data transfer depends on in which kind and way to exchange the data.

2.1.2 IEC 61499

IEC 61499 is an IEC standard for the use of function blocks in distributed Industrial-Process Measurement and Control Systems (IPMCSs). IEC 61499 is seen as a successor of IEC 61311, which is used for PLCs based process control systems. IEC 61499 is developed as a function block (FB) oriented programming model for distributed automation systems. As a new standard from IEC (International Electrotechnical Commission), IEC 61499 defines how function blocks can be used in distributed industrial process, measurement and control systems [Lew01].

The IEC 61499 standard family consists three parts [Chr07]:

IEC 61499-1: Part 1, Architecture. This part describes all general models of IEC 61499, and introduces different types of function blocks in the paradigm of application centered engineering, as well as their configuration specifications [IEC05a].

IEC 61499-2: Part 2, Software Tool Requirements. IEC Standard, January 2005 This part describes the exchange of library elements in IEC

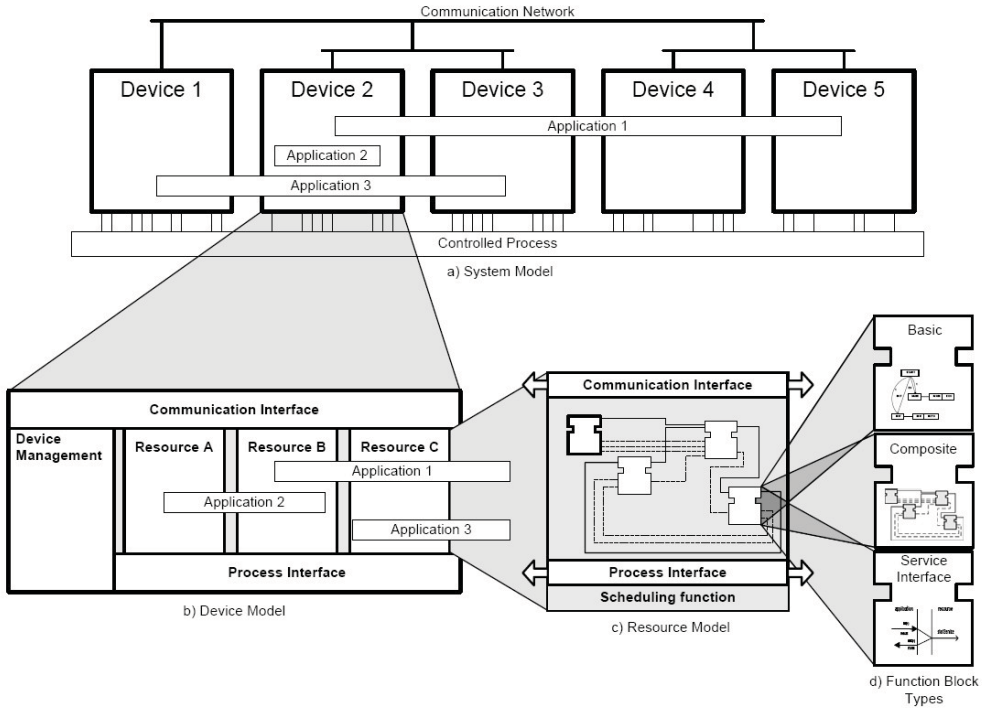

Figure 2.2: IEC 61499 Model Overview [Zoi07]

61499, along with display, modification, validation and implementation of declarations. It points out the main requirements in development of software libraries [IEC05b].

IEC 61499-4: Part 4, Rules for Compliance Profiles. IEC Standard, May 2005 This part defines rules for the development of compliance profiles which specify the feature of IEC 61499-1 and 61499-2 to be implemented in order to achieve the requirements from interoperability of devices, portability of software, and configurability of devices [IEC05c].

The IEC 61499 standard provides a set of models for describing distributed systems that are programmed using function blocks. As the architecture for a function block oriented distributed system, there are several reference models (see Figure 2.2)[Lew01] [IEC05a]:

System Model: It defines the relationship between communicating devices and applications. It describes the connection between devices and applications. An application can exist on one device or functionally distributed over many devices. A distributed application is designed for a network of function blocks, when the application is loaded, it can be loaded as series of function blocks fragments into different devices.

Device Model: A device contains at least one interface for process or communication; it also contains zero or more resources and function block networks. The interfaces for the process provides a possibility that information can be exchanged between the physical process (analog or digital Input/Output) and resources (as data or events).

Resource Model: A resource can be seen as a functional unit with independent control. The operation within itself such as being loaded, created, configured, parameterized, started up, deleted. must not affect other resources in the same device.

Application Model: In IEC 61499, application is defined as a network of interconnected function blocks, connected with each other via event connections and data connections. As described in System Model, an application can be distributed over many resources. Within each application, there can be also subapplications available. The behaviors of function blocks in subapplications are the same as they are in applications; therefore the subapplications can be equally encapsulated and identified in a system.

Distribution Model: When applications or subapplications distributed over different devices and resources, the Distribution Model defines two requirements for functionality and communication. The first one is requirements for where the applications or subapplications are distributed among multiple resources and devices, that is, to make sure that all elements contained in a given function block instance must be contained within the same resource. Secondly is that it describes requirements for communication services to support distribution of applications or subapplications among multiple devices to assure that the functionality among the function blocks of an application or a subapplication should remain the same.

Management Model: IEC 61499 defines a special form of application for creating and managing application fragments, the Management Application. It is used when a resource has many function block networks distributed among applications and subapplications. The Management Application has higher privileged functionality than normal applications that it can construct parts of other applications by creating function blocks and connections.

Function Block Model: Function block is the basic unit in IEC 61499. It is a functional unit of software specified by a function block type for its data structure. It can encapsulate its individual algorithms and operations, and it is connected with other function blocks via data connections and event connections.

Based on different kinds of function block specifications of its behaviors and interface instances, the functionality of function blocks can be categorized as follows [IEC05a] [Ham08]:

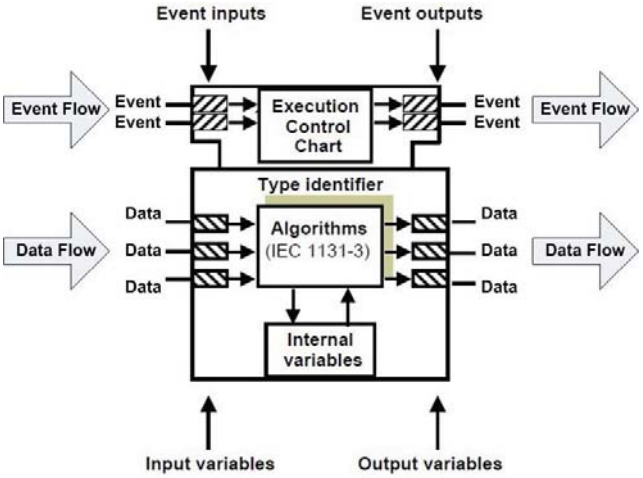

Figure 2.3: Basic Function Block based on [Chr04]

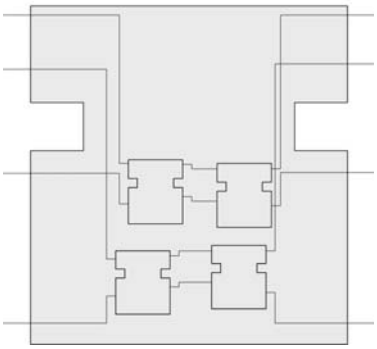

Figure 2.4: Composite Function Block

- **Basic Function Block**: This type of function block operates on the values of input variables, output variables, internal variables, and algorithms. The execution control chart (ECC) expresses the associations between the internal algorithms and the behaviors of event outputs (see Figure 2.3).
- **Composite Function Block**: Composite FBs encapsulate FB networks by using data connections and event connections among FBs. More FBs are connected with each other via event and data connections as component blocks. All the encapsulated FBs are connected to the outer FB Network via the event/data inputs and outputs (see Figure 2.4).
- **Service Interface Function Block**: A Service Interface Function Block

(SIFB) is used to provide functionality of controlling the device hardware and the services needed by the control system (e.g., QoS, I/O interface, communication interface). The inputs and outputs of service interface function block types have specialized semantics, and the behavior of instances are defined through sequences of service primitives. In this work, the SIFB will be only focused on its communication function. There are two types of generic service interface function blocks used to acquire communication between remote resources. The Requester SIFB is used to provide communication service to send requests, in order to obtain data. On the other hand, the Responder SIFB provides the communication service at the other end, which is to respond the request and send data back to requester. Together these two SIFBs can exchange data between two resources linked by communication functionality or network (see Figure 2.5).

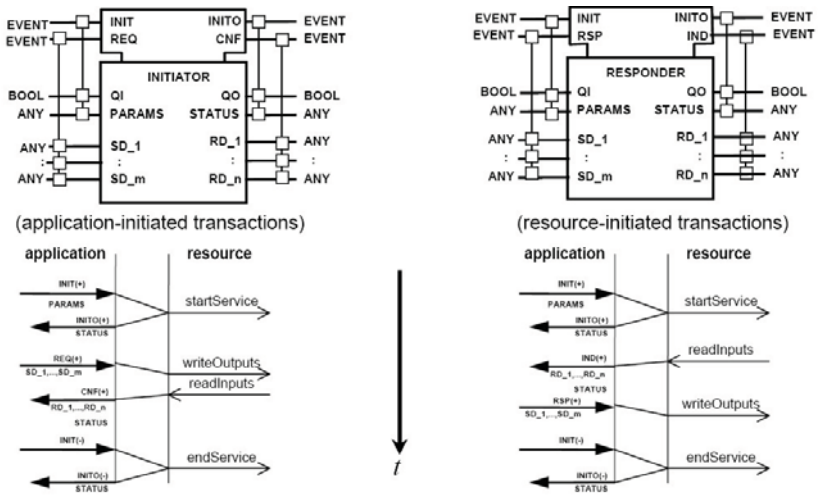

Figure 2.5: Requester and Responder [Chr04]

2.1.3 Multi Agent System

In order to achieve better reconfiguration ability in manufacturing systems, the Multi-Agent System (MAS) is introduced to the dynamic environments of industrial automation. From the previous work by Merdan [Mer09] at Automation and Control Institute, Vienna University of Technology, it is recognized as a system, which consists of a structure with High Level Control and Low

Level Control. Such a system is based on multi-agent structure and using distributed control to cope with dynamics in large complex systems.

By applying agents into the manufacturing control system, the MAS integrate the High Level Control (HLC) applications and Low Level Control (LLC) physical components. Such a system minimized the production time of jobs and costs, while increasing the resource utilization. Multi-agent system can be defined as a network of autonomous, intelligent entities/agents that cooperate and communicate together in order to achieve aims, which are beyond the individual capabilities and knowledge possessed by each agent. The agents are intelligent components that offer a decentralized control for modeling process and systems. Two main multi-agent architectures for dynamic scheduling that introduced in the literature are autonomous architectures and mediator architectures [MLHK08].

The HLC and LLC can be seen as hierarchical levels in the MAS. The HLC is controlled by an agent that coordinates and manages the manufacturing resources. The LLC governs the actions of the underlined physical system. The LLC collects the information and data from the physical components like sensors, conveyors, intersections, index stations, RFID readers and writers. The information collected by the LLC will be processed. Furthermore the results are performed as particular actions to inform the HLC about an event [MLHK08] [MKHFB08].

The IEC 61499 provides the event-driven function blocks and offering the framework for the integration of run-time control in the LLC. The implementation of the communication framework for agents, such as the Contact Agent and Order and Supply Agents, are built based on the Java Agent Development Environment (JADE) framework [JAD].

Here the focus is the communication between the HLC and the LLC. The aspects of such an interface can be interpreted as the principle of "Separation of concerns" [MLHK08]:

- Channel: Both partners have to use the same means for communication meaning that the communication channel has to be known by all involved participants.
- Message: Communication between the HLC and LLC is provided in an asynchronous message-based way, as this fits best for both parties.
- Message Content: The data transmitted within a message has to be understood by all communication partners and interpreted in the same way.

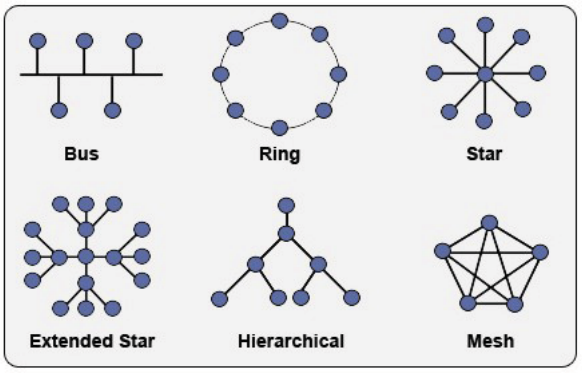

Figure 2.6: Typical Network Topologies [LN07]

As for the interface on the LLC side, the implementation is done with Service Interface Function Blocks. A generic interface for the HLC-LLC interface encapsulates the information and can be transferred via previously established communication channels. The status updates and commands are seen as messages to keep HLC and LLC synchronized [MVKZ08].

2.2 Industrial Network and Communication Systems

Networks and communication systems make the data exchange possible in industrial automation systems. The network is a broad idea that has many characteristics, such as dimension, protocols, interface, network address, address table, medium, operation kind, basic technology, topology [Fur00]. There are a few among them should be paid attention to, in order to understand how the data transmission works in a network.

From the physical structure of the network, the first thing to be seen is topology. Topology describes how the type and logic configuration of connections are. There are two parts that topology consists of, a geometric alignment of the members in the network and a logical configuration of the participants. The geometric configuration and the logical configuration are independent from one another [Sch00]. Some common and well known geometric topologies are shown in Figure 2.6.

Between the devices in the network (as shown in the physical topologies), there are certain rules available for the connections and routing paths in between, such rules are set by protocols.

Protocols are used to set rules and formats for communication between processes. As the definition of a protocol indicates, there are two important parts [CDK94]:

- A specification of the sequence of messages that must be exchanged;
- A specification of the format of the data in the messages.

In this way, the protocols can set a standard of connection, communication, and the way of data transfer between communication endpoints. In the field of industrial automation, there are a few commonly used protocols such as Ethernet-TCP/IP, Profibus, Bitbus, Interbus. For each protocol that is implemented, there is a regular defined format of the data packet to be transferred over the network. Protocols can be different depending on networks, which are based on different configuration, implementation, or physical medium.

With topology to set up the physical configuration of a network, and protocols to set up the rules of the data packet formats for transmitting, networks can now be differentiated amongst others by different topology, protocol, and physical medium.

2.2.1 OSI Model

The OSI model (Open Systems Interconnection) was developed by the ISO (International Standards Organization) as a model to illustrate functions in communication systems and as a basic model for standardization. It was conceived to provide a unified solution to interconnection of and interoperability between systems and networks developed by different parties [Fur00] [SJM90].

The OSI reference model is a 7-layer model, which is separated into two classes [Bor92]:

1. From layer 1 to layer 4 are net-orientated layers; the task is to bring data from one place to another.
2. From layer 5 to layer 7 are application-orientated layers; the task is to make the network in appropriate form available for the users.

The construction of the layer model that consists of seven layers is shown in Figure 2.7. From top to bottom the layers are: the application, presentation, session, transport, network, data link, and physical layer These seven layers offer interfaces, which are so called service access points to the upper and lower layers [Klo08].

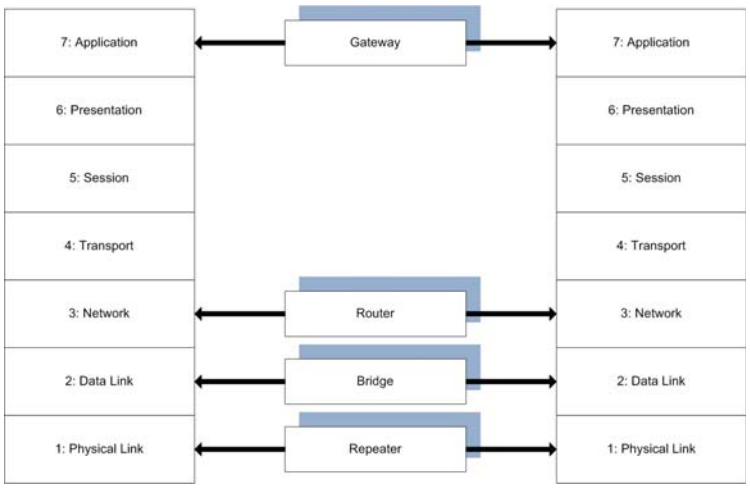

Figure 2.7: Repeater, Bridge, Router and Gateway in OSI model [Fur00]

There are service initial functions as Request, Indication, Response, and Confirmation for a task of the communication [Sch00]. For a confirmed service, there are all four initial functions available. For unconfirmed services, there are only Request and Indication available. The OSI model is used as a reference model. From this model, not every layer is always needed by different derivatives. For example, in TCP/IP model, the session layer (layer 5), presentation layer (layer 6) and application layer (layer 7) of the OSI 7-layer model are not implemented and are included in the application layer, which refers to the higher-level protocols. The reason is that OSI model is only an abstract model of network-based layer concept; it is not strictly constrained to every specified protocol or application [ZK09].

There are four important components of TCP/IP communication system: Repeater, Bridge, Router, and Gateway. These components improve flexibility in the installation and transmission between different transport networks, as well as for communication between partners with different protocols. The corresponding position of these components in OSI model can be seen in Figure 2.7.

The functions of these four components are introduced as follows [Sch00]:

- Repeater: it has no header information, can be seen either as a pure converter for electrical or optical signals of transmission system which allows the structure of topological nets further expanded, or a swap of the communication mediums.

- Bridge: it connects two networks and exchanges packets in between.

- Router: the ability for a router is routing between different communication systems, it decides the destination where the IP-datagram is supposed to be sent. There are two kinds of delivery in the routing path: direct and indirect delivery. Direct delivery is applied if the sender and the receiver are in the same network, or multiple networks, which are coupled by Bridges. Indirect delivery is applied if the sender and the receiver are in different networks.

- Gateway: the Gateway is applied when there are network interfaces with different protocols. It works as a protocol translator and converter to provide system interoperability.

2.2.2 Fieldbus

Fieldbus was introduced to industrial automation in the 1980s. The common meaning of fieldbus is a network for connecting field devices (such as sensors, actuators), field controllers (such as PLC's, regulators, drive controllers), and man-machine interfaces [Tho05].

Before fieldbus came into practice, as for conventional communications, there are a few properties bringing the considerations that needed to be solved [Bon95]:

- Information transmission only in one direction.

- Signal transmission cable from central control or process control system connected to field devices, with 2 or more connections to every device.

- Analogue signals have little information content.

- High amount of cabling to solve permanent power supplies for sensors and actuators.

- Converting many analogue signals to digital process control system is complex.

- Analogue signals transmission at increasing modern devices costs much.

The appearance and development of fieldbus provide possibilities that can solve the issues mentioned above and meet the requirements with increasing kind and quantity of devices, which also requires more configurable connection structures. The idea of the fieldbus is to replace point-to-point links from each

Table 2.1: Branches and Areas of Fieldbuses [Die09]

Aircraft and aviation:	MIL 1553, LON
Automation and Control:	PROFIBUS DP, CAN, INTERBUS, WFIP(USA, South Africa), Devicenet, P-Net (DK, Portugal), LON
Building Automation:	LON, EIB, BacNet, Ethernet
Process Control:	PROFIBUS PA
Vehicle:	CAN, TTP
Railway:	LON, CAN
Shipbuilding:	CAN

sensor or actuator to its controlling equipment to a single link on which all information is transmitted serially and multiplexed in time [PD98].

There are a few requirements from the end-user as the fieldbus is developing, which also become the goal of fieldbus [Tho99]:

- Safety, availability and, more generally, dependability.
- Better maintainability.
- Better modularity, and capacity for evolution.
- Openness, interoperability, interchangeability, long lifetimes.
- Better performances, and low cost.

Fieldbus works as a shared medium and a fundamental system component for the devices in industrial automation [Tho99]. The fieldbus has the ability that will complete the reading and writing variables to devices. Data exchange in fieldbus within automation systems supports the integrated system into environments with sensors, actuators and control devices. During normal operation of data exchanging, measured variables, control variables, and status information are communicated [Ebe07]. In other words, fieldbus enables the communication in industrial automation network.

Ever since the introduction of fieldbus, there are many standards and implementations from different manufactures. These different implementations can be also categorized into different applied fields or branches [Die09] (see Table 2.1).

Each of the fieldbus implementations mentioned in Table 2.1 supports its own standard, protocol, topology, limits to quantity of participants, access

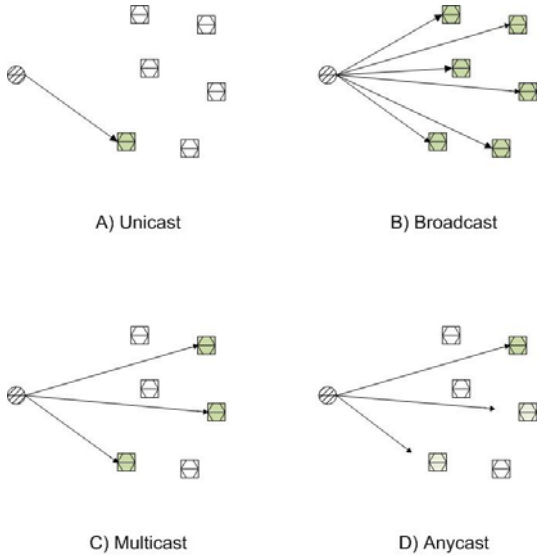

Figure 2.8: Unicast, Broadcast, Multicast, and Anycast

method, transfer rate, telegram format, data storage format, physical medium, etc. The configurations of the fieldbus can vary from these factors, which determines also the transmission characteristic of data.

2.2.3 Transmission Patterns

In a communication network, in order to transmit a packet from sender to receiver, there is more than one transmitting pattern available. A transmitting pattern is usually referring to the methods how the routing topology works when the sender, receiver, and data (message) are available. In a transmitting process, depending on the network addresses of different receivers and the number of receivers, there are four commonly used casting patterns (see Figure 2.8):

- **Unicast:** Unicast is a one-to-one connection between the sender and receiver, which means that there are only two communication partners available: one sender and one receiver. The bandwidth cost might be really high, because of unique resource is requested in Unicast [Mic03].
- **Broadcast:** Broadcast transmits the data packet sent from a host to a large set of hosts within the given address range without knowing what the other hosts can supply it. It is a one-to-many transmitting pattern. Broadcasting depends on the specific data link layer [Mog84][Mic03].

- **Multicast:** Multicast is a transmission pattern that delivers packet from the sender to a specific group of receivers (one-to-many). Sender sends out the same packet to each receiver, which has joined the same multicasting address group. The multicasting is useful when a group of receivers are about to receive the same packets from a specific sender over the network at the same time, which will save significant bandwidth [Dee89][APGD07].

- **Anycast:** Anycast is also a one-to-many transmitting pattern, each destination address identifies a set of receiver endpoints, but the data packet is routed to only one of them, which is either topologically the nearest or best [Abl06] [Woo02].

There are actually more casting terms used nowadays, but many of them are invented based on web-oriented applications being developed by different vendors focusing on consumer market.

2.2.4 DHCP

The Dynamic Host Configuration Protocol (DHCP) provides configuration parameters to hosts, which have the requirements of assigned parameter for communication addressing (e.g., IP address, Gateway, DNS-Server). It consists of two components [Dro97]: A protocol for delivering host-specific configuration parameters from a DHCP server to a host; and a mechanism for allocation of network addresses to hosts.

DHCP is built on a client-server model, the term "server" refers to a host providing initialization parameters through DHCP, and the term "client" refers to a host requesting initialization parameters from a DHCP server. There are three mechanisms for IP address allocation [Dro97]:

- Automatic allocation: which means the DHCP server assigns a permanent IP address to a client.

- Dynamic allocation: DHCP server assigns an IP address to a client for a limited period of time (or until the client explicitly relinquishes the address).

- Manual allocation: a client's IP address is assigned by the network administrator, and DHCP is used simply to convey the assigned address to the client.

During the process of configuring parameters between a client and a server, one or more of these mechanisms could be used, depending on the policies of the network administrator.

A typical process of DHCP configuration can be seen as several steps in Figure 2.9 [Dro97]. A general process of DHCP is focused on discovery, offer, request, and acknowledge. At each step, messages from Client and Server are exchanged. Each message contains their DHCP message format to give specifications of the content as data fields.

At first, Client broadcasts a `DHCPDISCOVER` message on its local physical subnet. In this `DHCPDISCOVER` message, a data field of `options` of preferred values for network address and leasing can be included. The data field of `options` can be used by the server as requirements of the requests to check if the request is a specified request with its configurations. The `DHCPDISCOVER` message is sent from the client to all available servers in its range. After the Servers receive the `DHCPDISCOVER` message, the client is discovered by the servers. Each server may respond with a `DHCPOFFER` message that includes an available network address. As for the network addresses, the server should check if the network address, which is going to be offered, is already in use when the server is allocating new network addresses.

When the server receives the `DHCPDISCOVER` message and sends its `DHCPOFFER` message, the client will receive one or more `DHCPOFFER` messages from one or more servers. Client will choose the server that matches the request configuration parameters based on offered parameters in `DHCPOFFER` message. If the `DHCPOFFER` is approved by the client, the client broadcasts a `DHCPREQUEST` with '`server identifier`' option to indicate which server it has selected. The `DHCPREQUEST` message is sent along with the data field of `options` for specifying preferred configuration values. The `DHCPREQUEST` message is also used as notification for the rejected servers. A timer is set for the process between `DHCPOFFER` and `DHCPREQUEST` for the client. If the client waits too long and is still not receiving `DHCPOFFER` message, the timer times out and retransmits `DHCPDISCOVER` message to the servers.

When the server receives the `DHCPREQUEST` broadcast from the client, a connection between the selected server and the client is established. The server should assign the client with the network address. By responding with a `DHCPACK` message, the server uses this message to contain all the configuration parameters for the requesting client. This `DCPACK` message is used as unique identifier for the client's lease and for both the client and server to identify a lease referred to in any DHCP messages. Configuration parameters in `DHCPACK` message should not be conflicted with the parameters in `DHCPOFFER` message. Therefore there is no need for the server to check the offered network address

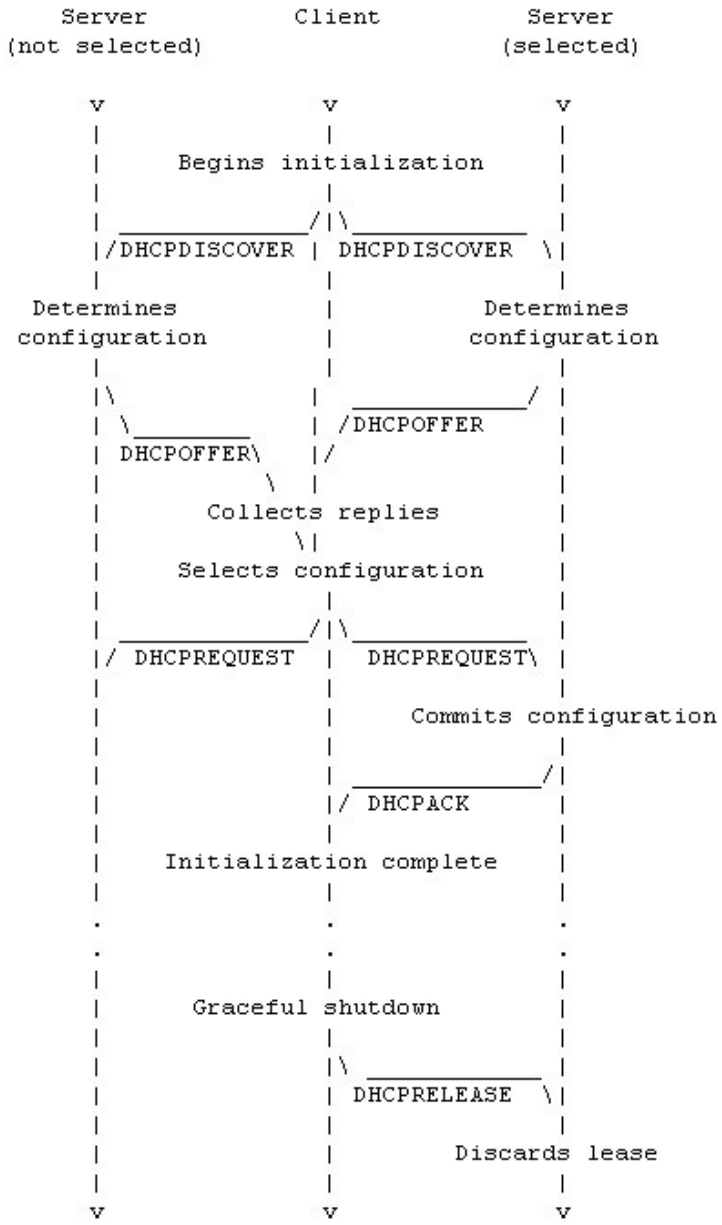

Figure 2.9: DHCP Diagram [Dro97]

at this point. The selected network address is already filled in the data field of the DHCPACK message.

If the server is no longer capable of sending configuration parameters that meet the request from the client (e.g., all addresses are already allocated), the server should send a DHCPNAK message to the client. If the client receives a DHCPNAK message, the client will restart the process of configuration.

When the client receives the DHCPACK message with configuration parameters, the client will recheck configuration parameters. Here the duration of leasing time in DHCPACK message should be noticed by the client. At this point, the client is configured.

If the client detects that the assigned address is being used, in order to avoid the confliction, the client must sends a DHCPDECLINE message to the server and restart process. As a design option, the client should wait 10 seconds before restarting in order to avoid excessive network traffic in case of looping.

If neither a DHCPACK message nor a DHCPNAK message is received, the timer of waiting times out at the client, the client will retransmit its DHCPDISCOVER message. The client should be set with a time period for retransmitting and waiting in order to coordinate a good timing balance. After repeating the transmission of DHCPREQUEST message, if there is still no DHCPACK or DHCPNAK response, the client reverts to INIT state and restarts the initialization process. A notification that the initialization process has failed and the process is restarting will be given by the client.

The client may also choose to relinquish its lease on a network address by sending a DHCPRELEASE message to the server. For the DHCPRELEASE message, the same client identifier should be given in this message.

DHCP also supports mechanisms that it can be used by hosts to renew their lease of parameter configuration [Eve02], or to reuse the allocated address. From the process of distributing the network addresses, DHCP is using the server to manage all network addresses and distributing them according to the requirements of the client. For such a structure, client is the one which provides the requests, and the server will use the parameters in the requests to select the correct configurations as response. The messages like DHCPDISCOVER, DHCPREQUEST, and DHCPACK are filled with the contents as data fields to meet the requirements. The DHCP mechanism also provides the possibility for additional requirements, such as options as data field, or DHCPNAK, DHCPRELEASE messages as additional message types.

2.2.5 Plug and Play

Plug and Play (PnP) is a general term widely used in computing and electronic products. The main focus of the Plug and Play is zero configuration, which means installing a peripheral in a computer or connecting a computer or device to a network without requiring any technical configuration by the user.

At the beginning of computing technology, the configuration of hardware is applied generally in two ways: either through redesign to accommodate different operations, or by manual reconfiguration to connect or disconnect the devices. Such methodology is also applied to solve the problem when there is a conflict between devices. Such a conventional way would no longer meet the requirements from the increasing number of devices. And the manual configuration is not an optimized option if considering its cost and complexity. In order to make automated configuration possible, Plug and Play defines the ability to add a new component to a system and to finish the configuration automatically without having to do any technical analysis or manual configuration. Plug and Play is aimed to solve not only the problems from configuring the system, but also to deal with resource conflicts caused by different devices.

The most widely used Plug and Play specification and term was developed by Microsoft with cooperation from Intel and many other hardware manufacturers. The goal is to create a computer whose hardware and software work together and to automatically configure devices and assign resources. Such a goal can allow hardware changes and additions possible without the need for large-scale resource assignment tweaking. The new devices will be able to be attached into the system and immediately usable. In other words, Plug and Play makes peripheral devices "out of box" functional without complications of setup.

A form of Plug and Play was actually first made available on the EISA (Extended Industry Standard Architecture Bus) and MCA (Micro Channel Architecture Bus) buses many years ago. For several reasons, however, neither of these buses caught on and became popular [Koz01] [Enc09].

In a general Plug and Play process, when a device is attached to the network, it broadcasts its services, other devices, hosts, or applications are notified by its appearance, a typical protocol stack provides certain service elements as follows [Neu06]:

- Resource allocation: certain identifiers from the available name space are assigned to the newly attached device.
- Self-description: the device must be able to provide information about the configuration parameters it requires.

- Discovery: it must be possible to discover the presence of devices along with their capabilities. The attachment and detachment of devices needs to be detected and announced. The discovery element can be split in device and service discovery.

There are many evolving standards based on the concept of the original idea of Plug and Play: Zero Configuration, such as Zeroconf, Jini, and UPnP. Most of the applications or protocols are developed for computer operating systems or electronic products in the consumer market. Here a brief introduction to UPnP and Zeroconf will be given.

UPnP (Universal Plug and Play) is a family of protocols from the UPnP Forum introduced in 1999 for automatically configuring devices, discovering services and providing peer-to-peer data transfer over an IP network. The UPnP is only related to Plug and Play by the original zero configuration concept: installation without manual configuration. UPnP covers many application areas including lighting, heating and air conditioning, quality of service (QoS), and consumer electronics [UPn09] [Hel02] [Mic00] [SL03].

A UPnP network is composed of: Control Devices, which provide services; and Control Points, which use the devices. There are six phases to UPnP networking [UPn00][SL03]:

- Addressing: Control point and device get addresses
- Discovery: Control Devices sends messages via multicast over UDP to advertise its services to Control Points
- Description: Control Points request device description from the Control Devices using URLs contained in the advertisement message. Control Points also retrieves service descriptions using the URLs in the device Descriptions.
- Control: Control Points asks Control Devices to invoke a service action by sending an HTTP request as specified in the device and service descriptions.
- Eventing: Control Point listens to state changes of device, and may subscribe to receive the descriptions from device.
- Presentation: Control Point controls and monitors device status using HTML-based user interfaces.

Table 2.2: Comparision of Zeroconf and UPnP based on [Che09]

	Zeroconf	UPnP
Addressing	IPv4 Link-Local Addressing	IPv4 Link-Local Addressing
Naming	Multicast DNS	UPnP has no equivalent to mDNS
Discovery	DNS Service Discovery	UPnP has no dependable Service Discovery protocol
Application	not defined	defines the particular application-layer protocol based on available devices

Zeroconf is an IETF (The Internet Engineering Task Force) specification that enables devices on an IP network to automatically configure them and to be discovered without manual intervention. Zeroconf can also assign an IP address and alternate host name to a device [Org09][Gut01].

There are three phases of Zeroconf configuration functions [LSKD07]:

- Addressing: automatic IP address assignment that Zeroconf can employ even without a DHCP server. This requirement is solved by Self-assigned Link-Local Addressing Standard (IPv4LL, RFC 3927).
- Naming: Zeroconf uses Multicast DNS to translate host names to addresses in the absence of a conventional Unicast DNS server.
- Discovery: Zeroconf adds DNS Service Discovery (mDNS) to discover what services are available on the network; Multicast DNS is employed by DNS Service Discovery (DNS-SD) to send packets to every node on the network for resolving duplicate host names and to query the network for services.

The comparison and description of differences between this three-layer structure (IPv4LL + mDNS + DNS-SD) from Zeroconf and UPnP can be seen in Table 2.2.

2.2.6 Quality of Service

Quality of Service (QoS) is defined by ITU (International Telecommunication Union) as a set of qualities related to the collective behavior of one or more objects [ITU95]. It sets prioritized network traffic, by using different predefined priorities. QoS can be seen as a control mechanism. It provides priority

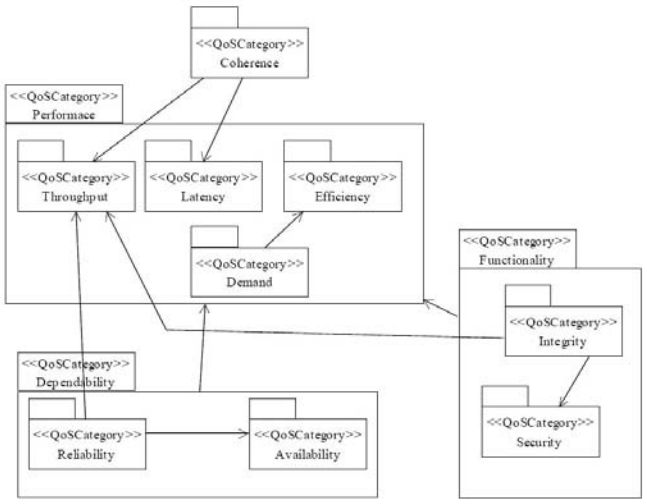

Figure 2.10: QoS Categories [OMG08]

levels based on different data streams and users according to the requirements of the applications. QoS guarantees a certain level of performance of the data transfer. It is especially useful for the network with limited capacity. As for the network and communication in industrial automation system, it is necessary to consider the QoS for the situation with a massive number of field devices available. As for the Plug and Play concept, the processes of addressing, distribution, and reconfiguration can be time critical depending on the communication patterns and the requirement of the devices. For some scenarios, it is necessary to ensure certain user data must not be dropped during the transmission. By applying QoS as service in industrial automation system, it can be used in network protocol filtering, channel selection, or requests configuration.

Depending on the different protocols applied in the industrial automation system, QoS can be used to control the certain sequence order for different communication patterns. Figure 2.10 provides an overview to different QoS categories.

Based on these categories of QoS, it is necessary to sort QoS parameters into different scenarios for industrial automation system. For each scenario, different kinds of QoS should be set to meet the requirements of the communication. Several cases can be taken as examples: as for initializing sequence for a client-server communication model, the clients can send their requests successfully only after the server is initialized; to ensure important data can be successfully sent before the demanding response times out; measuring if received response

arrives before it reaches the deadline of time limit; acknowledgment of data packets are dropped for some protocols with no reliable connection-oriented data transmission; synchronizing the sending and receiving rate of data packets between sender and receiver; validity of the data packets; consistent receiving time period for certain data packets [Vog07]. More scenarios could be available according to the types of data packets and the communication patterns.

2.3 Conclusion

This chapter gives a brief introduction and discussion focusing on the current technologies for industrial automation. A description covering the areas of the industrial automation systems from decentralized control to distributed control and Multi-Agent System is discussed.

Based on the development of the industrial automation system, the trend of this field is heading to a distributed system with more intelligent devices attached. The management and reconfiguration of these devices becomes a focus in order to reduce the complexity of the reconfiguration for the devices. The Multi-Agent System shows a framework consisting of the HLC and the LLC, and the agents can be applied as the "brain" of such a framework. But the interface between the HLC and the LLC are depending on the pre-defined or established channels. Such a communication interface still requires an architecture that can automatically configure communication interface based on the physical environment on the side of the LLC. The solution to such a requirement can reduce the costs and complexity of the reconfiguration during the process of attaching new devices into the system. The newly attached devices need to be discovered by the HLC agents and its presence should be acknowledged by other devices via channels among them. From the analysis of the industrial network and communication system, the approach of developing such a concept can be seen as developing a Plug and Play concept for industrial automation system.

3 Concept

The standards available now for a Plug and Play concept such as UPnP and Zeroconf are designed mainly for electronic products in consumer market; for each standard available there is a large alliance behind, which is formed with many manufacturers who wish to win larger market share. Besides them, there are also many new independent implementations and the number is increasing. Standards like UPnP and Zeroconf are developed for Internet based protocols such as HTTP, TCP, IP, or UDP, they are described as IP-based architecture and most of the applications are designed for web technologies. Such an orientation makes the available standards for Plug and Play quite unable to be directly integrated into the environment of industrial automation.

There are also some other limitations for the available standards, e.g., Zeroconf and UPnP. Zeroconf is aimed to handle the communication in a horizontal direction and the main task is IP-based networking oriented. UPnP is facing to a more vertical, device specific direction, which lead it to more complicated configurations as more applications are being developed. Another factor is that, because of the complexity of various applications from different vendors, it is not easy to implement a new application for a device based on such standards, not mentioning some of the standards are not even open to other development environments (e.g., Bonjour from Apple Inc.).

Based on such considerations and facts, there is a demand of developing a Plug and Play concept for industrial automation. This concept should support more network protocols and industrial automation standards.

The reason that this work is developing a new concept with its own mechanism instead of using directly the software developing tools to apply a communication channels distribution framework for devices, is that such an approach can lead to complications. The configurations of channel distribution from different engineering tools can be varied and can lead to a structure that is not open. If such configurations are applied to other development environments, the cost and complexity of reconfiguration can be an issue. Therefore, a concept with its prototypic and generic modules is necessary based on such a fact.

The essential part of a Plug and Play concept for industrial automation is to establish communications automatically among devices. In this chapter, the

requirements of such a concept will be brought forward, and a concept analysis will be discussed. Afterwards there will be an analysis about the main modules developed in this concept: **Channel Manager**, **Communication Manager** and **Generic Communication Module**. Based on such main modules as functional components for the system, a mechanism of configuring channels as well as possibilities for different scenarios that the main modules have to handle for communication partners will be discussed. At the end of this chapter, there will be a brief summary about the concept and its mechanism.

3.1 Requirements Analysis

In order to avoid complexity in modification and raise the configurability of industrial automation in manufacturing process, as described in the previous chapter, the requirements of industrial automation system have been much elevated to satisfy the need of increasing number of intelligent devices and a more flexible system. A flexible system should offer two characteristics: portability and interoperability. Portability is the ability to effectively move programs or services from one type of system or machine to another. Interoperability is the ability for existing programs and services to communicate effectively with new peers [vGH04]. Such definitions lead the communication system in industrial automation system to a new goal, which is to let the devices already existing in the industrial automation system communicate with each other. When a new device is attached, it should also obtain such a configuration of communication. To achieve this goal, there are two processes that the configuration of devices should be able to cover.

Firstly, the devices attached to the system should be easily discovered, as well as they should be configured correctly to make them fully functional. Secondly, it is possible that the devices are attached into the system from various network mediums using different network protocols, and some devices support more than one network protocol. Such a fact leads to an approach to find and to select an appropriate protocol based on the communication mediums between two devices. Only after the network protocol for transportation is correctly found and assigned, it is then possible for the devices to exchange data packets via a certain channel. Finally, the devices should be easily detached when their presence is no longer required.

An example can be seen from the Figure 3.1. Devices are connected into the industrial automation system via communication mediums, and there can be various network protocols supported by the communication medium. In Figure 3.1, there are Device 1 and Device 2 connected with Ethernet, while

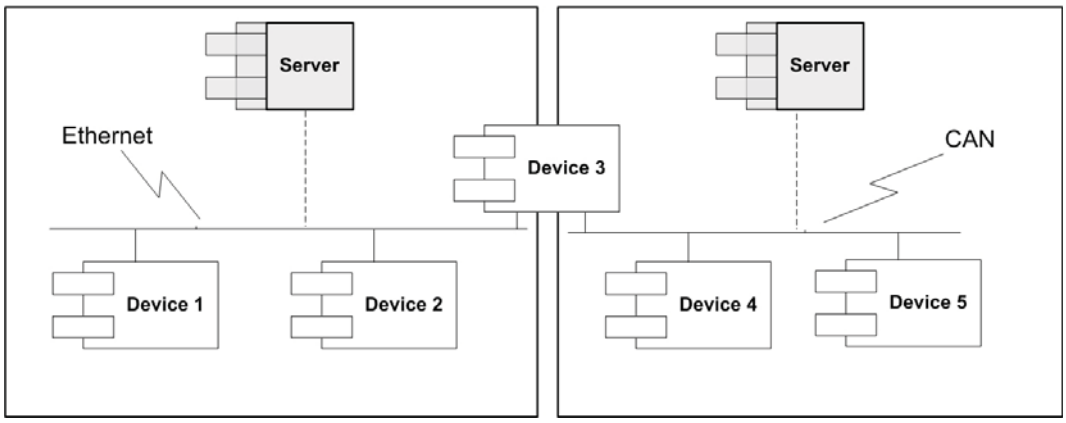

Figure 3.1: Devices in Industrial Automation System

Device 4 and Device 5 are connected into the network via CAN, and there is also Device 3 connected to both Ethernet and CAN. As Ethernet or CAN is providing technology and standard for network protocols, it is possible to have multiple protocols for the channel to choose from when the communication is about to be established among these devices. When the communication among devices is required, the communication has to choose one appropriate network protocol for the channel. Other than communication between two devices, e.g., Device 3 and Device 1, or Device 3 and Device 4, the communication among devices could also be one-to-many if the selected protocol is using broadcast or multicast as communication pattern. When Device 3 is trying to communicate with Device 1 and Device 2, the data packet transmission should be only applied with a network protocol based on Ethernet: TCP, UDP, EtherNet/IP, Powerlink, or ProfiNet. Even though Device 3 is also supporting other protocols based on CAN (e.g., CANopen, DeviceNet, ProfiBus, CANaerospace), the communication among Device 3, Device1 and Device2 in Ethernet has to filter out the protocols from CAN and only choose one protocol for Ethernet transmission. So there is a need for a Plug and Play concept that to develop mechanisms that can select appropriate network protocols in order to establish communications for devices.

Another complexity that in industrial automation can occur is that in one integrated system with multiple technologies applied. For a system with many field devices, there can be sub-systems or sub-applications run by different engineering environments using standards from IEC 61131-3, IEC 61499, and Multi Agent System (MAS), etc. Therefore, a Plug and Play concept for industrial automation has to provide a generic possibility that has compatibility among all different technologies without complicated reconfigurations.

3.1.1 Communication Characteristics

In order to meet such requirements described above, the focus now is to analyze the communication among devices and to discover what characteristics are necessarily needed and must be considered for the communication in industrial automation.

Participants: For the different cases of communication patterns, there have to be at least two partners standing at both ends of the communication. If the communication pattern is set to be broadcast or multicast, then it is possible that more than one communication participant is available at one end. There is a few definitions need to be declared: Requester and Responder, Sender and Receiver.

Requester and Responder can be seen as the role of the device, while Sender and Receiver can be seen as the character of the communication modules in device. **Requester** describes the communication participant who sends its requests of the communication. In this work, requesting communication means that the communication participant acts as a Requester and requires a channel to be assigned. The communication participant who responses the requests is the **Responder**, with whom the Requester will communicate via the assigned channel. Requester and Responder are all clients, and they represent the roles in the process of establishing the channels. After that, each Requester and Responder will exchange its data via its communication modules. The task of exchanging user data is carried out by Sender and Receiver. **Sender** is the communication module that sends the data, and **Receiver** is the one that receives the data.

The roles of these communication participants are not fixed, which means for either side of the communication, the role as a Requester or as a Responder is interchangeable, a Requester could be a Responder in another communication session and vice versa. It is all depending on the configuration of the network and the communication.

On the other hand, this rule of setting Requester and Responder, and furthermore setting Sender and Receiver has to be applied in correspondence to the communication direction. At the process of assigning the channel, the channel is established between a Requester and a Responder. When the communication direction is predefined for this channel, the transmitting of data packets must be realized with appropriate communication modules. When the transmitting of data packets between two devices is based on a unicast channel, the transmitting direction is bidirectional. For such a case, Requester and its Responder should consist of both Sender and Receiver so that the data can be exchanged from both ways. If the communication pattern among communica-

tion participants is multicast or broadcast, then one end of the communication will be set with communication modules of Receivers receiving data while the Sender sending data packets. For such a case, the transmission of data packets requires one Sender while the other communication participants consist of more Receivers.

The tasks of the sender are to send user data to the receiving terminal. In industrial automation, such a process of sending data and messages means normally to collect and send the user data from the device. Before the data is to be sent via Sender, the message or sequence of messages will be packed as one encapsulated data packet. The content of the data packet has to be operated appropriately for the transmission. The data can be converted, encoded, compressed, or sampled into the form of a packet that can be finally interpreted by the Receiver.

The tasks of the Receiver are to receive this data packet from the Sender and to decode the data packet so that the Receiver can process the data. As the inverse terminal to the Sender, the Receiver is the destination where the data packet should be sent, which means the Sender has to be aware of the destination address of the Receiver. The addressing method can be varied from the communication pattern in between. As for some possible communication patterns as broadcast, the destination address (Receiver) is not crucial. As for multicast, the Receiver should be assigned with a group address to make sure the data packages will be delivered to multiple destinations.

Transmission medium: After the Requester and Responder are assigned as participants for the service of communicating, there will be communications among their Senders and Receivers. Transmission medium is the carrier, which can be used to transfer the data between the Sender and the Receiver. In this work, the term **Channel** will be used to describe such a Transmission medium. As described in classical works such as [Sha48] and [SI98], the focus is about interpreting analog and digital signals. A channel is left merely as a physical medium such as a pair of wires, a coaxial cable, or a band of radio frequencies, etc. As for the concept of this work, a channel is not a physical medium but rather an abstract term that is used to describe a virtual transport medium. The difference between a data packet and a channel is that a data packet is used to encapsulate all the data, and channels encapsulate not only the data, but also the whole transmitting mechanism between the sender and the receiver. A channel can be seen as a container or a "pipeline" which consists of the configuration results, transport mechanism, addressing, and protocols.

A channel can be varied from one another based on three attributes:

- **Channel Identifier**: the name of a channel is the unique identifier that

indicates the basic attribute of a channel. It can be a description of the task of the channel, e.g., `Motion`, `Temperature`, etc., it can also be a description for the communication partners who use this channel, e.g., `Conveyor1_Conveyor2`, or `Divertor16_Lightsensor9`, etc.

- **Communication Pattern**: a channel is a medium that can indicate the communication patterns with its transmitting direction based on the communication pattern. For example, unidirectional channel is the channel that consists of single direction transmitting, and bidirectional channel is the one that supports bidirectional transmission. Different directions of channels are determined by different assigned network protocols.

- **Content**: the message content is the actual messages or data to be transferred between the sender and the receiver in communication. The content can be parameters from devices, status update, variables, or parameters for the communication specifications and requirements (e.g., QoS).

Transmission protocol: the transmission protocol for a communication can be seen as the tool of transporting used by a channel. When the transmitting medium, which is a channel, is established, a network protocol should be assigned to this channel in order to inform the Sender and the Receiver which transport protocol to use for sending and receiving the data packet. For the devices, there can be many network protocols supported based on their communication medium and protocol standards. The network protocols of the channels will be chosen from the supported protocols and must meet the requirements of the channels.

As can be seen from the Figure 3.1, it is possible for the devices that they are connected with multiple network protocols. If both of the communication partners can provide multiple supported protocols, then the channel has to be established with only one matched protocol that can be used by both of the Requester and Responder.

3.1.2 General Framework

At this point, a systematic overview can be concluded according to the primary analysis of the concept: each device is connected into the industrial automation using different network protocols, and they will exchange data via channels. When Plug and Play functionality for each device in such a network structure is required, every device should be seen as a Client. These clients will send their

requests to a central component where they can be assigned with channels for communication. This server-like component provides the configuration of the channel in corresponding to the requests of a channel from devices (clients). In this work, the term **Device** is used to describe the physical component as a field device in the industrial system. The term **Client** and **Server** is used for the description of the device in a abstracted way.

When the device is requesting communication with other devices, the role of the device is Requester. There the server provides the service of communication with configuring channels for devices and assign the channels for devices.

When the devices are connected via various network protocols, they are supposed to be connected via routers in order to make sure they can be connected with other networks. When a routing method is applied in the network system, it is possible that all communications among devices can be managed by a centralized server. But in this work, such architecture will not be discussed because of two considerations. At first, a system architectural structure with routing methods applied for devices is not the focus point of this work. There are many earlier available technical works based on such methods, therefore there is no need to include the part of analyzing possible routing methods in this concept. Secondly, for an industrial automation system, it is not optimized to leave only one centralized server to manage all the communications for transmitting data packages. Complexity of such a structure will have to use a monolithic module that is hard to gain the balance between functionality and simplicity. If a system failure occurs at the centralized server, it will cause a systematic failure upon all devices and the communications among them, which also raises the complexity and cost of maintenance and diagnosis.

Based on such considerations, a system structure with network protocols oriented framework that consists of one server in every individual communication medium is recommended (see Figure 3.1). By attaching a server component into the system, it will not only lower the risk of systematic communication failure, but will also make it easy to diagnose by splitting up a centralized server into individual servers based on different network protocols and medium when an error occurs. For each communication scope, a server will establish channels only for the clients available in its range.

In this case, the system can just focus on that single prototypic server and configure it. It will be also easy to establish communication among devices when new devices are connected into the system by simply duplicating the server module into the new network once the server is correctly configured.

3.2 Conceptual Structure

3.2.1 A Three Layer Structure

By introducing a client-server framework, it is clear to see there is a need to develop a system with a client-side module and a server-side module. Such a structure must consist of Requester, Responder, Sender and Receiver, whose tasks must be fulfilled by applying this structure. Therefore, a three-layer consisting of a server-side module, a client-side module, and generic module is introduced (see Figure 3.2).

Server-side Module: the module at the server side should be able to receive requests from clients, and configure channel assignments according to the requirements of the channel, which are included in the request message. Such a task brings certain requirements for this server-side module:

- This server-side module must be a fully functional server, which means it has to include a communication interface to receive requests and send responses.
- This module should have algorithms within that can be automatically operated to make functions such as selection, filtering, matching, reservation, and addressing possible.
- This module should have the ability to control the communication traffic when a massive amount of requests from clients reaches the server (there could be over thousands of field components and devices in an industrial automation system). This function has to be also extended when it deals with channel distribution, if the priority of assigning certain network protocols is crucial. Such a consideration requires the server-side module to use Quality of Service (QoS) to filter the incoming requests on registering and channel distribution.

Client-side Module: the client-side module should be able to act as Requester and Responder. It carries out the task of sending requests for channels to the server-side module, and receiving the response from the server-side module afterwards. At the step of initialization when the system starts, all client-side modules should be able to acquire device-oriented parameters for their requests.

Generic Module: the client-side module needs sub-modules, which work as Sender and Receiver, to exchange user data among other clients. In order to apply such a function, the client-side module should consist of communication modules. These communication modules are generic modules that are

initialized in devices when the system initializes. By giving messages of typification, these communication modules will be typified accordingly as Sender and Receiver based on the assigned network protocol.

In order to meet these requirements and descriptions above for the server-side module and the client-side module, a system with a three-layer structure is introduced. Two management modules and device-oriented communication modules are applied for this work: **Channel Manager** as a server-side module, **Communication Manager** as a client-side module, along with **Generic Communication Module** as generic module at client-side for communication.

3.2.2 Channel Manager

The Channel Manager is a server whose main function is to manage channels and eventually distribute the channels on communication mediums according to the requests from the clients. It can carry out the tasks of managing protocols and channel requests. From the description of transmitting the data packet between the server-side module and the client-side module indicates that the Channel Manager should at least include the following blocks with management function:

- **Identifier Management**: this function should be able to manage requests of clients based on various identifiers such as their `Client Names` and `Channel ID`.
- **Protocol Management**: a function that can select and assign the network protocols according to the requests from the clients.
- **Parameter Management**: this function should be able to configure the parameters for the channels from the `Parameter Supported` in requests. This management function could be extended to many more sub-functions rather than only collecting the available parameters from clients according to the requirements of devices. The parameters will be assigned with new values if necessary. There should be also a function for QoS parameter management in Channel Manager, that the Channel Manager can manage QoS priority level on the incoming requests and for channel distribution. The result of the assigned parameters for channels will be put in the response as the `Assigned Parameters`.

This structure with management functions is a general overview to describe which management functions should consist of for the Channel Manager (Fig-

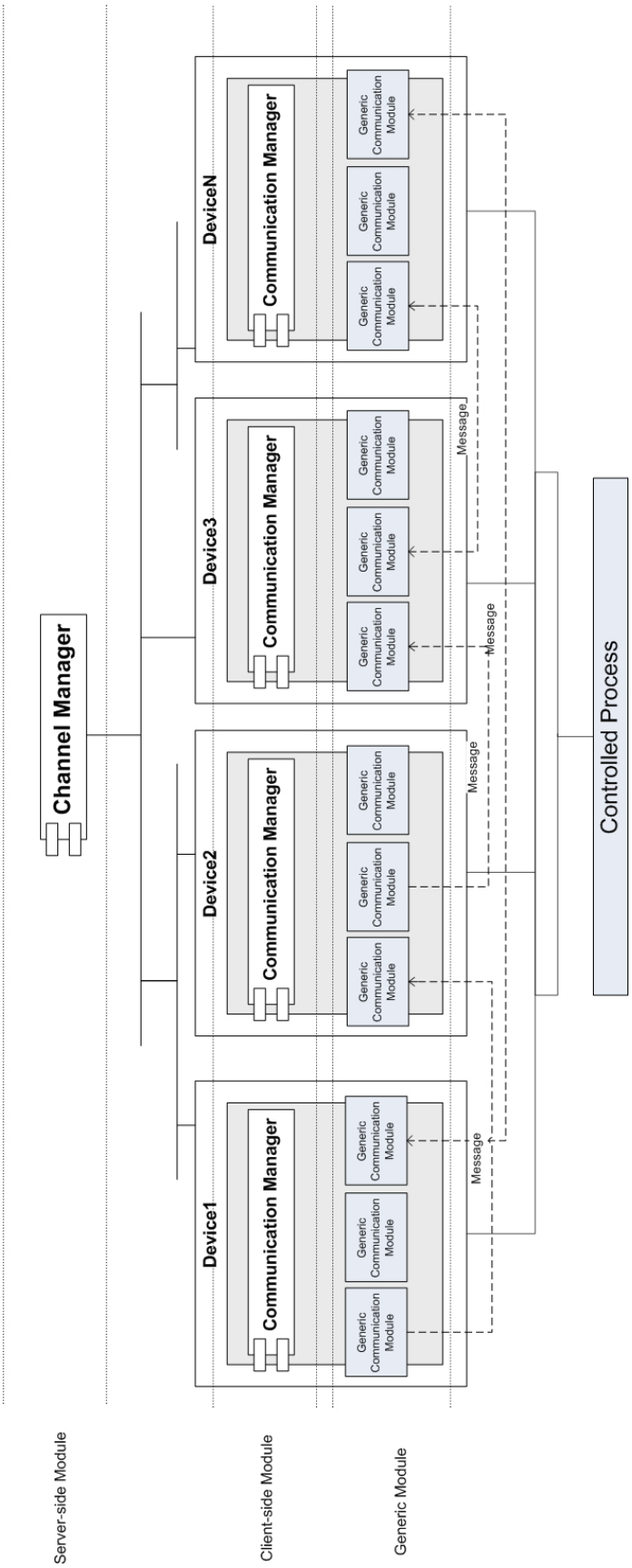

Figure 3.2: Overview of Major Components in the 3-Layer Structure

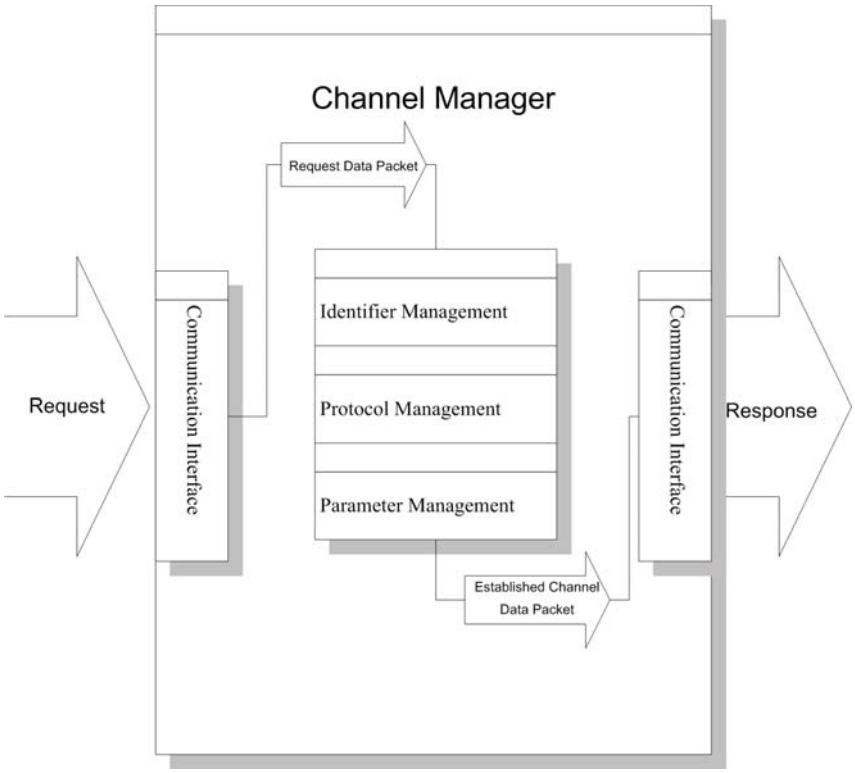

Figure 3.3: Channel Manager

ure 3.3). They should not be operated as individual processes, each management function could be available through the whole mechanism of distributing channels and are able to work with other functions cooperatively.

The data exchanged between the Channel Manager and clients (Requester and Responder) are message-oriented. The main concerns of such messages is their usability, which means that the content of a message needs to be specified so that it can be used by the Channel Manager, Requester and Responder for interpreting.

The messages between Channel Manager and device can be divided into two kinds, Request and Response, based on its original operator. **Request** is the message from all clients to the server asking for a channel (a request for communication service). **Response** is the message, which includes the information of assigned channel from server to clients (a response to a service request).

The content of request message and response message will determine what information the device should acquire for a channel. According to the description about the characteristics of a channel in previous section, such attributes

are needed for configuring a channel: **Channel Identifier**, **Client Identifier**, **Addressing**, **Protocols**, and **Parameters**. All these attributes can be seen as data fields in the message, each of the attributes can include its own specifications.

- **Channel Identifier**: when the Channel Manager starts its process of distributing channels, the first attribute it needs is the identifier of a channel. It can be named such as Channel '`Motion`', Channel '`RFID`' to show the attribute based on the functionality of the channel.

- **Client Identifier**: Client Identifier is the identifier of the device. For a process of channel distribution, the Channel Manager must always know the relationship between the channels and their corresponding clients. Therefore a device needs a unique name for the communication to show the server and other devices its presence. A client will indicate the Channel Manager and other clients by using its client identifier, for example, Device '`Dev3`', '`Conveyor1`', or '`Valve9`'.

- **Addressing**: the attribute of addressing of a channel means that the Channel Manager needs to assign channels with a method to tell the communication participants where to locate their partners. The requirement of addressing is based on the network protocol of a channel. If the assigned network protocol for a channel requires network addresses for the communication participants, the Channel Manager will assign an address to the client. The network address is useful only for the network protocols that support the Internet Protocol Suite. For such a case, the Channel Manager needs to distribute communication interface addresses for the Sender and Receiver. For the network protocols that do not require a network address for addressing, other possible methods should be specified according to the requirement of the network protocol.

- **Protocols**: in the request message between Channel Manager and Requester, the field of Protocols are a field that shows all supported protocols from a client. As can be seen from Figure 3.1, it is likely that an individual device can support more than one network protocols based on different communication mediums and network standards. In this case, the Channel Manager has to find a match for supported protocols among all the requests from clients that ask for a common channel. Channel Manager will sort these protocols and use its protocol management function to assign one network protocol for the assigned channel.

This assigned network protocol will be sent from the Channel Manager to the communication partners.

- **Parameters**: the Parameters field is included in the request message and the response message. Channel Manager will use the supported parameters from the Requester to configure the distribution process of channels, and the Channel Manager will put assigned parameters in the response message. Each parameter in the parameters field is an entry with its own semantics. In the request messages, some necessary parameters will be given by the clients after its initialization. As for the response message, the Channel Manager will also send necessary parameters as configuration of assigned channels to the Requester and Responder, so that the Communication Manager can use them to give typification messages for Generic Communication Modules. There are important parameters need to focus: `QoS indicator` and `Lease_Time`. `QoS indicator` contains QoS parameter as QoS priority, so that the Channel Manager can manage the traffic control by sorting their QoS priority levels when a massive number of requests are coming at the same time to the Channel Manager. Such a parameter for QoS can also be applied when the Channel Manager needs to sort network protocols in its own protocol management. Another extended `QoS indicator` is `QoS_Kind`, as developed in the work of B. Voglmayr [Vog07]. There are different kinds of QoS that should be handled in industrial automation systems according to different communication structures. These scenarios are protocol depended. For example, some `QoS_Kind` parameters are only necessary to be paid attention to when the communication is based on a client-server structure, some are necessary to be handled when the communication is based on a publisher-subscriber structure. The `QoS_Kind` parameter will be returned by Channel Manager according to the requirement of the assigned protocol. As for `Lease_Time`, it can be filled with a parameter of leasing time for the channel. Channel Manager can put this parameter in the Response Message. When the leasing time is out, the channel will no longer used by the devices and will be distributed again as available channel in the selection list of the Channel Manager.

Based on the analysis of attributes of channels that the Channel Manager manages, the Channel Manager needs to be designed with an algorithm that contains lists for selection, comparison, management and filtering. The Channel Manager will use these functions during the whole channel distribution process as a Plug and Play mechanism.

3.2.3 Communication Manager and Generic Communication Module

Communication Manager (see Figure 3.4) is the client-side module, which provides the ability for devices to support the Plug and Play concept. Each device has exactly one Communication Manager. It carries out the management function of handling established channels for the Generic Communication Modules.

Generic Communication Module is the communication module in the device, which carries out the task of exchanging user data with other communication participants. There can be many channels for a device to communicate with other devices. Therefore, each device will contain more Generic Communication Modules to handle all the channels (see Figure 3.2).

Just like Channel Manager, the Communication Manager has its own communication interface to send and receive messages. But the communication interface that Communication Manager needs is more than just one interface between Communication Manager and Channel Manager.

The Communication Manager obtains the parameters for the request of channels from the device itself. The initialization of the Communication Manager is applied by registering all necessary parameters for requests at the Communication Manager. After this initialization, the Communication Manager is able to send requests to Channel Manager.

From this aspect, the communication interface of Communication Manager must be divided with two different orientations: One communication interface is the interface between the Communication Manager and the Channel Manager, which is in charge of sending request and receiving response with established channel information from the Channel Manager. Another communication interface is the interface between the Generic Communication Module and the Communication Manager.

At the step of initialization, the Generic Communication Modules will register themselves at Communication Manager with the attributes as parameters from the device, this process is **Registration**.

All necessary attributes of the device such as `Channel Identifier`, `Client Identifier`, `Protocols` and `Parameters` should be saved in Communication Manager after the initialization of the device before it sends a request for a channel to the Channel Manager.

Such a requirement makes it necessary that the Communication Manager needs a Generic Communication Module to build a connection between it and the device.

Generic Communication Module should be working as a generic module in each device, so that the device can use it to communicate with other communi-

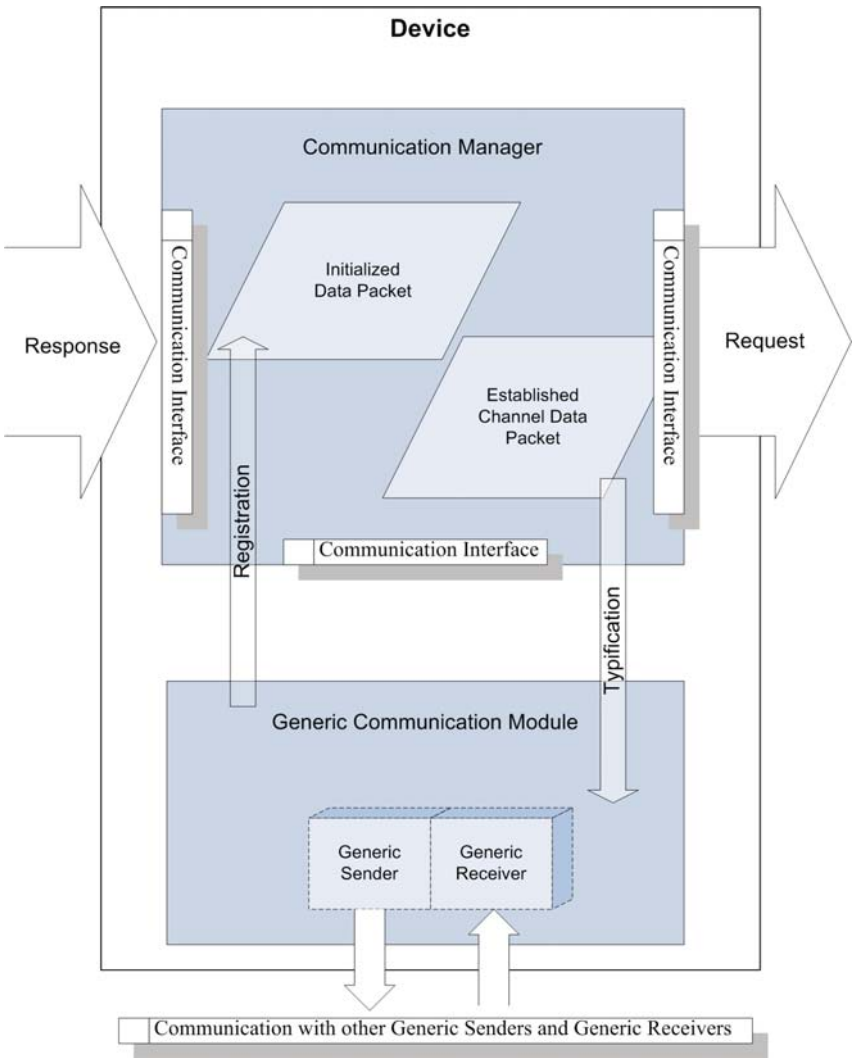

Figure 3.4: Device View: Communication Manager and Generic Communication Module

cation modules in all communication partners. The reason that these modules are called "generic" is because at the initialization step, they are not typified into the correct form of communication module according to the communication patterns at the registration. Such information can only be acquired according to the established channel data packet from the Channel Manager.

According to configurations provided for the established channel, Communication Manager should configure appropriate communication patterns along with its protocols for the Generic Communication Modules. Only after being correctly configured, the Generic Communication Modules can exchange data with other Generic Communication Modules from other devices.

For example, if the established channel requires a communication module with Client-Server structure, the Sender and Receiver should be given configurations that they will construct a relationship of Client and Server. Before this configuration is given, the Requester and Responder contain only Generic Sender and Generic Receiver. After the channel is established, the Requester and Responder should contain Generic Communication Modules as Client and Server.

From the aspects of Communication Manager and Generic Communication Modules, this process of configuring the Generic Sender and Generic Receiver is that Communication Manager gives messages of configurations to Generic Communication Modules. The Generic Communication Modules will be typified into the correct communication modules (Client and Server for this example) so that the user data can be exchanged. This process is called **Typification**.

The management function of the Communication Manager is applied after receiving the response from Channel Manager. The Communication Manager must determine which information should be used by Communication Manager itself and which information should be used for the Generic Communication Modules to configure the channels.

Based on the whole process of analyzing the Channel Manager, Communication Manager and Generic Communication Module, there is more types of messages available. Other than Request Message and Response Message for Channel Manager, there are other messages for Communication Manager and Generic Communication Module: Registration Message and Typification Message (see Figure 3.5).

For each of these messages, certain data and configurations are contained within. Generic Communication Module will use the data in Registration Message to initialize the Communication Manager. By using the Registration Message, the Communication Manager will send the parameters as Request Message to the Channel Manager. The Channel Manager will assign the chan-

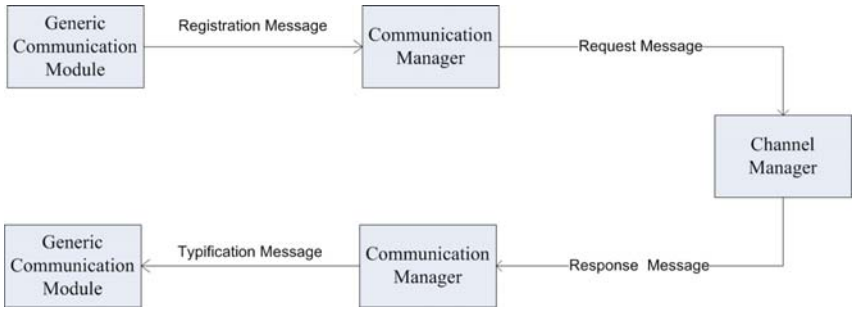

Figure 3.5: Flow of Messages

nels back to Communication Manager by sending a Response Message. After receiving the Response Message, the Communication Manager can send a Typification Message back to Generic Communication Module and typify the Generic Communication Module into required forms of Sender and Receiver. Therefore, there should be a complete mechanism developed for distributing the channels and configuring the Generic Communication Modules, which can support the Plug and Play concept.

3.3 Mechanism

A general overview of distributing a channel can be seen in Figure 3.6, it shows that after **Initialization**, the whole mechanism can be described as **Registration**, **Assignment**, and **Typification**. In this chapter, a detailed analysis of each step will be discussed.

3.3.1 Initialization

At the step of initialization, the device will register its Generic Communication Modules at Communication Manager. The device will register its data via Generic Communication Modules at the Communication Manager, e.g., `Channel ID`, `Client Name`, `Protocols Supported`, and `Parameter Supported`. The initialized Communication Manager will hold a list of these data to be sent and also a list for the data to be received. The list for received data will be used by the Communication Manager to interpret the Response Message from the Channel Manager for typification. As a Plug and Play concept for industrial automation, the Communication Manager can be designed with the function of holding its own data types for the system, i.e., `Datatype_Send(CIEC_INT, CIEC_BOOL, etc.)`, `Datatype_Receive(CIEC_INT, CIEC_BOOL, etc.)`. The

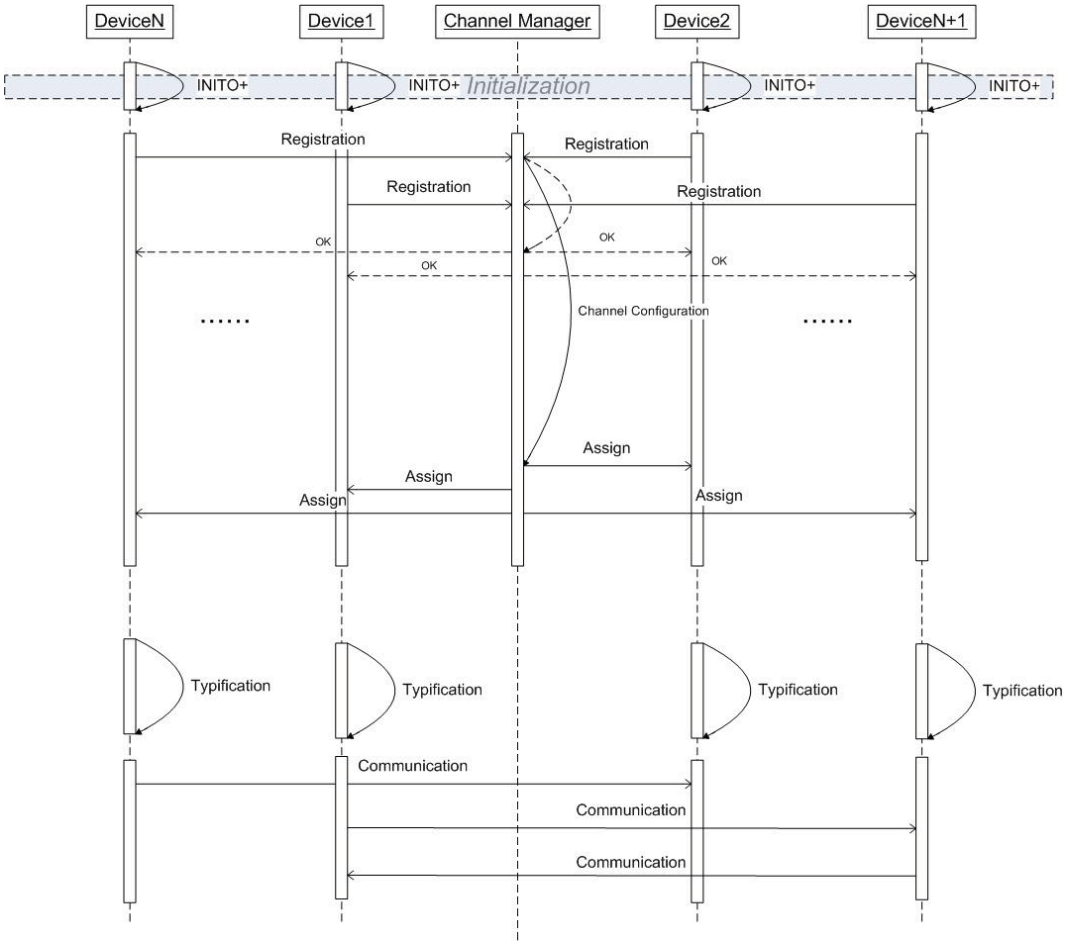

Figure 3.6: General Overview of Mechanism

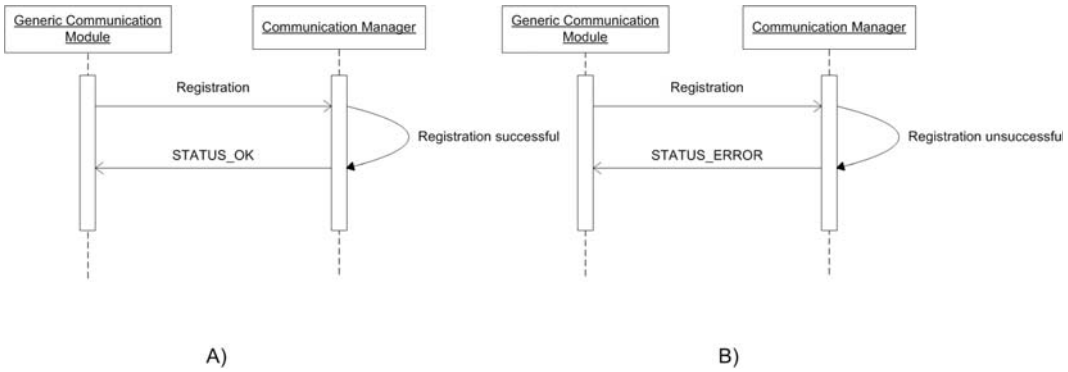

Figure 3.7: Initialization when A. Initialization Successful; B. Initialization Unsuccessful

data types are depending on the data that are necessary for the initialization. The `Datatype_Send` and `Datatype_Receive` will be used between the Channel Manager and the Communication Manager at initialization and typification.

During the initialization process, the Communication Manager should have a return message of `STATUS` to tell the Generic Communication Module if it has successfully initialized. A successful initialization should be finished with all parameters above registered at the Communication Manager. If the initialization is not successful, the whole procedure should start again. Here at the step of retrying, a `Retry_Time` should be set, this `Retry_Time` can be set as retry times or a time limit that should not be too many times or too long to hold the device in initializing status. If the value of the `Retry_Time` is reached, the `STATUS` should be returned with a `ERROR` message back to the Generic Communication Module so that the user would know the initialization of a Communication Manager is unsuccessful because of a defect function within the device (see Figure 3.7).

3.3.2 Registration

Once the initialization step is finished, which means that now the Communication Manager holds all the attributes for a channel request, the Communication Manager will start to send requests for channels via its communication interface to the Channel Manager. At this step, the client should be discovered and registered at the server. A general sequence of this process can be seen in Figure 3.8.

When the Channel Manager receives a request for a channel, it means that it receives a data packet with fields of `Channel ID`, `Client Name`, `Protocols`

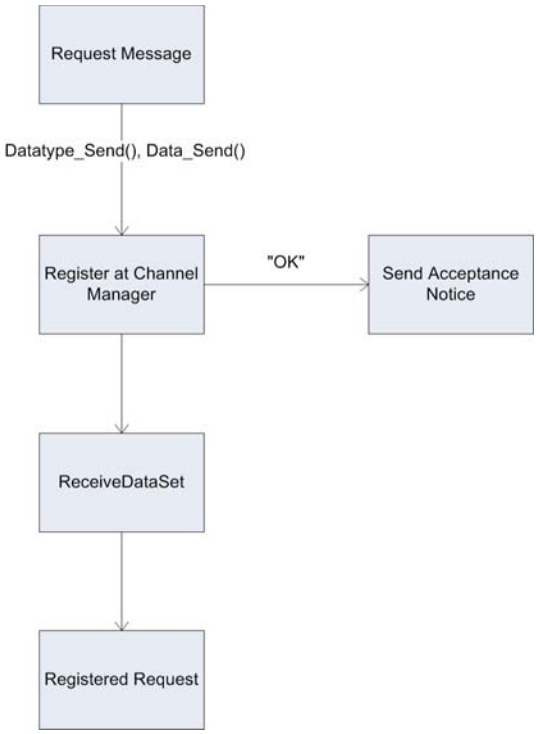

Figure 3.8: Registration

Supported, and Parameter Supported. The Channel Manager will interpret the data with its data types and data content. Here the client is registered at the Channel Manager. The Channel Manager will send an acceptance notice back to the Communication Manager with a confirmation of STATUS_OK.

After the request is registered at the Channel Manager, the Channel Manager will put the data it receives into a data set for received data. This set of data will be used for saving the incoming data for further channel distribution.

3.3.3 Assignment

The first task for the Channel Manager at the step of channel assignment and distribution is to sort the requests. The Channel Manager will apply QoS parameters of the request to decide if it is necessary to handle this incoming request at once or there is another request with a higher priority. After that, the Channel Manager will decide whether to set this request as a Requester or a Responder. In order to do that, the Channel Manager will use its identifier management function to sort the requests. In this concept, the Channel Manager uses Channel ID as the sorting criterium.

By using Channel ID as the data for identifier management, the Channel Manager can differentiate the requests. The roles of the clients will be set as the Generic Interface Type of Requester or Responder. If a request is set as a Requester, there is no match with another client asking for the same channel can be found. This request should be put into a list for new requests, waiting for the Responder. When Channel Manager finds a match between the saved request and a new request, the Channel Manager will set the new request as the Responder for the saved request. The Channel Manager will assign a channel between this Requester and Responder since they are requiring the same Channel ID. The reason that the Channel Manager should use Channel ID as the evidence for sorting the requests is that an identical ID is the criteria for requests to determine which communication participants are willing to have the communication with the same type and content. Therefore the Channel ID is the primary key for the Channel Manager to sort requests that have the same requirements for communication (see Figure 3.9).

Based on the communication medium and requirements of the devices, it is possible that waiting time for channel distribution should be considered. For such a case, it is necessary to consider that until the saved request is matched with a new coming request, the client which sends the saved request will hold the status as Waiting. During this period, the device is held as Registered_Request instead of being configured. The Waiting status can be set with a Timer_Waiting. If Timer_Waiting times out, the device will simply

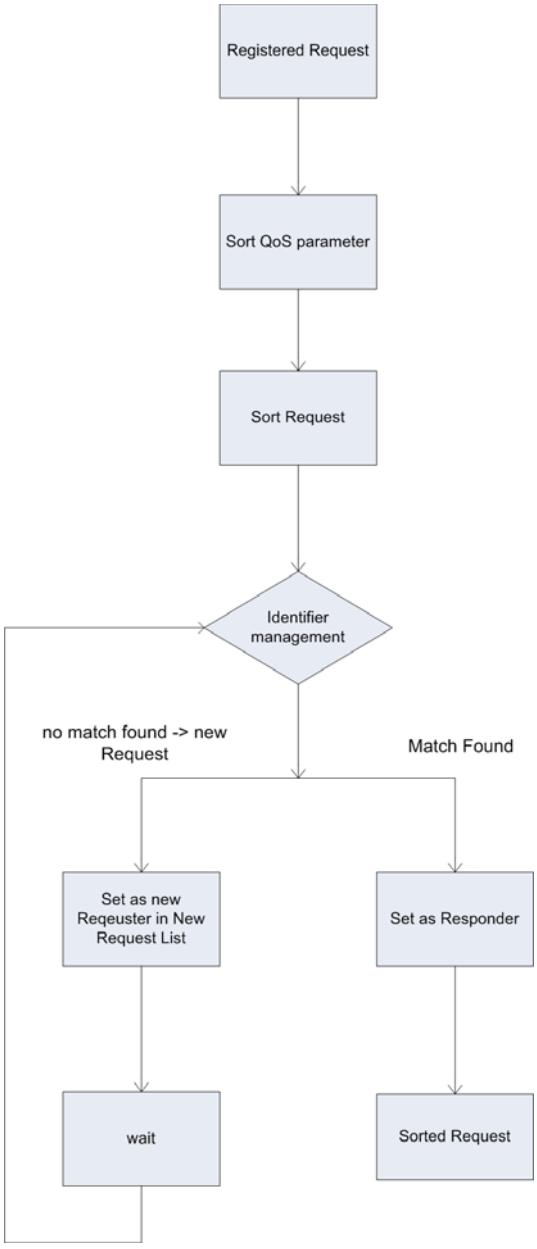

Figure 3.9: Identifier Management

quit the process of requesting a channel and send a new request for channel.

Assignment of a configured channel for the request must include functions of comparing and matching. Comparing is using the parameters in the data packet from requests for sorting the requests, matching is the result of search and the result can be used by the Channel Manager for further procedures.

Assignment of a configured channel for corresponding request requires the Channel Manager to apply the protocol management and parameter management after the requests are sorted based on identifier management. As for protocol management function, the Channel Manager will check if there is a matched protocol can be found among the supported protocols from requests. All supported protocols from requests can been seen as a collection protocol IDs and they can be put into a `Protocol ID List`. The Channel Manager can use this list and pick out the ID of the protocol for the protocol selection. There are three possibilities after the comparison:

- **No protocols matched**: in this case, the Channel Manager cannot find any match for the request in its `Protocol ID List`. This case means that even there are other requests which are also requesting the channel with the same `Channel ID`, but none of them supports the same protocols as this request does. Now the Channel Manager must put this request with its supported protocol ID into the `Protocol ID List` and wait for the next coming request. Here the Channel Manager should return an `ERROR` message to the Generic Communication Modules of registered Client so that the status of no matched protocol can be informed.

- **One protocol match found**: if only one protocol among the supported protocols for requests can found in the `Protocol ID List`, then this protocol will be used directly as `Protocol Assigned` if the QoS parameter of the protocol is also fulfilled.

- **More than one protocol match found**: the Channel Manager must determine only one protocol as `Protocol Assigned` for the response. Here the parameter management of the Channel Manager should be applied. If more than one matched protocol are found in the `Protocol ID List`, the Channel Manager can use QoS from each protocol as a selection criteria. Each protocol will set with a QoS parameter in correspondence with the request. If the request contains a higher priority level for its QoS parameter, then the Channel Manager must assign a corresponding protocol according to the QoS parameter of the request (see Figure 3.10).

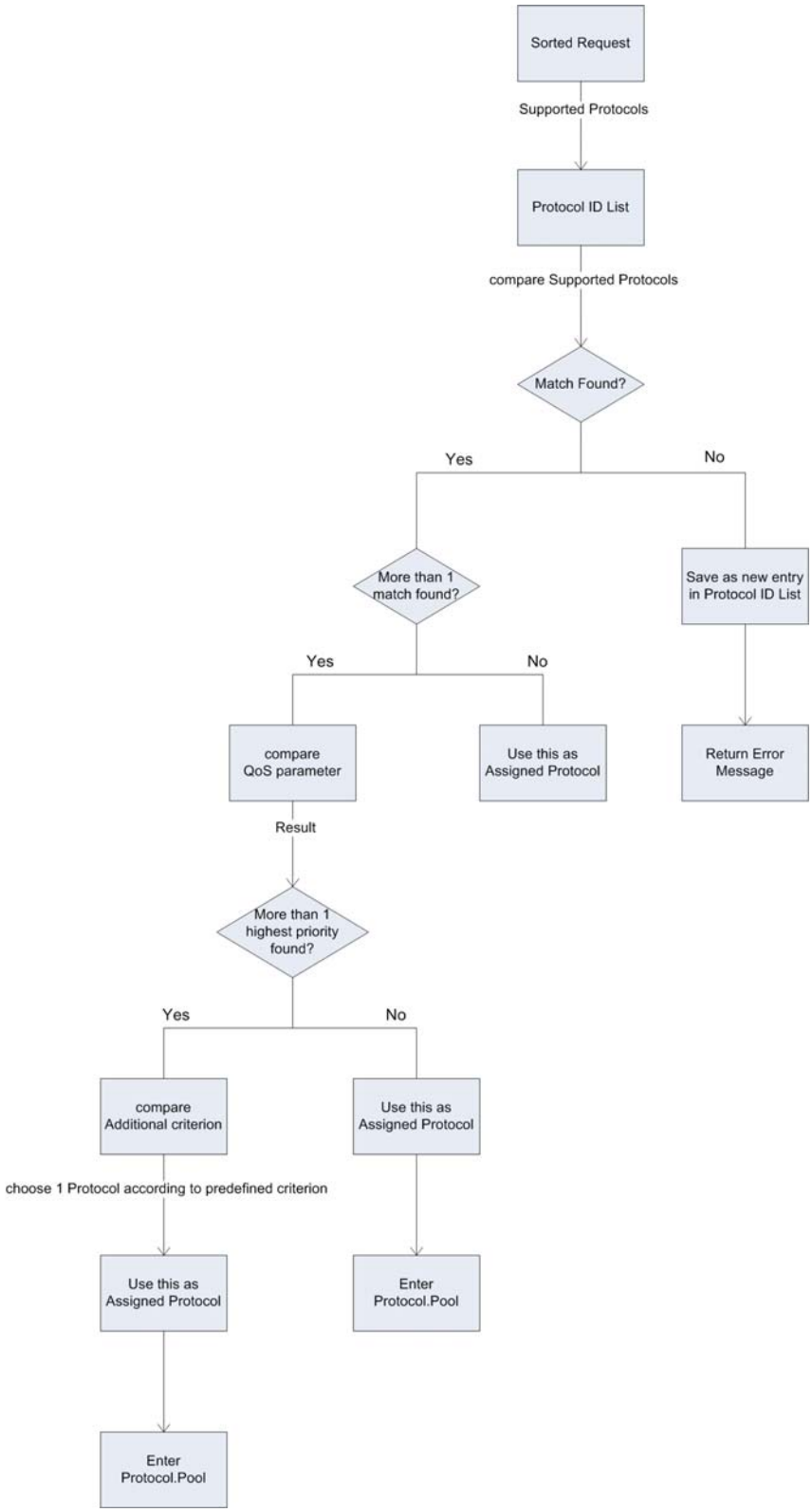

Figure 3.10: Protocol Management of Channel Manager

After getting the result of `Protocol Assigned`,the Channel Manager will use the corresponding protocol and choose channels to distribute. Here for each protocol, there should be a `Protocol Pool` defined. It is a pool for channels according to their corresponding protocols. The channels that are going to be distributed are saved in each `Protocol Pool`. Each channel is saved along with their Channel ID so that the Channel Manager can simply choose the correct channel to distribute along with this assigned protocol. Other parameters, e.g., `Lease_Time`, will be configured according to the corresponding `Protocol Pool`. The Channel Manager defines the methods of parameter configurations based on the result of the selected protocol and its `Protocol Pool`.

The `Protocol Pool` can be used more than just a pool of storing channels. There can be additional selecting criterion defined in each `Protocol Pool`. Other parameters and implementation methods as requirements of channel selection can be also defined in the `Protocol Pool` for channel distribution, e.g., `Lease_Time`.

When all the configuration is complete according to the corresponding `Protocol Pool`, Channel Manager can send the assigned `Channel ID`, `Client Identifier`, `Generic Interface Type`, `Assigned Protocol` and `Assigned Parameters` as a Response Message back to the clients. At the side of the clients, their Communication Managers will receive the Response Message and start the process of typification on the affected Generic Communication Modules.

After sending the response packet back to all the clients, the Channel Manager can add this packet as a configuration record of the assigned channel to a Assigned Channel List and keep it as a history record. Correspondingly, there will be only free channels available in the `Protocol Pool`. At this point, the clients who sent the requests will no longer need the presence of the Channel Manager as long as they have already a successfully configured channel. The Channel Manager will step out of the channel distribution process among these devices until the leasing time of the channel is over and the devices require new channels.

3.3.4 Typification

When the Channel Manager finishes the configuration of the channel and sends the Response Message to the Communication Manager of each client, the Communication Manager will start to process the Response Message it receives. The Communication Manager will now decide how to use the data within the Typification Message and establish communication with other communication modules that present the other communication partners.

When the Channel Manager sends the Typification Message to the Commu-

nication Manager, it sends the message to all clients since the communication structure between the Channel Manager and Communication Managers are client-server. Thus the Communication Manager needs an identifier to identify if the Typification Message is sent for itself. The Communication Manager uses the Client Identifier as a to check if the Typification Message is valid for this client.

The other data in Response Message from the Channel Manager will be used by Communication Manager as data for Typification Message:

`Channel ID`: Communication Manager will pass on this data directly to the Generic Communication Module as identifier of the channel.

`Assigned Protocol`: Communication Manager will use this protocol as transport protocol and decide which type of Generic Sender and Generic Receiver it should typify for the Generic Communication Module.

`Generic Interface Type`: by using the assigned `Generic Interface Type`, Communication Manager will decide the role of its Generic Communication Module. The role is protocol and communication pattern related.

`Assigned Parameters`: the assigned parameters will be used as configurations of the channel by Generic Communication Module, e.g., `Lease_Time`, `QoS_Kind`. If the `Assigned Protocol` is a protocol that requires communication interface addresses for the configured channel, the assigned communication interface addresses will be given from the Communication Manager to Generic Communication Modules as well.

When the whole process is finished, the configuration of the Generic Communication Modules is complete. From this point on, devices can communicate with each other through their Generic Communication Modules and established channels.

3.4 Summary

After analyzing the system structure and the mechanism, a systematic approach that can distribute channels for a device is complete. Such an approach includes Channel Manager, Communication Manager, and Generic Communication Module as components, and can establish communication among devices via steps of Registration, Assignment, and Typification.

As an overview of the whole structure, there are different types of messages applied:

- `Message_Registration`
- `Message_Request`

- `Message_Response`
- `Message_Typification`

Within these messages exchanged among Channel Manager, Communication Manager and Generic Communication Module, various kinds of data are being transferred. The data that discussed in this work can be seen as follows:

- `Channel Identifier`
- `Client Identifier`
- `Generic Interface Type`
- `Supported Protocols`
- `Assigned Protocol`
- `Supported Parameter` (e.g., `QoS_Priority`)
- `Assigned Parameters` (e.g., `Communication Interface Addresses`, `QoS_Kind`, `Lease_Time`)

This structure develops of a Plug and Play concept that devices can be communication function enabled with fully functional discovery and automatic channel assignment. Such a structure for the concept is open and flexible. It does not only support the new standard IEC 61499 for distributed industrial automation, but also supports the previous standards (e.g., IEC 61131 with central software module). Because the Channel Manager can run individually as an application crossing different devices, and the modularization conceptual structure can be easily configured based on various requirements from the user and the field devices. Unlike the available standards of Plug and Play technology, the structure with a Communication Manager and a Generic Communication Module can support multiple industrial network and communication systems.

4 Implementation

Based on the concept, the focus of the implementation is on the key components of Channel Manager, Communication Manager and Generic Communication Module. By implementing these components, a process covering initialization, registration, assignment and typification can be prototypically shown. In this chapter, an implementation approach for the concept will be discussed, and more details of development for each component will be given to provide possibilities of how the concept can be implemented for the industrial automation domain.

The implementation of the client-side modules of Communication Manager and Generic Communication Module is facing the requirements from low-level control (LLC) devices that are physically connected. The Generic Communication Module has to be able integrated into the device and register the message types and message contents to Communication Manager. The standard IEC 61499 provides the most logical system for defining the programming and offers the possibility of linking the devices with agents in high-level control (HLC) for a multi-agent system (MAS). By using such a framework, the integration of a run-time control and communication mechanism can be applied for the LLC devices in cooperating with the Channel Manager. When a new device is attached into the physical system governed by LLC, the Generic Communication Module can give registration message to Communication Manager, and further communication between Communication Manager and Channel Manager can be realized. However, based on the Plug and Play concept analysis, the implementation of such a concept should be designed for the industrial automation domain in general. Hence there is a need to design the interface of each module from the concept to be able to work with not only an IEC 61499 based distributed automation framework, but also with standard IEC 61131-3. The considerations of implementing consistent data types and interfaces have to be noticed so that the approach will be compatible with standard IEC 61131-3. By applying such a conceptual guideline, it is possible to implement the modules by using IEC 61499 development tools and still can make the implementation suitable for IEC 61131-3.

The implementation of Channel Manager is considered as an individual Java application, which simulates the process that different requests with different

configurations sent from clients with their data packets.

As for Communication Manager and Generic Communication Module, there is an IEC 61499 runtime environment FORTE has been developed at the Automation Control Institute, Vienna University of Technology (ACIN). FORTE (Framework for Distributed Industrial Automation and Control-Run-Time Environment) provides its own data types for IEC 61499, and it inherits the definition of data types from IEC 61131-3. The data over communication channel is defined according to IEC 61499 Compliance Profile. The ASN.1 encoding standard is applied so that the datatype is included in encoded data.

This runtime environment is implemented in C++ and enables several IEC devices within one control platform. Based on such a platform, 4DIAC-IDE (Framework for Distributed Industrial Automation and Control) and FBDK (Function Block Development Kit) are used as engineering tools to provide the environment of developing Function Blocks. The implementation of Communication Manager and Generic Communication Module can be applied by constructing a composite network function block that can be developed by using 4DIAC-IDE and then exported to FORTE for programming and configurations. As for FBDK, it can be used for building a set of different human-machine interface (HMI) function blocks to illustrate the Request/Response message exchange process with Channel Manager. Each of the modules can be implemented and tested separately while remaining consistent definitions of the data types and the interface.

4.1 Channel Manager

The general implementation process of Channel Manager is to develop a server with its own functionalities of automatically distributing channels, consisting of its own algorithm for management functions. In order to implement the Channel Manager as a component that can be adaptable for industrial automation, there are certain considerations that the implementation needs to focus.

Messages: according to the concept analysis, the messages for Channel Manager can be concluded as a Request Message and a Response Message. The data that need to be transferred are encapsulated in data packets, the data packets are differentiated based on their transferring directions via the interface, and the data in messages have certain data types. Additional messages such as `STATUS_OK` are considered as messages with status indicators as data.

Interface: the interface between Channel Manager and Communication Manager is considered as an interface that can exchange messages of the Request

and Response. The data types of these messages need to be configured as adaptable for FORTE. Since the concept of a plug and play is developed focusing the industrial automation field, the data types are defined also compatible with IEC 61311-3 data types. From such an interface, Channel Manager is can recognize the messages from either IEC 61131-3 or IEC 61499 environment.

For the Channel Manager, there are two data packets that need to be configured for the interface. A data packet of the Request Message from the Communication Manager requesting a channel, and a data packet of the Response Message to the Communication Manager after Channel Manager configured a channel. The Response data packet will be used by Communication Manager for typification, thus this Response data packet must consist of all the necessary parameters for the Generic Communication Module so that the Communication Manager will be able to use the data and to typify Generic Communication Modules.

The first task to implement the Channel Manager is to configure the sending and receiving interfaces. There are two kinds of data packets for Channel Manager to handle: receive Request data packet and send Respond data packet. The sending and receiving interface can be seen as two different parameter sets. The Receive Parameter Set consists of the data of the Request, and each entry of the parameter set represents an individual parameter that needs to be handled by Channel Manager for establishing a channel. After the configuration, the Channel Manager will use the configured parameters and put them into the Send Parameter Set. As for the receiving and sending interfaces, which transfer the Receive Parameter Set and Send Parameter Set, the communication pattern for the implementation is considered to use the Multicast. By applying the Multicast along with Publisher and Subscriber Service Interface Function Block (SIFB), the number of communication partners does not need to be fixed, which makes it better to show the message distributing ability of the Channel Manager.

The tasks of Channel Manager is to receive the Request data packet, then configure the channel according to the requirement of the Request, and finally return a Response data packet and sent it back to the Communication Manager. There are certain parameters must be fixed for the Request and Response data packets (Figure 4.1 and Figure 4.2).

Among the configurations of the channel from the Channel Manager for the Response data packet, the process of using `Generic_Interface_Type` and `Service_Interface_IDs` should be specified. Channel Manager will firstly compare the `Channel_ID` and check if a matched result can be found. The `Generic_Interface_Type` will be appointed at the same step. After setting the corresponding Communication Manager is either a Requester or Respon-

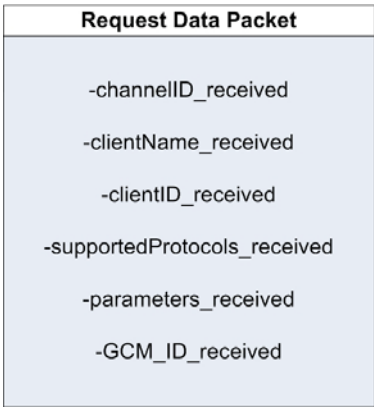

Figure 4.1: Request Data Packet

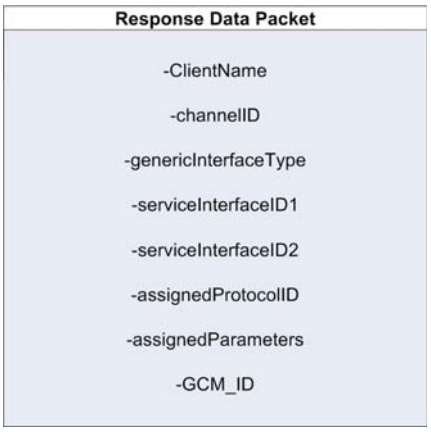

Figure 4.2: Response Data Packet

der, Channel Manager will put this data into the Response data packet so that the Communication Manager can use the data of assigned `Service_Interface_IDs` correctly. Each protocol is defined with a pool, cooperating with its configurations at the Channel Manager. The configurations include the channel reservation, channel QoS parameters, Service Interface ID list, and predefined Parameters. The Service Interface List is a pre-defined list filled with `Service_Interface_IDs`, or a list with available `Client_IDs` from other Requests. Depending on the different communication patterns that the channel supports, Channel Manager will assign appropriate `Service_Interface_IDs` and write them into the data output for Response data packet.

The value of `Client_Name` in the Request data packet will be returned by Channel Manager without changing its original value, but the corresponding data field for `Client_Name` will be identified as `Requester/Responder_Name`. The reason that Channel Manager needs to rearrange is that, as for Response data packet, the using of the data is not exactly the same for the Generic Communication Module as Requester and the Generic Communication Module as Responder. Thus it is necessary to differentiate the Response data packet into two categories: one for Requester and one for Responder.

`Assigned_Protocol_ID` represents the protocol that is to be used for the channel. Each `Assigned_Protocol_ID` can be distributed by Channel Manager after the comparison of the `Supported_Protocol_IDs` from Request data packets. A QoS sorting mechanism is defined in order to sort `Protocol_IDs` if multiple protocols are matched among Request data packets.

Just as the `Service_Interface_IDs`, the `Assigned_Parameters` are distributed by Channel Manager according to the pool of the `Assigned_Protocol`. For different scenarios and usages there can be various pre-defined configurations for `Assigned_Parameters` of channels based on the `Assigned_Protocol_ID` in Request data packet.

`Generic_Communication_Module_ID` (`GCM_ID`) is a special instance of the Generic Communication Module. In an IEC 61499 based structure, the Generic Communication Module will use the `GCM_ID` directing the Response Message to the correct function block. The Channel Manager only needs to receive this data and then return it with its original value in the Response data packet back to the Communication Manager.

4.2 Communication Manager and Generic Communication Module

The functionality of a client for industrial automation is applied as a client Function Block that has device-oriented management functions. As client-side components, the Communication Manager and the Generic Communication Module will be discussed together.

The interface and data types of Communication Manager are configured as the way of Channel Manager in order to meet the definition of data types in IEC 61499. Since IEC 61499 inherits the data type definitions from IEC 61311-3, there is no problem for integrating the Communication Manager into the general industrial automation system that is developed under these standards.

As a client-side component, the Communication Manager is implemented as a module with the ability of exchanging messages at initialization and typification. The communication at these two steps is a chain communication process among the function blocks as components from initialization to typification: Generic Communication Module-Communication Manager-Channel Manager-Communication Manager-Generic Communication Module (see Figure 4.3). The message exchange at initialization is the Registration Message, and the message at the typification is the Typification Message. In this process of exchanging messages, the Communication Manager is implemented as a functions block that is conducting the messages. But the Communication Manager is not capable of using the established channel from the Channel Manager directly. The communications among the devices are carried out by the Generic Communication Modules from the device. The Communication Manager must cooperate with the Generic Communication Module together so that the whole process of message exchange can be complete, and furthermore the communication among devices can be applied.

The structure consisting of the Communication Manager and Generic Communication Modules in device represents a framework of device management and communication. In order to communicate with the Channel Manager, the Communication Manager is implemented as a Service Interface Function Block in a Function Block Kernel. The data inputs and outputs of the Communication Manager are connected with a Subscriber and a Publisher for the Multicast in consistence with the Channel Manager. By using consistent data types as Channel Manager does, the Communication Manager can exchange messages for communicating with Channel Manager and for conducting typification messages with Generic Communication Module.

As a device management module, Communication Manager can be imple-

mented following the Singleton pattern that directs more than one Generic Communication Module (see Figure 4.3).

At the step of initialization, the Generic Communication Modules will register themselves at Communication Manager. The registration of the Generic Communication Modules will register the `data_send` and `data_receive` as parameter sets for requests and responses. When the Channel Manager sends its Response message back to the Communication Manager, the Communication Manager will use this `data_receive` parameter set to store the data of configurations.

At the step of typification, the Communication Manager module will send the data as typification messages back to the Generic Communication Modules to give them corresponding configurations of communication. Inside the Generic Communication Module, it holds SIFBs, whose types can be appointed by the Communication Manager. The typification of appointing the correct SIFBs depends on the `Assigned_Protocol_ID` and the `Generic_Interface_Type`.

By implementing the client-side module, it should be noticed that only one Communication Manager Kernel consisting of one Communication Manager is required. As for Generic Communication Module, there should be more prototypic Generic Communication Modules included for the communication with more channels.

As for an implementation using 4DIAC-IDE, the Communication Manager can be developed as a Communication Manager Kernel, which is a composite network FB with Communication Manager FB integrated. The Communication Manager FB is connected with SIFBs of Publisher and Subscriber as communication interface. Filtering messages with respect to the `Client_Name`) is implemented inside the Communication Manager Kernel. An illustration of the Communication Manager Kernel in 4DIAC-IDE can be seen in Figure 4.4, and the interface of the Communication Manager can be seen in Figure 4.5.

Within the illustration of the Communication Manager, each data from the Response data packet is appointed to the data inputs of the Communication Manager. Among these data, `GCM_ID_in` is used as indicator of directing the typification message to the correct function block, the other data will be used as configurations of the channel for the Generic Communication Modules.

The Generic Communication Module is also implemented as a composite FB. Inside this Generic Communication Module FB, SIFBs such as Publisher, Subscriber, Client and Server are included that can be used after the Generic Communication Module receives the message for typification. The SIFBs will be used to establish communication via several channels with other Generic Communication Modules in the same device or other devices (see Generic Communication Module in Figure 4.6).

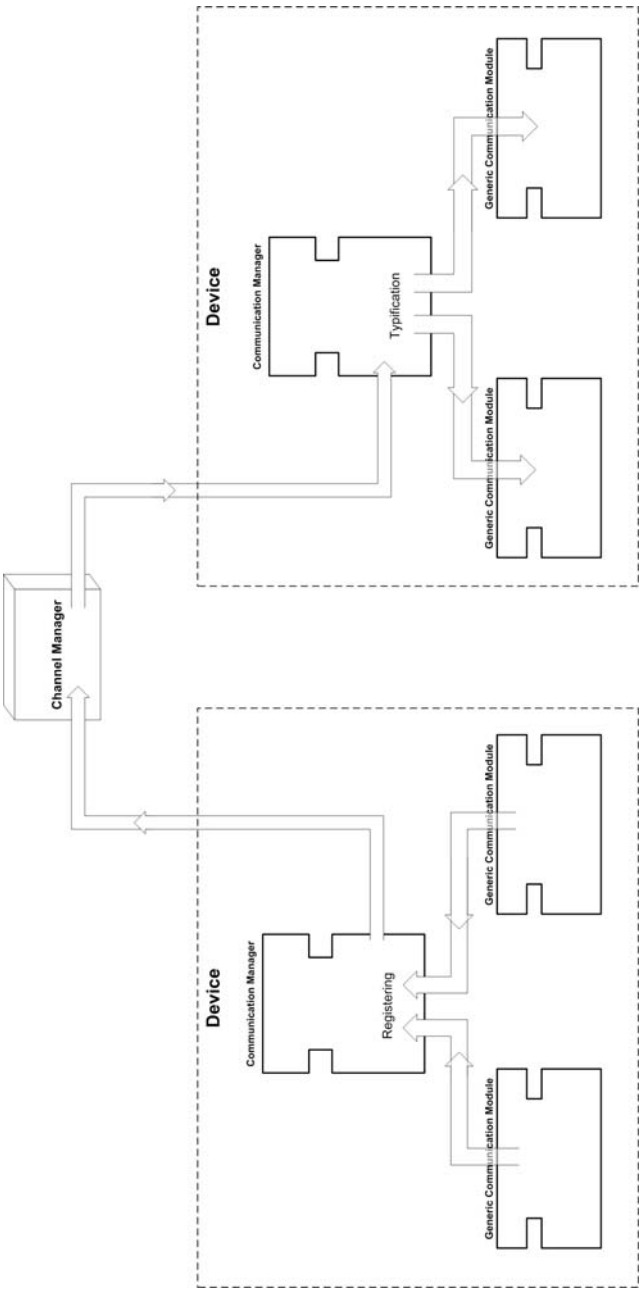

Figure 4.3: Approach of Implementing Communication Manager and Generic Communication Module

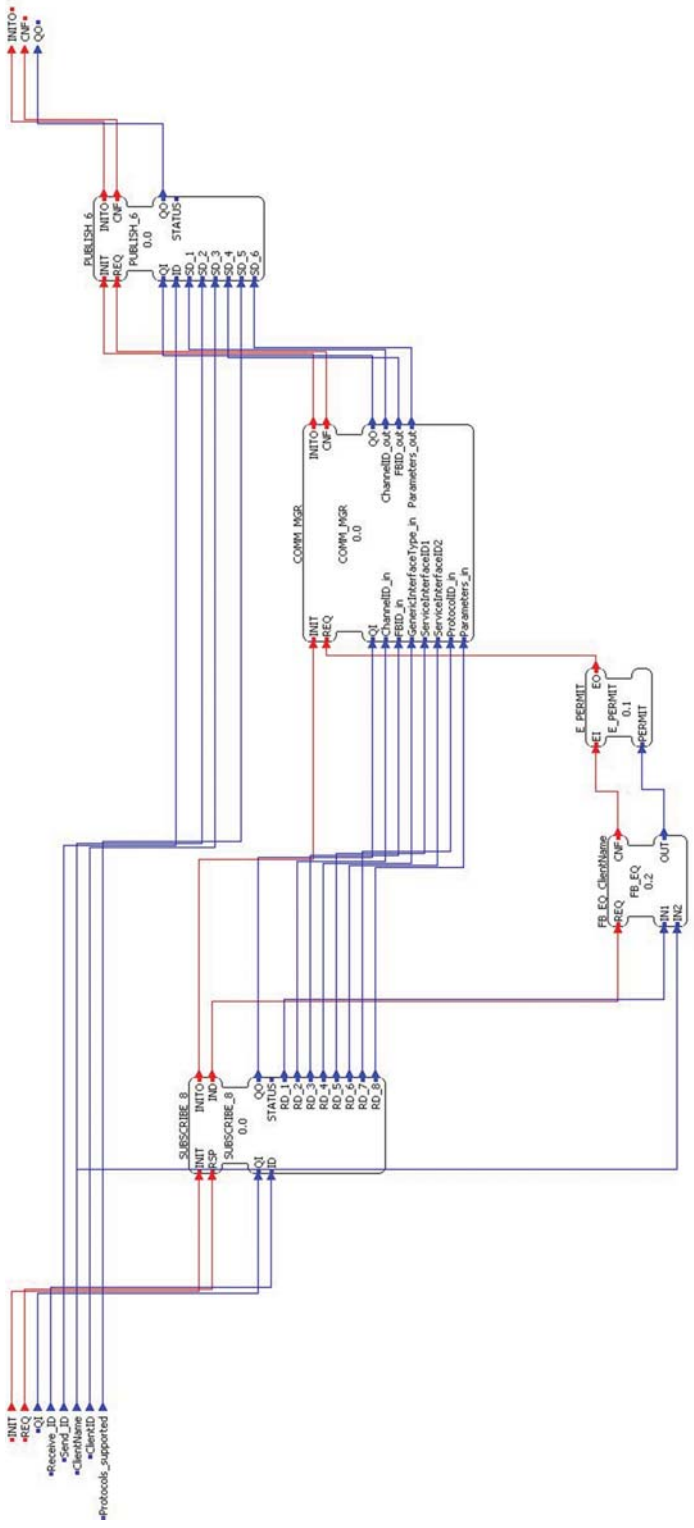

Figure 4.4: Communication Manager Kernel in 4DIAC-IDE

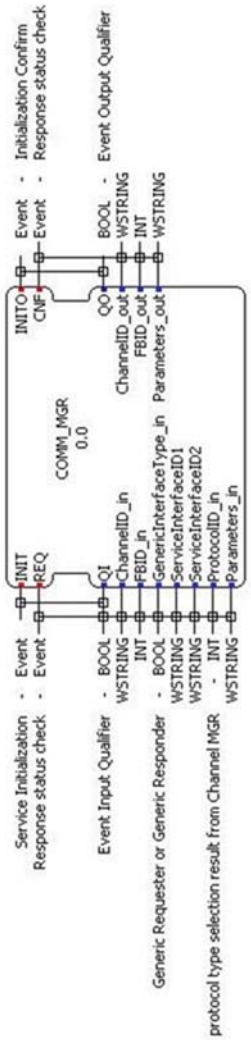

Figure 4.5: Communication Manager in 4DIAC-IDE

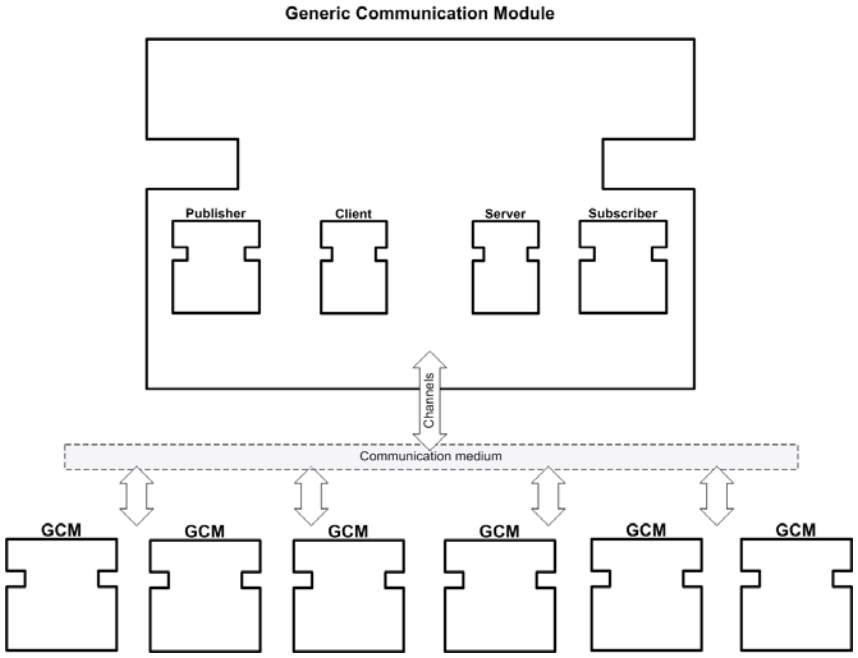

Figure 4.6: Generic Communication Module Function Block

4.3 Discussion

Based on the conceptual framework of the Plug and Play concept, the Channel Manager is implemented as an individual server that can distribute channels along with their configurations according to the requests from the Communication Manager. The Communication Manager is now able to use the data in the Response message and typify the Generic Communication Modules.

The Channel Manager uses its Identifier Management function for the comparison of requests. The parameters for Identifier Management and Protocol Management can be applied with other data provided by the requests. The Channel Manager is now able send a Response Data Packet after the selection and assignment process. By using FBDK to simulate the process of sending Request and receiving Response, the Channel Manager can sort the requests and assign correct channel data. By applying `Channel_ID`, `Client_Name`, `Protocols_Supported`, `Protocols_Supported`, `Parameters Supported`, and `GCM_ID` as parameters from the Request, the Channel Manager distributes automatically configured data as a Response data packet back to the Communication Manager.

As for the modeling tool of the process of sending request from the Publisher and receiving the results by the Subscriber, FBDK is used to develop the

interface of input and output. The data inputs and data outputs can be easily applied from the test windows of the Publisher and the Subscriber. Another possible application can be developed by the FBDK is to apply the application model with remote resources in the FBDK as a test system. Each data input for the Publisher can be predefined and the test windows of Subscribers can be developed as a HMI application that shows all the result in the preferred way from the user.

After the Registration and Assignment, the Communication Manager uses the content of the Response data packet for typification of Generic Communication Modules. The 4DIAC-IDE illustrates the interface of the Communication Manager Function Block and the Communication Manager Kernel.

From the aspect of exchanging data among devices, the Generic Communication Modules is pre-defined as a composite FB that includes Client, Server, Publisher and Subscriber SIFBs. The data inputs and outputs of this composite FB are defined for exchanging user data. By implementing the Communication Manager following the Singleton pattern, the Communication Manager conducts channel configurations to more Generic Communication Modules in the same device. The Generic Communication Modules are configured with fixed data inputs and outputs according to the interface of the composite FB of each Generic Communication Module. The event and data connections among the SIFB are not connected from the interface in 4DIAC-IDE, they are defined in FORTE according to the conductions of the Communication Manager. The structure of such a composite FB can be defined as a consistent interface of data inputs and outputs for all Generic Communication Modules.

5 Outlook

The three-layered framework in this concept specifies an open and flexible architecture for the industrial automation domain, which also provides further possibilities for the future development. Many potential functions can be applied based on this concept.

This concept provides the process of distributing and managing channels automatically based on specifications of the requests. The parameters of configuring channels are specified with a complete mechanism from discovering new devices to assigning channels. As for the next step, more parameters for channel configurations can be applied based on the mechanism in this concept. By applying different parameters, it is possible to develop various device based channel configuring approaches for more network and communication systems. The whole framework consisting of the Channel Manager, Communication Manager and Generic Communication Modules provides a solution of channel configuration and management to dynamic and complex network and communication systems. In the future, more implementations based on different runtime systems can be applied on this framework. More tests can be done for various industrial automation systems. For now, this work presents a prototypical implementation based on IEC 61499. The architecture from this work is able to be integrated into different platforms. Not only the whole framework is flexible to be adopted by other systems, each component is also easy to be applied and extended. The implementation can be applied in the test bed at the Odo Strugger Laboratory, Vienna University of Technology.

6 Conclusion

With more and more intelligent devices introduced into the industrial automation domain, exchanging process data via established channels must be ensured for the industrial automation systems. In order to bring automatic communication configurations among devices, and to avoid the complexity of integrating with different engineering tools, a Plug and Play concept for the industrial automation domain is required. Such a concept should support automatic channel configurations along with management functions.

At the beginning of this work, the current technologies in the areas of industrial automation and network communication systems were discussed. The available Plug and Play standards were briefly analyzed.

This work provided a development of the Plug and Play concept, which can make the automatically configured communication among the devices possible for the dynamic network and communication systems in the industrial automation environment. The concept in this work presented a three-layer framework that consists of the Channel Manager, Communication Manager and Generic Communication Module. The modular design of each component and the whole architecture based on this framework provided an open and flexible structure. This concept presented a complete mechanism consisting of Initialization, Registration, Assignment and Typification for discovering devices and eventually distributing channels. By given functions such as Identifier Management, Protocol Management and Protocol Management, the channel distribution process from Registration of devices to Assignment of channels was achieved by the Channel Manager. The development of the Communication Manager made it possible for the device to use the channel configuration data for typifying Generic Communication Modules.

The prototypical implementation showed that the process of establishing communication and managing channels for attached devices in the industrial automation domain is feasible.

Bibliography

[Abl06] J. Abley. *RFC4786 - Operation of Anycast Services*, December 2006. http://tools.ietf.org/html/rfc4786.

[APGD07] Fotis Andritsopoulos, Serafeim Papastefanos, George Georgakarakos, and Gregory Doumenis. Reliable multicast H.264 video streaming for surveillance applications. *The 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'07)*, 2007.

[Bon95] Karl Walter Bonfig. *Feldbus-Systeme*. Expert Verlag, 1995.

[Bor92] Walter Borst. *Der Feldbus in der Maschinen- und Anlagentechnik: die Anwendung der Feldbusnormen bei Entwicklung und Einsatz von Meß- und Stellgeräten*. München:Franzis, 1992.

[CDK94] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: concepts and design*. Addison-Wesley, 1994.

[Che09] Stuart Cheshire. *How does Zeroconf compare with Viiv/DLNA/DHWG/UPnP*. www.zeroconf.org, 2009. http://www.zeroconf.org/ZeroconfAndUPnP.html.

[Chr04] James H. Christensen. *Presentation: Automation Modeling, Design and Programming with the IEC 61499 Function Block Standard*. 2004.

[Chr07] James H. Christensen. *IEC 61499: A Standardized Architecture For Adding Value In Industrial Automation*. 2007.

[Dee89] S. Deering. *RFC1112 - Host extensions for IP multicasting*, 1989. http://tools.ietf.org/html/rfc1112.

[Die09] Dietmar Dietrich. Feldbussysteme Lecture Script, 2009.

[Dro97] R. Droms. *RFC2131 - Dynamic Host Configuration Protocol*. Bucknell University, March 1997. http://tools.ietf.org/html/rfc2131.

[Ebe07] Stephan Eberle. Adaptive internet integration of field bus systems. *IEEE Transactions On Industrial Informatics, Vol. 3, No. 1*, 2007.

[Enc09] PC Magazine Encyclopedia. *Plug and Play*. PC Magazine, 2009. http://www.pcmag.com/.

[Eve02] EventStudio. *DHCP - Dynamic Host Configuration Protocol (Normal Operation)*. EventHelix.com, 2002. http://www.eventhelix.com/RealtimeMantra/Networking/DHCP.pdf.

[FB04] Bernard Favre-Bulle. *Automatisierung Komplexer Industrieprozesse*. Springer-Verlag/Wien, 2004.

[Fur00] Frank J. Furrer. *Ethernet-TCP/IP für die Industrieautomation: Grundlagen und Praxis*. Heidelberg Hüthig, 2000.

[Gut01] Erik Guttman. Autoconfiguration for IP networking: Enabling local communication. 2001.

[Ham08] Reinhard Hametner. Modellierung von Regelungsstrategien in einer event-basierten Echtzeitsteuerungsumgebung. Master's thesis, Vienna University of Technology, 2008.

[Hel02] Sumi Helal. Standards for Service: Discovery and Delivery. 2002.

[IEC05a] IEC. *IEC 61499-1 Function blocks - Part 1: Architecture*. IEC, January 2005.

[IEC05b] IEC. *IEC 61499-1 Function blocks - Part 2: Software Tool Requirements*. IEC, January 2005.

[IEC05c] IEC. *IEC 61499-1 Function blocks - Part 4: Rules for Compliance Profiles*. IEC, May 2005.

[ITU95] ITU. *Data Networks and Open System Communications - Open Distributed Processing*. Telecommunication Standardization Sector Of ITU, November 1995.

[JAD] JADE. JADE - Java Agent Development Framework. http://jade.tilab.com/.

[Klo08] Maria Klonner. A concept for IEC 61131-3 programmable communication protocols. Master's thesis, Vienna University of Technology, 2008.

[Koz01] Charles M. Kozierok. *Plug and Play.* The PC Guide, 2001. http://www.pcguide.com/ref/mbsys/res/pnp.htm.

[Lew01] Robert Lewis. *Modelling control systems using IEC 61499, Applying function blocks to distributed systems.* IEE - The institution of Electrical Engineers, London, United Kingdom, 2001.

[LN07] Learn-Networking. A guide to network topology. 2007. http://learn-networking.com/network-design/a-guide-to-network-topology.

[LSKD07] Jae Woo Lee, Henning Schulzrinne, Wolfgang Kellerer, and Zoran Despotovic. Discovering zeroconf services beyond local link. *Globecom Workshops, 2007 IEEE*, 2007.

[Mer09] Munir Merdan. *Knowledge-based multi-agent architecture applied in the assembly domain.* PhD thesis, Technische Universität Wien, 2009.

[Mic00] Microsoft. *Understanding Universal Plug and Play: White Paper*, 2000. http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc.

[Mic03] Microsoft. Differences between multicast and unicast. Technical report, Microsoft Support, 2003. http://support.microsoft.com/default.aspx/kb/291786.

[MKHFB08] Munir Merdan, Gottfried Koppensteiner, Ingo Hegny, and Bernard Favre-Bulle. Application of an ontology in a transport domain. 2008.

[MLHK08] Munir Merdan, Wilfried Lepuschitz, Ingo Hegny, and Gottfried Koppensteiner. Application of a communication interface between agents and the low level control. 2008.

[Mog84] Jeffrey Mogul. *RFC919 - Broadcasting Internet Datagrams.* Network Working Group, Computer Science Department, Stanford University, October 1984. http://tools.ietf.org/html/rfc919.

[MVKZ08] Munir Merdan, Pavel Vrba, Gottfried Koppensteiner, and Alois Zoitl. Knowledge-based multi-agent architecture for dynamic scheduling in manufacturing systems. 2008.

[Neu06] Georg Neugschwandtner. Towards plug and play in home and building automation networks. 2006.

[OMG08] OMG. *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification*. OMG - Object Management Group, April 2008.

[Org09] Zeroconf Organization. *Zeroconf*, 2009.

[PD98] P. Pleinevaux and J.-D. Decotigde. Time critical communication networks: Field buses. *May 1988-Vo1.2, No. 3 IEEE Network*, 1998.

[Sch00] Gerhard Schnell. *Bussysteme in der Automatisierungs- und Prozesstechnik*. Friedr. Vieweg & Sohn, 2000.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379423, 623656, 1948.

[SI98] Claude E. Shannon and IRE. Communication in the presence of noise. *THE IEEE*, 86, 1998.

[SJM90] Liba Svobodova, Philippe A. Janson, and Eduard Mumprecht. Heterogeneity and OSI. 1990.

[SL03] Minyoung Sung and Gui-young Lee. A QoS-enabled Service Discovery and Delivery Scheme for Home Networks. *28th Annual IEEE International Conference on Local Computer Networks (LCN03)*, 2003.

[Sün04] Christoph Sünder. Integration of motion control in distributed automation systems according to iec 61499. Master's thesis, Vienna University of Technology, 2004.

[Tho99] Jean Pierre Thomesse. Fieldbuses and interoperability. *Control Engineering Practice 7*, pages 81–94, 1999.

[Tho05] Jean Pierre Thomesse. Fieldbus technology in industrial automation. *THE IEEE*, 93, NO. 6,, 2005.

[TQ08] Huaglory Tianfield and Feng Qian. Re-configurable Industrial Automation. *Proceedings of the 7th World Congress on Intelligent Control and Automation,June 25 - 27, 2008, Chongqing, China*, 2008.

[UPn00] UPnP. *UPnP Device Architecture*, 2000. http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20080424.pdf.

[UPn09] UPnP. *UPnP Forum*. UPnP Forum, 2009.

[vGH04] Albie F.J. von Gordon and Gerhard P. Hancke. Protocol conversion for real-time energy management systems. *2004 IEEE*, 2004.

[Vog07] Bernard Voglmayr. Middelware-Konzepte für verteilte Automatisierungssysteme basierend auf IEC 61499. Master's thesis, Vienna University of Technology, 2007.

[Woo02] Bill Woodcock. Best practices in ipv4 anycast routing. Technical report, Packet Clearing House, 2002.

[ZK09] Safaa Zaman and Fakhri Karray. TCP/IP model and intrusion detection systems. *2009 International Conference on Advanced Information Networking and Applications Workshops*, 2009.

[Zoi07] Alois Zoitl. *Basic Real-Time Reconfiguration Services for Zero Down-Time Automation Systems*. PhD thesis, Vienna University of Technology, Vienna, 2007.