MASTER

Universal communication interface architecture for Sattstore Warehouse Management System

van Hassel, T.F.M.; Knuit, I.A.P.C.

Award date:
2003

Link to publication

Disclaimer
This document contains a student thesis (bachelor’s or master’s), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
UNIVERSAL COMMUNICATION INTERFACE
ARCHITECTURE FOR SATTSTORE
WAREHOUSE MANAGEMENT SYSTEM
By T.F.M. van Hassel and I.A.P.C. Knuit

Master of Science Thesis
carried out from September 2002 to July 2003

Supervisor:
Ing. C. Albers (ABB BV)

Graduation professor:
Prof. dr. ir. E.R. Fledderus

The Faculty of Electrical Engineering of Eindhoven University of Technology disclaims all responsibility for the contents of traineeship and graduation reports.
Abstract

ABB’s SattStore Warehouse Management System (WMS) provides all functions needed to build an efficient warehouse operation, like warehouse administration and the control and monitoring of all activities in the warehouse, whether they are performed by automatic equipment or manual operators. Within SattStore the transport equipment communication handling part SattMate is used to set-up communication with i.a. cranes, conveyors and mobile terminals. As a result of the growing heterogeneity in communication interfaces towards the industrial controllers of these equipments, and the limitations of SattMate to deal with this heterogeneity, a Universal Communication Interface Architecture (UCIA) for SattStore has been developed.

The UCIA communication architecture development has focussed on crane and conveyor communication. Instead of using a crane sub system for crane communication and a conveyor sub system for conveyor communication defined by SattStore, the UCIA is based on the fact that SattStore has to communicate with controllers on both cell level (cell controllers) and control level (device controllers) according to the CIM (Computer Integrated Manufacturing) communication pyramid. The CIM pyramid divides the industrial communication environment into functional levels.

The UCIA provides a communication architecture, which makes it possible to connect SattStore to different cell controllers and device controllers (i.a. PLCs.), in a less time consuming way. For communication with device controllers, active or passive data servers are used to mask device controller interfaces to SattStore. The UCIA consists of a 3-tier client/server communication architecture, based on a developed server and cell controller access model, and middleware technologies. The first creates a generic interface towards SattStore. The latter offers i.a. location and access transparencies as well as additional services, like naming and logging. In addition they make communication implementations easier by the reuse of generic communication elements, like security and data persistency, and by masking underlying network details to the software engineers.

Samenvatting

ABB’s SattStore Warehouse Management System (WMS) biedt alle functionaliteit om op een efficiënte manier een magazijn te sturen, zoals administratie en het sturen en bijhouden van alle activiteiten in het magazijn, onafhankelijk of deze automatisch of manueel worden verricht. Om communicatie met bijvoorbeeld kranen, lopende banden en mobiele terminals op te zetten maakt SattStore gebruik van SattMate, dat het communicatie transport gedeelte afhandelt. Als gevolg van de groeiende heterogeniteit in de communicatie interfaces naar deze industriële controllers toe en de beperkingen van SattMate om deze heterogeniteit te ondersteunen, is een Universele Communicatie Interface Architectuur (UCIA) ontwikkeld.

De ontwikkeling van de UCIA communicatie architectuur is voornamelijk gericht op communicatie met kranen en lopende banden. SattMate gebruik een crane submodule voor kraan communicatie en een conveyor submodule voor communicatie met lopende banden. De UCIA heeft deze gedachte los gelaten en is gebaseerd op de gedachte dat SattStore moet communiceren met controllers op zowel cel niveau (cell controllers) als op control niveau (device controllers), zoals wordt onderscheiden in de CIM piramide. De CIM piramide verdeelt de industriële communicatie omgeving in functionele niveaus.

De UCIA biedt een communicatie architectuur die het mogelijk maakt SattStore te koppelen aan cell controllers en device controllers in een kortere project tijd. Voor koppelingen met device controllers worden actieve en passieve data servers gebruikt om de device controller interfaces voor SattStore transparant te maken. De UCIA bestaat uit een 3-tier client/server communicatie architectuur, gebaseerd op een ontwikkeld server en cell controller access model, en middleware technieken. Het access model zorgt voor een generieke interface naar SattStore toe. Middleware technieken zorgen ervoor dat locatie en toegang transparant blijven voor SattStore en bieden extra services, zoals naam en log services. Tevens maken deze communicatie implementaties eenvoudiger door het hergebruiken van generieke communicatie elementen, zoals “security” en data persistentie, en door het feit dat software ontwikkelers zich niet meer hoeven te bekommeren om onderliggende netwerk details.
Preface

The last 11 months we were overall busy with the developing of a concept of a universal communication interface architecture for SattStore. The main activities were finding generic communication elements in communications between SattStore and especially cranes and conveyor systems, and a profound research to different industrial standards and middleware techniques.

Because none of us has a software background and to gather knowledge about software architectures, how the guidelines and the system’s concept could be well documented and how to be sure that the concept satisfies all system requirements and objectives, the first part of our study consists of a short study to how to document software architectures. This study has resulted in a clear architecting process and functional and non-functional design aspects, which are important for development of a good software architecture.

The second part of our graduation report consists of distributed computing in general. In this part many definitions of the distributed computing environment are introduced, and two different implementation paradigms are distinguished with their related communication terms: protocol based versus object and component based implementation. This part concludes with an enumeration of functional and non-functional design aspects of distributed communication systems.

The third part introduces the SattStore transport environment. This study shows the position of the SattStore WMS in the CIM pyramid and concludes that SattStore always connects to two lower levels of the CIM pyramid: cell level and control level. The SattStore transport environment is followed by the present situation where the present communication solutions are shortly described.

The SattStore environment, the present situation and the introduction to distributed computing are input to the fourth part of our graduation report. With this knowledge we started different studies to industrial protocols and standards, and middleware technologies. Both studies resulted in conclusions, important trends and recommendations. Together with the assignment, requirements of the ABB Logistic Systems department and the results of the study to industrial protocols and standards, the systems’ architectural requirements are formulated. The architectural requirements consist of the system environment and functional and non-functional architectural requirements.

The system’s architectural requirements and the knowledge gained by the middleware study of the fourth part are input of the meta-architecture, which is the first phase of system structuring. System structuring forms the last major part of this graduation report. The meta phase combines the results of the middleware study and the architectural requirements to make decisions that will act as a guide during the next steps of structuring. The meta-architecture is followed by the conceptual architecture where the system’s concept is described and where the system is decomposed into high-level components. The responsibilities of each component, and interconnections between components are identified.

The conceptual architecture lays ground for the next R&D steps towards a universal communication interface architecture for SattStore. The conceptual phase is input to the logical phase, which adds precision to the conceptual architecture including component and interface specifications.

This graduation report has been realized in cooperation between the Technische Universiteit Eindhoven (TU/e) and ABB Logistic Systems. We like to thank prof. ir. F. van den Dool and prof. dr. ir. E. R. Fledderus from TU/e and ing. C. Albers from ABB for their supportive action during the project. Special thanks are due to Ir. A. van den Elshout, ing. R. Landa, ing. J. Hollebrandse and other colleagues of ABB Logistic Systems, who were always helpful and willing to think along with us during the architecting process.

I.A.P.C. Knuit
T.F.M. van Hassel

Etten-Leur, July 2003
# Table of contents

ABSTRACT ................................................................................................................................. I
PREFACE ........................................................................................................................................ III
TABLE OF CONTENTS ................................................................................................................. V
LIST OF ABBREVIATIONS ........................................................................................................ XI

1 INTRODUCTION ...................................................................................................................... 1
   1.1 ABB BV .............................................................................................................................. 1
       1.1.1 ABB Group ............................................................................................................... 1
       1.1.2 ABB Logistic Systems ........................................................................................... 1
   1.2 SATTSTORE WAREHOUSE MANAGEMENT SYSTEM ............................................. 1
       1.2.1 Introduction to SattStore WMS ............................................................................. 1
       1.2.2 SattStore WMS modules ...................................................................................... 2

2 ASSIGNMENT .......................................................................................................................... 5
   2.1 Problem Definition .......................................................................................................... 5
   2.2 Assignment ...................................................................................................................... 6

3 ARCHITECTURE DESIGN ....................................................................................................... 7
   3.1 INTRODUCTION .............................................................................................................. 7
   3.2 DEFINITION OF SOFTWARE ARCHITECTURES ......................................................... 7
   3.3 THE BENEFITS OF SOFTWARE ARCHITECTURE ...................................................... 7
   3.4 SOFTWARE ARCHITECTURE METAMODEL .............................................................. 8
   3.5 ARCHITECTURAL VIEWS, VIEWPOINTS AND STAKEHOLDERS ............................. 9
       3.5.1 Stakeholders ........................................................................................................... 9
       3.5.2 Viewpoints ............................................................................................................ 10
       3.5.3 Views .................................................................................................................... 10
       3.5.4 Software architecture view models ..................................................................... 10
   3.6 BREDEMEYERS DECISION LAYERED MODEL [BRED 2002] .................................. 12
       3.6.1 Meta-Architecture ............................................................................................... 12
       3.6.2 Architecture ......................................................................................................... 12
       3.6.3 Architecture guidelines and policies ................................................................... 13
   3.7 THE ARCHITECTING DESIGN PROCESS ..................................................................... 13
       3.7.1 The technical process ........................................................................................... 14
           3.7.1.1 Architectural requirements ........................................................................ 14
           3.7.1.2 Architecture specification .......................................................................... 15
           3.7.1.3 Architecture validation ............................................................................. 16
           3.7.1.4 Iterations ................................................................................................... 16
       3.7.2 The organizational process .................................................................................... 16
           3.7.2.1 Init/Commit .................................................................................................. 16
           3.7.2.2 Deployment ................................................................................................. 16
   3.8 DOCUMENTING SOFTWARE ARCHITECTURES ....................................................... 17

4 INTRODUCTION TO DISTRIBUTED COMPUTING ............................................................. 19
   4.1 DISTRIBUTED COMPUTING ....................................................................................... 19
   4.2 IMPLEMENTATION PARADIGMS ............................................................................... 20
   4.3 PROTOCOL-BASED ENVIRONMENT ........................................................................... 21
       4.3.1 Protocol layering .................................................................................................. 21
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.2 OSI reference model</td>
<td>22</td>
</tr>
<tr>
<td>4.3.2.1 Communication protocols</td>
<td>23</td>
</tr>
<tr>
<td>4.3.2.2 Data encapsulation</td>
<td>24</td>
</tr>
<tr>
<td>4.3.2.3 Service interface between layers</td>
<td>25</td>
</tr>
<tr>
<td>4.3.2.4 Connection-oriented and connectionless services</td>
<td>26</td>
</tr>
<tr>
<td>4.3.3 Internet Model</td>
<td>26</td>
</tr>
<tr>
<td>4.3.3.1 Protocol layers</td>
<td>26</td>
</tr>
<tr>
<td>4.3.3.2 Socket API</td>
<td>27</td>
</tr>
<tr>
<td>4.4 OBJECT AND COMPONENT-BASED ENVIRONMENT</td>
<td>28</td>
</tr>
<tr>
<td>4.4.1 DOC and DCC</td>
<td>28</td>
</tr>
<tr>
<td>4.4.2 Client/server computing</td>
<td>28</td>
</tr>
<tr>
<td>4.4.2.1 Logical view: software layers</td>
<td>29</td>
</tr>
<tr>
<td>4.4.2.2 Physical view: 2-tier, 3-tier, n-tier</td>
<td>30</td>
</tr>
<tr>
<td>4.4.3 Middleware</td>
<td>31</td>
</tr>
<tr>
<td>4.5 FUNCTIONAL AND NON-FUNCTIONAL DESIGN ASPECTS OF DISTRIBUTED SYSTEMS</td>
<td>33</td>
</tr>
<tr>
<td>5 SATTSTORE TRANSPORT ENVIRONMENT</td>
<td>37</td>
</tr>
<tr>
<td>5.1 COMPUTER INTEGRATED MANUFACTURING (CIM)</td>
<td>37</td>
</tr>
<tr>
<td>5.1.1 CIM functional levels</td>
<td>37</td>
</tr>
<tr>
<td>5.1.2 CIM information flow</td>
<td>39</td>
</tr>
<tr>
<td>5.2 MAPPING SATTSTORE ON CIM</td>
<td>40</td>
</tr>
<tr>
<td>5.2.1 Industrial cranes</td>
<td>40</td>
</tr>
<tr>
<td>5.2.1.1 Cranes and CIM functional levels</td>
<td>40</td>
</tr>
<tr>
<td>5.2.1.2 Crane information flow</td>
<td>41</td>
</tr>
<tr>
<td>5.2.2 Conveyor belts</td>
<td>41</td>
</tr>
<tr>
<td>5.2.2.1 Conveyors and CIM functional levels</td>
<td>41</td>
</tr>
<tr>
<td>5.2.2.2 Conveyor information flow</td>
<td>42</td>
</tr>
<tr>
<td>5.2.3 Crane versus conveyor communication</td>
<td>42</td>
</tr>
<tr>
<td>5.3 SATTMATE SYSTEM</td>
<td>42</td>
</tr>
<tr>
<td>5.3.1 Transport Assignments</td>
<td>43</td>
</tr>
<tr>
<td>5.3.2 The Core</td>
<td>43</td>
</tr>
<tr>
<td>5.3.3 Sub-systems</td>
<td>44</td>
</tr>
<tr>
<td>5.3.4 Transport sub-system</td>
<td>45</td>
</tr>
<tr>
<td>5.3.4.1 Transport Administrator</td>
<td>45</td>
</tr>
<tr>
<td>5.3.4.2 Transport Supervisor</td>
<td>45</td>
</tr>
<tr>
<td>5.3.4.3 Communication interface adapter</td>
<td>45</td>
</tr>
<tr>
<td>5.3.4.4 Crane transport sub-system</td>
<td>45</td>
</tr>
<tr>
<td>5.3.4.5 Conveyor transport sub-system</td>
<td>46</td>
</tr>
<tr>
<td>6 PRESENT SITUATION</td>
<td>47</td>
</tr>
<tr>
<td>6.1 CRANE COMMUNICATION</td>
<td>47</td>
</tr>
<tr>
<td>6.1.1 Standard crane interface</td>
<td>48</td>
</tr>
<tr>
<td>6.1.1.1 Process-IO</td>
<td>48</td>
</tr>
<tr>
<td>6.1.1.2 BCC records</td>
<td>49</td>
</tr>
<tr>
<td>6.1.2 Crane cell controller interfaces</td>
<td>49</td>
</tr>
<tr>
<td>6.1.2.1 BCC01 communication</td>
<td>49</td>
</tr>
<tr>
<td>6.1.2.2 ACC and MC01 communication</td>
<td>50</td>
</tr>
<tr>
<td>6.1.2.3 Daifuku cell controller communication</td>
<td>50</td>
</tr>
<tr>
<td>6.1.2.4 Protocol generics</td>
<td>51</td>
</tr>
<tr>
<td>6.1.3 Crane communication sequence</td>
<td>51</td>
</tr>
<tr>
<td>6.2 CONVEYOR COMMUNICATION</td>
<td>52</td>
</tr>
</tbody>
</table>
8 ARCHITECTURAL REQUIREMENTS .......................................................... 71

8.1 UClA-SYSTEM AND ENVIRONMENT ............................................. 71
8.2 UClA FUNCTIONAL REQUIREMENTS ACCORDING TO THE ASSIGNMENT ....................................................... 72
8.3 UClA NON-FUNCTIONAL REQUIREMENTS ACCORDING TO THE ASSIGNMENT ...................................................... 72
8.4 DEVICE CONTROLLER COMMUNICATION ........................................ 73

8.4.1 Functional requirements .................................................. 73
  8.4.1.1 Client/server architecture ........................................... 73
  8.4.1.2 Data Server Model (DSM) ...................................... 73
  8.4.1.3 Communication services ........................................ 74
8.4.2 UClA non-functional requirements ........................................ 75
  8.4.2.1 Object-oriented technology ...................................... 75
  8.4.2.2 Network communication ........................................ 75
  8.4.2.3 Reliability/Security .............................................. 75
  8.4.2.4 Heterogeneity/interoperability .................................. 75
  8.4.2.5 Scalability ......................................................... 75
  8.4.2.6 Usability and maintainability .................................... 76

8.5 CELL CONTROLLER COMMUNICATION ........................................ 76

8.5.1 Functional requirements .................................................. 76
  8.5.1.1 Client/server architecture ........................................ 76
  8.5.1.2 Communication services ........................................ 76
8.5.2 UClA non-functional requirements ........................................ 77

9 META ARCHITECTURE ........................................................................... 79

9.1 ARCHITECTURAL STYLE: 2-TIER, 3-TIER OR N-TIER ......................... 79
9.2 3-tier UClA ARCHITECTURAL ALTERNATIVES ................................ 80
  9.2.1 Introduction ..................................................................... 80
  9.2.2 Alternative 1 ............................................................... 80
  9.2.3 Alternative 2 ............................................................... 81
  9.2.4 Alternative 3 ............................................................... 82

10 MIDDLEWARE .................................................................................. 83

10.1 PURPOSE AND ORIGIN OF MIDDLEWARE .................................. 83
10.2 MIDDLEWARE CLASSIFICATION ............................................... 83
10.3 SUMMARY OF RELEVANT REQUIREMENTS .................................. 84
10.4 REQUEST ORIENTED MIDDLEWARE ........................................ 85
  10.4.1 Procedure Oriented Middleware ..................................... 85
    10.4.1.1 Introduction ....................................................... 85
    10.4.1.2 Functional design aspects ...................................... 85
  10.4.2 Object and Component Oriented Middleware ....................... 86
    10.4.2.1 Introduction ....................................................... 86
    10.4.2.2 Object versus component technologies ...................... 87
    10.4.2.3 Functional design aspects ...................................... 87
10.5 MESSAGE ORIENTED MIDDLEWARE (MOM) ................................ 90
  10.5.1 Introduction ............................................................... 90
  10.5.2 Functional design aspects ............................................... 90
10.6 WEB ENABLED MIDDLEWARE ................................................... 91
Table of contents

10.6.1 Introduction ................................................................................................................ 91
10.6.2 eXtensible Markup Language (XML) ................................................................. 92
10.6.3 Simple Object Access Protocol (SOAP) .............................................................. 92
10.6.4 Functional design aspects .................................................................................... 92
10.7 DATABASE ORIENTED MIDDLEWARES ................................................................. 94
10.8 NEXT GENERATION MIDDLEWARES ...................................................................... 94
10.9 MIDDLEWARE DEVELOPMENT PLATFORMS ....................................................... 94
10.10 MIDDLEWARE CONCLUSIONS AND RECOMMENDATIONS ......................... 95
  10.10.1 XML-SOAP ............................................................................................................ 95
  10.10.2 COM+ .................................................................................................................. 96
  10.10.3 CORBA 3.0 ......................................................................................................... 96
  10.10.4 EJB ..................................................................................................................... 96
  10.10.5 Message Oriented Middleware (MOM) products .................................................. 96
11 CONCEPTUAL ARCHITECTURE .................................................................................. 97
  11.1 PRIMARY PRESENTATION ...................................................................................... 97
  11.2 SUPPORTING INFORMATION ................................................................................. 98
    11.2.1 Context .............................................................................................................. 98
    11.2.2 Rationale ........................................................................................................... 98
    11.2.3 Catalogue .......................................................................................................... 99
      11.2.3.1 DSM Server Access Model (DSAM) ............................................................ 100
      11.2.3.2 Cell Controller Access Model (CCAM) ....................................................... 102
12 CONCLUSIONS AND RECOMMENDATIONS ................................................................ 105
  12.1 CONCLUSIONS ......................................................................................................... 105
  12.2 RECOMMENDATIONS ............................................................................................. 106
TERMINOLOGY ...................................................................................................................... 109
REFERENCES ........................................................................................................................ 125
List of abbreviations

The following table shows the abbreviations used in this graduation report:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>CCA</td>
<td>Cell Controller Adapter</td>
</tr>
<tr>
<td>CCAM</td>
<td>Cell Controller Access Model</td>
</tr>
<tr>
<td>CCF</td>
<td>Cell Controller Framework</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>CIM</td>
<td>Computer Integrated Manufacturing</td>
</tr>
<tr>
<td>CNC</td>
<td>Computer Numerical Control</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DCC</td>
<td>Distributed Component Computing</td>
</tr>
<tr>
<td>DCE</td>
<td>Distributed Computing Environment</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>DOC</td>
<td>Distributed Object Computing</td>
</tr>
<tr>
<td>DSM</td>
<td>Data Server Model</td>
</tr>
<tr>
<td>DSAM</td>
<td>DSM Server Access Model</td>
</tr>
<tr>
<td>DSF</td>
<td>DSM Server Framework</td>
</tr>
<tr>
<td>DSA</td>
<td>DSM Server Adapters</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>EBXML</td>
<td>Electronic Business XML</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Beans</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GIOP</td>
<td>General Inter-Orb Protocol</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>IOOP</td>
<td>Internet Inter Orb Protocol</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>J2ME</td>
<td>Java 2 Mobile Edition</td>
</tr>
<tr>
<td>J2SE</td>
<td>Java 2 Standard Edition</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Messaging Service</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MMS</td>
<td>Manufacturing Message Specification</td>
</tr>
<tr>
<td>MOM</td>
<td>Message Oriented Middleware</td>
</tr>
<tr>
<td>OLE</td>
<td>Object Linking and Embedding</td>
</tr>
<tr>
<td>OMA</td>
<td>Object Management Architecture</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OPC</td>
<td>OLE for Process Control</td>
</tr>
<tr>
<td>OPC-DA</td>
<td>OPC Data Access</td>
</tr>
<tr>
<td>OPC-XML-DA</td>
<td>OPC Data Access XML</td>
</tr>
<tr>
<td>ORB</td>
<td>Object Request Broker</td>
</tr>
<tr>
<td>PCI</td>
<td>Protocol Control Information</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable Logic Controller</td>
</tr>
<tr>
<td>POA</td>
<td>Portable Object Adapter</td>
</tr>
<tr>
<td>QOS</td>
<td>Quality Of Service</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RC</td>
<td>Robot Controllers</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SAP</td>
<td>Service Access Point</td>
</tr>
<tr>
<td>SAX</td>
<td>Simple API for XML</td>
</tr>
<tr>
<td>SCADA</td>
<td>Supervisory Control and Data Acquisition</td>
</tr>
<tr>
<td>SDU</td>
<td>Service Data Unit</td>
</tr>
<tr>
<td>SGML</td>
<td>Standard Generalized Markup Language</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architectures</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>VMD</td>
<td>Virtual Manufacturing Device</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>eXtended Markup Language</td>
</tr>
<tr>
<td>XMOP</td>
<td>XML Metadata Object Persistence</td>
</tr>
<tr>
<td>XSL</td>
<td>Extensible Stylesheet Language</td>
</tr>
<tr>
<td>XLST</td>
<td>XSL Transformation</td>
</tr>
</tbody>
</table>

Table: List of abbreviations

In the back of this graduation report a terminology and used references, categorized per chapter, have been inserted. In the report for the UCLA important new introduced terms are typed bold italic. The definition of a term is typed in italic under it as soon as the paragraph where the term is introduced has finished. References to publications, inserted in the reference list at the back of the report, are indicated by means of brackets, e.g. [ABB 2001].
Introduction

1.1 ABB BV

1.1.1 ABB Group

The ABB Group was formed in 1988, when the Swedish, Asia and the Swiss BBC Brown Boveri merged under the name ABB. Asia's history dates back to 1883. BBC Brown Boveri was founded in 1891.

ABB is a leader in power and automation technologies that enable utility and industry customers to improve performance while lowering environmental impact.

In power technologies, there is a greater focus today on high-quality, cost-effective products and processes that help power producers and distributors get the most from their existing plants or grids. ABB provides industrial and commercial customers, as well as electric, gas and water utilities, with a broad range of products, services and solutions for power transmission and distribution.

In automation technologies, ABB is creating new services to optimise existing production lines and increase returns from the supply chain. ABB enjoys both market and technical leadership in segments including electrical machines, drives and power electronics, low-voltage products, instrumentation, controls and robotics. ABB provides products, systems, software and services for the automation and optimisation of industrial, commercial and utility operations.

ABB is present in more than 100 countries. ABB is headquartered in Zurich, Switzerland. ABB Ltd shares are traded on the stock exchanges in London/ Zurich, Stockholm, Frankfurt and New York. More than half of ABB's revenues come from European markets, nearly a fifth from Asia, the Middle East and Africa, while about a quarter of revenues come from North and South American markets.

1.1.2 ABB Logistic Systems

The ABB Logistic Systems department in Etten-Leur is part of the worldwide ABB group and belongs to the division ABB automation technologies. The department Logistic Systems is responsible for development, engineering, sales, service and support of the SattStore Warehouse Management System, abbreviated as SattStore WMS. In Sweden the research & development of the SattStore WMS takes place, sometimes in cooperation with ABB Etten-Leur. In Etten-Leur people are overall busy with the engineering, sales, service and support of SattStore Warehouse Management Projects. Recent SattStore projects engineered by ABB Logistic Systems Etten-Leur are the automation of large complex warehouses at Heineken Zoeterwoude and Road Air Best.

1.2 SattStore Warehouse Management System

1.2.1 Introduction to SattStore WMS

The SattStore Warehouse Management System (SattStore WMS) provides all functions needed to build an efficient warehouse operation. SattStore is a software package for warehouse administration and the control and monitoring of all activities in the warehouse, whether they are performed by automatic equipment or manual operators. Tracking information is automatically captured in real time as the goods are handled.

Figure 1-1 shows SattStore WMS with its environment.
1.2.2 SattStore WMS modules

With its modular design SattStore easily connects to either automatic handling equipment such as high-bay cranes, automatic guided vehicles (AGV’s) or conveyors, or manual equipment like radio data terminals, label printers or bar code scanners. SattStore also connects easily to different ERP systems including BaaN Series, SAP R/3, Oracle Applications, etc.

Enterprise Resource Planning (ERP) is an industry term for the broad set of activities supported by multi-module application software that helps a manufacturer, or other business, manage the important parts of its business, including product planning, parts purchasing, maintaining inventories, interacting with suppliers, providing customer service, and tracking orders.

Figure 1-2 shows the modular design of SattStore, together with its modules.
Receiving
SattStore captures receipts as the goods arrive. The goods are registered either by the use of bar code readers or by manually entering data into the system. At this point relevant tracking information such as supplier id, supplier batch number and delivery order is entered. From this moment, the goods are visible and are tracked through the warehouse and out to the customer.

Put-away
Before storing the goods SattStore automatically selects a suitable location using the goods dimensions and the product characteristics. Several store types can be used: high-bay racking, low level racking, floor level storage etc. After selecting a suitable location the goods are put away. Put-away can be done using radio data terminals or paper lists. Full loads may be put-away by fork-lift trucks and smaller amounts by using a "reversed" picking method (placing parts on the shelves). Different areas of the warehouse may use different methods and equipment by default. As each individual storage operation is completed, the inventory core is immediately updated and the host system notified.

Order processing
Order processing involves several steps. In the warehouse office the orders to pick are planned using a variety of criteria such as delivery date, order type, order weight, order volume, shipment, consignee (recipient of an output order), etc. All criteria may be combined to form a specific search condition.

During the order planning process the planner knows the result of the planning in terms of weight, volume, expected expedition time and product shortages. Orders are then put into an order batch by the system and at the same time pick routes (a route, which is used to collect picking parts of output order lines, which should be picked together by the same person) and retrievals of full loads are created. Furthermore picking and full load retrieval may be generated as batch picking routes either per consignee, per shipment or per part in the order batch.

Pick routes are given to the pickers. Depending on the system parameter set-up, orders to several customers may be picked at one time and/or several pickers can pick the same orders. As picking commences, the weight and volume of orders to each customer are estimated and a unique label is printed containing consignee information and a unique identity to help identify the goods at a later stage.

It is also possible to set up the sequence in which different picking areas should be picked.

A picking area is an area in a store that defines the boundaries of picking and reversed picking routes. A store can be divided into several picking areas because of capacity requirements.

Shipment
The shipment procedure involves several activities: order assembly, packing and document retrieval and shipping. At the order assembly area all goods belonging to a customer are assembled. Full loads are brought by fork-lift trucks, whilst pick-to-loads from different pick areas are delivered by the pickers. After assembly the goods must be packed. Goods from different pick areas can be packed together as one load or one load can be split into several loads. A couple of packing documents can be printed like packing lists (contents of each load), delivery notes (results of order picking), consignment notes (contents of delivery in number of different package types), etc. After packaging the goods are shipped to e.g. a lorry. When loading the goods, the system may be set-up to require a check scanning of all loads being put into a lorry. This action secures that no mistakes are made when loading the lorry and thus, the correct goods are delivered to the customer.

Inventory-Core
The inventory core contains the basic inventory control functions. All location data, load data and product data belongs to this module. A large number of administration functions exist within SattStore to assist with the efficient management of the warehouse. Some examples are functions to handle shortages, perform cycle countings for locations that have not been counted in a certain period, resource management and equipment utilization and traceability, which function is to log all stock movements, full load inputs, full load outputs, picks and balance corrections.
**ERP Interface**

SattStore is designed for connection to one or more host systems although it may also operate stand-alone. This can be done with either an easy to implement *File Transfer Protocol (FTP)*, an XML-interface using the SattStore Collaboration Manager software or a more advanced message broker system, ABB Message Broker.

FTP is preferably used when just one connection is needed. The SattStore Collaboration Manager is a more general interface, using the *eXtendable Markup Language (XML)* as a common denominator, which retrieves and delivers data from a number of different systems. With the flexibility of the SattStore Collaboration Manager the interface work in the connected systems and maintenance work are minimized. ABB Message Broker is used in those cases where there is a need for a full communication centre where all communication is monitored and controlled on ERP-level.

The terms FTP and XML are mentioned here by means of completeness but are described in more detail later in this document.

**Equipment interfaces**

SattStore can be equipped with a combination of modules for the control of automatic and manual equipment in order to fulfil the needs of each installation. The group of modules that handles the communication with transport equipment like cranes, conveyors, AGV's, mobiles, etc is together called the *SattMate* system. Next to the SattMate system there are also modules for setting up communication with ABB industrial robots, barcode readers, label printers, etc.

The SattMate system consists of one main module called the Core and sub-modules for each type of transport equipment. More about SattMate can be found in chapter 4.

A SattStore project consists of a number of standard components, products, and customized software modules. SattStore is built modular and is developed with software tools and platforms, like Oracle RDBMS, Uniface (4GL), ADA (3GL), Visual C++ (3GL), Windows NT or AIX and Intel platform or IBM RS6000 platform. The products, used in a certain system, depend on the system configuration. Some products serves as a base for other products, and are compulsory in a system.
2 Assignment

2.1 Problem definition

Building communication interface adapters using guidelines and (re)using generic elements

Up till now a customer specific communication solution is built for each SattStore Warehouse Management project within ABB Logistic Systems. For each type industrial controller (or group of controllers) SattStore has to be connected to, a communication interface adapter is built within a SattMate sub-system. This communication interface adapter takes care of the communication between SattStore and the third party dependent industrial controller(s), which is often an exchange of application messages on top of a serial or socket connection. The interface adapter thus needs to support the interface with SattStore as well as the third party dependent industrial controller interface and must support technologies like parsing, mapping, routing, supervision and error control to provide proper communication between SattStore and third party industrial controller(s).

Often these customer specific solutions of building communication interface adapters are based on existing projects and are not built according to guidelines. Most important reason is that the interfaces towards third parties are different in such a way that it is hard for the software engineers to recognize the generic elements in every situation. As a result engineers start writing their own communication process to communicate with third party industrial controllers or adjust an existing communication solution. With it a standard way of developing and documenting of the software is not used.

Problem:
Every communication interface adapter is implemented in its own way, not using standard software developing and documenting rules. Generic elements, which consist in all interface adapters, are not reused in any way and standard guidelines to implement such a communication interface adapter within SattMate sub-systems are not used and even not available. Writing a complete new communication interface adapter or adjusting an existing one takes much more time than necessary. This results in high project throughput times, which results in higher costs and finally also leads to a weaker competitive position.

Limitations of the SattMate sub-systems
SattStore’s approach to provide communication with different industrial controllers, using crane, conveyor, mobile and other sub-systems, is an approach, which is more than 20 years old. The way SattStore exchanges messages with industrial controllers, using third party dependent application protocols on top of a serial or socket interface, is also becoming an old approach and is very dependent on the used industrial controller (vendor dependent).

Nowadays a couple of communication methods are available that solve the so-called vendor-dependency problem. Within the ABB group (not the department Logistic Systems) the MMS standard (Manufacturing Message Specification) has sometimes been used to create a more generic communication link between industrial controllers and applications. MMS uses a client/server model and creates genericity by making use of data servers. Data servers offer the controller’s data in a generic way towards the clients of the system. Next to MMS there are other communication solutions that create genericity by using data servers. In addition these communication methods provide more and more language and platform independency together with technology and location transparency by using middleware technologies.

On this moment SattStore does not support MMS communication with industrial controllers and there is little internal knowledge about this type of communication. Also other solutions to these more generic communication methods have never been examined.
Problem:
Within ABB Logistic Systems there is little knowledge available about communication with data servers, which is becoming the new industrial standard to communicate between industrial controllers and applications, and other vendor independent communication methods providing a generic, and more or less, location and technology transparent way of communication. In addition to this lack of knowledge it is also not sure whether the SattMate sub-systems have the possibility to make use of these communication methods and/or whether they will prevent an effective way of communicating using these technologies.

2.2 Assignment
Title: Universal Communication Interface Architecture for SattStore
Subtitle: The development of an UCIA for the SattStore Warehouse Management System for communication with industrial controllers

Within the ABB Logistic Systems department there is a need for a Universal Communication Interface Architecture (UCIA), which should reduce the time to write communication interface adapters for each industrial controller SattStore has to communicate with in a certain project. As a result of the UCIA-project it should be possible to achieve a simple connection between SattStore and different third party crane and conveyor controllers, without programming a complete new solution for each project. In addition it should be examined whether SattMate is capable using generic, vendor independent communication methods providing platform and language independency and location and technology transparency.

Standard guidelines and (re)use of generic elements
The UCIA should be a modular based communication interface Architecture, which can easily be adapted for proper communication with any type of crane or conveyor controller. When implementing the communication interface adapters standard guidelines should be used. Generic elements existing in the different communication interface adapters, like parse, map and supervise elements, should easily be reused.

Protocols & interfaces
Generic, vendor independent communication methods have to be examined, providing more or less programming language and platform independency and location and technology transparency. An example in the Industrial IT communication area is the growing trend towards the use of data servers. This part of the assignment consists of the following activities:

- Examine what generic, vendor independent communication solutions exist nowadays and which ones are interesting to integrate into the UCIA-project.
- Vendor independency is realized by the use of tight or loose coupled middleware technologies like CORBA or XML-SOAP. Examine what middleware technique(s) exist nowadays and which technique(s) can best be used in the UCIA-environment.
- Examine whether these vendor independent communication solutions also have generic elements, which can easily be (re)used in different projects of ABB Logistic Systems.

SattMate
Examine whether the use of SattMate sub-systems prevents an effective way of communicating using vendor independent communication technologies. If so, examine what elements within these sub-systems are not capable of using these communication technologies and whether there is a good, generic alternative communication solution for SattStore to communicate with crane and conveyor controllers.

Reliability
In the world of communication speed and reliability are very important. Therefore the UCIA has to guarantee these aspects. This can be realized by the use of sliding windows, packets including more messages, checksums, acknowledge mechanisms, retry mechanisms, watchdogs, etc. The delivering and reception of functional messages must have reliability, which is as high as possible.
3 Architecture design

3.1 Introduction

The major goal of the UCIA-project is to design an architectural concept of a system providing a generic connection between SattStore and different third party industrial controllers, together with guidelines describing how to achieve this connection in different situations. A system can be defined as follows:

A regularly interacting or independent group of items (e.g. components) forming a unified whole.

In this definition a component is defined as an object or program that performs a specific function and is designed in such a way to easily operate with other components and applications.

An object can generally be seen as any item that can be individually selected and manipulated.

To gather knowledge about the definition of these guidelines, how these guidelines and the system’s concept should be well documented and how to be sure that the concept satisfies all system requirements and objectives this chapter introduces the term “software architectures”. This chapter starts with a good definition of software architectures. Furthermore it shows the benefits of software architectures, deals with all elements related to software architectures and defines the used architecture design process. Finally it shows how to document software architectures in a way satisfying all system requirements and objectives.

3.2 Definition of software architectures

There is no standard, universally accepted definition of the term “software architecture”, but there is also no shortage of them. Because of its completeness the definition used in this document is the one from Bass, Clements, and Kazman:

“The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. By “externally visible” properties, is referred to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.” [BASS 1997]

First, architecture defines components. The architecture embodies information about how the components interact with each other. Second, the definition makes clear that systems can comprise more than one structure, and that no one structure holds the claim to being the architecture. Third, the definition implies that every software system has an architecture, because every system can be shown to be composed of components and relations among them. Fourth, the behaviour of each component is part of the architecture, insofar as that behaviour can be observed or distinguished from the point of view of another component.

3.3 The benefits of software architecture

From a business perspective the following benefits of using software architecture, for implementing communication interface adapters for SattStore, can be given:

- Using the standard guidelines to implement communication interface adapters will be less time consuming, resulting in lower project costs.
- Lower project costs and smaller throughput times can positively influence the competitive position.
- When software developers all document software architecture in a congruence, clear way, communication between software developers and engineers will be faster and more clear resulting in a more reliable and qualitative product.
From a developers perspective well-architected systems realize the following benefits:

- Developers can build and maintain loosely coupled components more independently, and even completely replace old components with new technologies using standard guidelines, showing detailed specifications of how to implement communication interface adapters to let SattStore communicate with industrial controllers, independent of what their specific interfaces look like. Interfaces and components are detailed described for both communications with industrial controllers using legacy interfaces as well as communication with data servers using middleware technologies.
- Clear definitions of responsibilities define where to put certain functionalities.
- The long-exclusive goal of reusable components is achievable. Generic elements, which exist in the different communication interface adapters, can easily be reused.
- Robust designs can survive partial failure, extension, requirements changes, platform changes, etc.
- Careful design of the application and the infrastructure yield high availability and performance.
- Standards-based technologies are more easily integrated with third parties and commercial "off the shelf" products, which have the benefits that they are already available and well tested.

### 3.4 Software architecture metamodel

To give an overview of all elements related to software architecture, the following software architecture metamodel can be drawn.

**Figure 3-1: Software architecture metamodel [BOOCH 2000]**

**System architecture**

*Software architecture is a subset of the overall system architecture. System architecture includes all design and implementation aspects, including hardware and technology selection.*

**Software architects**

*The person, team or organization responsible for systems architecting: the UCIA-project team.*
Views and stakeholders
A stakeholder is an individual, team, or organization with interests in, or concerns relative to, a system. A view is a representation of a whole system from the perspective of a related set of concerns.

Paragraph 3.5 discusses architectural views and stakeholders, and paragraph 3.6 describes the chosen view model. How views can be documented is described in paragraph 3.8.

Architecture design process
The architecture design process is the process incorporating the essential steps involved in creating a good architecture and gaining organizational support and compliance.

Paragraph 3.7 shows the used architecture design process.

Architectural requirements
Architectural requirements are a subset of the system requirements, determined by architectural relevance. Architectural requirements include functional and non-functional requirements. Functional requirements are those technical requirements the implemented system should provide. Non-functional requirements are aspects such as performance, cost and reliability.

Chapter 8 deals with the architectural requirements.

Architectural patterns, styles and style guides
Patterns are abstract, core solutions to problems that recur in different contexts but encountering the same forces each time. Styles represent known design approaches to architectures (e.g. layers, pipe & filters, event based communication). Styles are documented in a style guide. The style guide tells what the elements and relations are of the style, and when the style might be chosen for use in a system.

In chapters 9 till 11 these subjects will come up for discussion.

Software architecture description and architectural blueprints
The software architecture description is a collection of documents that together describe the architecture.

The production and delivery of blueprints, the specifications and the models, are crucial to the realization of an architectural vision, and to the emergence of software architecture as a profession (chapter 11).

3.5 Architectural views, viewpoints and stakeholders

3.5.1 Stakeholders
People and organizations interested in the construction of a software architecture are called stakeholders (e.g. customers, the end users, the developer’s organization and those who maintain the system). These stakeholders have different concerns that they wish the system to guarantee or optimise.

A concern is an interest of a stakeholder like purpose/mission, risks, maintainability, etc.

The stakeholders of the UCIA software architecture are people who are interested in the construction of the UCIA-software architecture or who are involved with the UCIA-project:

• Project managers of ABB Logistic Systems
• Sales managers of the ABB Group
• Software developers of ABB Logistic Systems and SattStore developers in Sweden
• Software engineers of ABB Logistic Systems
• Project lead engineers of ABB Logistic Systems
• Service engineers of ABB Logistic Systems
3.5.2 Viewpoints

The stakeholders together form the viewpoints of architecture.

A viewpoint is a definition of a view (contents, models to use, related to stakeholders and concerns).

Project manager stakeholders and sales managers demand short time to market, low costs and parity with computing products. In addition it is important that the software architecture description achieves business goals and is congruent with the business strategy. Software developers have benefits by a clear, congruent way of documenting software architecture. Software engineers, project lead engineers and service engineers also have benefits by a clear, congruent way of documenting software architecture and in addition they demand standard guidelines, for implementing communication interface adapters for SattStore, showing detailed specifications of how to implement communication interface adapters to let SattStore communicate with industrial controllers, independent of what their specific interfaces look like. Interfaces and components must be described in detail for both communications with industrial controllers using legacy interfaces as well as communication using more generic, vendor independent technologies. In addition the service engineer stakeholders are concerned with modifiability and maintainability.

The different stakeholders, together with their viewpoints, result in different architectural views.

3.5.3 Views

Each view addresses a specific set of concerns, which are of interest for different stakeholders. It represents a partial aspect of a software architecture that shows specific properties/concerns of a software system.

Following the IEEE 1471 practices a view corresponds to one viewpoint (from one or more stakeholders) and a view can be captured in one or more view models.

Documenting an architecture is a matter of documenting the relevant views and their relationships, and adding documentation that applies to more than one view.

The first task for an architect is to decide which views are relevant. One approach provides a simple three-step procedure for choosing the views relevant to a particular project's needs based upon determining the needs of the stakeholders in concert with IEEE 1471 [CLEM 2003]:

1. Produce a candidate view list
2. Combine views
3. Prioritise

A view does not have to be completed before starting another.

There are a number of well-known views, each revealing certain aspects of the architecture being analysed. Architecture should be described in several relevant architectural views. A view is relevant when it can unveil the properties that are of interest. Different views are suited for different analyses. A view describing the run-time objects may be used to estimate the system's performance and find bottlenecks, and a class view to e.g. estimate the maintainability from the number of dependencies between classes.

3.5.4 Software architecture view models

An architecture view model shows how significant properties of a system are distributed across its constituent parts. There are different models available for describing the architecture of software systems. Each model is based on other concerns and each model distinguishes different views, revealing different properties and suited for different analyses. Without going into details the most well known view models are described in the next paragraphs.
Architectural structures of a system [BASS 1997]
P. Clements, L. Bass and R. Kazmann summarize nine views and call these structures:

<table>
<thead>
<tr>
<th>Architectural structure</th>
<th>Useful for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module structure</td>
<td>Resource allocation and project structuring and planning</td>
</tr>
<tr>
<td>Conceptual or logical structure</td>
<td>Understanding the problem space</td>
</tr>
<tr>
<td>Process/coordination structure</td>
<td>Scheduling analysis; performance analysis</td>
</tr>
<tr>
<td>Physical structure</td>
<td>Performance, availability, security analysis</td>
</tr>
<tr>
<td>Uses structure</td>
<td>Engineering subsets; engineering extensions</td>
</tr>
<tr>
<td>Calls structure</td>
<td>Performance profiling; bottleneck elimination</td>
</tr>
<tr>
<td>Data flow structure</td>
<td>Irascibility of functional requirements</td>
</tr>
<tr>
<td>Control flow structure</td>
<td>Simulation and verification of timing and functional behaviour</td>
</tr>
<tr>
<td>Class structure</td>
<td>Producing rapid almost alike implementations from a common template</td>
</tr>
</tbody>
</table>

Table 3-1: Architectural structures of a system according to Clements et al

Kruchten’s 4+1 view model of software architecture [KRUC 95]
Kruchten distinguishes five different views in his 4+1 view model of architecture. Each view addresses a specific set of concerns, which are of interest for different stakeholders.

<table>
<thead>
<tr>
<th>Views</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical view</td>
<td>Supports the functional requirements: the services a system should provide to its end users</td>
</tr>
<tr>
<td>Process view</td>
<td>Addresses non-functional requirements, such as performance and availability of resources.</td>
</tr>
<tr>
<td>Development view</td>
<td>Focuses on the organization of the actual software modules in a software development environment. It concerns the internal requirements related to ease of development and software management.</td>
</tr>
<tr>
<td>Physical view</td>
<td>Also takes the systems non-functional requirements into account and makes the various elements identified in the logical, process and development view onto the various hardware elements. This mapping should be highly flexible and should have a minimal impact on the source code itself.</td>
</tr>
<tr>
<td>Scenarios</td>
<td>Help to demonstrate that the elements of the four views work together seamlessly.</td>
</tr>
</tbody>
</table>

Table 3-2: Kruchten’s 4+1 views

The SNH model [SONI 95]
Soni et al. distinguishes five different views, called architectures.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual architecture</td>
<td>Describes the system in terms of its major design elements and relationships between them. Typical elements are components and connectors.</td>
</tr>
<tr>
<td>Module architecture</td>
<td>Contains a functional decomposition and layers. Typical terms are sub-systems, modules, layers, imports and exports.</td>
</tr>
<tr>
<td>Execution architecture</td>
<td>Describes the systems dynamic structure. Typical elements are tasks, threads, Remote Procedure Call (RPC) and events.</td>
</tr>
<tr>
<td>Code architecture</td>
<td>Describes how the source code, binaries and libraries are organised in the development environment. Code resides in files, directories and libraries.</td>
</tr>
<tr>
<td>Hardware architecture</td>
<td>Describes the hardware components and their relations as far as they are relevant for making software design decisions. Processors, memory, networks and disks are typical hardware elements.</td>
</tr>
</tbody>
</table>

Table 3-3: SNH architectures
Bredemeyer software architecture views [BRED 2002]

The Bredemeyer view model is the model used in the UClA-project. This model is a combination of the Kruchtens 4+1 model and the SNH model. It includes all system views depicted in both models and categorizes these views in an even higher level of abstraction. The Bredemeyer view model is part of the Bredemeyer decision layered model. This model shows what views are relevant in different architecting phases and what views should be used for particular decisions. This model is shown in paragraph 3.6. According to this decision layered model the used architecting design process is introduced in paragraph 3.7.

3.6 Bredemeyers decision layered model [BRED 2002]

Architectural decisions during architectural development may be at different levels of abstraction. There are higher-level decisions that guide and constrain the system decomposition and structuring decisions (meta-architecture), and there may be lower level decisions that guide and constrain the next level(s) of design and implementation (architecture guidelines and policies). This is captured in the layered decision model of software architecture shown below.

![Figure 3-2: Bredemeyer software architecture decision layered model](image)

3.6.1 Meta-Architecture

The meta-architecture is a set of high-level decisions that will strongly influence the integrity and structure of the system, but is not itself the structure of the system. The meta-architecture, through style, patterns of composition or interaction, principles, and philosophy, rules certain structural choices out, and guides selection decisions and trade-offs among others.

In the meta-architecture one could think about the qualities the system should have, what components are needed in the conceptual architecture and it will help to make the architecture more vivid and understandable.

3.6.2 Architecture

Architecture is at the centre of the layered decision model, and at the centre of the architecting activity. It is where the system structures are created, taking into account system priorities and constraints, and ensuring that the system will achieve the system objectives and architectural requirements. This work is informed and constrained by the decisions made in the meta-architecture.

Within the architecture layer, there are different architecture views to enhance the understandability of the architecture and to focus on particular concerns separately: Conceptual, Logical and Execution architecture views. This is shown in figure 3-3 and described on the next page.
Architecture design

Overall system view

Blueprint for developers
- Unambiguous
- Precise

Conceptual Architecture
The purpose of the conceptual architecture is to direct attention at an appropriate decomposition of the system without delving into details. Moreover, it provides a useful vehicle for communicating the architecture to non-technical audiences, such as management, marketing, and users. It consists of the architecture diagram (without interfaces) and an informal component specification for each component, along with discussion and rationale.

With the term rationale the explanation and motivation for the architecture description is meant.

Logical Architecture
The logical architecture adds precision, providing a detailed blueprint from which component developers and component users can work in relative independence. It incorporates the detailed architecture diagram (with interfaces) and component and interface specifications, along with discussion and explanations of mechanisms, rationale, etc.

Execution Architecture
An execution architecture is created for distributed or concurrent systems. The process view shows the mapping of components onto the processes of the physical system. The deployment view shows the mapping of (physical) components in the executing system onto the nodes of the physical system.

3.6.3 Architecture guidelines and policies
To help maintain system integrity or to address cross-cutting concerns, architects may include decisions focused at guiding or constraining lower-level design or even implementation in the architecture decision set.

3.7 The architecting design process
This architecting process incorporates a technical process and an organizational process according to the Bredemeyer decision layered model. The technical process includes steps and heuristics for creating a good architecture. However, a technically good architecture is not sufficient to ensure the successful use of the architecture, and the organizational process is oriented towards ensuring support for, and adoption of, the architecture.

Figure 3-4 shows the technical architecting process (within the square) including the organizational process.
3.7.1 The technical process
The focal deliverable of the architecting process is the architecture document (set), motivating and describing the structure of the system through various views. However, though system structuring is at the heart of the architecting process, it is just one of the several activities critical to the creation of a good architecture. Architectural requirements are needed to focus the structuring activities. Different architectural approaches tend to yield differing degrees of fit to various system requirements, and evaluating alternatives or performing architectural trade-off analyses should be an integral part of the structuring phase. Lastly, a validation phase provides early indicators of, and hence an opportunity to resolve, problems with the architecture.

3.7.1.1 Architectural requirements

**Inputs**
The input of this phase consists of the business strategy and objectives, together with the system concept, documented in a project planning document, and an abstract architecture vision.

Before starting with formulating an architectural vision, the industrial communication environment of SattStore WMS will be studied in detail on different areas (chapter 5). With this knowledge the present situation will be studied and analysed (chapter 6). In parallel to this study a thorough study is done to the more vendor independent communication methods, as mentioned in the assignment, together with a study to trends in the industrial SattStore communication environment (chapter 7).

**Activities description**
Architectural requirements are a subset of the system requirements, determined by architectural relevance. The business objectives for the system, and the architecture in particular, are important to ensure that the architecture is aligned with the business agenda. The system context helps determine what is in scope and what is out of scope, what the system interface is, and what factors collide with the architecture. These goals are translated into a set of use cases, which are used to document functional requirements.

The system structure fails if it does not support the services or functionality that users value, or if the qualities associated with this functionality inhibit user performance or are otherwise unsatisfactory. System qualities that have architectural significance (e.g., performance and security) are therefore also important in directing architectural choices during structuring.

The architectural requirements can be found in chapter 8.

**Outputs**
The outputs of this phase are updated project plans and the architecture requirements.
3.7.1.2 Architecture specification

Input
The inputs of this phase are the architectural requirements, containing a description of the system’s environment, functional requirements and non-functional requirements, and a broad knowledge of the SatStore’s transport system, industrial protocols and standards, and middleware technologies.

Activities description
The architecture is created and documented in the architecture specification phase. This is decomposed into sub-phases, along the lines of the Bredemeyer decision layered model of software architecture.

Meta-architecture
Conclusions, important trends and recommendations from the researches to vendor independent communication methods existing in the industrial WMS environment in general, together with conclusions and recommendations from the present situation and architectural requirements and an extensive study to middleware techniques, will be used to formulate the architectural vision to guide decisions during the rest of system structuring.

Conceptual Architecture
The system is decomposed into high-level components and the responsibilities of each component, and interconnections between components are identified, without going into interface details. See chapter 11.

Logical Architecture
The conceptual architecture forms the starting point for the logical architecture. The logical architecture adds precision and incorporates the detailed architecture diagram and component and interface specifications, along with discussion and explanations of mechanisms, rationale, etc. Component specifications make the architecture concrete. These include a summary description of services the component provides, a description of the operations, constraints or pre-post conditions for each operation (in a state diagram), the concurrency model, constraints on component composition, a lifecycle model, how the component is instantiated, a typical use scenario, a programming example, exceptions, and a test or performance suite.

For reasons of completeness the complete logical architecture is described. It should, however, be mentioned that, as a result of limited time of the UCIA-project, the logical architecture will not be worked out. Priority for the UCIA-project is delivering a complete and well documented input for the logical phase, where component and interface specifications are described on a certain level of abstraction according to all views related to the conceptual architecture, rather than providing detailed component and interface specifications of some views in the logical phase. The relevant view of the logical architecture is the logical view, as mentioned in paragraph 3.5.4.

Execution Architecture
An execution architecture is created for distributed or concurrent systems. It is formed by mapping the components onto the processes of the physical system. Different possible configurations are evaluated against requirements such as performance and scaling.

The execution architecture is also described for reason of completeness but will not be no part of the UCIA-project for the same reason as mentioned above. As a result the UCIA-project will not deal with the process view mentioned in paragraph 3.5.4. Thy physical view is only mentioned on the highest level of abstraction i.a. in paragraph 4.4.2.2.
Architecture trade-off analysis

At each step in architecture specification, it is useful evaluating the different architectural alternatives against the prioritised architectural requirements. This is known as architecture trade-off analysis and it recognizes that different approaches yielding differing degrees of fit to the requirements. Selection of the best solution generally involves some compromise, but it is best to make this explicit.

Outputs

The outputs of this phase are the specified architectural guidelines and standards, where components and interfaces are described in detail, along with discussions and explanations of mechanisms and rationale.

3.7.1.3 Architecture validation

The architecture validation phase involves additional people from outside the UCIA-team to help provide an objective assessment of the architecture. In addition to make sure the architecture will meet the demands placed on it, including the right participants in this phase can help optimising the architecture.

3.7.1.4 Iterations

Though described sequentially above, the architecting process is best conducted iteratively, with multiple cycles through requirements, structuring and validation.

3.7.2 The organizational process

Architecture projects are sensitive for three major organizational sources of failure:

- The project is under-resourced or cancelled earlier by an uncommitted management
- It is stalled with endless infighting or a lack of leadership
- The architecture is ignored or resisted by product developers.

The organizational process helps address these pitfalls. Two phases, namely Init/Commit and Deployment, cover the technical process. However, the principal activities in these phases also overlap with the technical process activities.

3.7.2.1 Init/Commit

The Init/Commit phase focuses on initiating the architecture project and gaining strong commitment from upper management. The creation of the architecture vision is central both to aligning the architecture team and gaining management sponsorship. Not listening to the needs of the management, developers, marketing, manufacturing and user communities and not paying attention to gaining and sustaining sponsorship in the management and technical leadership of the organization, will lead to failure.

3.7.2.2 Deployment

The Deployment phase follows the technical process, and addresses the needs of the developers who are meant to use the architecture to design and implement products. These range from understanding the architecture and the explanation/motivation for the architecture description, to responding to the need for changes to the architecture. This entails consulting, and perhaps tutorials and demos, as well as the architects' involvement in design reviews.

The Deployment phase is described for reason of completeness but will not be any part of the UCIA-project for the same reason as mentioned for the logical and execution architecture. The development view will therefore not be described in this document.
3.8 Documenting software architectures

Architecture documentation can serve as a means of education, a vehicle for communication among stakeholders and as a basis for system analysis. Architecture documentation must balance these varied purposes. It should be sufficiently abstract that it is quickly understood by new employees. It should be sufficiently detailed so that it serves as a blueprint for construction. At the same time, it should have enough information so that it can serve as a basis for analysis.

There are different ways to document the architecture. One approach from F. Bachmann, L. Bass, P. Clements et al. is as follows [CLEM 2003]. All views describe system elements and the relationships among them. An architect can employ a common organizational scheme to document them containing a primary presentation and supporting documentation, as depicted in figure 3-5.

![Diagram of Contents of a view](image)

**Primary presentation**
A primary presentation shows elements and relationships. The primary presentation is usually a graphical architecture. If so, this presentation must be accompanied by a key that explains, or points to an explanation, of the notation used in the presentation. If the primary presentation is textual instead of graphical, it still carries the obligation to present a terse summary of the most important information in the view.

**Supporting documentation**
Supporting documentation explains and elaborates the information in the primary presentation. The supporting documentation includes the following:

- A catalogue that defines the elements and relationships of the view, including at a minimum those shown in the primary presentation. This explanation should include the interface(s) of the elements and might additionally include a list of properties, and/or a behavioural description. In addition, if there are elements or relations relevant to the view that were omitted from the primary presentation, the catalogue can present them.
- A context diagram describing the entity whose architecture is being documented in that particular view and its environment.
- Results of analyses that have been conducted, such as the results of performance or security analysis, or a list of changes required by a particular modification.
- Rationale for why the design decisions reflected in the view were made along with a list of rejected alternatives and why they were rejected.
- Glossary of terms used.
- Project management related information. The precise contents of this section will vary according to the standard practices of each organization, but this is the place where information such as authorship, configuration control data, and change histories are recorded.
4 Introduction to distributed computing

In the previous chapter, it is described how the system’s architecture will be documented and what project phases are walked through to develop a good software architecture, ensuring that the concept satisfies all system requirements and objectives. This chapter will be a logical sequence to the previous chapter as this chapter deals more with the implementation of a system’s architecture.

The UClA is an architecture which makes proper data sharing possible between SattStore and third party industrial crane and conveyor systems. Sharing data between different (sub)systems is better known as distributed computing. This chapter starts with an introduction to distributed computing in general. Different implementation paradigms will be introduced and described with their related terms and finally functional and non-functional design aspects are summarized. In spite of that this chapter mainly contains base-knowledge for electrical engineers, it could be very convenient for readers not familiar with distributed computing for understanding the rest of this graduation report.

4.1 Distributed computing

The Webopedia [WEB 2003] defines distributed computing as follows:

A type of computing in which different components and objects comprising an application can be located on different (sub)systems connected to a network.

According to this definition and the definition of a system (paragraph 3.1) any system may be considered as a distributed system. A distributed system is composed from relatively autonomous sub-systems, physically or logically distributed.

Purpose of distributed computing are:

- Share information/applications
- Distribute an application on different (sub)systems
- Distribute information on different (sub)systems
- Distribute workload on different (sub)systems

The underlying principle of all distributed computing systems is communication between sub-systems. Figure 4-1 shows the three basic communication elements that must be considered by communication between two applications on two different sub-systems according to the International Standard Organization (ISO).

![Figure 4-1: End system communication schematic](image-url)
Data network
The data network is the physical connection between two sub-systems. This can be a simple point-to-point wire link, a Local Area Network (LAN) or a Wide Area Network (WAN).

Communication sub-system
The combination of hardware and system-level software that enables sub-systems to communicate is referred to as the communication sub-system. When several sub-systems are connected to one another through a common communication sub-system, they can be considered as a network of sub-systems.

According to the ISO the complete communication sub-system is broken down into two generic function layers providing network-dependent functions and application-oriented functions. This gives rise to three operational environments:

1. The **network environment**, which is concerned with the protocols and standards relating to the different types of underlying data communication networks.
2. The **OSI-environment**, which embraces the network environment and adds additional application-oriented protocols and standards to allow end systems (computers) to communicate with one another in an open way.
3. The **real system's environment**, which builds on the OSI-environment and is concerned with a manufacturer's proprietary software and services, which have been developed to perform a particular distributed information processing task.

Application relation models
On top of the communication sub-system the real applications can be placed. The applications communicating with each other can have different application relational models. One relational model is based on which application (or component) starts the communication session. In this relational model three hierarchies can be distinguished:

- **Client/server model**, in which one application, the client, makes a service request from another program, the server, which fulfills the request.
- **Master/slave model**, in which one application is in charge of all other applications. This model is sometimes also called the server/client model, where the master (or server) pushes data to one or more slaves (or clients).
- **Peer-to-peer model**, in which either of two applications are able to initiate a transaction. Both applications (peers) have client and server functionality and can initiate a communication session.

This hierarchy, which distinguishes different node relationships, results respectively in the following connection relationships (also known as delivery mechanisms):

- **Many-to-one**: many clients can communicate with one server.
- **One-to-many**: one master (or server) is in charge of one or more slaves (or clients)
- **One-to-one**: peers communicate with each other in a one-to-one relationship.

The master/slave model is becoming an older relational model and is replaced by peer-to-peer models, which are evolving very fast. The last generations of peer-to-peer relational models have one-to-many connection relationships and even many-to-many connection relationships. Client/server and peer-to-peer relational models are further examined in the document Distributed Computing Architectures [ABB 2003-1].

4.2 Implementation paradigms
Distributed systems can be distinguished into different approaches towards implementation of distributed systems. The term *implementation paradigm* is introduced to distinguish between implementation approaches using distinct underlying implementation models, and in particular between the structural concepts being used.
Therefore, in this document, an implementation paradigm denotes an implementation approach that is characterized by its underlying structural concepts.

Two implementation paradigms are distinguished:

- Protocol-based implementation
- Object and component-based implementation

The important differences between these two implementation paradigms are listed in Table 4-1 very briefly:

<table>
<thead>
<tr>
<th>Key concept</th>
<th>Protocol-based</th>
<th>Object and component-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>and tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Related concepts</td>
<td>Protocol entity, PDU, required service, underly</td>
<td>Remote operations, stubs, skeletons, middleware</td>
</tr>
<tr>
<td></td>
<td>ing service,...</td>
<td>platforms, object layers,...</td>
</tr>
<tr>
<td>System structure</td>
<td>Stack of protocol layers where each layer consists</td>
<td>Set of objects, logically or physically distributed,</td>
</tr>
<tr>
<td></td>
<td>of multiple distributed protocol entities coopera</td>
<td>which cooperate by interacting via a middleware pl</td>
</tr>
<tr>
<td></td>
<td>ting by interacting via the underlying layers to pr</td>
<td>atform to provide some service.</td>
</tr>
<tr>
<td>Requirements</td>
<td>Specification, structuring, finite state machine</td>
<td>Types of middleware, object models, interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>definitions, transparencies.</td>
</tr>
</tbody>
</table>

Table 4-1: Protocol-based versus object and component-based

The UCIA-system, according to the assignment, must support legacy communication methods, as well as more or less vendor independent communication methods. The legacy SattStore environment towards industrial controllers is a protocol-based one, which shall become clear in chapter 6. The protocol-based environment is still evolving, however, there is an obvious trend towards the object- and component based computing environment, which uses technologies to achieve vendor independency. This object- and component based environment is used in most of the chapters of this graduation report. Both the protocol-based and object and component-based environment are introduced in the next paragraphs, together with their related terms mentioned in the table above.

4.3 Protocol-based environment

4.3.1 Protocol layering

There are many functions, which may be needed to be performed by a protocol. These range from the specification of connectors, addresses of the communications nodes, identification of interfaces, options, flow control, reliability, error reporting, synchronization, etc. In practice there are so many different functions, that a set (also known as suite or stack) of protocols are usually defined. Each protocol in the suite handles one specific aspect of the communication.

The protocols are usually structured together to form a layered design (also known as a "protocol stack"). The basic idea of a layered architecture is to divide the design into small pieces. Each layer adds to the services provided by the lower layers in such a manner that the highest layer is provided with a full set of services to manage communications and run distributed applications. A basic principle is to ensure independence of layers by defining services provided by each layer to the next higher layer without defining how the services are to be performed. This permits changes in a layer without affecting other layers.

Two leading reference models to describe the operation and design of layered protocol architectures are the OSI reference model and the Internet model or Internet protocol suite. An important difference between the models is the more architectural and theoretical approach of the OSI reference model against the more implementation-directed approach of the Internet model.
4.3.2 OSI reference model

Both the network-dependent and application-oriented (network-independent) functions of the OSI reference model are implemented as a number of layers. The boundaries between each layer and the functions performed by each layer have been selected on the basis of experience gained during earlier standardization activity.

The logical structure of the OSI reference model is made up of seven protocol layers. The three lowest layers (1-3) are network dependent and are concerned with the protocols associated with the data communication network being used to link the two communicating end systems or computers. The three upper layers (5-7) are application-oriented and are concerned with the protocols that allow two end-user application processes to interact with each other, normally through a range of services. The intermediate transport layer (4) masks the upper application-oriented layers from the detailed operation of the lower network-dependent layers. Essentially, it builds on the services provided by the latter to provide the application-oriented layers with a network-independent message interchange service. The transport layer together with the network-dependent layers are often called the lower layers of the OSI reference model, where the layers 5 till 7 are often called the upper layers of the model.

The OSI reference model layers according to the application- and network-oriented function distribution as well as to the upper and lower layer distribution is illustrated below.

![OSI Reference Model Diagram](image)

Each of the seven protocol layers performs a well-defined function, a collection of related services, in the context of the overall communication sub-system. The following functional layers are present:

**Application layer**

The application layer provides the user interface and implements the services to complete the application’s purpose. It is important to note that despite of the name, the “application” layer is related to services for applications and that it does not contain any application! End-user Application processes (APs) can be considered as forming the 8th layer, not addressed by the OSI reference model.

**Presentation layer**

This layer formats the data. For example encode an decode, compress and decompress and convert between different presentations.
**Session layer**
The session layer opens a session between hosts, controls the session between the two endpoints, then closes the session.

**Transport layer**
The transport layer provides the services that are typically associated with networking: “end-to-end” transfer of data. There is a possibility to use a reliable variant or an unreliable variant of transmitting data. The transport layer is the interface between the application-oriented layers and the network dependent layers.

**Network layer**
The network layer is responsible for establishing and clearing a network connection between two transport layer protocol entities. It includes functions such as network routing (addressing) and sometimes flow control.

**Link layer**
The link layer is built on top of the physical connection and provides functions such as error detection and, in event of transmission errors, the retransmission of messages.

**Physical layer**
This is the electrical interface. In fact this is the medium across which the bitstream between two equipments is transported.

### 4.3.2.1 Communication protocols
Figure 4-3 denotes the conversation between peer layers (similar layers). Peer layers “talk” to each other using the same communication protocol. The “conversation” between peer layers is logically independent of the conversation between other peer layers.

Between the upper peer layers of the OSI model application protocols are defined. An application protocol defines the rules governing communication between network applications. These rules describe the application’s information and how it is used by the applications sending and receiving the information. Two applications must have an application protocol in order to communicate, because it defines what the applications expect of each other. Application protocols can range from simple to complex. More complex information services require more complex application protocols.
To exchange the application data with other network systems the network dependent lower layers of the OSI model are used. The lower layer protocols, the transport-, network- and data link protocols, fill up these lower peer layers of the OSI model together with the physical interface. These lower layers control the transport of the application data up to the physical interface.

Between the application layer and the lower layers, application-support protocols can be defined. These application-support protocols support the application by defining session and/or presentation protocols.

There is thus an important difference between upper and lower layer communication, between application protocols and lower layer protocols. Communication systems that communicate with each other often use an application protocol (with or without application-support protocols) combined with a lower layer protocol or lower layer protocol stack. A communication system often can choose its “foundation”, the lower layer protocols, to transport the application data to another communication system. Communication systems have to support both the same lower layer protocols as well as the same application protocols (with or without application-support protocols) to communicate with each other. Only real peer layers can set up proper communication.

4.3.2.2 Data encapsulation

The peer-layer protocols illustrated in figure 4-3 are only conceptual, not the actual data path. The fat line shows an application passes its information to the layer below, which adds its own protocol information and passes it down to the next layer. Each layer’s information accumulates on the way down the stack. On the receiving end, each layer removes its information, interprets it, and responds according to the protocol. It then passes the remaining data up to the next layer. The application layer, at the top, receives only the application layer information. This process is called encapsulation or layering. Before explaining encapsulation in detail first the term PDU is introduced.

Data blocks sent between peer layers are called Protocol Data Units (PDU) consisting of:

- **Protocol Control Information (PCI)** in the header of the PDU
- **Service Data Unit (SDU)** in the payload of the PDU

A protocol on a layer defines the procedures which determine how the PDU will be processed at the transmit and receive nodes. The procedures specify the valid values for the PCI fields, and the action be taken upon reception of each PCI value (usually based on stored control information). Examples of procedures which are implemented in protocols include:

- Error recovery (e.g. the checkpoint procedure, the go-back-n-procedure)
- Flow control
- Segmentation
- Connection management

To explain encapsulation an OSI layer will now be considered indicated as layer N. A Service Data Unit (SDU) is a piece of information passed by a layer above (the N+1-Layer) to the current layer (the N-Layer) for transmission using the service of that layer. To transport the SDU, the current layer encapsulates the SDU by adding a protocol header containing PCI. The combined PCI and SDU is known as a PDU belonging to that layer. This forms the SDU of the layer below, etc. The reverse process happens on reception by the receiver at the corresponding peer layer N.

Figure 4-4 shows the encapsulation process.
The encapsulation procedure is being controlled by the Protocol Entity. The protocol entity is the code that controls the operation of a protocol layer. It performs the peer-to-peer procedures, encapsulate frames, multiplex/demultiplex traffic, etc.

Taken the OSI reference model the following happen by using the explained encapsulation. An application process sends data to the application layer that adds the PCI for that layer. The data together with the PCI forms the APDU (Application Protocol Data Units) of the application layer. This APDU serves as input for the layer below it (PSDU). The presentation layer adds his PCI. The presentation PCI together with the PSDU delivers the PPDU (Presentation Protocol Data Units), etc.

4.3.2.3 Service interface between layers

As seen in the previous paragraph during peer-to-peer communication, information flows down through lower layers in the same node, across the communications path and up through layers in the other node until it reaches the peer layer. Boundaries between adjacent layers in the same system are called interfaces in OSI terminology, with a Service Access Point (SAP) at the boundary. The SAP is used to access services provided by a lower layer to a higher layer (or vice versa). As seen in the previous paragraph the N+1-Layer uses the services of the N-layer for the exchange of the N+1-layer PDU data (Layer N+1 is the service user). Within the N-layer this N+1-layer PDU data was called N-layer SDU data as a result of the service the N-layer executes to transmit the data (Layer N is the service provider).

The SAP between layer N+1 and layer N as well as the data path of the N+1-layer PDU’s is shown below. As known from the previous paragraph this is a conceptual path, the real data path is different.
Services can be accessed through the use of *service primitives* defined in the SAP. A service primitive initiates an action or advises the result of an action. Each primitive may contain parameters to convey the PCI needed to perform its functions. There are four types of primitives used for communicating data:

1. **Request**: A primitive sent by the N+1-layer to layer N to request a service. It invokes the service and passes those parameters needed to fully specify the request.
2. **Indication**: A primitive returned to the N+1-layer from layer N to advise of activation of a requested service or of an action initiated by the layer N service provider.
3. **Response**: A primitive provided by the N+1-layer in reply to an indication primitive. It may acknowledge or complete a procedure previously invoked by the indication primitive.
4. **Confirm**: A primitive returned to the requesting N+1-layer by the N-layer to acknowledge or complete a procedure previously invoked by the request primitive.

Each type of primitive has a variety of forms and thus fields for different purposes. For example, there are Connect Request, Data Request, Flow Control Request, Disconnect Request, and a variety of other forms of Request primitives plus similar forms for Indication, Response, and Confirm primitives. Primitives are also identified by the protocol layers using them.

Figure 4-6 shows the basic communication time-sequence between two layers by using the four primitives as well as an example service primitive request layout.

![Figure 4-6: Basic service primitive time sequence + example service primitive](image)

### 4.3.2.4 Connection-oriented and connectionless services

The network, transport and upper layers in general offer two types of services:

- **Connection-oriented service**: Both service user and provider(s) (OSI layers) are involved with the party-agreement. The agreement is only achieved if all parties agree with the data-exchange conditions. Phases of connection-oriented services: establishment, data transfer and release.
- **Connectionless service**: No negotiations about an agreement between parties. Each data exchange is considered to be independent of other data exchanges.

### 4.3.3 Internet Model

#### 4.3.3.1 Protocol layers

TCP/IP stands for Transport Control Protocol/Internet Protocol. TCP was developed in the 1970s and its basic structure was established between the 1977 and 1979. TCP/IP development was initially funded by the U.S. Department of Defence (DOD) and is actually a collection of protocol, a protocol stack. The DOD had a wide assortment of communication equipments including radios, landlines, and satellite systems. As a result, the principal purpose behind TCP/IP is to link together different communication networks.

The in the previous paragraphs showed peer layer, data encapsulation and service concepts are also valid for the Internet model. The Internet model as well as the mapping on the OSI reference model is shown below.
Application layer
The application layer consists of application-protocols, which are sent over the TCP/IP network, like DNS (Domain Name server), SNMP (Simple mail transfer protocol), FTP (File transfer protocol), HTTP (HyperText Transfer Protocol), etc., which take care of the application-to-application data delivery.

Transport layer
The transport layer offers an end-to-end data-transfer service between application-processes (application-programs). This layer consists of the TCP or UDP protocol. TCP is a transport layer protocol that uses connection-oriented services, that moves multiple packet data between applications (UDP is the connectionless variant). With the TCP/IP suite, independent of the underlying data communication network, the Internet protocol (IP) is always present in the network/internet layer.

Network layer
The network layer offers an end-to-end data-transfer service between computersystems (called hosts). This layer consists of the IP protocol.

Subnetwork interface layer
The subnetwork interface layer hides the details of the physical network and take care of the sending and receiving of network-layer packets. Most used is LAN, WAN or Ethernet due to its ability to switch packets from computer systems on any network to another network, regardless of network peculiarities, operating system differences and other packet differences.

4.3.3.2 Socket API
The majority of open distributed applications are based on the client/server model. In the case of a TCP/IP suite, the user interface at the client side is normally an integral part of the client application protocol. The TCP/IP protocol stack can be used with the help of an API (Application programming interface). This API offers a standard set of command with which the application can use the protocol stack. There are different API's available. Most widely used API to the TCP/IP protocol stack is the Socket interface, which is based on the client/server architecture. Application protocols/processes can be directly sent and received with the help of sockets (combination of a TCP port number and IP address) using the TCP/IP protocol stack. Main service primitives used by the socket interface are:

- Creating/closing sockets, attaching to the network, setting options
- Sending/receiving (writing/reading) data through them
4.4 Object and component-based environment

4.4.1 DOC and DCC

A recent concept developed in distributed computing systems is distributed objects. As mentioned before in chapter 3 an object can be defined as any item that can be individually selected and manipulated. Distributed Object Computing (DOC) can be seen as a logical consequence to the invention of object-oriented programming. One of the major advantages of object-oriented programming is that the programs can communicate with one another and yet be independent of the program it is talking to, meaning that different object programs can use the common objects. Another advantage of DOC is that the objects can be written in any object-oriented language.

Object-oriented systems run on a single computer. This is where DOC comes into play. It enable object-oriented systems to allow their objects to be located on different (sub)systems and allows both business logic and data, to be distributed across a heterogeneous network.

The difficulties of creating reusable objects are similar to those of local object technologies, such as the Foundation Classes for Java or C++. In addition the designer of a distributed server object needs to use particular implementations of persistence, transaction, concurrency control and security services, all of which make it even more difficult to reuse a server object in a different setting.

Adding persistence to objects is essential since most application programs need to deal with persistent data.

A transaction differs from a simple sequence of commands in the integrity: provided that during its execution an error or another unrecoverable problem occurs, such as a denial of access to a resource, the call will automatically be returned and the system continues the execution from a well-defined state.

Concurrency is the ability to process multiple tasks at the same time.

These obstacles led to the development of Distributed Component Computing (DCC), which is the most recent concept developed in distributed computing systems.

A component is earlier defined as a small binary object or program that performs a specific function and is designed in such a way to easily operate with other components and applications.

DCC improves on DOC in several ways. In DOC, connections between objects are encapsulated into objects and cannot be configured easily by external software artefacts. The use of system services, i.e. non-functional aspects, must be hard coded into objects and mixed with the functional aspects of the application.

The basis for both DOC and DCC is client/server computing. Communication across a heterogeneous network providing location and technology transparency and programming language and platform independency is realized using middleware technologies. The terms client/server computing and middleware are discussed in the next paragraphs.

4.4.2 Client/server computing

The client/server computing paradigm came in the 1980s in response to the need to share centralized data stored in a mainframe computer with a large number of end users, all employing the processing power of increasingly powerful desktop computers. Sophisticated Graphical User Interfaces (GUI's) have been made possible through efficient utilization of increased processing power on the client computing systems. It helps to distribute business and application logic to a user's system and to increase the performance as processing is being distributed between server and client machines. To make the idea behind client/server computing clear on a meta-level, several views can be used. The ones discussed shortly in this paragraph are the logical view and the physical view.
4.4.2.1 Logical view: software layers

The logical view includes many levels of abstraction. On a high level of abstraction software applications can be divided into logical layers. All logical layers together form the application. Each layer in the software is responsible for a specific task in the application. A layer may have compile-time dependencies on the layers below it, but should not have such dependencies on layers above it. The logical layering of an application does not need to be the same as the physical layering (tiers) of an application. In theory 6 layers are distinguished in software, as shown below.

![Software Layering Diagram](image)

**Presentation manager**
A presentation manager defines how something is displayed to the user. It is also responsible for the infrastructure of user interface elements that are possible in an application. A presentation manager is not application dependent.

**Presentation logic**
The presentation logic layer is responsible for what is displayed to the user and is application dependent.

**Application logic**
The application logic layer contains the actual application logic. This is the layer where the application functionality is defined. The application logic layer is application dependent. In practice, this layer is the least recognizable of all layers. This layer is typically spread over all other layers with most of its logic ending up in the presentation logic layer.

**Business logic**
The business logic layer contains the business rules of an organization. All the business rules should be shared between all the applications of the organization. This will ensure that changes in business rules will propagate through all the organization's applications. This layer is not application dependent but organization dependent.

**Data logic**
The data logic layer contains the data dictionary of the application (or even organization). This layer is application (or even organization) dependent.

**Data manager**
This layer is responsible for the actual storage of data. This layer can be application/organization dependent but most of the time it is not. Most of the time the database manager is a commercial, off-the-shelf application, which is able to manage many applications. In practice, the data logic layer and the data manager layer are almost always one layer.
4.4.2.2 Physical view: 2-tier, 3-tier, n-tier

It is pleasant, though not always possible, to work at a logical level of design. An arrow between layers could mean A calls B for some service. The physical view shows how a request actually gets from A to B. In the physical view processes are allocated to processors and the network is defined. Physical choices impose real restrictions on logical options. Often the physical architecture is fairly well known ahead of time because of company standards or the need to build on existing structures. Therefore another view to see distribution is mentioned here as being based on tiers.

Tiers are functional layers of entities participating in the distribution.

Fat and thin clients

Architectures based on tiers all have the same basic idea with fat clients and fat servers. It is all about how to split the client/server application into functional units. The most typical functional units are:

- **Presentation logic**: user interface
- **Business (Application) logic**: actual task
- **Data logic**: shared data

Application or business logic can reside on the client (fat client) or on the server side (thin client, or fat server) in the form of stored procedures. Figure 4-9 gives an overview of fat and thin clients.

![Fat and thin clients](image)

2-tier architectures

A typical 2-tier client/server architecture partitions functionality in such a manner that the client application performs both business logic and user interface operations provided by the presentation and application logic and the server in the back-end is used as a data manager, a file server or even database repository. The 2-tier architecture works well in relatively homogeneous environments with business rules that do not change very often. Because of the homogeneity it is hard to integrate legacy applications. Scalability is also rather limited because the client and server exchange "keep alive" messages continuously, even when no work is being done, thereby saturating the network. This 2-tier design is functional in many business scenarios, but architectural limitations, development pitfalls and lack of support for software reusability have become apparent.

3-tier and n-tier architectures

A solution to the inflexibility of the 2-tier model is separating the responsibilities into more than two tiers. The logical separation and loose coupling of tiers isolates each tier from change in the other tiers. Typically a middle tier, which is encapsulating the business logic and application logic, is introduced in between the other two tiers, forming a 3-tier architecture. The 3-tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the 2-tier model) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. (The definitions of these terms are described in paragraph 4.5).
The three tiers of a 3-tier architecture are:

- **Tier 1: GUI tier.** Tier 1 (client tier) is typically a GUI, which is responsible for the visual aspects of the application, displaying the client objects and applying the client view of the data. The client interacts with the middle tier (tier 2) to perform any action or display data.

- **Tier 2: Middle tier.** Tier 2 (middle tier) is a server, which implements the "business logic" of the application. The middle tier server objects can interact with many clients but they rely on tier 3 to get data from equipment or from the databases. The server objects provide a unified view of the different data sources. The middle tier software concentrates on implementing domain-specific logic and therefore it is not limited by a specific access technique or a database being employed. The middle tier server also improves performance, flexibility, maintainability, reusability, and scalability by centralizing business logic. Centralized business logic makes administration and change management easier by localizing system functionality so that changes must only be written once and placed on the middle tier server to be available throughout the systems. With other architectural designs, a change to a function (service) would need to be written into every application.

- **Tier 3: Resource tier.** Tier 3 (resource tier) is typically a "legacy application", which can be used to extract data and perform actions, but which implements it in a specific way.

The trend to 3-tier systems is mainly driven by technological changes and necessity to integrate software systems, which were not originally designed to work together. Extending the middle tier by allowing accessibility to multiple application objects rather than a single one, results in a new architecture called n-tier. As the name implies an n-tier application is an application in which the six software layers are divided over an unknown number of tiers with the purpose to provide even more scalability and ease of integration.

Distributed client/server computing architectures are further described in Distributed Computing Architectures [ABB 2003-1]. It gives a more detailed technical overview of 2, 3 and n-tier architectures, together with their purposes, origin, examples and functional design aspects (as listed in paragraph 4.5).

### 4.4.3 Middleware

Communication techniques providing platform and language independency, remoteness transparency and services to help support the implementation of the middle tier in 3 or n-tier architectures have come to be known as middleware.

*Middleware is a distributed software layer, or platform, which abstracts over the complexity and heterogeneity of the underlying distributed environment with its multitude of network technologies, machine architectures, operating systems and programming languages.*

Different transport protocols have in common that they can transmit messages between different (sub)systems. If the communication between distributed systems is programmed at this level of abstraction, application engineers need to implement session and presentation layer (protocol-based). Components of distributed systems may be procured off-the-shelf and may include legacy and new components. As a result they are often rather heterogeneous. This heterogeneity comes in different dimensions: hardware and operating system platforms, programming languages and, when using middleware, the middleware itself.

Implementing session and presentation layer, together with platform and programming language independency, is too costly, too error prone and too time-consuming. Instead, application engineers should be able to request parameterised services from possibly more than one remote component and may wish to execute them as atomic and isolated transactions, leaving the implementation of session and presentation layer to middleware technologies.

Mapping the placement of middleware in the communication sub-system to the OSI-reference model, middleware can be seen as a replacement of the session, presentation and application layers. When using middleware the client application does not need any knowledge of the underlying network layers.
The mapping of middleware onto the OSI reference model is illustrated below.

![Middleware mapped onto the OSI-reference model](image)

When using middleware the client is also transparent for the location of distributed components, whether they are located on the same sub-system or on a different sub-system. Object oriented middleware techniques (DOC middleware) installs end points on the client side (client stubs or proxies) and on the server side (server stubs or skeletons) in order to mask remoteness and networking details. The client submits an invocation to the local stub. The local stub **marshals** the data and transmits it across to the server stub. The server receives the message and the remote server stub **unmarshals** the data. The results are returned to the client in the same indirect way. The return of results is marshaled by the server stub, sent to the client, and unmarshaled by the client stub.

Marshalling is the process of gathering data from one or more applications, putting the data pieces into a message buffer, and organizing or converting the data into a format that is prescribed for a particular receiver or programming interface. When the server receives the marshaled data, the elements are unpackaged back into an object or request (unmarshalling).

Component-oriented middleware techniques (DCC middleware) improves location transparency and platform and language independency by putting the distributed components (e.g. objects) in **containers**.

A container provides an application context for one or more components and provides management and control services for the components.

Containers are responsible to:

- Manage the lifecycle of components and notify components about lifecycle events such as **activation**, **passivation**, transaction progress.
- Provide components uniform access to services such as transactions, security and persistence.
- Register and deploy components.

Events are objects used to transmit state change information.

Activation and passivation are necessary because it is not feasible to have all server implementations running all the time. Thus, there must be a way to activate and deactivate servers on demand.

Because of the fast growing importance of location and technology transparency and programming language and platform independency, middleware technologies will be discussed in a separate chapter, chapter 10.
4.5 Functional and non-functional design aspects of distributed systems

In a distributed system it is important to pay attention how to implement different functional and non-functional aspects. The following design aspects are valuable for distributed systems in general but it should be noticed that most of these design aspects are addressed by middleware. The knowledge of this paragraph is used in all of the chapters, but mainly in the chapters 8 and 10.

Network communication

Different components of a distributed system may reside on different sub-systems. In order for the distributed system to appear as an integrated computing facility, the components have to communicate with each other. This communication can only be achieved by using network protocols, which are often classified by the ISO/OSI reference model (see paragraph 4.3).

Distributed systems are usually built on top of the transport layer, of which TCP or UDP are good examples. The layers underneath are provided by the network operating system. Different transport protocols have in common that they can transmit messages between different (sub)systems. If the communication between distributed systems is programmed at this level of abstraction, application engineers need to implement session and presentation layer. This is too costly, too error prone and too time-consuming. Instead, application engineers should be able to request parameterised services from possibly more than one remote component and may wish to execute them as atomic and isolated transactions, leaving the implementation of session and presentation layer to middleware technologies.

Coordination

As a result of the fact that components reside on different (sub)systems, distributed systems have multiple points of control. Components on the same host execute concurrently, which leads to a need for synchronization. This synchronization needs to be implemented in the session layer or can be provided by middleware technologies and can be achieved in different ways:

Synchronous

A component can be blocked while it waits for another component to complete execution of a requested service. Object-oriented developers prefer request/response semantics. Two message types are used:

- Requests contain in/in-out arguments
- Results carry out/in-out arguments and results

Deferred synchronous

After issuing a request, a component can also continue to perform its operations (non-blocking) and synchronize with the service-providing item at a later point. If this synchronization is initiated by the client this is often called deferred synchronous. To retrieve the result call-back or polling mechanisms can be used.

- Call-back The client supplies an additional object reference with each request invocation. When the response arrives this object reference is used to deliver the response back to the client.
- Polling The client invokes an operation that immediately returns a value type that can be used to either poll or wait for the response.

Asynchronous communication

Like deferred synchronous with the difference that synchronization is initiated by the server. Also the call-back mechanism can be used as described in deferred synchronous mode.

Asynchronous communication can also achieved by decoupling the client and the server by introducing intermediary message queues between senders and receivers.
Messages are objects send from a sender to a receiver. A sender sends messages, while a receiver receives them. A Message API is provided for senders and receivers to send/receive message. Queues are used to store messages persistently. They cooperate with other queues for message routing.

A possible asynchronous communication mechanism is publish/subscribe. An event queue is storing events. Publishers create events and store them in an event queue with which they have previously registered. Consumers register (subscribe) with event queues from which they retrieve events. Filters could be used to filter events on behalf of subscribers.

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. The use of a synchronous request/reply mechanism requires that the client and server are always available and functioning (i.e. the client or server is not blocked). The asynchronous mechanism does not guard against overloading a network. As such, a negative aspect of asynchronous messaging is that a client process can continue to transfer data to a server that is not keeping pace.

Reliability/security
Network protocols have varying degrees of reliability. Protocols that are used in practice do not necessarily guarantee that every packet that a sender transmits is actually received by the receiver and that the order in which they are sent is preserved. Thus, distributed system implementations have to put error detection and correction mechanisms in place to cope with these unreliabilities.

Unfortunately, reliable delivery of service requests and service results does not come for free. Reliability has to be paid for with decreases in performance. To trade-off reliability and performance in a flexible manner, different degrees of service request reliability are needed in practice.

For communication about service requests between two components, the reliabilities that have been suggested in the distributed system literature are:

- **Best effort**
  Best effort requests do not give any assurance about the execution of the request.

- **At-most-once**
  Guarantee the request to execute only once. It may happen that they are not executed, but then the requester is notified about the failure.

- **At least-once**
  Guarantee the request to be executed, possibly more than once.

- **Exactly-once**
  Provides the highest degree of reliability and guarantees the request to be executed only once.

Heterogeneity
The components of distributed systems may be procured off-the-shelf and may include legacy and new components. As a result they are often rather heterogeneous. This heterogeneity comes in different dimensions: hardware and operating system platforms, programming languages and, when using middleware, the used middleware technology.

Hardware platforms use different encoding for data types, such as numbers and characters. This heterogeneity should be resolved by middleware rather than by the application engineer. When integrating legacy items with newly built components, it often occurs that different programming languages need to be used. These programming languages may follow different paradigms. While legacy components tend to be written in imperative languages, such as ADA or C, newer items are often implemented using object-oriented programming languages. Even different object-oriented languages have considerable differences in their object model, type system and approach to inheritance. These differences need to be resolved by middleware.

There is not just one, but many approaches to middleware. The availability of different middleware solutions may present a selection problem, but sometimes there is no optimal single middleware, and multiple
middleware systems have to be combined. Thus middleware will have to be interoperable with other implementations of the same middleware or even different types of middleware in order to facilitate distributed system construction.

**Scalability**
Scalability denotes the ability to accommodate a growing future load. In centralized client/server systems, scalability is limited by the load that the server host can bear. This can be overcome by distributing the load across several hosts. The challenge of building a scalable distributed system is to support changes in the allocation of items to hosts without changing the architecture of the system or the design and code of any item. This can only be achieved by using distributed objects/components and respecting the different dimensions of transparency, which are listed below.

- **Access transparency**
  Demands that the way a component accesses the services of another component is independent of whether it is local or remote.

- **Location transparency**
  Demands that components do not know the physical location of the components they interact with.

- **Migration transparency**
  If components can access services without knowing the physical location and without changing the way they request it, load balancing mechanisms can migrate components between machines in order to reduce the load on one host and increase it on another host. It should again be transparent to users whether or not such a migration occurred.

- **Replication transparency**
  Replication can also be used for load balancing. Components whose services are in high demand may have to exist in multiple copies. Replication transparency means that it is transparent for the requesting components, whether they obtain a service from the original component or a copy.

- **Concurrency transparency**
  Several users or application programs can access objects simultaneously (for example shared data) without mutual influence.

- **Scalability transparency**
  Requires that it should be transparent to the designers and users of a component how scalability is to be achieved. It is achieved by replication and migration transparency.

- **Performance transparency**
  Requires that load is balanced across multiple copies of a component such that the user does not see any reduction in performance as the number of transactions processed increases.

- **Failure transparency**
  This facility requires that users are unaware that a failure has occurred, that recovery occurs automatically and that transactions are rerouted to other servers and processes.

The different transparency criteria that will lead to scalable systems are very difficult to achieve if distributed systems are built directly on network operating system primitives. To overcome these difficulties, it is demanded from middleware that they support all mentioned transparencies.

**Usability and maintainability**
Usability is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

Maintainability denotes the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment.
5 SattStore transport environment

To describe the environment of SattStore and the cranes and conveyors to be controlled, called the SattStore transport environment, two aspects are important: SattMate and Computer Integrated Manufacturing (CIM). Within SattStore the transport equipment communication handling part SattMate is used to set-up communication with i.a. cranes and conveyors. Next to this SattStore internals it is also important to gather more knowledge about the environment in which SattStore, cranes and conveyors operate. Both SattStore and industrial cranes and conveyors are part of the CIM environment represented by the CIM pyramid. CIM divides the industrial environment into well-defined functional levels. SattStore as well as the cranes, conveyors and matching crane and conveyor controllers can thus be mapped onto the CIM pyramid. CIM will be explained first before looking at the SattMate system.

5.1 Computer integrated manufacturing (CIM)

5.1.1 CIM functional levels

Manufacturing systems demand efficient and reliable transmission of information from measurement and control devices on the plant floor all the way up to applications across the manufacturing enterprise. The task of specifying, designing and implementing the information system for a manufacturing enterprise is complex. It is simplified by considering sub-components of the system, with well-defined interfaces. The task is further simplified by arranging these components hierarchically in layers, where each layer only communicates with the layer directly above and directly underneath. The different activities or functional levels of a manufacturing company, from the manufacturing process up to the enterprise direction, comprising sales, purchasing, design engineering, industrial engineering, production planning, etc., can be represented by an organizational pyramid, called the CIM pyramid as shown below. All entities (applications and/or equipments) belonging to a specific functional level of this pyramid have the same sort of functionality in the manufacturing environment.

![CIM hierarchical pyramid](image)

**Figure 5-1: CIM hierarchical pyramid**

**Enterprise level**

At enterprise level, the whole enterprise is considered. Starting from marketing, research, the strategic enterprise objectives, long-range forecasting and planning are obtained. The purchases and sales section can be found at this level.
Factory level
The factory level deals with a single factory and is responsible for managing the whole plant. Its main activity is production planning and control. Starting from the definition of a product, it designs and defines the processes, their sequences, the material required, the machine tools and part programs which are necessary in order to obtain the product. Because of the correspondence in applications on the enterprise and factory level, both levels are often taken together. Typical entities at the enterprise/factory level are enterprise/factory controller applications like ERP and other management applications.

Area level
At area level, each area is responsible for coordinating the activities of all the store, transport or production cells it is composed of. This level mainly deals with shop floor order handling (receiving, transporting, controlling, storing, etc.). Typical entities at the area level are area controller applications like WMS for the control of the warehouse area, production management system for the control of the production area, etc.

Cell level
The cell level deals with groups of machines or handling units called cells. Some important tasks of a manufacturing cell can be goods storage and retrieval, transportation of goods and the manufacturing of parts. Typical entities used at the cell level are cell controller applications that supervise and control a group of machines or device controllers. Cell controllers contain SCADA (Supervisory Control and Data Acquisition) functionalities.

A handling unit is a vehicle or other mechanical equipment that moves a load (e.g. goods) from one position to another. Different handling unit types are cranes, conveyer belts and forklift trucks equipped with an onboard terminal.

A cell controller is an application that supervises and monitors the resources in a particular industrial manufacturing cell. Resources are a combination of device controllers, intelligent equipment, sensors/actuators, etc. Cell controllers are in general complex applications, but offer in most of the cases a comprehensible interface towards supervision systems (systems on top of the cell controller) and resources in the cell. A cell controller allows the following main functions:

- Integration of the cell resources
- Control and supervision of the cell
- The generation, storage and exchange of the different commands to the different resources in the cell
- The reaction to fault conditions in the cell

SCADA is the level of applications that monitor and control devices such as programmable controllers. These systems are usually PC or workstation based.

Control level
The control level deals with one machine or handling unit type. At the control level, signals from the sensors on the handling unit are processed in a machine or device controller and the commands to the actuators on the handling unit are generated (this is called a control loop). The usual entities at the control level are machine or device controllers like PLC, CNC, RC, etc.

PLC (Programmable Logic Controller) is a class of industrially hardened devices that provides hardware interface for input sensors and output actuators. PLCs can be programmed using relay ladder logic to control the outputs based on input conditions and/or algorithms contained in the memory of the PLC.

CNC (Computer Numerical Control) allows the control of motion in an accurate and programmable manner through use of a dedicated computer within a numerical control unit, with a capability of local data input such that machine tools are freed from the need for "hard-wired" controllers.

RC (Robot Controller) takes care of the data exchange with industrial robots. RCs can be loaded with a default or custom program that will handle most robot control needs.
Field level
This level contains sensors and actuators for direct action on the manufacturing process. The sensors/actuators provide discrete or continuous analogue signals. The usual entities at the field level are temperature, flow, pressure and digital I/O sensors/actuators.

A sensor is a device, which responds to an input quantity by generating a functionally related output usually in the form of an electrical or optical signal (this signal is sent to a control device).

An actuator is a device, which responds to an input electrical or optical signal (descended from a control device) by generating a functionally related output in the form of an output quantity.

The functional CIM layers give directives to the manufacturing environment. It is possible that a specific functional level is not present in the manufacturing environment according to a specific manufacturing demand. If there is for example a simple transport operation of goods (involving one handling unit with one device controller) the cell level could not be present, because there is no need for a cell controller that covers and coordinates device controllers in this situation. In this case an area level application directly addresses a device controller and offers the information the device controller needs.

5.1.2 CIM information flow
As mentioned before the entities on the different functional levels exchange information according to well-defined interfaces. An entity of a specific pyramid level gets its information from a level-entity lying hierarchically on top of the level. This on top lying entity is often called the host system, because of the hierarchical role in the pyramid.

An enterprise/factory level controller application (e.g. an ERP application) sends an order job to the area controller application on the area level according to the area controller interface. The enterprise/factory controller application is in this case the host of the area controller application. This area controller application creates on the basis of the order job on his turn one or more cell jobs for specific manufacturing cell controllers that will be sent to the different cell controllers according to the cell controller interfaces. The cell controllers on the cell level understand these jobs and generate the right device jobs for the different device controllers belonging to the cell. Finally the device controllers take care of the communication with the sensors/actuators, the field data. Next to the exchange of jobs also status information could be exchanged. A status is for example a reply on a specific job (e.g. job finished) to the controller that generated that job. It should be noticed that statuses are not always a reply on a job but can also be created as a reaction on a specific condition in the manufacturing environment. The here described chain of data exchange, or information flow between the different levels of the industrial manufacturing environment, is often called message passing and can be represented as a tree as shown below.
An interface between the enterprise or factory controller and the area controller is called an area controller interface, an interface between an area controller and a cell controller a cell controller interface, an interface between a cell controller and a device controller a device controller interface and finally an interface between a device controller and the connected sensors/actuators is called a sensor/actuator interface.

To exchange information between the different entities each interface has got communication requirements to fulfil the communication needs of the entities connected to it. Most important is the type and frequency of the data being exchanged between the entities. The protocols used, define the principal network characteristics, such as the speed of data transmission, length of data packets, degree of data security, etc.

5.2 Mapping SattStore on CIM

SattStore is a warehouse management system that takes care of the administration and control of all activities in the warehouse area of a manufacturing plant. SattStore WMS is thus according to the CIM functional layering an area level application. SattStore gets its order jobs from enterprise/factory ERP applications with the help of an ERP interface or interfaces. Next to the manufacturing environment SattStore can also be used in a real distribution environment, without being a part of a manufacturing process (there is for example no production area). In this case the environment consists only of one area, the warehouse area, getting again information from an on top lying ERP application or applications.

Next to the ERP communication SattStore has to control different transport equipment in the warehouse to transport, store and retrieve goods. The equipments considered in the UCIA project are industrial cranes and conveyor belts. First take a look at these equipments and the matching functional position in the CIM pyramid and the matching information flow or message passing.

5.2.1 Industrial cranes

The industrial cranes used in a warehouse are automatic crane and models, such as box cranes and pallet cranes, from different makes. These cranes have as main purpose to pickup goods from a specific store-location in the warehouse and put them on for example a conveyor belt or the other way around. To do this the cranes can horizontally move to the pickup or deposit location according to a pre-defined working-range.

A working range is a range of locations within which a vehicle (crane, transfer car) is working.

To control the cranes, jobs are sent to the cranes. Typical jobs sent to the cranes are positioning (drive to a location), pickup (pickup specific goods) and deposit (deposit specific goods) assignment commands. Typical status information from the cranes are assignment acknowledge commands (crane has received the assignment command) and completed commands (job finished by crane).

5.2.1.1 Cranes and CIM functional levels

Industrial cranes always communicate with the help of a crane cell controller application (CIM cell level) as master of a crane cell. This crane cell controller is the host system of a couple of crane device controllers (CIM control level) that make the connection with the sensors/actuators on the cranes (CIM sensor/actuator level) and controls the cranes. To communicate with the different industrial cranes SattStore has to set-up communication with the crane cell controller. Figure 5-3 shows the mapping of the industrial crane communication on the CIM pyramid, together with SattStore and the ERP application.
5.2.1.2 Crane information flow

The information flow (the message passing) used when communicating with cranes is also visualized in figure 5-3. The ERP application sends an ERP order job to SattStore. This job may imply the transportation of goods with the help of cranes. When a crane action is needed the right crane cell job is sent to the crane cell controller. The crane cell controller generates the right crane device jobs and waits for status replies from the different crane device controllers. The crane device controllers generate the right crane data and receive the right statuses from the sensors/actuators on the crane. When the complete crane cell job has been executed a status command is sent back to SattStore.

A crane device job is a command for a specific crane. This command is positioning, pickup or deposit. A crane device status is a status from a specific crane. This can be acknowledgment or completed (see the introduction of paragraph 5.2.1). The crane data/status imply the control of the sensors/actuators on the crane e.g. for moving the fork of the crane.

A crane cell job is often a bundling of device jobs for a specific crane (identified with that specific crane’s number and other needed information). If SattStore for example likes a crane to go to a specific position, wants the crane to pickup goods and further deposit other goods, this can be done by sending one crane cell job to the crane cell controller. The crane cell controller unpacks the cell job and sends and receives the commands and statuses (the device jobs and statuses) to and from the right device controller. If the job has been completed a status report is sent back to SattStore. Notice that this is often the case, but not all crane cell controllers want the crane information to be a bundling of device jobs.

5.2.2 Conveyor belts

Conveyors used in a warehouse move goods from one location to another. To control the conveyors, jobs are sent to the conveyors. The typical type job sent to the conveyors is a positioning command (transport goods to a location). Optional status information can be received from the conveyors when accessing the location.

5.2.2.1 Conveyors and CIM functional levels

Conveyor belts are controlled by conveyor device controllers, in the case of SattStore always PLCs (CIM control level). These PLCs take care of the communication with the sensors/actuators on the conveyor belts (CIM sensor/actuator level). Notice that conveyor belts do not use a cell controller that covers a conveyor device controller or a group of conveyor device controllers (CIM cell level not used). A conveyor PLC can control more conveyor belts (sensors/actuators on the conveyor belts) dependent on the needs of the system. To communicate with the different conveyors SattStore has to set-up communication with the conveyor PLC.
Figure 5-4 shows the mapping of the industrial conveyor communication on the CIM pyramid, together with SattStore and the ERP application.

![Diagram of conveyor communication mapped onto the CIM pyramid]

**5.2.2.2 Conveyor information flow**

The information flow (the message passing) used when communicating with conveyors is also visualized in figure 5-4. The ERP application sends an ERP order job to SattStore. This job may imply the transportation of goods with the help of conveyors. When a conveyor action is needed the right conveyor PLC jobs (position jobs) are sent to the conveyor PLC (or different PLC types) on request of the conveyor or on request of SattStore. When a PLC job has been executed a status command is sent back to SattStore. This status command is optional.

A conveyor PLC job is a command for a specific conveyor, in this case a positioning command. A conveyor PLC status is a status from a specific conveyor when a position has been reached (see the introduction of paragraph 5.4.2). From SattStore’s point of view the conveyor PLC jobs / statuses for a specific conveyor belt sent to the PLC are always the same and there is never an exchange of bundled commands or jobs in this situation. Because of this simplicity of the commands the absence of a cell controller is explicable.

The conveyor data/status imply the control of the sensors/actuators on the conveyor e.g. for starting and stopping the conveyor-motor.

**5.2.3 Crane versus conveyor communication**

There is thus a clear difference between crane and conveyor communication. When communication with cranes the crane cell jobs being exchanged with the crane cell controllers dependents on the capabilities of these crane cell controllers and can be different in each situation. When communicating with conveyors always the same conveyor PLC jobs are exchanged.

**5.3 SattMate System**

SattMate is the transport equipment communication handling part of the SattStore WMS. The SattMate system consists of a core and different sub-systems.

The **SattMate Core** is the base level needed for all possible configurations of a SattMate system. It also contains basic software packages used by the other sub-systems in the system. The Core is an integrated part of the SattMate concept and complies with the disposition of other components in the concept.
A **SattMate Sub-system** is a part of the complete SattMate system that controls or administrates a handling unit type. Several sub-systems for several handling unit types are allowed in the SattMate system. A sub-system is identified by a unique sub-system number and a unique sub-system name and belongs to a specific sub-system object type.

Before deepening the core and sub-systems first the term transport assignment is explained to obtain a better understanding of the functionalities of the core and sub-systems.

### 5.3.1 Transport Assignments

A **transport assignment** is a requirement to move a load from one location to another. There are two types of assignments managed by the system: **Automatic assignments** and **Semi-automatic assignments**.

**Automatic assignments are assignments under control of the SattMate system ensuring safe restart. Automatic assignments are initiated by supervisory administrative software and dispatched automatically between the different SattMate sub-systems.**

**Semi-automatic assignments are assignments are issued to one sub-system using a semi-automatic assignment. These assignments are not automatically restored after a system breakdown.**

Information about the automatic assignments is stored in an assignment table. The assignment table holds one record for each automatic assignment created. An automatic assignment is accessed with an assignment identifier and will result in one or many **sub-assignments** that will be executed according to a particularly **sequence**. Information about semi-automatic assignments is not saved to disc and will result in only one sub-assignment. No sequence will be used in this case.

A *Sub-assignment is a part of an assignment, which is carried out in one sub-system. A sub-assignment is e.g. the transport of a load by a conveyor belt from one location to another in the conveyor layout.*

A **Sequence** is a set of sub-system identities and function entries that should manage an automatic assignment for a given path. A sequence is accessed with a sequence identifier and consists of different steps.

Each step is one step in the sequence. A step holds the following information:

- Sub-system identity and name that has to execute the sub-assignment.
- Location data: Next location of a specific load.
- Assignment type: Type of the sub-assignment e.g. transport.
- Entry identities for send and receive: Definition of the function, which is to be performed in the step. An entry identity points out a piece of program code to be executed in a sub-system. The send entry let the function start, the receive entry specifies which mention has to be given back from the sub-system to continue the sequence. This receive entry is optional.

Each step in the sequence has a step alternative, which describes what is to be done when a specific sub-system reports an error.

An automatic assignment can contain a sub-system address with an assignment queue. In this case the sub-assignments are inserted in the corresponding assignment queue table. To each assignment queue is tied a specific process which is the receiver of assignments in the assignment queue. Each assignment queue can hold a specified maximum number of assignments, which can be specified.

### 5.3.2 The Core

As mentioned the Core is base level needed for all possible configurations of a SattMate system. The main components of the Core are: **VDU routines** and the **Dispatcher**.
VDU routines are functions which can be configured by the user for maintenance and supervision of the common objects (of a specific object type) in the SattMate system.

The Dispatcher takes care of the distribution of the sub-assignments to the different sub-systems and receives statuses back from these sub-systems (therefore the full-name of the dispatcher is the assignment and status dispatcher).

The main function of the Dispatcher is to distribute the sub-assignments to the different sub-systems. The Dispatcher internally keeps track of the next sub-system in an assignment's route through the system: the sequence. The Dispatcher saves data about all active assignments in the system. This data is used to recover from system breakdown.

Next to this functionality the Dispatcher controls the assignment queues and saves them on disc whenever they are updated. The Dispatcher controls the assignment restart of each sub-system. The Dispatcher distributes status changes in stores, racks and locations between the different sub-systems and also saves the status on disc. This is data used to recover from system breakdown. Finally the Dispatcher distributes changes in working ranges (the range of racks within a vehicle is working) between the different sub-systems. The Dispatcher is thus the generic main element in the SattMate system and supervises the assignment sequences and sub-system handling. Figure 5-5 shows the SattMate Core schematically.

5.3.3 Sub-systems
The sub-system's main function is to control and administrate a handling unit or equipment type. Each sub-system is divided into two different function levels:

The Sub-system Supervisor supervises the sub-system e.g. sub-system monitoring and status modification. This data is used to recover from system breakdown.

The Sub-system Administrator takes care of the assignment administration. These are functions needed to control the handling units e.g. handling unit optimisation. This level does not perform any VDU- or disc-operations.

A sub-system has got a sub-system status that defines if the sub-system is currently accepting assignments or not. Each sub-system is part of standard object type and offers a set of entries, which can be used in a sequence to address and handle the different sub-systems. The available object types are:

- Selector: Location selection sub-systems selects transfer addresses and final store location addresses
- AD. Administrative sub-systems, which requests assignments to be created
- Application. User written sub-systems with user written objects
- Transfer car. Car Transfer sub-systems
- Mobile. Mobile sub-systems
- Graph. Graphic display sub-systems
- Transport. Load transportation sub-systems (Crane, Conveyor, AGV, etc).
5.3.4 Transport sub-system

Important for the UCIA project are the Crane and Conveyor transport sub-systems. The transport sub-systems always consist in general of the following main parts: The administrator, the supervisor with database connection and a communication interface adapter that makes the connection with the industrial controllers that control the cranes/conveyors. The industrial controllers and matching cranes/conveyors are third party dependent and not part of a SattMate sub-system.

5.3.4.1 Transport Administrator

The main function of the transport Administrator is to control the third party handling units (for example to transport pallets from one location to another given by the information in the received assignments).

The main functions of the transport Administrator are:

- Administrate the assignment queues. Assignments are received from the Dispatcher. They are acknowledged to the Dispatcher when they are completed by the handling unit.
- Select assignments from the queues to be sent to the handling units.
- Control the transmission of jobs to the handling units.
- Perform retry actions for the handling unit when needed.
- Perform necessary time supervision for the handling unit.
- Receive and send information on handling unit status from and to the Supervisor at status change and system startup.
- Send information about handling unit assignments and statistics on request from Supervisor VDU routines.
- Receive parameter values from Supervisor at system startup.

5.3.4.2 Transport Supervisor

The transport supervisor contains functions needed to supervise the sub-system e.g. sub-system monitoring and status modification. The supervisor saves dynamic data such as handling unit status on disc. This data is used to recover from system breakdown. The main functions of the Supervisor are:

- Configuration of handling units using VDU routines.
- Maintain status on handling units using VDU routines.
- Display statuses, received from the Administrator.
- Load the Administrator with configuration data, from the project database, during startup.

5.3.4.3 Communication interface adapter

To let the Administrator communicate with the different types of handling units, different communication interface adapters are used. In the communication interface adapter conversion and mapping takes place to get a proper exchange of commands between the Transport Administrator and third party controllers connected to the handling units (i.e. cranes and conveyors).

5.3.4.4 Crane transport sub-system

The crane and conveyor Administrator control the transmission of commands to the handling units (cranes and conveyors). The commands sent between the crane Administrator and a crane are called standard crane jobs. The interface between the crane Administrator and the crane communication interface adapters is indicated as the standard crane interface.

The standard crane jobs are positioning, pickup & positioning, and deposit. Statuses from the cranes are assignment completed and acknowledgement. The crane jobs and statuses are described in paragraph 5.2.1.
The task of the crane communication interface adapter in the crane transport sub-system is now to build up the crane cell jobs on the basis of the standard crane jobs and send them to the crane cell controller according to the crane cell controller interface. The communication interface adapter must thus interact with the crane administrator as being a crane handling unit (with the mentioned standard crane jobs and status exchange and an identification of the crane), react to the cell controller as being a host system (with the crane cell controller dependent crane cell job and status exchange), and map the interfaces/protocols used. Notice that for each type cell controller (with different crane cell jobs and possible a different crane cell controller interface) a new communication interface adapter is needed.

5.3.4.5 Conveyor transport sub-system

The commands sent between the conveyor admin and a conveyor are called standard conveyor jobs. The interface between the conveyor Administrator and the conveyor communication interface adapters is indicated as the standard conveyor interface.

A standard conveyor jobs is the positioning command and statuses from the conveyors are exchanged using the status command (optional), like described in paragraph 5.2.2.

The conveyor communication interface adapter receives jobs from the conveyor admin and sends the jobs to the PLC according to the PLC interface/protocol and receives statuses from the PLC, which are sent back to the Administrator as standard status commands. The communication interface adapter is thus a sort of service hatch. Notice that for each type conveyor PLC (with different conveyor PLC interface) a new communication interface adapter is needed.

The crane communication interface adapter is thus more complex as the conveyor communication interface adapter. Figure 5-6 shows a SattMate system with a crane and conveyor transport sub-system. The blue terms are the interfaces, the red terms the sort of data being exchanged over the interfaces.

![Diagram of SattMate crane and conveyor transport sub-system](image)

Figure 5-6: SattMate crane and conveyor transport sub-system
6 Present situation

This chapter will look at the present and past situation regarding communication between the SattStore WMS application and crane cell controllers and conveyor PLCs. The chapter deals with the following:

- Layout and properties of the standard crane/conveyor interfaces and protocols used within SattStore to make the connection between the crane/conveyor communication interface adapter processes and the Administrator processes.
- Layout, properties and generics of the (vendor-dependent, non-standard) crane cell controller and conveyor PLC interfacing protocols to make the connection to the third party controllers.
- Layout, properties and generics of the crane/conveyor communication interface adapters that map the standard crane/conveyor interfacing protocols onto crane cell controller/conveyor PLC interfacing protocols.

6.1 Crane communication

The communication with a crane cell controller in the past and present situation can be visualized according to the figure below.

![Diagram showing crane communication](image)

**Figure 6-1: Present situation crane communication**

Between the crane administrator process and the crane communication interface adapter the standard crane interface can be found that takes care of the exchange of the standard crane jobs and statuses. Between the crane communication interface adapter and the crane cell controller (the crane cell controller interface) third party dependent cell controller protocols are used to exchange the crane cell jobs and statuses with the crane cell controller in a synchronous, deferred synchronous and asynchronous way.

In former days the crane sub-system had the task to setup communication with a simple crane cell controller named BCC01 coupled to mobile device controllers MC01 on the cranes. The communication between SattStore and the BCC01 uses the BCC protocol. Therefore the crane sub-system and the standard crane interface were adapted to the communication needs of that type of controllers. For this reason the standard crane interface contains BCC records (BCC protocol) exchanged with the help of the interprocess mechanism process-IO. The BCC records are universal enough to exchange data with all sorts of crane cell controllers, because the cranes always receive and send the same commands and statuses (see chapter 5). Nowadays the task of the crane communication interface adapter is to map the standard crane interface (BCC protocol in record form) onto the crane cell controller interface (Third party dependent protocol with PDUs), where in the former days the communication interface adapter only mapped the BCC records onto BCC protocol PDUs being exchanged with the BCC01 (this was a relative simple mapping process).
6.1.1 Standard crane interface

6.1.1.1 Process-IO

Process-IO carries messages from one Windows NT process to another. Such a message contains data and is declared in a record. Every message contains a unique identity number (message ID) that is unique for a specified type of process.

Every process has one queue for messages to receive (according to the FIFO principle). A queue (see also the assignment queue in paragraph 5.2) is a mailbox in a shared memory area. All messages that are sent to a process are put in that queue (mailbox). A queue is created when the receive process is executed and the PROCESS_IO package is elaborated or when a process calls a SEND to a process that has not been executed. This means that a message will never be lost even if the receiver process has not been executed while the sender calls SEND. Figure 6.2 visualizes the queue function.

The operations RECEIVE, SEND and SIGNAL are used to communicate between these processes. The SEND procedure puts a message in the queue of the receiver. The RECEIVE procedure gets a message out of this queue and the SIGNAL procedure is used when there is no data to be sent but one process wants to notify another. It is not possible to send records directly from one process to another. First a message record sent by SEND has to be translated to a byte string message. This is done using the function PACK. And on the other side there is a function UNPACK to translate the byte string message received with RECEIVE to a message record. The function IDENTITY returns the message identity from a message and the function SENDER returns the process name that has sent the message.

Figure 6.3 shows the mentioned relations in the case that A sends data and B receives that data.

A process that defines a number of messages (in message records) using Process-IO to send and receive them, must supply a package for handling the Process-IO messages. This package must supply the message data types and PACK, UNPACK and SEND/RECEIVE operations on each message. Each message defined is identified with a unique identity number (message ID) matching the possible numbers for that particular process. For example the mentioned BCC protocol records do all have a specific ID.
6.1.1.2 BCC records

Data is transferred between Crane_Admin and the crane communication interface adapter in Process-IO records containing the BCC-protocol. The crane sends a message to Crane_Admin because of an event, e.g. the crane wants a new assignment, and Admin responses by sending a message to the crane, e.g. acknowledgement or a new assignment. It’s not recommended that Admin polls the cranes (the crane cell controllers), since all the not useful data being sent heavily load the communication system.

BCC protocol records:

- **Connect to Mobile and Disconnect Mobile record.** In case of a Connect-to-Mobile message the Crane_Admin process notifies the crane communication interface adapter that all messages from a particular BCC should be sent to him. The crane communication interface adapter reacts with a positive acknowledgement (ACK) or a negative acknowledgement (NAK). In case of a Disconnect-Mobile message the Crane_Admin process notifies the crane communication interface adapter that he will disconnect from the BCC. The crane communication interface adapter reacts with an ACK or a NAK.

- **STX record.** These messages contain a data part (standard crane jobs/statuses) consisting of a fixed length array of bytes (byte-string data type, size about tens of bytes). Conversion functions should be used to pack and unpack data into/from this byte array.

- **ACK record.** Every STX Message from Crane_Admin to the crane communication interface adapter is followed by an ACK when the STX message is received successfully, while STX messages from the crane communication interface adapter to the Crane_Admin process are never ACK’ed.

- **NAK record.** Every STX Message from Crane_Admin to the crane communication interface adapter is followed by a NAK when the STX message is not received successfully, while STX messages from the crane communication interface adapter to the Crane_Admin process are never NAK’ed.

**Important PCI (Protocol Control Information) fields (all integer fields):**

- **Mark.** Identifies to which message an ACK/NAK corresponds. The crane communication interface adapter will move the Mark value from the message record received to the ACK/NAK reply record.

- **BCC.** The BCC01 cell controller with which has to be communicated.

- **Mobile.** The MC01 device controller with which has to be communicated.

6.1.2 Crane cell controller interfaces

6.1.2.1 BCC01 communication

In the introduction to crane communication is mentioned that the communication with the crane micro controllers BCC01 application is based on the BCC application protocol and fits in a protocol-based environment. The services of a character-oriented asynchronous full duplex data link protocol are used to exchange the BCC PDUs to and from the crane cell controllers and the other way around. A RS232 physical interface is used. The following BCC01 communication OSI reference model can be drawn:

![Figure 6-4: BCC OSI reference model](image)
**BCC protocol PDUs:**
See the BCC protocol records. Only the *Connect to Mobile* and *Disconnect Mobile* messages are SattStore internals and only exchanged between Crane_Admin and the crane communication interface adapter.

**Important PCI (Protocol Control Information) fields:**
See the BCC protocol records. Only the BCC field is not present, because the crane communication interface adapter sends the message to the right BCC. A Mobile identification suffices.

### 6.1.2.2 ACC and MC01 communication
The communication with the crane micro controllers ACC (series 1) and MC01 (series 2) is based on the series 1A and 2A application protocol and fits in a protocol-based environment. The services of a character-oriented asynchronous full duplex data link protocol are used to exchange the series 1A and 2A PDUs to and from the crane cell controllers and the other way around. A RS232 physical interface is used. The same figure as figure 6.4 can be drawn with the series 1A or 2A protocols instead of the BCC protocol.

#### Series 1A protocol PDUs
- **Series 1A message to/from crane.** Always same PDU is used to send data (crane cell jobs/statuses, no bundling of standard crane jobs/statuses), byte string data type, between SattStore and the Crane and the other way around.

#### Series 2A protocol PDUs
- **Series 2A message to crane.** PDU sent between SattStore and Crane (crane cell jobs, no bundling). The PDU contains a byte string field for the data.
- **Series 2A message from crane.** PDU sent between Crane and SattStore (crane cell statuses, no bundling, byte string). The PDU contains a byte string field for the data.

**Important PCI (Protocol Control Information) fields:**
- **STAMP.** This must be incremented every time a new message is sent (integer).
- **Message Type.** This indicates the type of message being transmitted or received (integer).
- **Crane number.** Number of the crane for which the functional message has intended (integer).
- **Status fields.** Indicates errors, etc (Byte string)

### 6.1.2.3 Daifuku cell controller communication
The communication with the Daifuku cell controllers is based on the Daifuku application protocol. In the former days again the services of a character-oriented asynchronous full duplex data link protocol were used to exchange the Daifuku application PDUs to and from the crane cell controllers and the other way around. Nowadays Socket communication (TCP/IP-Ethernet) is used and therefore the socket service primitives are used to invoke the transport layer. The following Daifuku communication OSI reference model can be drawn (Using Socket communication):

![Daifuku OSI reference model](image)

**Figure 6-5: Daifuku OSI reference model**
Daifuku protocol PDUs
- *Daifuku message frame.* Always the same PDU is used to send data, byte string data type, between SattStore and the Daifuku Crane and the other way around (crane cell jobs/statuses which are a bundling of standard crane jobs/statuses).

**Important PCI (Protocol Control Information) fields:**
- *Sequence number.* This must be incremented every time a new message is sent. Otherwise it’s regarded as a resending (integer).
- *ID.* This indicates the Identity of the message being transmitted or received (integer).
- *Transmission time fields.* To set the transmission time of the host (SattStore) or Crane (byte string).

### 6.1.2.4 Protocol generics

In general crane cell jobs are about tens/hundreds of bytes (and sometimes a bundling of device jobs). The required exchange time is about hundreds/thousands of milliseconds dependent on the used type crane cell controller. As seen the application protocols used consist of data-frames (PDUs) that contain different fields for the crane cell jobs (data-fields) and different fields for protocol control information. In general the following generic PCI fields are present in the examined protocols:

- *Security field(s).* Sequence number or Send/Ack/Retry number combination as security mechanism to ensure that the different cell jobs and statuses are exchanged with reliability as high as possible: At-most-once reliability (always integer fields used).
- *Job identification field(s).* An identification of the type of job(s) that is (are) being exchanged (always integer fields used).

Next to these two generic elements each application protocol exchanged between the communication process and the industrial controller contains other PCI elements. Often a number/address/location of the industrial controller(s) the job(s) is (are) intended for is added, as well as priority information.

### 6.1.3 Crane communication sequence

The crane communication interface adapter makes the mapping between both the BCC and the third party dependent application protocols. This implies the mapping of the Protocol Control Information (PCI) fields of the incoming BCC protocol records onto the right PCI fields of the third party PDUs and the other way around as well as the right reaction to the current state of the protocols in both cases. Sometimes the crane communication interface adapter must be capable to bundle jobs (e.g. Daifuku cell controller communication).

A sequence can be drawn between the Administrator process, the communication interface adapter and the third party dependent controller involving the mentioned BCC protocol records and the application protocol PDUs to the third party controller. The crane communication interface adapter is the main element in such a sequence and must operate correctly according to the sequence. Figure 6.6 shows an example with a SattStore-ACC crane cell controller communication sequence (the crane communication interface adapter is called Crane_IO in this case)

Notice that next to the application level mapping also lower OSI layer protocols (like the datalink protocols) used by the third party must be supported by the crane communication interface adapter (interface adapter must be a real peer for the third party controller application on all OSI layers) as well as the Process-IO package. The Datalink/Network/Transport layer protocols used are standard protocols and therefore not further explained here.
The exact tasks of the crane communication interface adapter in the present situation dependents thus on the used crane cell controller. Only the SattStore site (BCC protocol Process-IO records) is always the same and there are a couple of third party cell controller application protocol generic PCI fields that always must be generated/supported by the crane communication interface adapter (there are thus some correspondences in the application protocol entities). But of course this is only valid for the here mentioned situations and other protocols might have other fields, and not the ones here indicated as generic.

**6.2 Conveyor communication**

The communication with a conveyor PLC in the past and present situation can be visualized according to the figure below.

Figure 6-6: Communication interface adapter as main part of a sequence

Between the conveyor administrator process and the conveyor communication interface adapter the standard conveyor interface can be found that takes care of the exchange of the standard conveyor jobs and statuses. In the present situation always the Signal-IO system is used. Between the conveyor communication interface adapter and the conveyor PLC, the conveyor PLC interface, third party dependent PLC protocols are used to exchange the conveyor jobs and statuses with the conveyor belts in a synchronous, deferred synchronous and asynchronous way.

Conveyor PLC jobs are always represented as a set of variables. A bit variable is used by the conveyor to trigger when a position is needed from SattStore. SattStore sends this position as an integer variable to the conveyor with optional an assignment identification integer variable if a status request (position reached) is required. When a position is reached another bit variable is used to trigger SattStore together with the assignment identification integer. The complete conveyor communication from the conveyor Admin to the conveyor PLC application and the other way around is based on the exchange of these bits and integer variables.
6.2.1 Standard conveyor interface

6.2.1.1 Signal-IO system

IO signal information is transferred between the Conveyor_Admin process and the conveyor communication interface adapter process using the Signal-IO system (Signal-IO is a SattMate sub-system). In general different SattMate sub-systems needs to read and write IO signal information handled by external equipment. A substation is external equipment that can handle IO signals (for example a PLC). An IO signal is defined as a unit of information read from and/or written to a substation. The following IO signal external data types are possible: Bit/Byte array and Integer_2 array (16 bits or WORD).

Each sub-system in SattMate can use IO events to access the IO signal information or to monitor it. The Signal-IO system is used by sub-system programmers, substation communication process programmers and project engineers that define substation IO signals for a specific customer. The Signal-IO system consists of two main parts: A project database and the ADA Signal-IO package. The Project database contains information about all connected substations and about all IO Signals. The ADA Signal-IO packages contain procedure interfaces both for the sub-systems and the substation communication process. The signal-IO transactions (records) are based on the Process-IO record exchange mechanism.

Initiation

All substations that a sub-system process uses must be defined by the sub-system process. This is done automatically when the signals are initiated. At sub-system initiation the sub-system supervisor or main process (if supervisory and administrative functions are done in the same process) reads signal base data from the project database that will be stored in the Signal-IO system Administrators internal signal database.

Signal Monitoring

Signal monitoring can be done trough declaration of events. An event is a request to the Signal IO system to monitor signal transitions for a specific IO signal (bit/byte array or integer_2 array) specified in the Signal IO system Administrator. The substation communication process (that exchange data with the PLC according to a PLC dependent protocol) monitors the requested IO signals for a specific transition.

There are two types of events: permanent and temporary. A permanent event once declared will notify all future transitions of the type requested. A temporary event ends when the transition occurs and is notified. Possible transition are: zero-one, one-zero, to one (digital signals = bit array with range 1) and change value (other types). In case of a permanent signal monitoring the values returned by the event notifications should be stored in the internal signal database.

Read and Update

(Deferred) synchronous read and write of the IO signals is possible. A specific procedure will read the current value in the internal signal database of the Signal-IO system. A put procedure will send an update message to the substation communication process. It will also update the signal in the internal signal database if the signal is monitored with a permanent event.

Signal-IO records:

- **Substation Definition.** This message contains information about the communication interface adapter and the connected PLC’s. For example ID’s, polling interval, timeout-interval etc.
- **Signal Definition.** This message contains signal base data that will be stored in the Signal-IO system Administrators internal signal database.
- **Event Declaration.** This message is sent when an event is to be declared on IO signals. All signals that are part of an event are requested from the communication process.
- **Notification.** A notification message is sent when a communication process has detected the requested signal value transition.
- **Event Cancellation.** This message is sent when an event has to be cancelled.
- **Signal Update.** This update message is sent to the substation communication process. It will also update the signal in the internal signal database if the signal is monitored with a permanent event.
6.2.2 Conveyor PLC interfaces

6.2.2.1 Comli-based PLC communication

The Comli protocol is a conventional communication data link protocol using serial, asynchronous data transmission in half duplex mode, i.e. one direction at a time, and according to the master/slave polling principle. Master and slave can be linked together in different ways to achieve the desired function, i.e. point-to-point, multipoint (multidrop) or radial configuration. Up to 32 Slave systems can be connected to a master system in a multipoint. The communication with the substations is dependent of the received Event Declaration Messages and Signal Update Messages from Conveyor_Admin (See signal-IO). The following Comli OSI reference model can be drawn:

![Comli OSI reference model](image)

Notice that the SattStore application uses directly the services of the data link layer (Comli protocol) to exchange data with the PLCs. The layers 3 till 7 are empty in this case.

Comli protocol PDUs

- **Request for data.** SattStore requests the PLC to send specific data.
- **Data transfer message.** Transfer of data (bits and integers) between PLC and SattStore or the other way around.
- **ACK message.** Acknowledgement from PLC as reaction on a specific data transfer.

Important PCI (Protocol Control Information) fields:

- **Identity.** Indicates which PLC the message is intended for.
- **STAMP.** This transmission mark indicates if the message is being sent for the first time or whether it is a retransmission. The stamp value changes for each new message. A retry is not treated as a new message.
- **Message type.** This indicates the type of message being transmitted or received. The characters can be considered as a function code indicating how the master or slave interprets the data received. The message type is included in all message categories, i.e. request, data transfer, acknowledge.
- **Address.** Start address indicates the first address from which the requested data is to be read or write.
- **Quantity.** Quantity indicates the number of items requested, e.g. the number of I/Os, number of registers, number of limiter modules etc.

6.2.2.2 Siemens-3964R PLC communication

The Siemens 3964R protocol has got the same communication properties as the Comli protocol and thus also the same OSI reference model filling up (see figure 6.7). The datalink PDUs sent to and received from the Siemens PLC are a little different, but the used PCI fields in the PDUs are again the same.
Present situation

Siemens 3964R protocol PDUs
- Write request. Write n words from SattStore to the PLC.
- Write request response. Response from PLC after a write request.
- Read request. Read n words from SattStore to the PLC.
- Read request response. Response from PLC after a read request.

6.2.2.3 Protocol generics
In general there is often an exchange of bit-fields and integers represented as 2-byte(word)-fields. The required exchange time of the variables is about hundreds of milliseconds. Next to size and frequency requirements, reliability plays also a considerable role, like mentioned above by crane communication (also at-most-once reliability). The data link protocols used consist of data-frames (PDUs) that contain different fields for the integers and bits and different fields for protocol control information. In general the in 6.2.2.2 mentioned PCI fields are present.

6.2.3 Conveyor communication sequence
The conveyor communication interface adapter gets commands from the Signal-IO system to monitor or update specific data (signals) in the PLC. The conveyor interface adapter executes these tasks according to the third party dependent PLC protocols (in the present situation Comli and Siemens 3964R) and reports specific event transitions on signals back to the Signal-IO system. Figure 6.9 shows such a monitor sequence.

![Figure 6-9: Monitor signal sequence (with Comli protocol)](image)

The conveyor administrator wants signal X (bit) to be monitored for an event transitions (0-1) and sends an event declaration record with the signal X in it to the Signal-IO system. Notice that the signal X must be announced by the Signal-IO system (with a signal definition record) for it can be used. The Signal-IO system knows the memory area (start address – stop address) from a particular device belonging to that signal (in this example a memory address Mem A in PLC 1). The Signal-IO system gives the conveyor communication interface adapter (in this case Comli_IO) the assignment to poll the area Mem A in the PLC 1 for event transitions (0-1). If the communication interface adapter detects the transition, this will be reported to the Signal-IO system, which, on his turn, sends an event notification message to the conveyor administrator with the right changed value. If the event is permanent this sequence will be repeated.

The lower OSI layer protocols (Comli or Siemens 3964R and the physical line) used by the third party must thus be supported by the conveyor communication interface adapter (interface adapter must be a real peer for the third party PLC application on all OSI layers) as well as the Signal-IO/Process-IO package.
The conveyor communication interface adapter in the present situation must setup communication with a third party PLC according to the PLC dependent interface, to monitor/exchange variables defined by Signal-IO. The SattStore interface (Signal-IO sub-system) is always the same and there are PLC datalink protocol generic PCI fields that always must be generated/supported by the conveyor communication interface adapter (there are thus correspondences in the datalink protocol entities). Off course, just as the crane communication, these claims are based on the two mentioned protocols, and need not always to be true.

6.3 Conclusions
In the present (and past) situation the communication between SattStore and the Cranes is different from the communication between SattStore and the conveyor belts. The difference is a result of the different placement in the CIM pyramid.

6.3.1 Crane communication
In the case of crane cell controllers always an application protocol is used on top of a serial or socket connection to let SattStore and the cell controllers understand each other (the upper layers are thus cell controller specific, while the lower layers are standard). The data field of the protocol consists of a fixed length byte string containing the cell jobs/statuses, often accompanied with a job or status identification field. The cell controller defines the layout of the protocol and the matching PCI fields. For this reason each protocol has in general other PCI fields, in spite of the two mentioned generic fields (for the examined protocols).

The main task for the crane communication interface adapter is to map the BCC protocol records (used on the standard crane interface) onto the crane cell controller dependent protocol and the other way around. This is often very complex (e.g. job bundling) and the functionality of the crane communication interface adapter is thus strongly dependent on the used crane cell controller.

The conclusions after studying the present (and past) situation are:

- Between the Crane Admin and the crane communication interface adapters always the same BCC protocol records are exchanged by using Process-IO.
- Protocols used between the crane communication interface adapter and the third party crane cell controllers are always different, and contain less equal/generic components.
- The layout and functionalities of the crane communication interface adapters differ per situation. There are very little generic components to be distinguished in the adapters.

6.3.2 Conveyor communication
In the case of conveyor PLCs communication always an exchange of bits and integers take place. From SattStore’s point of view always the Signal-IO system is used. This system let the conveyor communication interface adapter send/receive or monitor the bit and integer variables according to the third party dependent PLC protocol (only the physical layer is the same). The conveyor communication interface adapter is thus less complex. But off course third party PLC protocols could be different each time.

The conclusions after studying the present (and past) situation are:

- Between the Conveyor Admin and the conveyor communication interface adapters always the same Signal-IO system (with Process-IO records) is used.
- Protocols used between the conveyor communication interface adapter and the third party conveyor PLC are always different, but do contain generic components.
- The layout and functionalities of the conveyor communication interface adapters are clear and do not completely differ per situation. Only the used protocols to the third party PLC or PLCs contain variable components.
7 Industrial protocols and standards

Next to vendor-dependent, non-standard communication protocols used in the present situation to make the connection to the third party controllers, there are also standard vendor independent network protocols and standards available on the market nowadays. This chapter examines which standards are interesting for the communication with cell controllers and device controllers from the point of view of possible (dis)advantages by using and implementing the solutions, market support of the different solutions (now and in the future) and possible generic components within the solutions. First of all the possible solutions are gathered and explained. Next to this the solutions are compared and considered according to the mentioned viewpoints and clear preferences, that best fit the demands, are given.

7.1 Industrial networks

7.1.1 Overview of industrial network types

As mentioned in chapter 5.1 of the graduation report different entities on adjacent CIM layers exchange information. To exchange the information between the different entities, standard communication networks are extensively used in the manufacturing environment nowadays. In general the CIM environment can be divided into three network interface areas, each addressed by a specific type of network. Figure 7-1 shows the interface areas.

Interface area 1: Cell controller / Area controller interface

Because of the low frequency and large data size of the data being exchange between the area controller applications and the cell controllers (complete cell jobs, status replies and possible other information), the performance of a communication system making the connection between the area level and the cell level must guarantee the reliable transfer of large amounts of information in times that are not critical. The same sorts of requirements are present between the area level applications and the enterprise/factory level applications (ERP data, office data, etc).

Standard networks used at this level are called plant networks, which are either LANs, or intranets because their application protocols, such as FTP, HTTP, based on TCP/IP are particularly suitable to implement the communication requirements of this level. Moreover, intranets use the well-known World-Wide-Web functionalities. Also XML and other middleware techniques are important at this area.
Industrial protocols and standards

**Interface area 2: Device controller interface**

Device controller must provide a connection with a host system (often a cell controller) for getting control commands, sending statuses, monitoring data and sometimes a connection with another device controller for exchanging coordination information. This implies the transfer of considerable quantities of data (called *messaging*) for controlling, configuration, calibration, trending etc. (typical transfers of some hundreds of bytes and sometimes even kilobytes). The communication between a device controller and a host system or other device controller is not real-time. The times allowed for the data transfer can be on the order of hundred of milliseconds or more.

Standard networks used at this level are called *control networks* which can be addressed by two types of protocols: fieldbus network protocols and industrial Ethernet (LAN) protocols. Standards like MMS (Manufacturing message specification) and OPC (OLE for process control) play also an important role on this interface area.

**Interface area 3: Sensor/actuator interface**

The typical operation of the device controllers consist in the *cyclic data exchange* with the sensor/actuators performed at very high speed, which can be interrupted by alarms/events from the field, *asynchronous traffic*. Usually the amount of data exchanged by the device controllers with each sensor/actuator is small (some bytes), but the cycle times required are of milliseconds. The communication systems used between device controllers and sensors/actuators must be *real-time* (respond in a timely predictable way to unpredictable external stimuli arrival).

Standard networks used at this level are called field networks categorized into *sensor networks* (that encompass protocols initially designed to support discrete I/O) and *device networks* (encompass protocols initially designed to support continuous analog I/O). Both field networks can be addressed by fieldbus protocols. There are thus types fieldbus network protocols for discrete, analog and control environments.

Notice that the mentioned categories are a simplification because the separation between the categories is often blurred. In many cases, a network protocol that best fits in one category can perform some or many of the functions from another. E.g some fieldbus protocols address the discrete, analog and control environment.

### 7.1.2 SattStore and industrial network types

SattStore is an area level application that needs communication with cell controllers and device controllers. The communication between SattStore and the cell controllers fall into interface area 1 and can be addressed by plant networks using standard protocols or middleware techniques like XML (Because plant networks are in principal office networks, there are no specific industrial protocols that address interface area 1. Standard protocols like FTP, HTTP, or middleware techniques can be used, but are not part of this chapter). The communication between SattStore and the device controllers fall into interface area 2 and can be addressed by some types of fieldbus protocols and industrial Ethernet protocols or by more generic solutions like MMS and OPC (Sometimes called data-server solutions). All of these solutions have (more or less) the in paragraph 7.1.1 mentioned messaging capabilities. From SattStore’s point of view there is never a coupling on field networks (interface area 3).

### 7.2 Fieldbus protocols

#### 7.2.1 Introduction

*Fieldbus* is a generic-term, which describes a digital communications network, which is used in industry to replace the existing 4 - 20mA analogue signal. The network is a digital, bi-directional, multidrop, serial bus, communications network used to link isolated field devices, such as controllers, actuators and sensors. The use of fieldbuses at different levels (interface areas 2 and 3) of the manufacturing environment is only possible if they are capable of satisfying the specific functions required by these levels: Cyclic data exchange and asynchronous traffic (interface area 3) and messaging (interface area 2).
7.2.2 Communication profile

Cyclic data exchange and asynchronous traffic functions are realized by real-time protocols that use the services of the data link layer directly (sometimes a transport and network layer is used). The messaging function, however, is mostly realized by suitable application (layer 7) protocols, sometimes combined with an object model. Figure 7-2 shows the typical fieldbus communication profile. In practice it is not usual to find a station connected to a fieldbus implementing the three functions simultaneously. This is obvious because, for example, the messaging system function, often a bulk data transfer, could dramatically slow the cyclic data exchange being performed on the same network.

From SattStore’s point of view the messaging function is important for the communication with device controllers according to interface area 2. The cyclic data exchange and asynchronous traffic will not further examined in this chapter, because there is, as told, never a coupling with sensors/actuators.

7.2.3 Messaging services

The application protocol has to handle messaging services. Messaging services are for example read/write of simple and complex variables, domain upload/download (e.g. the memory in a PLC in which a program is stored), events, remote start/stop of programs, etc. In most of the cases the application protocol implements only the simple and complex variable access and the event services, because they are at most used in general. Often the implementations of services are based on a subset of MMS (See paragraph 7.4). The messaging system often uses the peer-to-peer communication technique. To keep nodes from dominating the network, most peer-to-peer networks use some kind of token rotation algorithm. But response times vary considerably for any given message, depending on load and on how "far away" one is from the token holder when there is a need to speak.

Sometimes the application protocol introduces an object model by means of which each station is considered as a container of communication objects. These communication objects are the only entities that can be invoked on the fieldbus and they represent variables, domains, programs, events, etc dependent on the possibilities of the fieldbus (often thus only variable and event objects). A client/server model is used to exchange the data in this case and a connectionless or connection-oriented type of transmission can be used.

Groups of objects are often bundled (in object libraries) and form a so-called device profile for a specific device or application. Device profiles are used in automation technology to define specific properties and behavior for devices, device families or entire systems in such a way that this determines their largely unique characterization. Only devices and systems with the same vendor-independent profile provide interoperability on a fieldbus, thus fully exploiting the advantages of fieldbuses for the end-user.
7.2.4 Reliability
There is in general the necessity of protecting the operation of a fieldbus protocol against several sources of failures: Physical medium failures, Device failures and transmission errors. Physical medium failures can only be addressed by creating a redundant network, what is supported by some fieldbuses. Device failures must be recognized and most important is that the defective device has to maintain the plant under its control in a safe state. Each fieldbus can handle these device failures. Finally, every fieldbus is able to detect, and sometimes correct, any possible loss or corruption of information during transmission. This is often achieved using suitable control bits of check fields, which are added to the useful data to transmit. Two types of checks may be executed: on the character (e.g. the block checksum by character oriented protocols) and on the frame (frame check sequence FCS or cyclic redundancy check CRC on the basis of a polynomial function applied to the frame). Important technique to correct errors is the Hamming Distance. In general an at-most-once reliability is used: Guarantee the request to execute only once. It may happen that they are not executed, but then the requester is notified about the failure and have sometimes the possibility to retransmit the data.

7.2.5 Fieldbus solutions
As mentioned fieldbuses are a combination of the layer 1, 2 and 7 (sometimes 3 and 4) of the OSI-reference model. A fieldbus solution specifies the physical layer, the data link layer protocols and services and the application layer protocols and services. There are a couple of fieldbus solutions included in European and International standards and some proprietary standards not officially recognized by standards organizations.

The International Fieldbus standards are issued by the IEC61158 and the IEC61784 standards which partly overlaps the European CENELEC EN50170, EN50254, EN50325 and EN 50295 standards. European standards can thus be mapped on the IEC standards. The IEC61158 is a specification of Fieldbus layers and consists of eight parts. The most important parts are the IEC61158-2, which includes 8 physical layer specifications, the IEC61158-3/4 which includes 8 datalink layer service/protocol types and the IEC61158-5/6 which includes 10 application layer services/protocol types. A specific Fieldbus protocol is issued by a so-called IEC61784 profile (CPF), which is a combination of an IEC61158 Physical, data link and application layer service/protocol. The document Industrial protocols and standards [ABB 2003-3] contains an overview of the European and International fieldbus standards as well as proprietary solutions.

Important solutions that are capable of messaging are ControlNet (IEC61784 CPF-2/1), Foundation Fieldbus H1 (IEC61784 CPF-1/1), WorldFIP SubMMS (IEC61784 CPF-5/2) and the proprietary solution MODBUS. Each solution has thus a physical layer specification (sometimes the same) as well as different datalink and application services and protocols to fulfill the messaging needs. MODBUS is an exceptional case and implements messaging capabilities directly via the datalink layer services without using an application layer and object model. A detailed work out of ControlNet and Foundation Fieldbus H1 can be found in the document Industrial protocols and standards [ABB 2003-3]. This document provides knowledge about the filling up of the mentioned OSI layers in these situations. Next to this it also offers a short description of MODBUS.

7.3 Industrial Ethernet protocols

7.3.1 Introduction
The impressive growth of Ethernet is modifying the scenario of manufacturing communication systems. At interface area 2, the replacement of fieldbus with Ethernet-TCP/IP is usually straightforward, because performance of the latter equals, or often betters, that of the most popular fieldbuses. At interface area 3, the use of Ethernet has to be considered very carefully because this network is non-deterministic (in other words the networks can not exactly predict when information is available) and, consequently, it may not be able to perform the real-time functions required at the interface area 3. However, in recent years the performance of Ethernet has been significantly enhanced by the introduction of new standards like Fast Ethernet (IEEE802.3u 100Mbit/s), Gigabit Ethernet (IEEE802.3z 1000Mbit/s) and switch-technology.
7.3.2 Communication profile

The typical industrial Ethernet communication profile looks like the fieldbus communication profile but now with the layers 2, 3 and 4 of the OSI model standardized (Ethernet-TCP/UDP/IP). As told an application layer implementation (and sometimes layers 5 and 6) must take care of the messaging, next to well-known OSI application protocols like HTTP, SMTP and FTP. Most suitable messaging solutions are represented by the use of MMS or subsets of MSS or other vendor-dependent protocols and object models. An unsolved problem is the absence of a well-defined/established, worldwide standards-based, real-time protocol, to implement the functions needed in interface areas 3, in spite of the presence of a fast network. This explains the absence of the cyclic data exchange and acyclic traffic functions in the typical communication profile. Transmission reliability is acquired by using TCP/IP on top of LAN (CSMA/CD) and by using application protocol (or by using Middleware techniques).

7.3.3 Industrial Ethernet solutions

Some of the industrial Ethernet protocols (interface area 2 protocols) are already issued by the IEC61158 and IEC61784 standards (Not in European standards). Of course the datalink layer (and network/transport layer) is always Ethernet-TCP/IP. The application layer is in this case of a specific IEC61158-5/6 application service/protocol type. Issued protocols are Foundation Fieldbus HSE (FF HSE) (IEC61784 CPF-1/2, based on a new application layer specification), Ethernet/IP (IEC61784 CPF-1/2, same application protocol as ControlNet) and Profinet (IEC61784 CPF-3/3, based on a new application layer specification).

Next to the IEC61158 standardized industrial Ethernet protocols there are also a couple of other standards like MODBUS TCP/IP (MODBUS protocol on top of TCP/IP), IAONA (Industrial Automation Open Networking Alliance) and IDA (Interface for Distributed Automation). Also the rising OPC standard (Ethernet-TCP/IP-based, see paragraph 7.5) becomes more and more part of different industrial Ethernet standards. Together with the use of TCP/IP-Ethernet also Middleware techniques become part of the communication protocols on interface area 2 (See also chapter 10 Middleware). A clear transition from protocol-based to object/component-based communication can thus be perceived. In the document Industrial protocols and standards [ABB 2003-3] a work out of the different mentioned industrial Ethernet protocols can be found.

As told many fieldbus protocols as well as many industrial Ethernet protocols use MMS or subsets of MMS for implementing the messaging services on the application layer. Now look at MMS and services in detail.

![Figure 7-3: Typical communication profile of Industrial Ethernet protocols](image-url)
7.4 Manufacturing Message Specification (MMS)

7.4.1 Purpose and introduction

MMS (Manufacturing Message Specification) is an internationally standardized OSI application protocol for exchanging data and supervisory control information (messaging data) between networked devices (like PLCs) and/or computer applications. MMS is an international standard (ISO 9506) that is developed and maintained by Technical Committee Number 184 (TC184), Industrial Automation, of the International Organization for Standardization (ISO).

The primary goal of MMS was to specify a standard communications mechanism for devices and computer applications that would achieve a high level of interoperability, independence and supports data access. Interoperability is the ability of two or more networked applications to exchange useful supervisory control and process data information between them without the user of the applications having to create the communications environment. Independence allows interoperability to be achieved independent of the developer of the application, the network connection and the function performed. Data access is the ability of networked applications to obtain the information required by an application to provide a useful function.

7.4.2 MMS technical details

In order to achieve the in the introduction mentioned goals, it would be necessary for MMS to define much more than just the format of the messages to be exchanged. MMS precisely defines:

- **Objects.** MMS defines a set of common objects (e.g., domains, variables) and defines the network visible attributes of those objects (e.g., name, value, type).
- **Services.** MMS defines a set of communications services (e.g., read, write) for accessing and managing these objects in a networked environment.
- **Behaviour.** MMS defines the network visible behavior that a device should exhibit when processing these services.
- **Messages.** MMS defines as set of messages, PDUs (Protocol Data Units), that carry the requests and responses associated to these devices (ASN.1 format).

This definition of objects, services, and behaviour comprises a comprehensive definition of how devices and applications communicate in which MMS calls the *Virtual Manufacturing Device (VMD) model*. Before looking at this VMD model, a couple of MMS communications basics are explained first.

7.4.2.1 MMS communication basics

An **MMS user** is an application program that uses the MMS services to communicate with the MMS user of a target system, for example a PLC system, according to the client/server principle. The **MMS provider** is that part of the communication system that provide the users with the MMS services. Essentially the MMS provider is responsible for transferring MMS requests and responses between MMS users. There are two types of services possible:

- **Confirmed services:** Services which require a confirmation (uses the 4 basic communication service primitives as mentioned in chapter 6.3)
- **Unconfirmed services:** Invoked by a server to report a significant event to the client (only uses request and indication, response and confirm are not used in this case)

An **MMS server** in the sequel is an application, which maps physical resources onto MMS object, which may be accessed from the network. This notion of mapping is fundamental in MMS; clients do not access physical resources directly but the MMS objects that represent them. An **MMS client** in the sequel is an application, which accesses the MMS objects of a server via the network and thus makes use of the available physical resources provided by the MMS server.
7.4.2.2 The MMS VMD object model

Every manufacturer of programmable devices or applications that will implement MMS, has to map the real devices and objects (e.g. a PLC and internal variables) into virtual ones defined by the VMD model according to a so-called *executive function* or translation function. Figure 7-5 shows the VMD model and this translation.

> Because MMS clients always interact with the virtual device and objects defined by the VMD model, the client applications are isolated from the specifics of the real devices and objects. A properly designed MMS client application can communicate with many different brands and types of devices in the same manner.

7.4.2.3 MMS object classes & services

As told MMS defines a variety of objects that are found in many typical devices and applications, requiring manufacturing communication. MMS makes use of a template to define the properties of an object class. Each instance of an object class has the same set of attributes, but has its own set of attribute values. There is one key attribute, the name attribute that is used in each object class. The attributes of MMS objects are accessible by the MMS services, defined per object class. The behaviour of objects is defined using finite state machines that indicate the state transitions resulting from the reception of service invocations.

There are object classes for program control, variable access, event management and other purposes. Each object class has matching services. Because of the possible importance of the variable access for the UCIA project, these object classes will be shortly explained here. The document Industrial protocols and standards [ABB 2003-3] contains a more detailed description of all object classes/services and a detailed overview of the variable access system.
Variable access object classes
Real variables are mapped to MMS variables, which may be seen as an abstraction to describe the external view of real variables (simple or complex). In order to access the variables, MMS defines the object classes *Unnamed Variable* (using an address) and *Named Variable* (using an object name) containing a typed data element (e.g. simple like integer, floating point, etc or complex like array, structure etc). Also the object classes *Named Variable List* (list of variables that is named as a list) and *Named Type* (description of the format of a simple or complex variable's data) are possible.

Most important variable access services are *read*, to obtain the value of a variable(s), *write*, to change the value of a variable(s) and *InformationReport*, to report values to a MMS client in an unconfirmed manner (can for example be used as alarm notification mechanism).

7.4.2.4 MMS Application interfaces
The MMS protocol does not specify how application programs interface with the MMS services: ISO standard protocols do not specify how each of the seven layers of the OSI model interfaces with its adjacent layer. MMS specifies the service that is to be provided at the application level but does not specify the user's view of the MMS services. Vendors are free to define the interface between the user application program and the MMS services. The level of support offered by an Application Programming Interface (API) can vary widely between protocol implementations both in functionality and independence from the hardware platform and the operating system.

7.4.2.5 MMS and reliability
Confirmed services offer transmission reliability by reporting (confirmation) if an action (request) has been executed. Unconfirmed services from the server to the client do not have this transmission reliability. (Comparable with TCP and UDP). The way in which the MMS provider ensures correct data transmission, depends on the used methods and lower layers of the OSI model.

7.5 OPC (OLE for process control)

7.5.1 Purpose and introduction
*OPC (OLE for Process Control)* is an industry standard for communication between applications and field devices. The figure below illustrates the so-called 'I/O driver problem'. Clearly each fieldbus or industrial Ethernet protocol requires a unique, fieldbus-specific IO driver. The problem that OPC deals with is that each of these IO drivers also presents a different API to the calling application. As a result, each application also requires a unique code for each driver, device or network it wants to connect to. This obviously makes it difficult to reuse applications with different sets of process interface equipment. OPC solves this problem by creating a 'software bus' as shown in figure 7.6. Applications, the *OPC clients*, only need to know how to get data from OPC data sources, the *OPC servers*. These OPC servers encapsulate the IO driver logic and provide the data in a single format.

![Figure 7-6: The OPC software-bus solution](image)
Industrial protocols and standards

OPC consists of a series of standard specifications. As told the original is the **OPC Data Access (OPC DA)** specification (meanwhile version 2.05 and 3). This specification tells how to move real time data from PLC's, DCS's and other control devices to applications like HMI or for example SattStore. There are also a couple of derived specifications like OPC alarms & events (OPC-AE), OPC Historian Data access (OPC-HDA), OPC security, OPC Data exchange (OPC-DX) and OPC XML Data Access (OPC-XML-DA).

### 7.5.2 OPC technical detail

OPC is a client/server architecture based on Microsoft’s OLE/COM and therefore Ethernet-TCP/IP and Windows-based (See chapter 10 Middleware). OPC (COM) uses a component based implementation paradigm and includes DCOM (Distributed COM) that allows creation of network-based applications with a minimum of development effort. The OPC specification describes the OPC COM objects and their interfaces implemented by OPC servers. An OPC client can connect to OPC servers provided by one or more vendors. Vendor supplied code determines the devices and data to which each server has access, the data names, and the details about how the server physically accesses that data.

#### 7.5.2.1 General OPC architecture and components

An OPC client application communicates to an OPC server through the specified custom and automation interfaces. OPC servers must implement the custom interface, and optionally may implement the automation interface. In some cases the OPC Foundation provides a standard automation interface wrapper. This “wrapperDLL” can be used for any vendor-specific custom-server.

![Figure 7-7: The OPC architecture](image)

There are several unique considerations in implementing an OPC Server. The main issue is the frequency of data transfer over non-sharable communications paths to physical devices or other databases. Thus, expected is that OPC Servers will either be an inprocess, local or remote EXE (See chapter 10 middleware) that includes code that is responsible for efficient data collection from a physical device or a database.

#### 7.5.2.2 OPC specifications

As told there are a couple of specifications, with the OPC-DA specification as most important. The specifications include a set of custom COM interfaces for use by client and server writers (C++) and references to a set of OLE Automation interfaces to support clients developed with higher-level business applications such as Excel, Visual Basic, etc. Because of the possible importance of OPC-DA and OPC-XML-DA for the UClA project these specifications are shortly explained here. Other specifications and more details about OPC-DA and OPC-XML-DA have been included in the document [ABB 2003-3].

**OPC Data Access (OPC-DA) specification**

At a high level, an OPC Data Access Server is comprised of several objects defined in a logical object model: the server, the group, and the item. The OPC server object maintains information about the server and serves as a container for OPC group objects. The OPC group object maintains information about itself and provides the mechanism for containing and logically organizing OPC items.
The OPC Groups provide a way for clients to organize data. For example, the group might represent items in a particular operator display or report. Data can be read and written. Exception based connections can also be created between the client and the items in the group and can be enabled and disabled as needed. Within each Group the client can define one or more OPC Items. The OPC Items represent connections to data sources within the server. An OPC Item, from the custom interface perspective, is not accessible as an object by an OPC Client. Therefore, there is no external interface defined for an OPC Item.

Variables are of the type VARIANT, which means that all types of variables are possible (simple and complex). There are several possibilities to read/write values: by a synchronous get/set (read/write), by an asynchronous get/set (read/write) or by subscription. When a client subscribes to an item it can be specified that the update should be delivered only if the value changed by a percentage (called a deadband).

The OPC Data Access server provides management of the OPC groups/items and also controls, mediates and optimizes access to the physical devices by multiple clients.

**Figure 7-8: The OPC Data Access logical object model**

**OPC XML Data Access (OPC-XML-DA) specification**

Many OPC member companies have participated in the development of the specification and indicated they are developing products based on the OPC-XML standard. The OPC-XML specification provides the foundation for multi-platform interoperability using XML, and is one of the key architectural and strategic specifications that are a stepping-stone for several other OPC initiatives that are under way. Microsoft worked closely with the OPC Foundation in developing the OPC XML specification to take full advantage of the Microsoft .NET platform. OPC XML leverages Microsoft .NET technologies that will help open up information flow throughout the manufacturing enterprise and beyond.

The OPC-XML-DA specification has the same properties as the "normal" COM-based OPC-DA specification. But now based on an XML-SOAP interface instead of a COM interface to provide multi-platform interoperability. More information about XML-SOAP can be found in paragraph 10.6.3.

**7.5.2.3 OPC and reliability**

Transmission reliability is guaranteed by COM/DCOM/XML-SOAP and not further explained here (See chapter 10 about Middleware).

**7.6 Evaluation of industrial protocols and standards**

There are different possibilities to exchange device jobs and statuses between SattStore and device controllers like PLCs. Each solution has its own way of dealing with the problem area. For example the messaging capabilities of fieldbus and industrial Ethernet protocols are often limited to a subset of MMS, which is of course something that must be taken into account when using and implementing a solution.
As discussed in view of the purpose of this chapter important issues by evaluating/comparing the different solutions are the possible (dis)advantages by using and implementing the solutions, the market support of the different solutions (now and in the future) and the possible generic components within the standard solutions.

7.6.1 Fieldbus / Industrial Ethernet protocols

A couple of fieldbus protocols as well as the Industrial Ethernet protocols support messaging services (e.g. the in paragraph 7.2.4 mentioned solutions) and are suitable to interconnect SattStore with third party device controllers. In most of the cases the messaging services offered by the protocols consists of an exchange of data (variables) and the defining of events (and are often a subset of the MMS services).

7.6.1.1 Advantage of using fieldbus protocols

- Protocols are standard communication protocols and offer thus also a standard IO driver and API per protocol type. If SattStore implements the standard IO driver it can communicate with all the different device controllers (that also support the protocol).
- Reduce of wiring and installation. The existing 4-20mA analogue signal standard requires each device to have its own set of wires and its own connection point. Fieldbus eliminates this need so only a single twisted pair wiring scheme is required.

7.6.1.2 Disadvantages of using fieldbus protocols

- Each type fieldbus protocol needs a different IO driver and matching API. If SattStore for example wants to connect to three types of field networks, three different IO drivers are needed.
- The messaging capabilities depend on the used protocol and is often only the exchange of data (variables) and the defining of events (not domains, program invocations, etc)
- Fieldbus protocols often fit in a (rather obsolete) protocol-based implementation paradigm. Each fieldbus vendor implements the necessary interface area 2 and/or 3 functions on the different layers of the OSI model without using Middleware solutions.
- There is very little heterogeneity present in the fieldbus solutions (see previous item). The only way the provide interoperability on a fieldbus, thus fully exploiting the advantages of fieldbuses for the end-user, is by using the same vendor-independent device profile on top of the application protocol.
- Integration of different networks is often only possible if the fieldbus networks are from the same vendor (e.g. DeviceNet, interface area 3, and ControlNet, interface area 2).

7.6.1.3 Advantage of using industrial Ethernet protocols

- See 7.6.1.1
- The widespread use of Ethernet led to a great availability of low-cost components and know-how.
- Intranet networks are increasingly adopted at the interface area 1, if Ethernet is used both at the interface areas 2 (and in the future 3), the different communication networks can be easily integrated, as they may use the same TCP/IP protocols.
- Remote access to all levels of the manufacturing environment can be facilitated by using World Wide Web functionalities.
- The OPC standard (Ethernet-TCP/IP-based) becomes more and more part of different industrial Ethernet standards. Together with the use of TCP/IP-Ethernet also Middleware techniques become part of the communication protocols on interface area 2 (See also chapter 10 Middleware). A clear transition from protocol-based to object/component-based communication can be perceived what increase heterogeneity.

7.6.1.4 Disadvantages of using industrial Ethernet protocols

- Each type industrial Ethernet protocol needs a different IO driver and matching API (See 7.6.1.2).
- The messaging capabilities depend on the used protocol (See 7.6.1.2).
- Some industrial Ethernet protocols fit in a (rather obsolete) protocol-based implementation paradigm. (See 7.6.1.2)
Industrial protocols and standards

7.6.1.5 Fieldbus / Industrial Ethernet protocol market-support and future

Some types fieldbus protocols are widely used in the manufacturing environment for connecting device controllers and sensors/actuators. For example profibus-DP is an important standard on interface area 3. On interface area 2 there is no fieldbus standard that dominates the area. ControlNet is a regular choice, as well as Foundation Fieldbus H1 or MODBUS. Because other solutions offer a lot of advantages with regard to fieldbus protocols, there is no future for fieldbus protocols on interface area 2.

Many industrial Ethernet protocols are still in its infancy and therefore not yet widely used. In spite of this it's the question whether industrial Ethernet solutions like Ethernet/IP, Profinet and Foundation Fieldbus HSE will be a success in the future or not. This because MMS and OPC are dominating the market in interface area 2 on this moment and very probable OPC also in the future.

7.6.2 MMS and OPC

A lot of the mentioned disadvantages are solved by using MMS or OPC. In the case of MMS SattStore and the device controllers are the MMS users. SattStore must be the MMS client application and the device controllers the MMS servers implementing the mentioned VMD object model. In this case variables, events, programs, etc in the device controller can become virtual VMD objects by means of the mentioned execution function and can be exchanged with SattStore according to the MMS services provided by the MMS provider. In the case of OPC SattStore is the OPC client application and the device controllers/fieldbus networks/etc are connected to OPC servers implementing the OPC object model and COM/DCOM (or XML) interfaces.

7.6.2.1 Advantages of using MMS

- If SattStore has MMS client functionality it can exchange useful supervisory control and data information with MMS servers without creating the communications environment. This interoperability is achieved by independence of:
  - The developer of the application or device: Other communications schemes are usually specific to a particular brand (or even model in some cases) of application or device. MMS is defined by independent international standards bodies with participation from many leading industry experts and vendors.
  - Network connectivity: MMS becomes the interface to the network for applications, thereby isolating the application from most of the non-MMS aspects of the network and how the network transfers messages from one node to another. (Any communication link and platform is possible)
  - Function Performed: MMS provides a common communications environment independent of the function performed. An inventory control application accesses production data contained in a control device in the exact same manner as an energy management system would read energy consumption data from the same device.
- While many communications schemes only provide a mechanism for transmitting a sequence of bytes (a message) across a network, MMS does much more. MMS also provides definition, structure, and meaning to the messages that significantly enhances the likelihood of two independently developed applications interoperating.
- An MMS implementation (with a well-defined MMS API) called OpenMMS is available at ABB Etten-Leur. OpenMMS is a subset of MMS and has server functionality for variable access services, journal services, event services and file transfer services. It has client functionality for variable access services and journal services.

7.6.2.2 Disadvantages of using MMS

- MMS is on the one hand flexible (No platform, language, etc demands) but at the same time also complex to implement and use. E.g. MMS does not specify a standard application interface, the defining of such an interface could be difficult in spite of the rules offered by MMS.
Device controllers that want to "speak" MMS must create an execution function to map the real objects onto the virtual objects. Often this implies that a controller card (with the execution function) must be positioned next to the device controller to support MMS.

### 7.6.2.3 Advantages of using OPC

- **OPC** offers a mechanism to provide data from a data source and communicate the data to the SattStore client in a standard way. A vendor can develop a reusable, highly optimized server to communicate to the data source, and maintain the mechanism to access data from the data source/device efficiently. Providing the server with an OPC interface allows SattStore to access their device data. A clear API is specified on client (SattStore) and server side.
- In the case of OPC hardware manufacturers only have to make one set of software components for customers to utilize in their applications, software developers will not have to rewrite drivers because of feature changes or additions in a new hardware release and customers will have more choices with which to develop integrated manufacturing systems.
- Next to this the underlying component-based techniques are also an important advantage of OPC. DCOM offers the necessary interoperability, transparency, etc but has language and platform restrictions. The release of OPC-XML-DA overcome these restrictions and offers a complete platform and language independent OPC solution based on XML-SOAP.

### 7.6.2.4 Disadvantages of using OPC

- A device controller must be connected to an OPC server to work with the OPC approach (no OPC controller cards). Vendor-built OPC servers are (sometimes) expensive and it is not easy to build an (often complex) OPC server. This because the OPC server needs all logic from the device controller(s) connected as well as all COM-based interface information and the mapping of the COM object model to the real world data connections.
- OPC has Windows NT/XP platform requirement (Microsoft technology) with an Ethernet-TCP/IP protocol stack as communication link. But as told the OPC-XML-DA specification solves this restriction.

### 7.6.2.5 MMS / OPC market-support and future

MMS is widely used in the manufacturing environment for connecting PLC, cell controllers, applications etc. In spite of this nowadays there is a clear trend towards object/component-based environments. Protocol-based environments, where the original MMS belongs to, lose ground and are often replaced by middleware solutions. To make MMS ready for the future the MMS standard can be lifted up by using an object oriented approach. There are several MMS adaptation possibilities to do this:

- Implementing of all MMS services over TCP/IP sockets (see also chapter 4 of the graduation report)
- Using of RPCs (Remote procedure calls) to request the services (see also chapter 10 Middleware)
- Translating of MMS PDUs from ASN.1 into CORBA IDL (allows the conversion of conventional MMS service requests into object-VMD invocations) (see also chapter 10 Middleware)

Another possibility is to us XML as message format instead of ASN.1 to create interoperability, transparency, etc between client and server (See also chapter 10 Middleware). All of these solutions are possible because of the flexibility of MMS. Because most of them are in the conceptual phase, today none of these solutions are available for the general public. And off course it's the question whether they will become important in the future or not.

The OPC-DA 2.x is very popular and widely used in the industrial environment. The OPC-DA 3 specification is just released and not yet widely used. But because of the backwards compatibility and the success of the predecessor version 2.x it is very likely that this new version 3 will become a new mile-stone in industrial communication in the future. Unlike OPC-DA the derived specifications (like OPC-AE) are at the moment less used in the industrial environment. Reason for this is that some of them are still being developed and others are simply not that popular as OPC-DA. Fact is that, listening to the industrial heterogeneity needs, OPC-XML-DA is going to play an important role in the future for data exchange between applications and devices.
7.6.3 Protocol generics

As told fieldbus protocols that are capable of messaging are not widely used and will disappear in the near future. Now look at the remaining solutions: industrial Ethernet protocols, MMS and OPC-DA. In spite of the different viewpoints all solutions do have generic elements (assuming that the industrial Ethernet protocols define an object model):

- Client/server communication relationship. This relationship allows client to request something from the server. The server responds on his turn. It is also possible that the server sends something to the client without receiving a request.
- Servers are standardized Object-Oriented representations of real world device data. Real world device data is represented as objects in the servers. The servers use an object-oriented data model to organize these objects.
- Clients have the possibility to read and write data (variables) in the server.
- Clients have the possibility to subscribe to a specific event (e.g. value-change of a variable). If the event happens the clients which have subscribed gets a response from the server.
- Ethernet-TCP/IP can be used as communication backbone between the clients and servers.

7.7 Conclusions & recommendations

The following main conclusions can be drawn on the basis of the preceding:

- Fieldbus protocols are not widely used on interface area 2 and very likely also not in the future. Next to this each type fieldbus protocol needs an IO driver implemented by SattStore. There are little generic elements in these protocol IO drivers (each fieldbus has got a different IEC61784 profile with different physical, data-link and application layer properties) and therefore combining fieldbus IO drivers is too hard to realize (only complex protocol converters could make the conversion between protocols).
- Industrial Ethernet protocols are more important because of the transition from protocol-based to object/component-based, but are, just like fieldbus protocol, (still) not widely used. Because of the use of object oriented client/server models interoperability is higher. Often an MMS subset is used that supports data access and events (E.g. domains and programs invocations not often implemented). But again a protocol-dependent IO driver is needed to implement the industrial Ethernet protocol (Different IEC61784 application specifications).
- MMS is a generic and widely used solution. It’s flexible but at the same time it is also complex to make an MMS implementation. Dependent on the implementation MMS supports all kind of messaging from data access to domain and program invocations as well as event, file and journal services. An MMS implementation called OpenMMS is available at ABB logistic Systems. Drawback of MMS is that MMS is on this moment not (yet) ready for the future (protocol-based)
- OPC-DA (version 2.x and in the future 3) is a very popular and widely used solution. Each device, fieldbus or industrial Ethernet protocol can be connected to an OPC server. Because each supplier of such devices or protocols has nowadays an OPC server OPC is the most generic solution available. The messaging is in this case based on variable access (data access). Variables can be read/write (synchronous and asynchronous) and monitored (subscription). To implement other messaging services (program invocations, etc) using OPC, there is a need for other specifications, which are not available or widely used, and thus also not interesting. OPC-DA is COM-based and thus based on Middleware techniques. The future will offer OPC-XML-DA with the XML-interface.

From SattStore point of view the implementation of an OPC Data Access client (OPC-DA 2.x, 3 or XML) implies the possibility to make a (data access) coupling with (hardly) any device controller supporting a fieldbus or industrial Ethernet protocol. Next to this the implementation of an MMS client (e.g. OpenMMS based) makes it possible to address all MMS servers. Industrial Ethernet protocols become in this case superfluous and are not value-added looking at the disadvantages and implementation issues.
8 Architectural requirements

According to the Bredemeyer decision model, described in chapter 3, the next step in the architecting process is to define the architectural requirements according to the architecture's goal (assignment), the SattStore communication environment (chapter 5), the present situation (chapter 6), the industrial protocol and standard research (chapter 7), and additional requirements defined by the ABB Logistic Systems. The architectural requirements are input to the meta-phase where the system internals will be described according to these architectural requirements.

First the UCIA-system and environment as well and functional and non-functional requirements will be considered from a high abstraction level with knowledge obtained from the assignment, the SattStore communication environment and additional demands of ABB Logistic Systems. After this the UCIA-system and environment as well as the functional and non-functional requirements are further deepened and a distinction is made between UCIA cell controller communication requirements and UCIA device controller communication requirement on the basis of the knowledge acquired in the chapters about the present situation and the industrial protocols and standards with additional demands of ABB Logistic Systems.

8.1 UCIA-system and environment

The initial goal of the UCIA-project was to develop an architecture, which software engineers can use to implement communication interface adapters for SattStore to communicate with industrial cranes and conveyors, in a way that generic elements can easily be reused, resulting in shorter project throughput times.

Paragraph 5.1 and 5.2, about CIM and the mapping of SattStore on the CIM pyramid, specifies the UCIA-architecture further to an architecture, which software engineers can use to implement communication interface adapters for SattStore to communicate with industrial crane cell controllers, exchanging crane cell jobs/statuses, and with industrial conveyor PLCs, exchanging conveyor PLC jobs/statuses.

From paragraph 5.3, the study of the transport environment of SattStore, SattMate, it follows that legacy communication interface adapters are restricted to the communication between crane Admin and crane cell controllers and between conveyor Admin and the conveyor PLCs. The use of these administrators interfere the development of an UCIA, as a crane administrator has to be used to communicate with crane cell controllers and a conveyor administrator to communicate with conveyor PLCs.

Instead of using a crane sub system for crane communication and a conveyor sub system for conveyor communication, the UCIA will be based on the fact that SattStore has to communicate with controllers on both cell level and control level according to the CIM communication pyramid. The UCIA is thus not restricted to the communication between crane Admin and crane cell controller and not restricted to the communication between conveyor Admin and conveyor PLCs.

The mentioned decisions result in the following definition for the UCIA:

The UCIA is an architecture, which software engineers can use to implement communication interface adapters for SattStore. A SattStore process X (transport administrator or other) communicates with cell controllers using cell jobs/statuses and a SattStore process Y (transport administrator or other) communicates with device controllers using device job/statuses.

The UCIA must be transparent for the contents of the jobs/statuses being exchanged. If a SattStore process sends for example a cell job A, this command must be understood by the cell level application (the application must know how to interpret the cell job A) for which the command is meant. If a SattStore process sends a device job B (e.g. an integer value), this command (integer) must be understood by the control level entity (device controller) for which the command is meant. There is thus no mapping between contents of data.
The UCIA-environment consists of:

- SattStore processes sending/receiving cell jobs/statuses (if SattStore processes exchange data with cell level applications that understand these jobs and generate the right statuses).
- SattStore processes sending/receiving device jobs/statuses (if the SattStore processes exchange data with device controllers that understand these jobs and generate the right statuses).
- Third party cell level applications sending/receiving cell jobs/statuses.
- Third party control level applications (device controller) sending/receiving device jobs/statuses.

### 8.2 UCIA functional requirements according to the assignment

The following UCIA functional requirements are added on the basis of the assignment, the UCIA communication environment and additional demands of ABB Logistic Systems.

- The UCIA sets up a connection between SattStore processes (administrator or other) sending/receiving cell jobs/statuses type X and third party cell level applications sending/receiving cell jobs/statuses X and sets up a connection between SattStore processes (administrator or other) sending/receiving device jobs/statuses type Y and third party control level applications (device controllers) sending/receiving device jobs/statuses type Y.
- To SattStore applications Process-IO must be used (if possible) as Inter Process Communication (IPC) mechanism to interface SattStore processes with the UCIA. UCIA supports in this case the Process-IO package, which implements Process-IO. This is an ADA package using a C++ package for handling all of the operation system specific commands.
- The UCIA is restricted to the communication between SattStore processes and third party crane/conveyor controllers. The UCIA should therefore not support additional functionalities like monitoring, message scheduling and other “Business logic functionalities”. Only error logging should be supported in the architecture.

Paragraph 8.4 and 8.5 describe the functional requirement added to the UCIA after the present situation and industrial protocols and standards research.

### 8.3 UCIA non-functional requirements according to the assignment

Next to the functional requirements there are also non-functional requirements. The following UCIA non-functional requirements are added on the basis of the assignment, the UCIA communication environment and additional demands of ABB Logistic Systems.

- The UCIA components must run on the Microsoft Windows NT/XP platform and it must be possible to run UCIA software architecture implementations next to each other within one Windows NT environment.
- To provide i.a. location and technology transparency towards SattStore middleware technologies must effectively be used.
- The UCIA should be built in such a way that software reuse is easy when using the UCIA.
- The reliability of the exchanged jobs/statuses must be as high as possible.
- The communication between SattStore processes and a third party controller is not real-time (there is no prediction demand for when specific information must be available for a controller).

8.4 and 8.5 describe the non-functional requirement added to the UCIA after the present situation and industrial protocols and standards research.
8.4 Device controller communication

After examining the present situation as well as the industrial protocols and standards and demands from ABB Logistic Systems, a lot of UCIA functional and non-functional requirements can be added in relation with device controller communication. Most of the requirements are derived from OPC-DA and MMS, explained in chapter 7.

8.4.1 Functional requirements

8.4.1.1 Client/server architecture

As discussed in chapter 6 the communication between SattStore and device controllers (in the present situation conveyor PLCs) is always based on an exchange of bit and integer variables. A master/slave datalink protocol (Siemens 3964R or Comli) is used on top of a serial connection to let SattStore and the conveyor PLCs understand each other and exchange the necessary bit and integer variables that represent the device jobs and statuses. A polling method is used in this case. The used protocols are dependent on the third party conveyor PLCs but do in general contain generic components.

Next to these vendor-dependent solutions used by ABB, there are a couple of standard industrial (network) solutions to connect third party device controllers, such as PLCs, to SattStore. As discovered in chapter 7 OPC and MMS are the most generic solutions. OPC and MMS are client/server software architectures based on middleware techniques. OPC is based on a standard COM or XML-interface, MMS does not specify a particular middleware by itself. Clients connect to OPC or MMS servers, which encapsulate the third party dependent device controller logic (fieldbus protocols, industrial Ethernet protocols or other vendor-dependent protocol like the mentioned Siemens 3964R and Comli protocols).

As discussed in chapter 7 the support of OPC-DA (2.x/3.0 and XML) together with MMS will best meet the industrial communication requirements, in the case of communication with device controllers, from a viewpoint of using and implementing, market-support (now and in the future) and generic elements. To get generic communication between SattStore and device controllers the device controllers must thus be addressed by so-called data servers and for that reason the UCIA must be a client/server architecture.

8.4.1.2 Data Server Model (DSM)

Although OPC and MMS are the most interesting data server solutions, to be able to support other future solutions the UCIA will not be restricted to OPC-DA 2.x/3.0, OPC-XML-DA and MMS data servers. In the case of communication with device controllers, it must be possible to support other data server types that look like the mentioned data servers (This has been decided in consultation with ABB BV). This can be for example vendor-dependent data servers or future OPC-DA or MMS versions.

On the basis of the generic elements of the OPC-DA and MMS servers a server model called the Data Server Model (DSM) has been defined. A data server is a DSM server if the following requirements have been met:

- The data server contains an object-oriented software representation of real world device controller variables and takes care of the connection with the variables in the physical devices.
- The object-oriented representation of the device controller variables consists of variable classes and methods that describe the interface to the variables in the variable classes.
- Variables in the data server are identified by variable names.
- The data server offers clients the possibility to read/write the variables with the help of method calls.
- The data server offers clients (optional) the possibility to subscribe/unsubscribe on variable-transitions (value-changes) with the help of method calls (variable monitoring). If a transition appears, the data server is able to notify interested clients with the help of a push call.
- Supported variable data types are at least bits and integers.
There are two types of DSM servers possible:

- **Active DSM servers**: Data servers which support subscriptions on variable-transitions and are capable of notifying the interested clients on a specific subscribed variable-transition.
- **Passive DSM servers**: Data servers which do not support subscriptions on variable-transitions and are not capable of notifying the interested clients on a specific subscribed variable-transition.

OPC-DA 2.x/3.0 servers, OPC-XML-DA servers and MMS servers are all active servers capable of notifying interested clients on a specific subscribed variable-transition. Because the UCIA has to become a very generic architecture, where both legacy and new servers could be integrated, and with as little restrictions as possible, the support for both active and passive servers will be integrate. The UCIA is thus a client/server architecture capable of connecting SattStore to all kind of active and passive DSM servers. Variables in DSM servers can be read, write and optional monitored.

On the basis of these findings the following UCIA functional requirement can be added when communicating with device controllers:

- The UCIA is a client/server architecture, where SattStore has client functionality to invoke services and methods on different DSM servers.

Notice that DSM is an abstract model and only specifies properties for servers. Each DSM server could have its own implementation of the required properties. For example each DSM server could have its own method calls by which a client can read and write variables in the server and each server could support a different underlying communication technology (protocol or middleware-based).

### 8.4.1.3 Communication services

On the basis of the DSM model together with requirements from ABB the following communication services have been defined and added to the functional requirements of the UCIA:

**Synchronous communication services**

- SattStore is able to read/write variables in active/passive DSM servers through a blocking synchronous call.
- For any synchronous read/write operation, value and status information shall always be delivered to SattStore.
- SattStore is able to specify a timeout (in milliseconds) after which a blocking synchronous operation is abandoned.

**Deferred synchronous communication services**

- SattStore is able, to read/write variables in active/passive DSM servers through a non-blocking deferred synchronous call.
- For any deferred synchronous read/write operation, value and status information shall always be delivered to SattStore.
- SattStore supplies an additional object reference with each read or write call. When the response arrives this object reference is used to deliver the response back to SattStore.

**Data subscription services**

- SattStore is able to subscribe/unsubscribe to value-transition of variables in active DSM servers through a non-blocking deferred synchronous call.
- For any deferred synchronous subscribe/unsubscribe operation, status information shall always be delivered to SattStore.
- SattStore is able to specify value-transitions.
- SattStore shall asynchronously be notified by the active DSM server each time the value-transition it has subscribed to happens.
- SattStore supplies an additional object reference with each subscribe call. When the value-transition happens this object reference is used to deliver the value-transition back to SattStore.
8.4.2 UCIA non-functional requirements

8.4.2.1 Object-oriented technology
There are several aspects of using object-oriented technology in the UCIA architecture even if there is no object-oriented software used in the present and past situation. Data servers are object-oriented representations of the connected device controllers consisting of classes and methods that describe the interface to these device controllers. Next to this there is an obvious current industrial trend from protocol-based environments to object and even component-based environments improving software reuse. Finally the most dominant environments for software development today, especially for GUIs, are object and component-oriented, like Microsoft Visual C++, Visual Basic and Java.

8.4.2.2 Network communication
For data exchange with DSM servers the following network communication requirements exist:

- The UCIA is built on top of a TCP/IP connection.
- By using middleware, application engineers do not need to implement session and presentation layer which is often too costly, too error prone and too time-consuming.

8.4.2.3 Reliability/Security
For data exchange with DSM servers the following reliability requirements exist:

- Exactly-once reliability or at-most-once reliability should be guaranteed. This guarantees the request from SattStore to execute only once. For at-most-once reliability SattStore is notified about the failure and the request is send again.
- SattStore and the DSM servers shall not be in a position to crash or deadlock the UCIA.
- The UCIA shall provide timeout mechanisms preventing SattStore or DSM servers to enter deadlock situations when communicating.
- SattStore shall be informed about errors in any synchronous, deferred-synchronous and asynchronous operation.

8.4.2.4 Heterogeneity/interoperability
For data exchange with DSM servers the following heterogeneity/interoperability requirements exist:

- DSM data servers and UCIA-system components can be written in different programming languages, which should be transparent to SattStore.
- For interoperability with DSM servers vendor and technology independency and transparency must be supported.

8.4.2.5 Scalability
For data exchange with DSM servers the following scalability requirements exist:

- The UCIA supports access transparency. The way in which SattStore access an object/component in the DSM server is independent of whether it is local or remote.
- The UCIA supports location transparency. SattStore does not know the physical location of objects/components it interacts with and must thus be transparent for the location of DSM servers.
- The UCIA must support Concurrency transparency. Several users (e.g SattStore) can access objects simultaneously (shared data) without mutual influence. This is mainly based on future assumptions.
8.4.2.6 Usability and maintainability
For data exchange with DSM servers the following usability and maintainability requirements exist:

- The UCIA must support object/component reuse. For this reason it must be possible to easily adapt to a specific communication environments.
- The usability must be as high as possible. The UCIA must be easy to learn, it must be easy to create inputs (from SattStore) and interpret outputs (by SattStore), and must be easy to maintain.

8.5 Cell controller communication
After examining the present situation as well as the industrial protocols and standards UCIA functional and non-functional requirement can be added in relation with cell controller communication.

8.5.1 Functional requirements

8.5.1.1 Client/server architecture
As discussed in chapter 6 in the present (and past) situation the communication between SattStore and cell level entities (in the present situation crane cell controllers) is always based on an exchange of data-frames or functional message frames. An application protocol is used on top of a serial or socket connection to let SattStore and the cell controllers understand each other and exchange the necessary cell jobs and statuses. The used protocol is strongly third party cell controller dependent and contains in general less generic components.

Next to these strongly vendor-dependent solutions used by ABB, there are no standard industrial (network) solutions to connect third party cell controllers to SattStore. As explained in chapter 7 the communications between SattStore and the cell controllers fall into interface area 1 and can be addressed by plant networks using standard office protocols or middleware techniques. Standard protocols are for example FTP, HTTP, etc, possible middleware techniques are enumerated in chapter 8.

The communication with device controllers clearly has a suitable industrial solution: DSM servers. For the communication with cell controllers it is harder to define functional and non-functional requirements. Because of the differences in cell controllers, and the communication support of these cell controllers, it is not possible to define a sort of cell controller model with support for a large group of cell controllers.

However, fact is that plant networks, where the coupling between SattStore and Cell controllers belongs to, hardly always use a client/server relationship between applications and also middleware techniques are often part of the communication between entities on the plant network. For this reason it must be defined that for the communication with cell controllers also a client/server relationship should be used. This conforms to the present situation experiences where, in spite of the different protocols, SattStore always makes a request to the cell controller that executes the commands and generates the right responses back to SattStore.

On the basis of these findings the UCIA is also a client/server architecture for cell controller communication, where SattStore has client functionality to invoke services and methods on different cell controllers.

8.5.1.2 Communication services
As discussed in chapter 6 byte-string data types, of about tens of bytes, are exchanged between SattStore and the cell controllers, often accompanied with an integer field or fields containing the job or status identification. On the basis of this present situation as well as demands of ABB Logistic Systems the following communication services have been defined for the communication with cell controllers and added to the UCIA functional requirements.
Architectural requirements

Synchronous communication services
- SattStore is able to send a byte-string (with the cell job) and integer (job identification) together to the cell controller in a blocking synchronous way.
- For any send operation status information shall always be delivered to SattStore.
- SattStore is able to specify a timeout (in milliseconds) after which a synchronous send operation is abandoned.

Deferred synchronous communication services
- SattStore is able to send a byte-string (with the cell job) and integer (job identification) together to the cell controller in a non-blocking deferred synchronous way.
- For any send operation status information shall always be delivered to SattStore.
- SattStore supplies an additional reference for each deferred synchronous send-operation. When the response arrives this reference is used to deliver the response back to SattStore.

Asynchronous communication services
- The cell controller can asynchronously send a byte-string (with the cell status) and integer (status identification) together to SattStore.
- SattStore supplies an additional reference for data coming from a cell controller. When the cell controller sends data this reference is used to deliver the data to SattStore.

8.5.2 UCIA non-functional requirements
The same non-functional requirements as mentioned in paragraph 8.4.2 hold for the UCIA when communicating with cell controllers. Only object-oriented technology is not a demand, but strongly recommended.
9 Meta architecture

This chapter introduces the meta-phase where information from all preceding chapters, especially the architectural requirements, are combined, compared and used to formulate the architectural vision to guide decisions during the rest of the system structuring in the conceptual phase. Important trends and recommendations from different researches in the communication area to vendor independent communication methods existing in the industrial WMS environment in general, together with conclusions and recommendations from the present situation and architectural requirements will be used.

The following subjects are part of the UClA meta-phase and influence the UClA internals:

- The choice of an UClA architectural style: 2-tier, 3-tier or n-tier (This chapter)
- A comparison of different 3-tier generic UClA architectural alternatives. (This chapter)
- A comparison of different middleware technologies and clear preferences for the UClA (Chapter 10)

9.1 Architectural style: 2-tier, 3-tier or n-tier

As discussed in the architectural requirements middleware technologies must effectively be used for obtaining access transparency, location transparency and concurrency transparency between SattStore and DSM servers and cell controllers. Also software reuse must be easy when using the UClA as well as offering at-most-once reliability and error-logging functionalities. Next to this technology independency and transparency it is important to acquire interoperability and in the case of DSM sever communication object-oriented technology must be supported. From the preceding it is also known that synchronous as well as deferred synchronous and asynchronous traffic is a demand for the client/server interaction.

As mentioned in chapter 4 distribution is based on tiers. There are 2-tier architectures, 3-tier architectures and n-tier architectures. The next paragraphs explain why a preference goes out to the use of a 3-tier architecture with an application server beyond 2-tier and n-tier architectures.

2-tier architecture

The communication mechanisms used in a 2-tier architecture are in general synchronous SQL calls. Middleware techniques are hardly used in 2-tier architectures, which make 2-tier client/server architectures work best in a homogeneous environment, where all servers and the interactions with these servers are of the same type, and where the user itself must implement transmission reliability. A 2-tier architecture will not be used for the UClA, because a 2-tier architecture could never meet the architectural requirements.

3-tier architecture

A 3-tier architecture offers many advantages compared to the 2-tier architecture. The UClA should support a heterogeneous environment, in this that SattStore will be able to use this architecture to exchange information on both control and cell level. In addition the cell controllers on the cell level and the DSM servers have different interfaces. By using a 3-tier architecture, middleware implements a middle tier between SattStore and all controllers. The middleware masks remoteness to SattStore and network details to both SattStore and software engineers, and supports heterogeneous environments very well because of the platform and language independency. An application server implemented by middleware technologies can provide a complete set of configuration, security and maintainability functions. (Read more in chapter 10).

N-tier architecture

As discussed in the previous paragraph a 3-tier UClA architecture fulfills the need for the communication between SattStore and DSM servers and cell controllers. By splitting up the software layers, as mentioned in paragraph 4.4.2.1, in more than 3-tiers the system become too complex and do not add any further value to the UClA. Therefore it is not recommended to divide the UClA into more than 3 tiers.
9.2 3-tier UCIA architectural alternatives

From the previous paragraph it is known that a clear preference goes out to a 3-tier UCIA. This paragraph compares three different 3-tier UCIA architectures and enumerated their advantages and disadvantages.

9.2.1 Introduction

OPC-DA 2.x/3.0 and OPC-XML-DA both have a 3-tier ORB architecture based on respectively COM and XML-SOAP. In the OPC-DA case the OPC-DA client API is essentially a combination of the COM API and the OPC-DA client API. The COM API implements the COM-interface while the OPC-DA client API specifies the used interfacing commands (for example the OPC read and write in/in-out arguments, etc). The OPC-DA server API is also a layered API, which is part of the OPC-DA server, and consists of both a COM-interface API and the OPC-DA server side interfacing commands. This learning about client and server APIs can also be applied to OPC-XML-DA.

MMS does not specify a middleware interface. To fit MMS in a 3-tier architecture a middleware technique will be used to exchange the different service requests and responses. Initiatives for this have been mentioned in paragraph 7.6.2.5. The MMS client and server API should thus be extended with a suitable middleware API that again strongly affects the way in which the MMS client and server send and receive commands.

Next to the known OPC-DA and MMS solutions other active and passive DSM servers can also be based on a particularly middleware technology. In general a DSM client API will thus be a layered API consisting of the middleware API and a DSM client API. The middleware API implements the middleware-interface while the DSM client API specifies the used interfacing commands exchanged over that middleware interface. The DSM server API is part of the DSM server and DSM servers that do not define a middleware technology in the specification must thus be extended with a suitable middleware API to be part of a 3-tier architecture.

The same counts for cell controllers. Most of them do not specify a specific middleware-based interface. They should thus be extended by a suitable middleware API to exchange data over the middleware.

9.2.2 Alternative 1

Implement all passive and active DSM client APIs (like the OPC-DA client API, etc) as well as the cell controller client APIs in SattStore. For each type DSM server and each type cell controller a communication interface adapter is needed (indicated as Name_IO) that implement the DSM server or cell controller dependent client APIs as well as the matching middleware APIs. If DSM servers or cell controllers do not specify a middleware technology by itself a middleware technique will be chosen on the basis of the architectural requirements. If DSM servers or cell controllers are addressed by the same middleware a combination of these middlewares must be possible. Figure 9-1 shows the architectural diagram of this solution with two types of DSM servers and two types of cell controllers.

Advantages: From SattStore point of view each type active and passive DSM server and cell controller is directly addressed by its “own” client API. Dependent on the supported DSM server or cell controller middleware technologies, the middle tier of different solutions could be combined and middle tier elements therefore reused.

Disadvantages: To support all types of DSM servers and cell controllers (existing an new types) SattStore needs to implement the different client APIs, and for each type DSM server or cell controller a communication interface adapter must be created (indicated as Name_IO). As discussed in chapter 6 these communication interface adapters, containing the client APIs, are in general rather complex to implement and contain in the case of cell controller communication less generic elements. Next to this the UCIA architecture must support different types of DSM server and/or cell controller dependent middleware technologies. All in all this is thus definitely not a generic UCIA solution and solve little of the problems of the present situation.
9.2.3 Alternative 2

Use the client API of DSM server type 1 (e.g. OPC-DA), implemented in DSM_1_IO, to address all types of DSM servers. Gateways are needed to convert DSM type 1 client calls to other DSM server type calls as well as convert underlying technologies. Because all servers are DSM servers this is possible. From SattStore point of view all types DSM servers act like the DSM type 1 server. To use the UCIA the DSM server type 1 must be an active server to support subscription on variable transitions. DSM Servers addressed by gateways could be active or passive and only have subscription functionalities if they are active. On the middle tier only one middleware technology is present to address all DSM servers. This middleware is specified by the DSM server type 1 or chosen on the basis of the architectural requirements.

For the cell controller communication a same sort of mechanism can be used. A client API of a cell controller type 1 (e.g. Daifuku) is used, implemented in CC_1_IO, to address all types of cell controllers. Gateways are needed to convert cell controller type 1 client calls to other cell controller type calls as well as convert underlying technologies. From SattStore point of view all types cell controllers act like the cell controller type 1. On the middle tier only one middleware technology is present to address all cell controllers. This middleware is specified by the cell controller type 1 or chosen on the basis of the architectural requirements.

If the DSM server type 1 and the cell controller type 1 are addressed by the same middleware a combination of these middleware must be possible. Figure 9-2 shows the architectural diagram of this solution with two types of DSM servers and two types of cell controllers.

Advantages: The SattStore client only needs to support the DSM server type 1 client API while all types of DSM servers are supported. Next to this the SattStore client only needs to support the cell controller type 1 client API while all types of cell controllers are supported. The implementation of one generic communication interface adapter for DSM server communication (DSM_1_IO) as well as the implementation of one generic communication interface adapter for cell controllers (CC_1_IO) suffice to address all DSM servers and cell controllers now and in the future. Finally only one or two generic middleware technologies are present in the UCIA architecture.

Disadvantages: The developing of the generic communication interface adapters is rather complex. The development of the gateways is also a complex activity. Both the rather complex DSM server type 1 calls (e.g. OPC-DA calls) must be converted to rather complex calls to other DSM servers, as well as the underlying technologies (e.g COM-based OPC-DA to XML-based OPC-XML-DA). The same counts for the cell controller gateways. Next to this also subscription management is difficult, which DSM server keeps which subscriptions? And there is still not a generic model because from SattStore's point of view passive and active servers could not be addressed by the same set of commands, because subscription is not possible in the case of passive servers. Adaptations to the model could solve these latter DSM problems.
9.2.4 Alternative 3

The main problems of alternative 2 are the rather complex communication adapters and gateways. For this reason alternative 3 contains simple user-defined client APIs that are able to address all types of DSM servers and cell controllers (alternative 3 look like alternative 2). One client API, implemented in a relative simple communication interface adapter, is used to address passive and active DSM servers. Another client API, implemented in a relative simple communication interface adapter, is used to address cell controllers. Because of this simplicity gateways are less complex. Only the third party DSM server or cell controller commands and technologies could be complex but are in this case hidden from SattStore.

Advantage: By defining simple user-defined interfaces both DSM server and cell controller communication could be addressed by a generic middleware technology on the middle tier. This middleware technology must fulfil the architectural requirements.

Disadvantages: See alternative 2.
10 Middleware

This chapter starts with the purpose and origin of middleware, followed by a possible middleware classification. According to the functional and non-functional design aspects described in chapter 4, the present communication solutions described in chapter 6, new communication solutions described in chapter 7, the architectural requirements described in chapter 8 and the meta-architecture subjects discussed in the previous chapter, a summary of the most relevant requirements is listed in paragraph 10.3 to guide the preliminary middleware choice(s). According to these requirements, market trends and other stakeholder concerns, the most interesting middleware technologies for the UCIA-system are examined and compared with each other, with their advantages and drawbacks. This comparison will be used to formulate a part of the architectural vision to guide decisions during the rest of system structuring.

A short description of all middleware classes can be found in this chapter and a more detailed description about the different middleware technologies, including purpose and origin, technical details, functional and non-functional design aspects and possible trends can be found in document Middleware [ABB 2003-4].

In spite of the fact that the term middleware and the matching solutions are often mentioned in the preceding chapters, the architectural requirements and the alternative discussion of the previous chapter are necessary to come to substantiated choices. For that reason the middleware chapter is positioned here and not sooner in the graduation report.

10.1 Purpose and origin of middleware
The first middleware products were not known as middleware per se. They were considered as helper tools for communication applications. When large companies started to work on creating distributed systems, they needed a middleware product to aid in performance, control, data integrity, and ease of use. In the early 1980s Sun Microsystems developed a product based on Remote Procedure Call (RPC) protocol, which allows one program to request that another program (across the network) performs a task without having to be troubled about the network particulars. The ADA programming language was also considered the pioneer of the middleware concept since it provided runtime libraries that allowed for portability of the software across a wide variety of hardware platforms and Operating Systems (OS). The advances in ADA programming language’s engineering features such as reusability, portability, and encapsulation made for a better class of reliable middleware products.

Middleware products were increasingly being used as the industry moved from the client/server systems to the multi-tiered-distributed systems. In 1989, the Object Management Group (OMG) was founded. This group was the first to develop middleware specifications called Object Request Brokers (ORB), which is a middleware technology that manages communication and data exchange between distributed objects. OMG’s Common ORB Architecture (CORBA) was influential in the development of other object-oriented middleware technologies, and more recently for component-oriented middleware technologies. At the same time Message Oriented Middleware products (MOMs) conserve their importance and are being adapted to object and component oriented technology.

The purpose of all middleware technologies is to mask remoteness and underlying networking technologies to the client and to provide platform and programming language independency, together with increasing usability, flexibility, interoperability, maintainability, heterogeneity, scalability, portability and reliability.

10.2 Middleware classification
There are many ways to classify middlewares. One of the problems of classifying is that middlewares are not just one service, but are a combination of several. Another problem is that there are a large variety of middlewares themselves and many different technologies behind them. The middleware category from
Wolfgang Emmerich [EMME 2000] is the one, which is the inspiration of the classification used in this document. The motivation for this choice can be found in the document [ABB 2003-4]. The main difference between the different classes is the way applications exchange information with each other, e.g. using a request/response mechanism, using database calls/transactions or with the help of messages. The last class “next generation middlewares” addresses limitations in the other classes.

The middleware classification used in this document is illustrated below.

![Middleware classification](image)

A particular middleware can sometimes be placed in different middleware categories, but the descriptions of each category will make this major division clear. Below all categories are shortly described, according to the functional design aspects and relevant requirements of the UCIA-system, which are described first.

### 10.3 Summary of relevant requirements

By using a 3-tier architecture middleware technologies are used on the middle tier. To fulfill all architectural requirements there are clear demands on the used middleware technologies (whether defined by the DSM server solution or user-defined). This paragraph shows summarizing the relevant middlewares categorized in the functional and non-functional design aspects (see paragraph 4.5).

**Network communication**

For data exchange with both cell controllers as well as DSM servers the following middleware network communication requirements exist (notice that instead of a middleware technology also middleware technologies could be used if there is a need to).

- The middleware shall allow invocation of both local and remote methods/objects.
- The middleware shall provide the same access capabilities to both local and remote methods/objects, thus masking remoteness to the client.
- The middleware technology must support both communications between SattStore and cell controllers as well as between SattStore and device controllers.
- The middleware technology masks underlying network technologies to the client.

**Coordination**

Independent of whether SattStore communicates with cell controllers or DSM servers, both synchronous, deferred synchronous and asynchronous communication must be supported by the middleware. Asynchronous publish/subscribe will especially be important towards active DSM servers where clients can subscribe to server variables, which are than published by the server (pushed).
Reliability
As discussed in the architectural requirements data exchange must have a reliability as high as possible resulting in the following middleware requirements:

- SattStore and third party processes shall not be in a position to crash or deadlock a middleware system process.
- The middleware shall provide timeout mechanisms preventing client or server applications to enter deadlock situations when communicating via the middleware.
- The middleware offers the possibility to inform SattStore about errors in any synchronous, deferred synchronous and asynchronous operation.
- The middleware supports at least at-most-once reliability.

Heterogeneity/interoperability
The following middleware requirements are part of the obtaining of heterogeneity/interoperability:

- At least the Microsoft Windows NT/XP platform must be supported by the middleware
- The middleware makes it possible to write DSM servers, cell controllers, and UCIA-system components in different programming languages.
- For interoperability with DSM servers and cell controllers vendor and technology independency and transparency must be supported.

Scalability
Access transparency, Location transparency and Concurrency transparency must be supported by the middleware.

Usability
The middleware must be easy to implement, maintain, adapt and use.

10.4 Request Oriented Middlewares
Middleware technologies in this class provide communication between applications designed according to the request/response paradigm.

10.4.1 Procedure Oriented Middleware

10.4.1.1 Introduction
Procedure oriented middleware focuses on integration via procedure or method calls (RPC mechanism) across different hardware and operating system platforms in a language independent way, where the complexity involved in the development of distributed processing is reduced by keeping the semantics of a remote call the same whether or not the client and server are located on the same system.

10.4.1.2 Functional design aspects
Method invocation/Network communication
Clients that reside on other hosts can invoke procedures across the network. Procedural middleware implements these procedure calls by marshaling the parameters into a message that is sent to the host where the server item is located. The server item unmarshals the message and executes the procedure and transmits marshaled results back to the client, if required. Marshaling and unmarshaling are implemented in client and server stubs (endpoints) that are automatically created by a compiler from an RPC program definition.
Client and server stubs communicate with each other using an Interface Definition Language (IDL).

IDL is a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language.

**Coordination**
RPCs are synchronous, request/reply interactions between one client and one server, which involve blocking of the client until the server fulfills its request. Asynchronous and multicast communication is not supported directly by procedural middleware. RPC can be combined with MOM for asynchronous processing. Because most RPC implementations do not support peer-to-peer or asynchronous client/server interaction RPC is not well suited for applications involving distributed objects/components or object-oriented programming.

**Reliability**
RPCs have an at-most-once reliability. The procedural middleware supports exceptions if an RPC fails by returning a message that a failure occurred. Exactly-once semantics or transactions are not supported. If the network goes down, the server is slow, or the message is lost, the client is left waiting unless time-outs are programmed, which could create additional problems in that the message may be sent more than once.

**Heterogeneity**
Procedural middleware can be used with different programming languages and moreover across different hardware and operating system platforms.

**Scalability**
The scalability of RPCs is rather limited. Unix and Windows RPCs do not have any replication mechanisms to scale RPC programs. Thus replication has to be addressed by the designer of the RPC-based system, which means in practice that RPC-based systems are only deployed on a limited scale.

**Usability**
By using RPC, the complexity involved in the development of distributed processing is reduced by keeping the semantics of a remote call the same whether or not the client and server are located on the same system. However, RPC increases the involvement of an application developer with the complexity of the master-slave nature of the client/server mechanism.

### 10.4.2 Object and Component Oriented Middleware

#### 10.4.2.1 Introduction
This class provides communication mechanisms to interact with objects/components that may reside on other machines. Object technologies have had a large impact on industrial software development process and came with features like encapsulation and abstraction, which lead to more flexible software products. Object oriented programming has several limitations, especially in the creation of reuse. Component technologies were developed providing components that isolate and encapsulate specific functionalities and offer them as services in such a way that they can be adapted and reused without having to change them programmatically.
10.4.2.2 Object versus component technologies

Starting from object oriented programming this class can be distinguished into the following sub classes:

**Local component technologies**

Component models provide a mechanism by which software engineers can develop applications by composing components through their well defined interfaces rather than developing new or changing existing components. Local component technologies, e.g. Microsoft’s **Component Object Model (COM)**, and Sun’s **JavaBeans** have in common, that they do not provide any mechanisms for distributed deployment nor do they provide for distributed communication mechanisms to interact with distributed components.

**Distributed object technologies**

Concurrently with the evolution of object models into local component models the industry realized that it was no longer viable to assume that object communication could be connected to just one host. To address that problem, OMG developed a middleware specification, called ORB, which provided for communication and data exchange between distributed objects. OMG’s **Common ORB Architecture (CORBA)** was influential in the development of Java's **Remote Method Invocation (RMI)** specification, which provides for invocation of Java methods across machine boundaries, and for the definition of a distributed invocation capability for Microsoft’s COM that is commonly known as **DCOM**. CORBA, DCOM and RMI, also known as Distributed Object Computing (DOC) technologies, evolved from and address the limitations of RPC.

**Distributed component technologies**

Creating reusable objects is rather difficult and in addition DOC middleware does not provide sufficient mechanisms to prevent tight coupling among collaborating object implementations. In addition the designer of a distributed server object needs to use particular implementations of persistence, transaction, concurrency control and security services, which make it even more difficult to reuse a server object in a different setting. These obstacles led to the development of Distributed Component Computing (DCC) frameworks, which isolate the developer from implementation of persistence, transaction, security, etc.

The three main distributed component frameworks are Microsoft’s **COM+**, which supports the execution of COM components and Microsoft's Transaction Server (MTS) and thus implements transactional and security capabilities as well as distributed communication. Sun defined the **Enterprise Java Beans (EJB)** specification as part of their Enterprise Edition of the Java 2 platform. The OMG defined the **CORBA Component Model (CCM)**, introduced in CORBA 3.0, which uses the CORBA object model as its underlying object interoperability architecture. CCM is compatible with EJB in order to extent the power of EJB and at the same time provide a distributed component model for languages other than Java.

10.4.2.3 Functional design aspects

**Object invocation/Network communication**

DCOM/COM and CORBA make use of an Object Request Broker (ORB) technology, which promotes the goal of object communication across machine, software and vendor boundaries. Java RMI allows Java objects to be executed remotely, and this provides ORB-like capabilities. A recent trend in distributed object middleware techniques is using the more loose coupled capabilities of the Web (see paragraph 10.6).

**Object Request Broker (ORB)**

ORBs promote interoperability of distributed object systems because they enable users to build systems by piecing together objects, from different vendors, that communicate with each other via the ORB. The ORB must support many functions in order to operate consistently and effectively, but many of these functions are hidden from the user of the ORB. Developers are only concerned with the object interface details, improving system maintainability. It is the responsibility of the ORB to allow objects to hide their implementation details from clients, like programming language, operating system, host hardware, and object location.
To do this the middleware runtime installs end points on the client side (client stubs or proxies) and on the server side (server stubs or skeletons). The client object submits an invocation to the local image of the distant server object. The local ORB stub marshals the data and transmits it across to the server object. The server object receives the message and the remote server stub unmarshals the data. The results are returned to the client object in the same indirect way.

![Figure 10-3: Object Request Broker (ORB)](image)

This appears to be similar to procedural oriented middleware, but this middleware operates with objects. For this scheme to work several conditions are required, at least a language independent IDL for describing interfaces and generating stubs for various target languages. A broker acts as a middleman that contacts a number of data sources, obtains their reference IDs, collecting data, and sometimes reorganizes data.

CORBA uses the Internet Inter ORB Protocol (IIOP) as the communication protocol between distributed objects. Java RMI relies on a protocol called the Java Remote Method Protocol (JRMP). Using DCOM client and server are connected through an underlying RPC protocol called the Object Remote Procedure Call (ORPC). TCP, HTTP, UDP or IPX/SPX can be used as underlying transport protocol.

**Component models**

There are two different layers at which co-operating components are tied together. In the component link layer, the code that comprises the cooperating components is linked in a way that enables them to interact. In the component invocation layer, the components interact by dynamically invoking operations defined in component interfaces. Components are primarily constructed around the invocation layer above.

Central to the notion of distributed components is the idea that the designer of a component should only be concerned with the application or business logic and not be burdened with the implementation of location, persistence, transactional capabilities and security. Usually, an interface specification is used to describe a component, where IDLs describe the services provided by a component and in order to shield components from the underlying infrastructure specifics, containers are introduced. A component provides event interfaces the container automatically invokes when particular events occur (such as activation or passivation of server implementations). The container provides generic interfaces to the component that it can use to access container functionality offered by the underlying middleware technologies such as transactions, security, persistence and notification.

CCM uses the CORBA object model as its underlying object interoperability architecture, thus using IIOP as the transport protocol to communicate between distributed components. EJB uses both IIOP and RMI. COM+ clients communicate with the server using either DCOM or COM+, depending on the Windows platforms installed, or SOAP on top of the HTTP protocol. The application can use both communication techniques in the same environment. DCOM/COM+ can be used in LANs or WANs where security and firewalls do not cause problems for communication. SOAP can be used when clients are connected to the server environment using internet/intranet connections. (SOAP is described in paragraph 10.6.3).
Coordination
CORBA, COM/DCOM and RMI all support synchronous, deferred synchronous and asynchronous remote object invocations.

Component models extend the object models with better reuse of components and transparencies for e.g. persistence and security but also extend the support for asynchronous messaging (pass/queue and publish/subscribe). EJB supports the Java Message Service (JMS) to support asynchronous messaging. COM+ uses queued components for asynchronous messaging and CCM includes the CORBA Messaging, which extends CORBA with Asynchronous Method Invocations (AMI). AMI allows decoupling client from server operation (non-blocking communication). There are two models, namely the polling model and the callback Model. Store and forward” semantics are supported by Time-Independent Invocations (TII).

Reliability/security
When communicating in synchronous and deferred synchronous mode CORBA, DCOM and RMI have at-most-once reliability. Communicating in a one-way mode guarantees a best effort delivery. CORBA messaging, or the Notification service can be used to achieve exactly-once reliability.

The security model of DCOM is based mainly on the LAN Manager security, while COM+ defines its own security. CORBA has also defined an own security service to handle authentication of clients. Since Java RMI works with java programming language it inherits the security built into Java. Use of RMI Security Manager can enable dynamic class loading thus providing additional security.

Heterogeneity
CORBA is a specification and can therefore be used on heterogeneous platforms, operating systems and programming languages as long as there is an ORB implementation for the platform and a language mapping (IDL) for the programming language. On the negative side CORBA has been created by standard bodies and is full of compromises.

COM/DCOM also supports multiple programming languages by using different IDLs. COM/DCOM is mainly supported on the Windows platform. COM+ also uses SOAP for distributed communication improving heterogeneity.

RMI is only applicable as middleware between clients and servers implemented in the Java language since it relies heavily on Java object serialisation, but is nearly supported on all hardware platforms as long as there is a Java Virtual Machine (JVM) implementation for that platform. There is some recent integration between CORBA and RMI. RMI can use the CORBA IIOP protocol and there is a mapping between the Java and the CORBA IDL, which make use of IDL for Java applications optional. Exposing Java objects directly through an ORB or the Java-IDL mapping makes it possible to invoke Java objects from non-Java environments. A CCM component can appear as an EJB bean to EJB clients, an EJB bean can appear as a CCM component by using appropriate bridging techniques and EJB also support CORBA IIOP as its communication framework.

Scalability
The support of object oriented middleware for building scalable applications is still somewhat limited. Object oriented middleware supports load balancing and scalability by determining which server object has the least load for the requested time, but support for replication is still rather limited. Component models improve scalability by using reusable components in the form of containers and by promoting n-tier architectures.

Usability
Object oriented middleware technologies allow programmers to know what is happening but makes it transparent to the operation. The CORBA and COM/DCOM technology have a great disadvantage that they have been too low-level and complicated, and therefore hard to learn, requiring very skilled developers. The CORBA and COM objects have therefore been hard to reuse effectively. The CCM is a great step in the direction of defining and grouping CORBA objects to higher abstractions as components. COM+ also uses containers to isolate software developers from security, persistence, etc. RMI and EJB are easier in use. They work for Java language only and is fine-tuned for it, which can also be seen as a disadvantage.
10.5 Message oriented middleware (MOM)

10.5.1 Introduction

MOM is a specific class of middleware that supports the communication between distributed system components by facilitating message exchange. It is a client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces.

Data is exchanged by message passing and/or message queuing supporting both synchronous and asynchronous interactions between distributed computing processes, or the publish/subscribe paradigm allowing “publisher” processes to push data into the system and “subscriber” processes to consume data.

10.5.2 Functional design aspects

Network Communication

Clients use MOM to send a message to a server across the network. The message can be a notification about an event, or a request for a service execution from a server component. The content of such a message includes the service parameters. The server responds to a client request with a reply-message containing the result of the service execution. The MOM software (kernel) must run on every platform of a network.

Coordination

A strength of MOM is that this paradigm supports asynchronous message delivery. The client continues processing as soon as the middleware has taken the message. Eventually the server will send a message including the result and the client will be able to collect that message at an appropriate time. The weakness, at the same time, is that the implementation of synchronous requests is cumbersome, as the synchronization needs to be implemented manually in the client. This asynchronous communication is not good for time critical applications either. MOM can effectively be combined with RPC for synchronous support. A further strength of MOM is that it supports group communication by distributing the same message to multiple receivers in a transparent way. MOM is also well suited for object oriented systems because it furnishes a conceptual mechanism for peer-to-peer communications between objects.

Reliability

Another weakness of MOM is that it only supports at-least-once reliability. Thus the same message could be delivered more than once. MOM achieves fault-tolerance by implementing message queues that store messages temporarily on persistent storage. The sender writes the message into the message queue and if the receiver is unavailable due to a failure, the message queue retains the message until the receiver is available again. A drawback is that persistent messages are slower than passing the message just through the memory.

In the publish/subscribe MOM, the sender does not specifically address the receiver, but sends out a message to all subscribers. This may cause problems with security.

Heterogeneity

MOM does not support data heterogeneity very well, as the application engineers have to write the code that marshals. In addition MOM implementations are nominally incompatible with other MOM implementations. Using a single implementation of a MOM in a system will most likely result in a dependence on the MOM vendor for maintenance support and future enhancements, which could have a highly negative impact on a system’s flexibility, maintainability, portability, and interoperability.

Scalability

MOMs do not support access transparency very well, because client components use message queues for communication with remote components, while it does not make sense to use queues for local
communication. This lack of access transparency disables migration and replication transparency, which complicates scalability.

Since it sends the same message (event) to all subscribers and isn’t a point-to-point communication, this messaging system offers superior performance and scalability for multi-destination messages.

**Usability**
MOM insulates developers from connectivity concerns. The application developers write to APIs that handle the complexity of the specific interfaces. The administrative and maintenance burden would increase significantly with a large distributed system, especially in a mostly heterogeneous system.

### 10.6 Web Enabled Middleware

#### 10.6.1 Introduction
An alternative approach, capable of avoiding shortcomings of tight coupled object and component middleware techniques, are the concepts of the Web.

DCOM, CORBA, and Java RMI are binary-based complex distributed communication architectures that have tightly coupled object and component models. The tight coupling requires much agreement and shared context among different applications from different organizations in order to be communicable and reliable. Furthermore, the use of distributed computing systems requires high availability, adaptability, and maintainability, and, in order to remain useful, they have to cope with advances in technology, modifications of their operating environment and ever-changing human needs.

There are two ways for an application framework to integrate distributed legacy applications. First, the framework could be written so that it knows how to work with each technology that was used and is able to associate each remote application or object with the appropriate technology (tight coupling). Alternatively, each remote application or object could be wrapped in a common technology. In this case, the client application will only need to know how to work with the wrapper technology (loose coupling). The second solution is much better in that it limits the developer of the client application to only having to worry about a single communication mechanism. However, the problem with the integration technologies already examined is that they are fairly complex to implement and carry a certain degree of overhead. Wrapping any of these technologies with another one of these technologies doubles that overhead.

Web Services have emerged as an important new field, distinguished from conventional distributed computing by its focus on a loosely-coupled, self-describing, modular applications that can be published, located and invoked based on a set of Web-enabled standards. Over the last two years, research and development efforts within the Web Services community have produced protocols, services and tools that address precisely the challenges that arise when seeking to build interoperable integration. And considerable progress has been made on the construction of such an infrastructure.

The overall aim of the Web Services is to enable interoperability, enhance integrated problem solving in adaptive, dynamic and multi-institutional organizations. It focuses on allowing services, which can be composed of resources, systems, programs, applications, raw data, etc., to be published, found, and invoked by other applications at run-time, as well as design-time, language and operating systems independent. During the process, all services can be encapsulated and provided to one another without knowing the underlying implementation. This facilitates the development of distributed applications by providing distribution transparencies to application developers, and shielding the developer from the heterogeneity of operating systems and communication systems.

The exact technical details of Web services can be found in the document [ABB 2003-4]. The most important techniques used in the Web service technology, **XML** and **SOAP** are shortly described below.
10.6.2 eXtensible Markup Language (XML)

XML is a project of the World Wide Web Consortium (W3C), which started in 1996. XML is designed to improve the functionality of the Web by providing more flexible and adaptable information identification. In its simplest form, XML is a powerful medium of data exchange. It not only describes the data and its structure but it’s also open, which allows one to define one’s own tags and format. XML was proposed as a way of extending HTML. XML, like HTML, is text-based, which has an advantage over binary formats because it can be visually inspected and easily understood by humans.

10.6.3 Simple Object Access Protocol (SOAP)

Several wire protocols have been defined to transport XML payloads across a network. The wire protocol that is currently getting the most attention and being integrated into most existing technologies is the Simple Object Access Protocol (SOAP). SOAP was originally conceived by Gopal Kakiwaya at Microsoft. It is an open standard like the HyperText Transport Protocol HTTP, improving heterogeneity. SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC.

The SOAP message structure, SOAP envelope, has two sub elements, SOAP:Header and SOAP:Body. The SOAP:Header element contains information about the transaction. This information is user defined, and its content depends on your application. The SOAP:Body element contains the actual transaction data.

![SOAP message structure](image)

Figure 10-4: SOAP message structure

10.6.4 Functional design aspects

**Network communication**

A service provider publishes a service description in order to be accessible and so that the service requester knows where to find the service provider. The service requester retrieves a service description from the service registry in order to find the required business type. The **UDDI** and the **WSDL** are responsible for these publish and find operations.

**UDDI** (Universal Description, Discovery, and Integration) is an XML-based registry for businesses worldwide to list themselves on the Internet.

**WSDL** (Web Services Description Language) is an XML-based interface definition language that provides a way to catalogue and describe Web Services.

Then the service needs to be invoked. In the bind operation the service requester invokes or initiates an interaction with the service at runtime using the binding details in the service description. Often SOAP is used for invoking Web Services or for exchanging data between the Web Service and the client application,
respectively. SOAP is an XML-based specification for network communication between services. It is used for sending and receiving documents between end-points (RPC or documents). It can be sent over different data transport, e.g. HTTP, HTTPS and JMS.

**Coordination**

Web Services can be both synchronous and asynchronous. Web Services that rely on synchronous communication are usually RPC-oriented. A problem with accessing synchronous Web Services via the Internet is that any required service must be available so the code can be used immediately.

Reliable SOAP messaging is a framework whereby an application running in one Web server instance can asynchronously and reliably invoke a Web service running on another Web server instance. Asynchronous Web Services would allow the required code to be unavailable, without interrupting the business process.

**Reliability/security**

Along with security, reliable asynchronous communications has been one of the gaping holes in today's Web Services architecture. Lack of reliability, due to the inherent nature of using SOAP over protocols such as HTTP, is one of the biggest obstacles to the adoption of Web Services for mission-critical communications between applications and services, such as complex business-to-business transactions or real-time enterprise integration. Since SOAP is a wire protocol, not a system, it does not address security. SOAP's security is currently limited to application-level security and determined by the transport protocol that is uses. For example, HTTPS using Secure Socket Layer (SSL) when HTTP is the transport protocol.

"WS-Reliability" is a standalone specification for reliable SOAP that can provide the missing link to bridge the gap between organizations and help making Web Services a truly enterprise-capable technology for standards based systems integration. WS-Reliability is a specification for an open, reliable SOAP-based asynchronous messaging protocol, which enables reliable communication between Web Services. WS-Reliability includes well-known characteristics of reliable messaging such as guaranteed delivery, duplicate message elimination, and message ordering. Delivery options such as at-least-once, at-most-once, and exactly-once are available. Message expiration and delivery retries are also possible.

**Heterogeneity**

Like other distributed systems, Web Services builds upon two fundamental blocks: a data exchange protocol and a set of metadata for the functional representation of Web applications:

Basic interoperability at the information exchange level is built on top of XML. In the Web Services model, the most basic level of application interaction involves the ability to exchange XML data.

An IDL provides a common, platform independent, functional representation of applications. In the Web Services framework , the IDL representation uses XML Schema as its data definition language, and application functionality is described in terms of the XML messages that can be exchanged.

SOAP by design provides interoperability between heterogeneous operating systems. SOAP is an open standard that is built upon open technologies such as XML and HTTP. It is not vendor-specific and therefore less intimidating to smaller players in the industry. As a result it is being accepted uniformly by the industry, thus improving its chances of being the standard for true distributed interoperability. XML is also open. It is very easy to create XML parsers and SOAP nodes in languages that do not already have them.

**Scalability**

Web Services are scalable and interoperate with each other. This is made possible by the protocols and standards in place to make web services work. The most important are UDDI, WSDL, and SOAP. They are characterized by i.e. "physical scalability", which refers to the ability to handle increasing volumes of usage without requiring drastic reconfiguration, testing and new hardware, and "developmental scalability", which addresses the ability to spread development projects among multiple teams and locations without compromising source integrity, reuse and quality assurance.
Usability
The concept behind Web Services is easy to understand. And there are a lot of free toolkits from vendors like IBM, Microsoft and Apache open source allow developers to quickly create and deploy Web Services.

Interdependency is a weakness. Since the components of a Web Service can again be (smaller) Web Services, the Web Service depends on the proper functioning of the smaller Web Services. When unexpected results are encountered from the Web Services, it will be a difficult task to trace the origin of the problem. A Web Service is essentially a black box for which the user has no way to control or inspect, maybe only by requesting the service and see if it delivers the desired output.

XML and SOAP are especially useful for moving large amounts of data. Unlike object oriented distribution technologies, which must keep objects in memory in order to work with them, XML payloads can be saved to disk. SAX can then be used to parse the XML file. SOAP-based distributed systems are loosely coupled. As a result they are easier to maintain because they can be modified independently of other systems.

10.7 Database Oriented Middlewares
Database oriented middleware is a technological function that enables access to databases of different types, on different platforms and/or in multiple locations. Data oriented middleware is developed for access to databases. The UClA-system environment is not a database environment and therefore database oriented middleware technologies are not further examined.

10.8 Next Generation Middlewares
While middleware products are already successfully employed in industrial practice, they still have several shortcomings, which prevent their use in many application domains. This middleware class addresses the current weaknesses that will influence the next generation of middleware products.

The document [ABB 2003-4] gives a short definition of the subclasses in this class but next generation middleware technologies will not further be examined because it falls beyond the scope of this project.

10.9 Middleware development platforms
By means of completing the middleware muddle this paragraph shows very shortly the main n-tier middleware development platforms.

Microsoft's DNA and .NET
Windows DNA is Microsoft's term for its three-tier architecture based on COM+ and MTS. It includes client technologies like ASPs, server technologies like COM+ and the SQL Server back-end database. DNA uses a tightly coupled messaging architecture, which means that a message's recipient and the sender must both be running DNA-compliant applications. DNA 2000 was intended to replace Windows DNA. The main difference is that DNA 2000 uses a loosely coupled messaging architecture based on BizTalk's XML and SOAP. DNA is still on the drawing board, but it has already been superseded by the .NET architecture, which completely replaces the COMs and the DNAs. .NET supports a loosely coupled architecture for distributed applications and remote access based on BizTalk XML and SOAP technologies.

Sun's J2SE, J2EE and J2ME
J2SE is Sun's premier solution for rapidly developing and deploying mission-critical, enterprise applications. The J2EE technology and its component-based model simplifies enterprise development and deployment. The J2EE platform manages the infrastructure and supports the Web Services to enable development of secure, robust and interoperable business applications. J2EE provides scalability, interoperability, reliability and security. J2ME is a highly optimised Java runtime environment. J2ME technology specifically addresses the vast consumer space, which covers the range of extremely tiny commodities such as smart cards.
10.10 Middleware conclusions and recommendations

This paragraph describes the preferences of middleware classes and technologies for the UCIA. From the wealth of middleware classes and products it is intended to retain some and discard others. Making choices is a difficult task. Object-oriented middleware techniques, like CORBA, RMI, COM/DCOM and XML/SOAP each have their benefits and drawbacks.

From these techniques XML/SOAP is the most heterogeneous, scalable, widely accepted and usable middleware technique and also provides communication using the Web. Because XML is intended for marking up human-readable, textual data, it is by the same token a rather inefficient way of storing information that only ever needs to be read by machines. This is why XML is probably not becoming a replacement for the other object-oriented middleware techniques but is very interesting to cooperate with them for communicating using the Web and for human-readable logging.

Component frameworks, which are evolving very fast at this moment, still make use of these object-oriented techniques to communicate between distributed objects and in addition provide more transparencies and better reuse. Where object oriented middleware techniques had several dependencies (RMI is a Java only solution and COM/DCOM is for use on the Windows platform only) component frameworks address these dependencies and limitations and have the ability to use components from other vendors or use bridging techniques to become interoperable with other object oriented distribution architectures. The differences between component frameworks are much smaller, which makes a well-funded choice more difficult.

10.10.1 XML-SOAP

For both synchronous as well as deferred-synchronous request/reply communication (RPC style read/write interactions) and asynchronous communication, the combination XML-SOAP will be retained as a preference for the used middleware solution:

- It is a loose coupled open standard, easy to use, well adapted to heterogeneous environments and improving usability, interoperability, heterogeneity and maintainability.
- Both the communication with DSM servers as well as cell controllers could be addressed by XML. On the plant network, between the enterprise level and the cell level, XML is already an often used communication technique to exchange data. For communications with the device controllers OPC/XML-DA will provide OPC-compliant solutions with plant-level capabilities and functions.
  o OPC servers can easily be integrated using XML. The OPC/XML-DA specification, which builds on the OPC DA 3.0 specification defining an industry standard for exchanging manufacturing data in secure way using COM/DCOM, is based on XML-SOAP.
  o XML can also be used towards MMS servers, as these devices do not specify an application interface by itself, and towards cell controllers, where the different cell controller interfaces could be wrapped to the standard XML interface.
- XML-SOAP provides the possibility to use the Web for distributed communication.
- XML is text-based and could easily be understood by humans.
- XML-SOAP based distributed systems are loose coupled. All applications only need to know how to wrap to the common XML interface unlike other tight coupled middleware solutions like, CORBA, COM/DCOM and RMI, where the middleware framework should know how to work with each technology that is used and is able to associate each remote application or object with the appropriate technology. A developer can either write an XML parser/wrapper or use an existing parser. Two popular XML parser APIs available today are the Document Object Model (DOM) and the Simple API for XML (SAX).

The Microsoft .NET development platform supports a loosely coupled architecture for implementing distributed architectures using XML and SOAP, and provides the functionality to define (reusable) components and is in addition the underlying communication architecture where the OPC XML specification is based on. For future requirements also Web services could be built to exchange data via the Web.
10.10.2 COM+

COM+ remains a complementary solution applicable for communication between SattStore and DSM servers and cell controllers. Next to XML-SOAP COM+ is a suitable middleware.

- Using COM/DCOM as the main distributed object middleware technology makes the integration to the DSM servers easier than using CORBA or RMI because the OPC DA 2.x and 3.0 specifications are based on COM/DCOM.
- MMS servers do not specify their own application interface. Therefore the same COM/DCOM/COM+ interface could be used towards these servers. Otherwise a simple gateway could be used.
- Using COM+ as a coupling to the cell controllers will be more complicated because using the COM/DCOM object model between more applications demands the ORBs to be installed on all applications communicating. COM+ also supports SOAP and asynchronous component queuing, which could be a solution towards cell controllers.
- COM/DCOM is a proprietary solution owned by Microsoft, which makes it a Windows-only solution. For the UCIA-project this will not be a problem, as only the Windows platform should be supported. For future requirement changes this could form a problem. COM+ already addresses the vendor dependent problem by supporting the SOAP protocol.
- COM+ masks several service implementations to the developer and has the possibility to reuse components.

10.10.3 CORBA 3.0

For the following reasons CORBA 3.0 will not be used as middleware technology:

- CORBA is, just like COM/DCOM, a good technology to integrate diverse programming languages and even better to integrate diverse platforms. When CORBA is to be used to communicate with OPC servers a bridge is needed to enable CORBA to exchange data with the on COM/DCOM-based OPC server.
- CORBA is an industrial standard, endorsed by a consortium of over 800 companies and has been created by standard bodies to provide heterogeneity, which makes it full of compromises.
- CORBA is only a loosely specified standard.

10.10.4 EJB

For the following reasons EJB will not be used as middleware technology:

- EJB is a 100% Java solution.
- When RMI is to be used to communicate with OPC servers a bridge is needed to enable CORBA to exchange data with the on COM/DCOM-based OPC server.

10.10.5 Message Oriented Middleware (MOM) products

MOMs will be discarded as middleware technology because of the following reasons:

- MOMs have a number of capabilities that are highly useful for implementing the asynchronous publish/subscribe paradigm, which are not present in any of the three ORB products. For synchronous communication RPCs still have to be implemented.
- Component frameworks like COM+, EJB and CCM have a MOM product implemented. COM+ uses queued components, CCM the CORBA message service and EJB uses JMS.
11 Conceptual architecture

This chapter discusses the UCIA conceptual architecture represented in primary presentations accompanied with the necessary support information, as described in paragraph 3.8. The UCIA conceptual architecture is based on the architectural requirements (chapter 8) combined with the architectural styles, 3-tier alternatives and middleware preferences obtained in the metaphase (chapter 9 and 10).

11.1 Primary presentation

The following figure shows the primary architectural diagram of the UCIA architecture.

![UCIA architectural diagram and legend](image)

The UCIA-architecture is a 3-tier client/server architecture, which consists of a SattStore tier, a middle tier and a data tier. The architecture has been split up into a DSM server communication part and a cell controller communication part, which use a generic middle tier based on COM+ and XML-SOAP. For the communication with DSM servers the UCIA has a simple generic interface between the SattStore client (called the SattStore DSAM client) and each active or passive DSM server. For the communication with cell controllers the UCIA has a generic interface between the SattStore client (called the SattStore CCAM client) and the different cell controllers.

The simple generic interface to DSM servers is based on an object-oriented communication model called the Data Server Access Model (abbreviated as DSAM). DSAM supports synchronous, deferred synchronous read and write method calls on variables in DSM servers as well as monitoring of variable transitions (asynchronous publish/subscribe communication). The SattStore DSAM client must support the relative simple SattStore DSAM Client API, implemented in a communication interface adapter called DSAM-IO.
DSM servers are addressed by gateways, which implement the DSM Server Framework (also called the DSAM server API), together with DSM server dependent adapters that contain the third party DSM server dependent interface and technology information.

The simple generic interface to cell controllers is based on an object-oriented communication model called the **Cell Controller Access Model** (abbreviated as CCAM). CCAM is derived from the DSAM model. It supports sending byte-strings (cell job) and integers (job identification) together to the cell controller in a synchronous and deferred-synchronous way. Cell controllers can asynchronously send byte-strings (cell status) and integers (status identification) together back to the SattStore CCAM client. The SattStore CCAM client must support the relative simple SattStore CCAM Client API, which is implemented in the CCAM-IO communication interface adapter. Cell controllers are addressed by gateways, which implement the Cell Controller Framework (also called CCAM server API), together with cell controller dependent adapters that contain the third party cell controller dependent interface and technology information.

The coupling between the SattStore processes and DSAM-IO is realized with the help of Signal-IO (See paragraph 6.2.1.1), based on Process-IO. SattStore processes can use Signal-IO to define signals and read, write and monitor these signals. DSAM-IO is responsible for converting Signal-IO signals to SattStore DSAM client requests and the other way around.

The coupling between the SattStore processes and DSAM-IO is done with the help of BCC-IO (See paragraph 6.1.1.2), based on Process-IO. SattStore processes can use BBC-IO to send the cell jobs and identifications to CCAM-IO. This component converts the information to SattStore CCAM client requests and the other way around.

### 11.2 Supporting information

#### 11.2.1 Context

The architectural representation is part of the conceptual view, as described in paragraphs 3.6 and 3.7. It shows a decomposition of the system on the highest level of abstraction, which is interesting for both engineers and management.

#### 11.2.2 Rationale

This paragraph clarifies why particular architectural design decisions have been taken together with possible alternatives and motivations why these alternatives are not suitable or less suitable for the UCIA architecture.

**3-tier client/server architecture**

The UCIA architectural requirements defines that the UCIA is a client/server architecture. SattStore has client functionalities and can address active and passive DSM servers and cell controllers. As discussed in the meta-phase a 3-tier architecture best meets the architectural requirements of the UCIA. By using a 3-tier architecture the following tiers will be distinguished:

**Tier 1: SattStore tier**

The SattStore tier is the one, in which the SattStore WMS client application is implemented. The SattStore client can invoke objects/components of DSM servers and make calls to cell controllers using the middleware, via a SattStore COM/DCOM/XML-SOAP API.

**Tier 2: Middle tier**

Middleware techniques are used to provide platform and language independency of the data servers as well as remoteness transparency. An application server implements services, like configuration, and other functionalities necessary for proper and efficient client/server communication. The middleware technology chosen is COM+/XML-SOAP. The SattStore client can interact with the middle tier server objects but rely on tier 3, the data tier, to get the real data from the data servers.


Tier 3: Data tier

The data tier is used to extract data from the DSM servers or cell controllers and perform actions in a data server or cell controller dependent way. DSM servers or cell controllers are connected to the objects/components of the middle tier, the middleware, via a COM/DCOM/XML-SOAP API.

Simple generic interfaces

Alternate 3, as discussed in paragraph 10.4, has been chosen for the communication with DSM servers as well as for the communication with cell controllers. From SattStore’s point of view simple generic interfaces are sufficient to address all kind of DSM servers and cell controllers. Problems, like the subscription problem as well as the passive and active DSM server problem, are not solved by this alternative and for this reason alternative 3 has been extended in the concept with the DSAM model. This model solves the problems without losing the strength of alternative 3, the simple generic interfaces. The CCAM model is derived from this model and later added. Paragraph 11.2.3 will look at these models.

COM+ /XML-SOAP

As discussed in paragraph 10.10 making middleware choices is a difficult task. Object-oriented middleware techniques, like CORBA, RMI, COM/DCOM and XML/SOAP each have their benefits and drawbacks, but do all meet the architectural requirements. Also discussed are component frameworks, which are evolving very fast at this moment, make also use of object-oriented techniques to communicate between distributed objects and in addition provide more transparencies and better reuse. The differences between component frameworks are much smaller, which makes a well-funded choice more difficult.

As discussed COM+ is a complementary solution applicable for communication between SattStore and DSM servers and cell controllers. Using COM/DCOM as the main distributed object middleware technology makes the integration of the most important DSM servers, OPC servers, easier than using CORBA or RMI because the OPC DA 2.4 and 3.0 specifications are based on COM/DCOM. Therefore the gateways to these servers do not need to convert underlying technologies but only have to address the conversion between standard object invocations and OPC invocations. MMS servers and cell controllers do not specify their own application interface. Therefore the same COM+ interface could be used towards these servers and cell controllers.

COM+ is a proprietary solution owned by Microsoft, which makes it a Windows-only solution. For the UCIA-project this will not be a problem, as only the Windows platform should be supported. Services like security, naming and repository, which are all supported by COM+, can be reused without any difficult conversions. This in contrast with EJB and CORBA, where the supported services should first be converted to the Windows platform. These issues make COM+ the best solution for the Windows environment.

Finally COM+ masks the technologies behind above mentioned service implementations to the developer, which makes the reuse of these components a lot easier and time consuming. Configuration services (like naming services for locations of DSM servers and cell controllers) and logging functionalities could be integrated in the model.

As discussed XML-SOAP is probably not becoming a replacement for other object-oriented middleware techniques. XML will be used for communicating using the Web and is an interesting solution for loose coupled communications with for example the cell controllers. In addition XML will be used for human-readable logging.

11.2.3 Catalogue

This paragraph describes the components of the UCIA architectural diagram in detail as well as their mutual relationships. Most of the used components in the UCIA architecture are determined by the DSAM and CCAM communication models.
11.2.3.1 DSM Server Access Model (DSAM)
The DSM Server Access Model (abbreviated as DSAM) is a client/server communication model that allows one generic SattStore client to read, write and monitor variables (bits, integers and other variables dependent on the DSM server) in active and passive DSM servers in a simple generic way. Next to this it also offers a generic subscription manager. To use the DSAM communication model the UCIA architecture must contain the following components:

- **DSAM-IO (SattStore DSAM client API):** A simple generic interface allows the SattStore DSAM Client to request information and receive responses from active and passive DSM server according to the DSAM model. The SattStore DSAM client API is implemented in the DSAM-IO communication interface adapter.

- **DSM Server Framework (DSF):** The DSF is DSM server independent software and must be implemented each time a third party dependent active or passive DSM server needs a connection with the SattStore DSAM client. The DSF is also called the DSAM server API.

- **DSM Server Adapters (DSA):** The DSA is DSM server dependent software and implements the DSM server dependent interface and underlying communication technology.

**DSAM-IO (SattStore DSAM client API)**
The SattStore client identifies the variables by their DSM server and variable name. The server name identifies the DSM server, which contains the variable class containing the variables. The variable name is the name of the variable itself. The SattStore client can synchronously and deferred-synchronously read and write the variables in the data servers as well as deferred-synchronously subscribe and unsubscribe to transitions on variables. Variable transitions are asynchronously called back to SattStore. For each subscribed variable it is possible to specify which variable value change (called transition) is monitored (e.g. 0-1 bit-transition).

Each synchronous action implies a specific response from the data server. The SattStore client waits for this response (and can specify a time out). A synchronous read call returns the value of the requested variable together with status-information. A synchronous write call does not return data, but only status-information. Status-information contains transfer errors, data server errors, etc.

Each deferred-synchronous action also implies a specific response. The SattStore client doesn't wait for this response and continues processing. To use deferred synchronous communication a call-back mechanism must be used between the DSM server and the SattStore client. On the basis of this call-back mechanism DSM servers are able to send data and status information back to the SattStore client. To use a call-back mechanism the SattStore client must implement data listener interfaces which handle the call-back information.

A deferred-synchronous read call returns the value of the requested variable together with status-information on the specified data listener. A deferred-synchronous write call does not return data, but only returns status-information on the specified data listener.

A deferred-synchronous subscribe or unsubscribe does not return data, but only returns status-information on the specified data listener. If a specific subscribed variable transition or an error appears this is asynchronously published to the matching data listener of the SattStore client together with status-information.

All read, write, subscribe and unsubscribe calls as well as the call-back and data listener functionalities are performed according to standard generic methods and method calls. A standard SattStore DSAM client API defines the standard method calls the SattStore client can use to access variables in the DSM servers and defines data listeners for call-backs. Call-back functionalities are supported by COM/DCOM.
The DSAM-IO communication interface adapter implements the SattStore DSAM client API and has thus a coupling with SattStore processes and with the middle tier. For the connection with the middle tier DSAM-IO contains a SattStore COM/DCOM/XML-SOAP API.

**DSM Server Framework (DSF) and DSM Server Adapters (DSA)**

From the SattStore client's point of view all DSM servers can be addressed with the same set of method calls and support the same sort of functionalities. In reality the different types of supported DSM servers support different types of variable read and write calls (as discussed in chapter 9). Also subscribe and unsubscribe calls as well as the way in which the DSM servers implement call-backs and data-listener functionalities is different per type DSM server. When using passive DSM servers there is even no publish/subscribe mechanism present.

The generic DSM Server Framework (DSF) solves these problems together with third party DSM server dependent DSM Server Adapters (DSAs). The DSF is DSM server independent software and must be implemented each time a third party dependent active or passive DSM server needs a connection with the SattStore client. A DSA is used to implement the third party DSM server dependent methods and method calls relevant for the DSF (the DSA implements DSM server dependent client functionalities). Each type DSM server has its own DSA connected to the generic DSF. Figure 11-2 shows the DSF conceptual overview.

The generic DSF has the following functionalities:

1. Incoming synchronous read and write calls from the SattStore client are received by the DSF and coupled to DSM server specific read and write calls implemented by the DSA. Responses are directly sent back to the SattStore client. The read/write interface provided by the DSM server is thus directly implemented.

2. Incoming deferred synchronous read and write calls from the SattStore client are received by the DSF and coupled to DSM server specific read and write calls implemented by the DSA. Responses are sent back to the SattStore client by means of a call-back mechanism on the basis of an identified data-listener. The read/write interface provided by the DSM server is thus directly implemented.

3. Incoming subscriptions from the SattStore client are always stored in the generic subscription manager. If an unsubscribe call is received the relevant subscription is removed from the subscription manager.
4. If the variable to be monitored in the incoming subscription is part of an active server the subscription manager executes the DSM server specific deferred synchronous subscribe (or unsubscribe) calls as well as activate or de-activate the DSM server specific listener (which detects transitions on the subscribed variable). Status responses are sent back to the SattStore client by means of a call-back mechanism on the basis of an identified data-listener. The subscribe/unsubscribe interface provided by the DSM server is thus directly implemented.

5. When the active DSM server listener detects a variable transition the DSF forwarding mechanism is invoked. This mechanism forwards/publishes the transition to all subscribed data-listeners by means of a call-back on the basis of the generic subscription manager.

6. If the variable to be monitored in the incoming subscription is part of a passive DSM server an internal polling mechanism must be used to detect transitions on the variable. The subscription manager exchanges commands with the polling mechanism to indicate which subscribed variable transitions must be polled. The polling mechanism polls for the variable transitions which consist of a sequence of synchronous read calls to the DSM server and interpreting of the responded data. When a variable transition is detected the transition is sent to the data listeners by means of a call-back mechanism on the basis of the generic subscription manager.

The DSF makes the SattStore DSAM client completely transparent for the used type DSM servers and let each connected DSM servers acts like a generic DSM servers. From the SattStore client’s point of view there is thus always a generic coupling on each third party dependent DSM server, while in reality the DSF and the matching DSA are addressed. Notice that if the DSM server directly supports the generic SattStore client calls a DSA is not needed. This can be the case if a data server implementation is adjusted to the DSAM.

The DSF has on the one hand a coupling with the middle tier and implements the necessary COM/DCOM/XML-SOAP APIs and on the other hand a coupling with the DSAs according to the mentioned DSF rules. The DSAs are coupled to the DSM servers and implement DSM server dependent logic and are thus also coupled to the DSF.

11.2.3.2 Cell Controller Access Model (CCAM)

The Cell Controller Access Model (abbreviated as CCAM) is a client/server communication model that allows one generic SattStore client to send byte strings (cell jobs) together with integers (job identifications) to cell controllers and receive byte strings (cell statuses) together with integers (status identifications) from the cell controllers on a fixed data listener. To use the CCAM communication model the UCSIA architecture must contain the following components:

- **CCAM-IO (SattStore CCAM client API):** A simple generic interface allows the SattStore CCAM Client to send and receive information from cell controllers according to the CCAM model. The SattStore CCAM client API is implemented in the CCAM-IO communication interface adapter.
- **Cell Controller Framework (CCF):** The CCF is cell controller independent software and must be implemented each time a third party dependent cell controllers needs a connection with the SattStore CCAM client. The CCF is also called the CCAM server API.
- **Cell Controller Adapters (CCA):** The CCA is cell controller dependent software and implements the cell controller dependent interface and underlying communication technology.

The CCAM with its components are derived from and based on the discussed DSAM. Because cell controllers do not support subscription, the model only supports the sending of data from SattStor to cell controllers and the sending of data from the cell controllers to SattStore. This latter as a response or spontaneously.

**CCAM-IO (SattStore CCAM client API)**

The SattStore client identifies the cell controllers by a cell controller name. The SattStore client can synchronously and deferred-synchronously send byte-string (cell job) and integer (job identification)
information together to the cell controller. Always a status is sent back to the SattStore CCAM client, directly (synchronous communication) or by means of a call back mechanism on a data-listener (deferred synchronous communication).

A cell controller can asynchronously send (publish) byte-string (cell status) and integer (status identification) information together back to the matching (fixed predefined) data-listener of the SattStore CCAM client together with transmission status-information.

All send, call-back and data listener functionalities are performed according to standard generic methods and method calls. A standard SattStore CCAM client API defines the standard method calls the SattStore client can use to send and receive the byte-strings (cell job) and integers (job identification) together and defines data listeners for call-backs. Call-back functionalities are supported by COM/DCOM.

The CCAM-IO communication interface adapter implements the SattStore CCAM client API and has thus a coupling with SattStore processes and with the middle tier. For the connection with the middle tier CCAM-IO contains a SattStore COM/DCOM/XML-SOAP API.

**Cell Controller Framework (CCF) and Cell Controller Adapters (CCA)**

From the SattStore client's point of view all cell controllers can be addressed with the same set of method calls and support the same sort of functionalities. In reality the different types of cell controllers support different types of send and receive functionalities.

The generic Cell controller Framework (CCF) solves these problems together with third party cell controller dependent Cell Controller Adapters (CCAs). The CCF is cell controller independent software and must be implemented each time a third party dependent cell controller needs a connection with the SattStore CCAM client. A CCA is used to implement the third party cell controller dependent methods and method calls relevant for the CCF (the CCA implements cell controller dependent client functionalities). Each type cell controller has its own CCA connected to the generic CCF. Figure 11-3 shows the CCF conceptual overview.

![Figure 11-3: CCF conceptual overview](image)

The generic CCF has the following functionalities:

1. Incoming synchronous send calls (containing a byte string and an integer field) from the SattStore CCAM client are received by the CCF and coupled to cell controllers specific send commands (containing a byte string and an integer field) implemented by the CCA. Status responses are directly sent back to the SattStore CCAM client.
2. Incoming deferred synchronous send calls (containing a byte string and an integer field) from the SattStore CCAM client are received by the CCF and coupled to cell controllers specific send commands implemented by the CCA. Status responses are sent back to the SattStore CCAM client by means of a call-back mechanism on the basis of an identified data-listener.

3. When the cell controller listener (which is always active) detects that the cell controller like to send information, the CCF forwarding mechanism is invoked. This mechanism forwards/publishes the information (containing a byte string and an integer field) to a fixed predefined data listener in the SattStore CCAM client.

The CCF makes the SattStore CCAM client completely transparent for the used type cell controllers and let each connected cell controller act like a generic cell controller. From the SattStore client's point of view there is thus always a generic coupling on each third party cell controller, while in reality the CCF and the matching CCA are addressed. Notice that if the cell controller directly supports the generic SattStore client calls a CCA is not needed. This can be the case if a cell controller implementation is adjusted to the CCAM.

The CCF has on the one hand a coupling with the middle tier and implements the necessary COM/DCOM/XML-SOAP APIs and on the other hand a coupling with the CCAs according to the mentioned CCF rules. The CCAs are coupled to the cell controllers and implement cell controller dependent logic and are thus also coupled to the CCF.
12 Conclusions and recommendations

This chapter contains the conclusions and recommendations of the UClA project after finishing the conceptual architecture. The UClA project is mainly a theoretical research project and has laid the foundation of a new communication concept for the communication between SattStore and device controllers and SattStore and cell controllers. Because it is only the foundation the ideas must be further developed and worked out in the next phases of the conceptual phase, the logical phase and finally the execution phase. On the basis of this graduation report engineers must be able to continue the system structuring and have a complete theoretical backbone and holds to do this.

12.1 Conclusions

The paragraph shows an enumeration of the conclusions of the UClA project. Conclusions are results of performed and finished researches as well as choices made during the UClA system structuring till the logical architecture.

- To gather knowledge about software architectures, how the guidelines and the system's concept could be well documented and how to be sure that the concept satisfies all system requirements and objectives a study about how to build up and document software architectures has been performed and finished. Four most well known architecture view models have been shortly described and for the UClA is chosen for the Bredemeyer view model, which is part of the Bredemeyer decision layered model. From the technical process of this Bredemeyer decision layered model the architectural requirements phase and the meta architecture and conceptual architecture of the architectural specification have been finished.

- A general study to distributed computing has been performed and finished. There are two implementation paradigms discussed: protocol-based implementation and object and component-based implementation. Protocol-based implementation is based on OSI reference model layers, PDU exchange and services. Object and component-based implementation is based on (distributed) objects and components and object- and component oriented middleware technologies. Next to this also functional and non-functional design aspects of distributed systems have been studied and documented.

- The SattStore transport environment has been studied on the basis of CIM and SattMate. On the basis of the CIM pyramid it has been proven that SattStore is an area controller application that communicates with industrial crane cell controller on the cell level (exchange of cell jobs / statuses) and industrial conveyor belt PLC on the control level (exchange of device jobs / statuses). Within SattStore always the crane transport sub-system with the crane administrator and a crane communication interface adapter is used to address industrial crane cell controllers and always the conveyor transport sub-system with the conveyor administrator and a conveyor communication interface adapter is used to address industrial conveyor PLCs.

- The present (always vendor-dependent) communication situation of SattStore has been studied and documented in the case of communication with crane cell controllers and conveyor PLCs. The exchanged crane cell jobs / statuses with crane cell controllers are byte string data types and often accompanied with a job or status identification integer field. The other protocol PCI fields are different each time and there are thus less generic elements in the protocols and crane communication interface adapters except the BCC-protocol records used between SattStore and the communication interface adapters. The exchanged PLC jobs / statuses with the PLCs consists of bit and integer communication. The used protocols to the PLCs, the conveyor communication interface adapters as well as the coupling SattStore and the communication interface adapters (Signal-IO) contain a lot of generic elements.

- A study to industrial protocols and standards have been performed and finished to find generic vendor-independent ways to communicate with cell controllers and device controllers. Criteria were advantages and disadvantages of using and implementing the solutions, market support of the different solutions
Conclusions and recommendations

The solutions have been gathered, explained, compared and considered according to the mentioned viewpoints. OPC-DA (2.x/3.0 and XML) and MMS data-server solutions came out as best meeting the criteria.

With the knowledge of the SattStore communication environment, the present situation and the industrial protocol and standard research together with additional requirements defined by ABB Logistic Systems the architectural requirements have been defined containing the UCIA-system environment, the UCIA functional and the UCIA non-functional requirements. Main conclusion is the leaving of the SattMate sub-system approach, because this construction holds up effective communication. Chosen is for an architecture that addresses device controllers and cell controllers from a SattStore process whether it is an administrator process or not. Next to this it has been proven that the UCIA must be a client-server architecture in both the communication with active and passive DSM servers (defined model for data servers that connect to device controllers) and cell controllers.

The meta phase of the architectural specification has been performed and finished on the basis of all mentioned researches and results. In this phase a strong preference has gone out to a 3-tier UCIA with a simple generic interface towards SattStore with which SattStore can address all kind of active and passive DSM servers as well as address all kind of cell controllers. On the generic middle tier a middleware technique must be used to fulfill all architectural requirements, like logging, transparencies, reliabilities, etc. For this reason an extensive middleware research has been performed and finished. This research has given a clear preference to the use of COM+ in combination with XML-SOAP on the middle tier.

With the knowledge of the architectural requirements and the meta phase a UCIA concept has been defined and documented. All choices have been substantiated in a rationale and an element description. The UCIA architecture is a 3-tier client/server architecture, which consists of a SattStore tier, a middle tier and a data tier. The architecture has been split up into a DSM server communication part and a cell controller communication part with use a generic middle tier based on COM+ and XML-SOAP. Important models used are the Data Server Access Model (DSAM) for a generic communication with DSM servers and the Cell Controller Access Model (CCAM) for a generic communication with cell controllers. The in the present situation used BCC-records and Signal-IO records could be reused.

12.2 Recommendations

The paragraph shows an enumeration of the recommendations after finishing the UCIA project. As discussed the UCIA project provides a foundation for the logical and execution phase of the UCIA system structuring. This list of recommendations provide therefore recommendations for engineers that continue the system structuring and next to that a couple of general recommendations for ABB Logistic Systems

The Data Server Access model (DSAM) and its components must be further worked out and specified. This implies the following structuring steps:

1) First of all the generic interface must be defined on basis of COM/DCOM. From the conceptual phase it is known what kind of generic calls must be present (synchronous/deferred synchronous read and write calls, deferred synchronous subscribe and unsubscribe calls) as well as how responses must be given back to SattStore (data listener reference object per deferred synchronous call and asynchronous push call). By defining simple generic commands (simple in/out-parameters) the interface between SattStore and all kind of active and passive DSM servers is established.

2) DSAM-IO must be created. DSAM-IO must make the right conversions between Signal-IO read, write and event declaration/event cancellation records and the in (1) mentioned simple generic read, write and subscribe/unsubscribe calls. Answers back from the DSM servers are converted to Signal-IO notifications.
Conclusions and recommendations

(3) The DSM server Framework (DSF) must be created. On the basis of the in the conceptual architecture phase described functionalities and the in (1) defined simple generic interface the DSF can be built up. Interesting research areas are the subscription manager, the polling mechanism and the forwarding mechanism.

(4) On the basis of the developed DSF, a DSM server adapters (DSA) must be created and implemented for each connected active or passive DSM server type. The DSM server read, write, optional subscribe/unsubscribe calls and the optional active DSM server listener's push calls must be correctly connected to the DSF according to the formulated software rules in (3). For the creation of the OPC-DA 2.x/3.0, OPC-XML-DA and MMS DSA the document Industrial protocols and standards [ABB 2003-3] could be very helpful.

- The Cell Controller Access model (CCAM) and its components must be further worked out and specified. This implies the same sort of steps as discussed above. A simple generic interface must be defined on the basis of COM/DCOM. From the conceptual phase it is known what kind of generic calls must be present (synchronous/deferred synchronous send calls containing a byte string together with an integer) as well as how responses must be given back to SattStore (data listener reference object per deferred synchronous call and fixed pre-defined data listener reference object per asynchronous push call). Next to this CCAM-IO must be created that convert BCC protocol records to request calls and the other way around as well as the CCF and matching CCAs must be created and implemented.

- The CCAM is in the actual conceptual architecture not capable of using dynamic data listeners for the asynchronous push calls from the cell controllers. By extending the CCF it would maybe possible to create data listeners for asynchronous information from the cell controller in a dynamic way. Maybe by using a sort of asynchronous call-back subscription manager.

- The marshalling of the two models, the DSAM and the CCAM, could be examined because the CCAM is derived from the DSAM and both frameworks do contain a lot of generic elements.

- The COM+ middle tier must be exactly defined and created. This implies the integration of COM/DCOM APIs in DSAM-IO and CCAM-IO as well as define and implement the naming service for resolving DSM server and cell controller locations. Also logging functionalities must be implemented in the COM+ middle tier as well as all other in the architectural requirement mentioned functions. Next to this the support for XML-SOAP is used to extend COM with web functionalities as well as technology transparency functions and for this reason the XML-SOAP API could be integrated in DSAM-IO, CCAM-IO, DSF and CCF. The document Middleware [ABB 2003-4] could be very helpful to discover more information about COM and XML-SOAP.

- Because of the dominance of the OPC-DA specification, a simplified not totally generic solution could be the use of an OPC-IO communication interface adapter instead of DSAM-IO, that converts Signal-IO to COM-based OPC-DA calls and the other way around. The advantages and disadvantages as well as the feasibility of this OPC-DA solution could be examined.

- The knowledge of chapter 3 could be very helpful in the system structuring and as discussed the architecting process is an iterative process, with multiple cycles through requirements, structuring and validation.

- Because the industrial protocol and standard research (chapter 7 and reference document [ABB 2003-3]) as well as the middleware research (chapter 10 and reference document [ABB 2003-4]) are mostly general, they could be used for all kinds of other purposes.
Terminology

Active DSM servers
Data servers which support subscriptions on variable-transitions and are capable of notifying the interested clients on a specific subscribed variable-transition.

Actuator
An actuator is a device, which responds to an input electrical or optical signal (descended from a control device) by generating a functionally related output in the form of an output quantity.

Agent
A piece of software which acts to accomplish tasks on behalf of its user

API (Application program interface)
A formalized set of software calls and routines that can be referenced by an application program in order to access supporting system or network services

Application protocols
Protocol used at the application layer of the OSI/internet reference model. An application protocol defines the rules governing communication between network applications. These rules describe an application’s information and how it is used by the applications sending and receiving the information.

Application-support protocols
Protocol used on the layers between the lower and application layers of the OSI reference model. These application-support protocols support the application by defining session and/or presentation protocols.

Architecture
A top-down description of the structure of the system.

Architecture description
Collection of documents that together describe the architecture.

Area controller interface
Interface between an area level entity (area controller application) and a host system.

Asynchronous traffic
A sensor detecting a critical situation uses this function to notify the event to the device controller, which, in turn, will undertake the appropriate actions.

Automatic assignments
Assignments under control of the SattMate system ensuring safe restart

Availability
The degree to which a system or component is operational and accessible when required for use

Bean
A Bean, also knows as JavaBean, is loosely defined as a reusable software component that can be manipulated visually in a builder tool.

Cell controller
A cell controller is an application that supervises and monitors the resources in a particular industrial manufacturing cell. Cell controllers do in most of the cases offer a comprehensible interface towards supervision systems (systems on top of the cell controller) and resources in the cell. A cell controller allows
the following main functions: Integration of the cell resources, control and supervision of the cell, generation, storage and exchange of the different commands to the different resources in the cell and the reaction to fault conditions in the cell.

**Cell Controller Access Model (abbreviated as CCAM)**
Client/server communication model that allows one generic SattStore client to send byte strings (cell jobs) together with integers (job identifications) to cell controllers and receive byte strings (cell statuses) together with integers (status identifications) from the cell controllers on a fixed data listener.

**Cell controller interface**
Interface between a cell level entity (cell controller application) and a host system.

**CCAM-IO**
SattStore communication interface adapter that converts BBC protocol records to SattStore CCAM client calls.

**Cell Controller Framework (CCF)**
The CCF is cell controller independent software and must be implemented each time a third party dependent cell controllers needs a connection with the SattStore CCAM client. The CCF is also called the CCAM server API.

**Cell Controller Adapters (CCA)**
The CCA is cell controller dependent software and implements the cell controller dependent interface and underlying communication technology.

**Cell job**
Assignment command for a specific cell controller.

**CGI**
CGI, or Common Gateway Interface, is the standard programming interface between web servers and external programs.

**CIM**
CIM (Computer Integrated Manufacturing) organized the different activities or functional levels of a manufacturing company, from the manufacturing process up to the enterprise direction, comprising sales, purchasing, design engineering, industrial engineering, production planning, etc. Can be represented as a pyramid.

**Client**
Systems that run application processes to access the servers over the network, requesting services.

**CNC**
CNC (Computer Numerical Control) allows the control of motion in an accurate and programmable manner through use of a dedicated computer within a numerical control unit, with a capability of local data input such that machine tools are freed from the need for "hard-wired" controllers.

**COM**
Component Object Model. Microsoft's framework for developing and supporting objects that can be accessed by any compliant application.

**Compatibility**
The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment.
Terminology

Complexity
The degree to which a system or component has a design or implementation that is difficult to understand and verify.

Component
A unit of distribution and reusable program building block that can be combined with other components to form an application.

Component model
A component model defines the basic architecture of a component, specifying the structure of its interfaces and the mechanisms by which it interacts with its container and with other components. The component model provides guidelines to create and implement components that can work together to form a larger application. Application builders can combine components from different developers or different vendors to construct an application.

Conceptual architecture
The purpose of the conceptual architecture is to direct attention at an appropriate decomposition of the system without delving into details.

Concern
Interest of stakeholder.

Concurrency
Concurrency is the ability to process multiple tasks at the same time.

Configurability
The ratio of behaviour variation to the effort of defining variable properties.

Connectionless service
No negotiations about an agreement between parties (OSI/internet model layers). Each data exchange is considered to be independent of other data exchanges.

Connection-oriented service
Both service user and provider(s) (OSI/internet model layers) are involved with the party-agreement. The agreement is only achieved if all parties agree with the data-exchange conditions. Phases of connection-oriented services: establishment, data transfer and release.

Consignee
Recipient of an output order.

Container
Components execute within a construct called a container. A container provides an application context for one or more components and provides management and control services for the components. In practical terms, a container provides an operating system process or thread in which to execute the component.

Control networks
Networks typically used as backbones for communication between I/O systems, device controllers, cell controller, operator stations, and sometimes even upper CIM level applications.

Conveyor PLC interface
Interface between SattStore and a type conveyor PLC (conveyor belt used in a warehouse).

CORBA
Common Object Request Broker Architecture. A standard architecture, defined by the OMG, for communication between distributed objects.
Crane cell controller interface
Interface between SattStore and a type crane cell controller (industrial crane used in a warehouse).

Cyclic data exchange
At every cycle a device controller sends the output signals to the actuators and reads the input signals from the sensors.

Database
A database is a collection of data that is organized so that its contents can easily be accessed, managed, and updated.

Database manager system
A database management system (DBMS), sometimes just called a database manager, is a program that lets one or more computer users create and access data in a database.

DCE
Distributed Computing Environment

DCOM
Distributed Component Object Model. Extensions to COM which allow remote access to COM components.

Device controller interface
Interface between device controller and a host system.

Device job
Assignment command for a specific device controller.

Device networks
Networks that encompass protocols initially designed to support analog continuous I/O with status information.

Device profile
Device profiles are used in automation technology to define specific properties and behavior for devices, device families or entire systems in such a way that this determines their largely unique characterization. Only devices and systems with the same vendor-independent profile provide "interoperability" on a fieldbus, thus fully exploiting the advantages of fieldbuses for the end-user.

Directory service
Directory services provide a way of looking up names just like the naming service, but goes further by providing a general facility for looking things up.

Dispatcher
Generic main element in the SattMate system and supervises the assignment sequences and sub-system handling.

Distributed computing
In general, distributed computing is any computing that involves multiple computers remote from each other that each has a role in a computation problem or information processing.

DTD
A document type definition (DTD) is a specific definition that follows the rules of the Standard Generalized Markup Language (SGML)
Terminology

Data Server Model (DSM)
Model for data servers, based on OPC-DA and MMS, consisting of demands a data server needs to satisfy to be a DSM server. DSM servers can be used in the DSM Server Access Model (DSAM).

DSM Server Access Model (DSAM)
Client/server communication model that allows one generic SattStore client to read, write and monitor variables (bits, integers and other variables dependent on the DSM server) in active and passive DSM servers in a simple generic way.

DSAM-IO
SattStore communication interface adapter that converts Signal-IO records to SattStore DSAM client calls.

DSM Server Framework (DSF)
The DSF is DSM server independent software and must be implemented each time a third party dependent active or passive DSM server needs a connection with the SattStore DSAM client. The DSF is also called the DSAM server API.

DSM Server Adapters (DSA)
The DSA is DSM server dependent software and implements the DSM server dependent interface and underlying communication technology.

Efficiency
The degree to which a system or component performs its designated functions with minimum consumption of resources (CPU, Memory, I/O, Peripherals, Networks)

ebXML
Electronic Business XML is a project to use XML to standardize the secure exchange of business data.

Efficiency
The degree to which a system or component performs its designated functions with minimum consumption of resources (CPU, Memory, I/O, Peripherals, Networks)

EJB
Enterprise Java Beans. An API specification of a framework for developing and executing distributed applications written in the Java Programming Language.

Encapsulation
The process in which an application passes its information to the layer below, which adds its own protocol information and passes it down to the next layer. Each layer’s information accumulates on the way down the stack. On the receiving end, each layer removes its information, interprets it, and responds according to the protocol. It then passes the remaining data up to the next layer. The application layer, at the top, receives only the application layer information.

Environment
Environment in which the system operates and that influences the system.

Executive function
Translates the real devices and objects into the virtual ones defined by the MMS VMD model when communicating with MMS client applications and devices.

Fieldbus
Fieldbus is a generic-term which describes a digital communications network which is used in industry to replace the existing 4 - 20mA analogue signal. The network is a digital, bi-directional, multidrop, serial bus, communications network used to link isolated field devices, such as controllers, transducers, actuators and sensors. Can be used on interface area 2 and 3.
Field data
Data intended for a specific field sensors.

Flexibility
The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

FTP
File Transfer Protocol is a protocol used for transmission of files over a TCP/IP network. The protocol contains functions such as: converting character sets, copying files and logging on to the network. FTP is intended to handle binary files, which means that all kinds of files can be transmitted without coding and decoding.

GIOP
General Inter-ORB Protocol. ORBs communicate through a high-level standard protocol (GIOP), which is independent of the underlying protocol stack.

GUI
Graphical User Interface. The application, or part of it which is responsible for the interaction with the user.

Handling unit
A handling unit is a vehicle or other mechanical equipment that moves a load (e.g. goods) from one position to another. Different handling unit types are cranes, conveyors belts, and forklift trucks equipped with an onboard terminal.

Heterogeneity
The degree in which components are from different vendors.

Host system
System hierarchically on top of another system. A system can be an application or device.

HTTP
The Hypertext Transfer Protocol (HTTP) is the set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. Relative to the TCP/IP suite of protocols (which are the basis for information exchange on the Internet), HTTP is an application protocol.

IDL
IDL (interface definition language) is a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language. In distributed object technology, it's important that new objects be able to be sent to any platform environment and discover how to run in that environment. An Object Request Broker (ORB) is an example of a program that would use an interface definition language to "broker" communication between one object program and another one.

IIOP
Different ORBs can communicate using the Internet Inter ORB Protocol (IIOP). IIOP describes how GIOP messages should use the Transmission Control Protocol and Internet Protocol (TCP/IP) stacks.

Industrial Ethernet
Fieldbus based on TCP/IP-Ethernet. Can be used on interface area 2.

Internet Model
Layered protocol architecture (protocol stack) addressed to the Internet environment and the matching protocols (TCP/IP, Ethernet, HTTP, etc)
Interoperability
The ability of two or more systems or components to exchange information and to use the information that has been exchanged

Introspection
This mechanism allows components to publish the operations and properties, that they support, and the mechanism by which these are discovered by the developer tool.

J2EE
J2EE technology and its component based model simplifies enterprise development and deployment.

J2ME
A highly optimised Java runtime environment.

J2SE
J2SE is the specification for the Java Platform, Standard Edition. It consists of a standard API. JRE and Java 2 SDK (JDK) are used to implement the specification.

JDK
Java Development Kit, also called Java 2 Software Development Kit (SDK). JDK provides a development environment for Java applications.

JMS
The Java Messaging Service API supports asynchronous communications through various messaging systems, such as reliable queuing and publish-and-subscribe services.

JVM
A Java Virtual Machine is the program that runs programs that are written in the Java language.

Load (in terms of SattStore)
A load contains a number of part loads. It is identified by a unique load identity, which is printed on a load label.

Load balancing
Load balancing is a technique for distributing service requests evenly among servers that offer the same service. This avoids overburdening some servers while leaving others idle or infrequently used. Before sending a request to a service routine, the system identifies all servers capable of handling the request and selects the one most appropriate for maintaining a balanced load across all the servers in the configuration.

Logical architecture
The logical architecture adds precision, providing a detailed "blueprint" from which component developers and component users can work in relative independence.

Lower layer protocols
Protocols used at the four lower layers of the OSI reference model. They control the transport of the application data up to the physical interface.

Marshaling and unmarshaling
Marshaling is the process of gathering data from one or more applications putting the data pieces into a message buffer, and organizing or converting the data into a format that is prescribed for a particular receiver or programming interface. When the client receives the object, the elements are unpackaged back into an object or request (unmarshaling).
Maintainability
The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment.

Message passing
Chain of data exchange or information flow between the different levels of the industrial manufacturing environment

Meta-architecture
The meta-architecture, through style, patterns of composition or interaction, principles, and philosophy, rules certain structural choices out, and guides selection decisions and trade-offs among others.

Meta data
Metadata is a definition or description of data.

Meta language
Meta language is a definition or description of language

Middleware
Connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network, providing platform and language independency and location and technology transparency.

MMS
Manufacturing Message Specification. An internationally standardized application layer protocol for exchanging real-time data and supervisory control information between networked devices and/or other computer applications.

MMS client
An application, which accesses the MMS objects of a server via the network and thus makes use of the available physical resources provided by the MMS server.

MMS provider
That part of the communication system that provides the user with the MMS services.

MMS server
An application, which maps onto MMS object the physical resources, which may be accessed from the network.

MMS user
An application program that uses the MMS services to communicate with the MMS user of a target system, for example a PLC system.

Model
Piece of (structured) information in a particular representation/language.

MOM
Message Oriented middleware. A general term for middlewares, which are based on messages rather than on remote object invocation.

Naming service
The naming services like DNS (Domain Naming Service) list the readable names associated with the numbers (IP address and port number).
Object
In object-oriented programming, a single software entity that consists of both data and procedures to manipulate that data.

OLE
Object Linking and Embedding. A set of COM-based technologies now normally referred to as ActiveX.

OMA
Object Management Architecture (OMA) provides interoperability between object oriented distributed components even in highly heterogeneous networks.

OMG
Object Management Group. A consortium that aims to define a standard framework for distributed, Object-oriented programming. The OMG is responsible for the CORBA specification.

OPC
OLE for Process Control. An industry standard created with the collaboration of a number a leading worldwide automation hardware suppliers. OPC defines a standard common interface for communicating with diverse process-control devices.

OPC client
Client side of the OPC software bus. On the basis of a well-defined OPC client API the client knows how to address OPC servers.

OPC server
Server side of the OPC software bus. OPC servers are OPC data source which encapsulate the IO driver logic and provide the data in a single format on the basis of a well-defined OPC server API.

OPC Data Access (OPC-DA)
The original standard of OPC. This specification tells how to move real time data from PLC’s, DCS’s and other control devices to applications like HMI (COM-based).

OPC Data Access XML (OPC XML-DA)
This specification is being developed to provide flexible, consistent rules and formats for exposing plant floor data using XML, leveraging the work done by Microsoft and others on BizTalk, SOAP, and other XML frameworks.

Operability
The ease of operating the software

ORB
Object Request Broker. An ORB is a middleware component which acts as an intermediary between a client and a distributed object. The ORB is responsible for delivering messages between the client and object across a network.

ORBlets
Java applets that invoke the services of remote CORBA objects

Order job
Assignment command for a specific area controller.

ORB
Object Request Broker. An ORB is a Middleware component which acts as an intermediary between a client and a distributed object. The ORB is responsible for delivering messages between the client and object across a network.
OSI reference model
The Open System Interconnection (OSI) reference model is a layered protocol architecture dividing the communication sub-system into small well-defined pieces. Each layer adds to the services provided by the lower layers in such a manner that the highest layer is provided a full set of services to manage communications and run distributed applications.

Passive DSM servers
Data servers which do not support subscriptions on variable-transitions and are not capable of notifying the interested clients on a specific subscribed variable-transition.

PCI
Protocol Control Information (PCI) is header information of a PDU containing fields that are coupled to the stored control information of a protocol.

PDU
Protocol Data Units (PDU’s) are data block sent between peer layers.

Peer
A device that is capable of both initiating communications and accepting communications initiated elsewhere.

Peer-to-Peer
Any relationship in which multiple, autonomous devices interact as equals.

Plant networks
Networks used for normal office programs, such as text processing, spreadsheets, databases, and e-mail. Next to this in a manufacturing environment this network is used to interconnect ERP applications, area controller applications and cell controller applications.

PLC
PLC (Programmable Logic Controller) a class of industrially hardened devices that provides hardware interface for input sensors and output actuators. PLCs can be programmed using relay ladder logic to control the outputs based on input conditions and / or algorithms.

POA
The Portable Object Adapter (POA) the standard object adapter and is part of the CORBA 2.3 specification. It is used together with IIOP to allow ORB implementations from different vendors to communicate. A single ORB may have multiple POAs each tailored to a certain task using parameterized characteristics.

Portability
The ease with which a system or component can be transferred from one hardware or software environment to another.

Protocol Entity
The code that controls the operation of a protocol layer. It performs the peer-to-peer procedures, encapsulate frames, mux/demux traffic, etc.

Protocol stack
Protocols structured together to form a layered design.

QoS
Quality Of Service is the idea that transmission rates, error rates, and other characteristics can be measured, improved, and, to some extent, guaranteed in advance.
Rationale
Explanation/motivation for the architecture description.

Real-time
A communication system responds in a timely predictable way to unpredictable external stimuli arrivals.

Reliability
The ability of a system or component to perform its required functions under stated conditions for a specified period of time.

Replication
Replication is the real-time backup of data stored in the database to an alternate store.

Reusability
The degree to which a software module or other work product can be used in more than one computing program or software system.

RMI
Remote Method Invocation. The ORB specified by Sun for Java. It allows method invocation on objects. Interfaces are specified in Java and not in a separate IDL.

Robot Controllers (RC)
RC (Robot Controller) takes care of the data exchange with industrial robots. RCs can be loaded with a default or custom program that will handle most robot control needs.

RPC
Remote Procedure Call. The system which allows to call a procedure in a remote (server) program and receive the results of that call.

SAP
Service Access Point (SAP) is an interface between two adjacent layers of the OSI reference model. The SAP is used to access services provided by a lower layer to a higher layer (or vice versa).

SattMate
The transport equipment communication handling part of the Warehouse Management System SattStore.

SattMate Core
Base level needed for all possible configurations of a SattMate system.

SattMate Sub-system
Part of the complete SattMate system that control or administrate a handling unit type.

SAX
SAX (Simple API for XML) is an API that allows a programmer to interpret a Web file that uses XML.

Scalability
The ease with which a system or component can be modified to fit the problem area.

SCADA
SCADA is the level of applications that monitor and control devices such as programmable controllers. These systems are usually PC or workstation based.

Scalability
The ease with which a system or component can be modified to fit the problem area.
SDU
Considering a layer N a Service Data Unit (SDU) is a piece of information passed by a layer above (the N+1-Layer) to the current layer (the N-Layer) for transmission using the service of that layer.

Security
The ability of a system to manage, protect, and distribute sensitive information

Semi-automatic assignments
Assignments issued to one sub-system.

Sensor
A sensor is a device, which responds to an input quantity by generating a functionally related output usually in the form of an electrical or optical signal (this signal is sent to a control device).

Sensor/actuator interface
Interface between sensors/actuators and a device controller.

Sensor networks
Networks that encompass protocols initially designed to support discrete I/O.

Sequence
Set of sub-system identities and function entries that should manage an automatic assignment for a given path

Server
A server is defined as the provider of services

Service primitive
A service primitive initiates an action or advises the result of an action. Each primitive may contain parameters to convey the Protocol Control Information (PCI) needed to perform its functions.

Servlet
A servlet is a small program that runs on a server. The term was coined in the context of the Java applet, a small program that is sent as a separate file along with a Web (HTML) page. Java applets, usually intended for running on a client, can result in such services as performing a calculation for a user or positioning an image based on user interaction.

SGML
SGML (Standard Generalized Markup Language) is a standard for how to specify a document markup language or tag set. Such a specification is itself a document type definition (DTD). SGML is not in itself a document language, but a description of how to specify one. It is metadata.

SMTP
SMTP (Simple Mail Transfer Protocol) is a TCP/IP protocol used in sending and receiving e-mail. However, since it's limited in its ability to queue messages at the receiving end, it's usually used with one of two other protocols, POP3 or Internet Message Access Protocol, that let the user save messages in a server mailbox and download them periodically from the server.

SOA
Service-oriented architectures (SOA) support a programming model that allows service components residing on a network to be published, discovered, and invoked by each other. Typically these services components interoperate with each other in a platform-and language independent manner.
SOAP
Simple Object Access Protocol (SOAP) is an XML-based protocol that defines a framework for passing messages between systems over the Internet. It's typically used for executing remote procedure calls. SOAP was originally intended and defined for use on top of HTTP to make SOAP more easily incorporated into Web-based applications, but other transport protocols, such as SMTP, can also be used.

Socket interface
Most widely used API to the TCP/IP protocol stack.

SSL
The Secure Sockets Layer (SSL) is a commonly-used protocol for managing the security of a message transmission on the Internet.

Stakeholder
Person of organization with an interest in the system.

Standard crane commands
Assignment and status commands for a specific crane exchanged between the Crane Administrator and the crane communication interface adapter.

Standard crane interface
The interface between the Crane Administrator and the crane communication interface adapters.

Standard conveyor commands
Assignment and status commands for a specific conveyor belt exchanged between the Conveyor Administrator and the conveyor communication interface adapter.

Standard conveyor interface
The interface between the Conveyor Administrator and the conveyor communication interface adapters.

Status
A status is reply back on a specific job (e.g. job finished) to the controller that generated that job. Notice that statuses are not always a reply on a job; statuses can also be created as a reaction on a specific condition in the manufacturing environment.

Stubs and Skeletons
A stub is a client-side helper class. The client communicates with the stub as if it were a real object, even though it is only an interface. A skeleton is a server-side helper class that receives method invocations from the client, unpacks the arguments and carries the method call into the actual object instance.

Sub-assignment
Part of an assignment, which is carried out in one sub-system.

Sub-system Administrator
Part of a sub-system that takes care of the assignment administration and handling in that sub-system.

Sub-system Supervisor
Part of a sub-system that supervises the sub-system e.g. sub-system monitoring and status modification.

System
A set of different elements so connected or related as to perform a unique function not performable by the elements alone. In this context it is the system for which the architecture is described.
TCP/IP
TCP/IP (Transmission Control Protocol/Internet Protocol) is the basic communication language or protocol of the Internet. It can also be used as a communications protocol in a private network.

Transaction
A transaction differs from a simple sequence of commands in the integrity: provided that during its execution an error or another unrecoverable problem occurs, such as a denial of access to a resource, the TP-monitor automatically returns the call and lets the system continue the execution from a well-defined state.

Transparency
Making something invisible to the client.

Transport assignment
Requirement to move a load from one location to another given to the SattMate system.

UDDI
UDDI (Universal Description, Discovery, and Integration) is an XML-based registry for businesses worldwide to list themselves on the Internet.

UDP
UDP (User Datagram Protocol) is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is an alternative to the Transmission Control Protocol (TCP) and, together with IP, is sometimes referred to as UDP/IP.

UML
Unified Modeling Language. UML is a compilation of "best engineering practices" when designing and creating large, complex software systems in an object oriented fashion.

Usability
The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

VDU routines
Functions which can be configured by the user for maintenance and supervision of the common objects (of a specific object type) in the SattMate system.

View
Part of an architecture description.

Viewpoint
Definition of a view (related to stakeholders and concerns).

Virtual Manufacturing Device (VMD) model.
The definition of objects, services, and behaviour comprises a comprehensive definition of how devices and applications communicate in which MMS calls the Virtual Manufacturing Device (VMD) model.

Web services
Web services are a new class of applications that can talk and work with one another over the Internet.

Working-range.
A working range is a range of locations within which a vehicle (crane, transfer car) is working.
Terminology

Wrapper
Modern Object Oriented jargon for an object that delegates most of the work to another object (the wrappee) that the wrapper object is said to wrap. The wrapper may present additional or diminished authority to its clients over that of the wrappee. It may present the same authority by transforming the input or output.

WSDL
The Web Services Description Language (WSDL) is an XML-based interface definition language that provides a way to catalogue and describe Web Services.

XML
eXtended Markup Language. The universal format for structured documents and data on the Web defined by WWW consortium. It is similar to HTML but much more flexible and powerful.

XMOP
XML Metadata Object Persistence

XSL
XSL (Extensible Stylesheet Language), formerly called Extensible Style Language, is a language for creating a style sheet that describes how data sent over the Web using the Extensible Markup Language (XML) is to be presented to the user.

XSLT
XSL Transformations (XSLT) is a standard way to describe how to transform (change) the structure of an XML (Extensible Markup Language) document into an XML document with a different structure. XSLT is a Recommendation of the World Wide Web Consortium (W3C).
References

Chapter 1
[ABB 2002] ABB Manufacturing & Consumer Industries
SattStore System overview

Chapter 3
Software architecture documentation in practice: documenting architectural layers

Documenting software architectures: Organization of documentation package

Software Architecture in Practice.
Addison-Wesley, 1997.

[BRED 1999] Bredemeyer D.
Architecting process
Bredemeyer consulting
http://www.bredemeyer.com

[BRED 2002] Bredemeyer D.
Bredemeyer consulting
http://www.bredemeyer.com

[BOOC 1999] Booch, Rumbaugh, and Jacobson
The UML Modeling Language User Guide.
Addison-Wesley, 1999.

Software architecture and the UML
Rational software, white paper. 2000

Documenting software architectures: Views and beyond

A practical method for documenting software architectures

[FLOR 2002] Florijn G.
Describing software architectures
Whitepaper, 2002

[FLOR 2002 2] Florijn G.
Architectural styles and patterns
Whitepaper, 2002
**Design patterns: Elements of reusable object-oriented software.**  

**Documenting software architectures**  
Report ABB B.V. 2003

[KRUC 95] Kruchten P.  
**The 4+1 view model of architecture.**  
IEEE software, pages 42-50, 1995

[Land 2002] R. Land  
**A brief survey of software architecture**  
Department of Computer Engineering, Malardalen University, Vasteras, Sweden, Feb. 2002

[SONI 95] Soni D., R. Nord and C. Hofmeister  
**Software architecture in industrial application.**  

**Chapter 4**

[ABB 2003-1] ABB Manufacturing & Consumer Industries  
**Distributed Computing Architectures**  
Report ABB B.V. 2003

[BARK 2000] Barkai, D.  
**An Introduction to Peer-to-Peer Computing.**  

[CAMB 2001] Cambiotis J.  
**An Introduction to Client/Server Architecture Goals and Specifications.**  
Lancaster University, 2001

[EMME 2000] Emmerich W.  
**Software engineering and Middleware: A Roadmap**  

[HALS 1996] Halsall F.  
**Data communications, computer networks and open systems (4th edition)**  
Essex: Addison Wesley, 1996

[Jaza 2000] Jazayeri M. and A. Ran, F. van der Linden  
**Software Architecture for Product Families: Principles and Practice.**  

[Lee 2000] Lee P.  
**Client server architecture evolution.**  
COMP9117 Software Architecture Seminar

[MURP 2002] Murphy G.  
**Client/Server Software Architecture**  
2002
[OFFU 2002] Offutt J.
Notes on N-Tier Architectures
Software Engineering for the World Wide Web
http://www.ise.gmu.edu/~ofut/

[Quar 2002] Quartel D. and M. Harkema
Implementation of Telematics Systems
University of Twente, department Telematic Systems and Services; Lecture notes, course 265200
http://arch.cs.utwente.nl/courses/its

[QUIN 1996] Quinn B. and D. Shute
Windows socket network programming
Massachusetts: Addison Wesley, 1996

[STOC 2001] Stock, M.
Doctoral Dissertation
Technologies for thin client architectures
Zurich, January 2001

Chapter 5

[ABB 1998] ABB Manufacturing & Consumer Industries
SattMate Core User Specification version 5.2
Report ABB B.V. 1998

[ABB 1999-1] ABB Manufacturing & Consumer Industries
SattMate Core Design Specification version 5.2
Report ABB B.V. 1999

[ABB 1999-2] ABB Manufacturing & Consumer Industries
SattMate Crane Design Specification version 5.2
Report ABB B.V. 1999

[ABB 1999-3] ABB Manufacturing & Consumer Industries
SattMate Conveyor Design Specification version 5.2
Report ABB B.V. 1999

Computer Integrated manufacturing (2nd edition)
New York: Prentice Hall College Division, 2000

Chapter 6

[ABB 1998] ABB Manufacturing & Consumer Industries
Comli System Description
Report ABB B.V. 1998

[ABB 1999-4] ABB Manufacturing & Consumer Industries
SattMate Siemens_IO Specification
Report ABB B.V. 1999

[ABB 1999-5] ABB Manufacturing & Consumer Industries
SattMate Comli_IO (Master/Slave) Specification version 5.1
Report ABB B.V. 1999
References

[ABB 1999-6] ABB Manufacturing & Consumer Industries
SattMate Signal_IO Specification version 5.2
Report ABB B.V. 1999

[ABB 1999-7] ABB Manufacturing & Consumer Industries
SattMate BCC_IO Specification version 5.2
Report ABB B.V. 1999

[ABB 1999-8] ABB Manufacturing & Consumer Industries
SattMate Crane_IO Specification version 5.2
Report ABB B.V. 1999

[ABB 2003-2] ABB Manufacturing & Consumer Industries
SattMate Daifuku_IO Specification 9.2
Report ABB B.V. 2003

Chapter 7

[ABB 2003-3] ABB Manufacturing & Consumer Industries
Industrial protocols and standards
Report ABB B.V. 2003

[BART 2001] Barton C.S.
PLC proprietary and open networks

[BERR 2000] Berrie P.G. and K.P. Lindner
An introduction to networks in process automation

[ESPR 1995] Esprit Consortum
MMS: A communication language for manufacturing
Esprit Research Report CCE-CNMA, project 7096, CCE-CNMA, vol. 2
Berlin: Springer, 1995

[FELS 2002] Felser M.
The Fieldbus standards: History and structures
Report University of Applied Science Bern

[FIEL 2003] Fieldbus Foundation
Foundation Fieldbus Technical Overview (FD-043 Revision 3.0)
www.fieldbus.org

[HASS 2003] Hassel T.F.M
Industrial protocols and standards research
Report ABB B.V. Etten-Leur 2003

[INTE 2001] Intellution Inc.
A technical overview of the OPC Data Access Interfaces
[MOSS 2000] Moss, B
Real-time control on Ethernet

[NETW 2000] Network Associates
A guide to communication protocols
Overview poster protocols

[OPC1 1998] OPC Foundation
OPC overview version 1.0 October 27, 1998
www.opcfoundation.com

[OPC2 1998] OPC Foundation
OPC Common definition and interfaces version 1.0 October 27, 1998
www.opcfoundation.com

[OPC3 2002] OPC Foundation
OPC Data access custom interface standard version 2.05 June 28, 2002
www.opcfoundation.com

[OPC3 2002] OPC Foundation
OPC XML-Data specification version RC (Release Candidate) 1.8 June 13, 2002
www.opcfoundation.com

[PRAT 2002] Pratt W.A.
Sorting out the protocols
By B.G. Liptak

[POTT 2000] Potter D.
Using Ethernet for industrial I/O and Data acquisition
Report National Instruments Corporation

[PROF 2002] Profibus organization
Technical overview of Profibus
www.profibus.com

[SISC 1995] Sisco Inc.
Overview and introduction to the Manufacturing Message Specification (MMS)
(revision 2)
Technical report

[VITT 2001] Vitturi S.
Overall fieldbus trends
By B.G. Liptak

[VITT 2002] Vitturi S.
Fieldbus networks catering to specific niches of industry
By B.G. Liptak
Chapter 10

Middleware
Report ABB B.V. 2003

[BISH 2002] Bishop T.
A survey of middlewares.
Towson University College of Graduate Education and Research
Master Thesis, August 2002

[BRIT 2001] Britton C.
IT architectures and middleware – Strategies for building large, integrated systems
Addison Wesley, 2001

STR Technology Descriptions.
Software Engineering Institute
http://www.sei.cmu.edu/str/descriptions/index.html

DCOM and CORBA Side by Side, Step by Step, and Layer by Layer.

Engineering distributed objects (EDO 99) Workshop summary.
In: Proceedings of the 1999 International Conference on 16-22 May 1999

[EMME 2000] Emmerich W.
Software engineering and Middleware: A Roadmap
University College London, Department of Computer Science

Component technologies: Java beans, COM, CORBA, RMI, EJB and the CORBA
Component model.
In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002),
Orlando, Florida.

[EMME 2002-2]Emmerich, W.
Distributed Component technologies and their software engineering implications.
In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002),
Orlando, Florida.

[GOGI 2002] Gogis
A J2EE Introduction

[LI 2002] Li, T.
Web services technology: design, implementation and evaluation.
References

[LINT 1997] Linthicum D.
Next generation middleware

[LUYT 2000] Luyten, K.
Component based Software development for the internet.

[NAND 2001] Nandrajog I.
Simplified Object Access Protocol
CIS 679 Management of IS, Spring, 2001

[OMG 1995] OMG
OMG technical document formal/00-11/07, July 1995
http://www.omg.org

The essential distributed objects survival guide.
Chichester: John Wiley & Sons, 1996.

[PARK 2003] Park J.
middleware.
School of Information Sciences & Technology, The Pennsylvania State University
http://ist.psu.edu/park

Middleware for Distributed Systems - Evolving the Common Structure for Network-centric Applications.
In: The Encyclopedia of Software Engineering, ed. by J. Marciniak and G. Telecki.

[SORE 2001] Sorensen, Carl-Fredrik
Object-Oriented Systems: A Comparison of Distributed Object Technologies.
The Norwegian University of Science and Technology, 2001
Lecture notes no. DIF8901

Consensual trends for optimizing the constitution of middleware
In: Periodical-Issue-Article

[TARI 2001] Tari Z. and O. Bukhres

[VINO 1997] Vinoski, S.
CORBA: Integrating diverse applications within distributed heterogeneous environments.
[W3C 2000] W3C
Simple Object Access Protocol (SOAP) 1.1
W3C Note 08 May 2000
http://www.w3.org/TR/SOAP/

http://webopedia.internet.com