IAS-Forschungsbericht

Forschungsbericht
Institut für Automatisierungs- und Softwaretechnik
Hrsg: Prof. Dr.-Ing. Dr. h. c. P. Göhner

Vicente Ferreira de Lucena Jr.

Flexible Web-based Management of Components for Industrial Automation

Band 4/2002

Universität Stuttgart

Flexible Web-based Management of Components for Industrial Automation

Von der Fakultät Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von
Vicente Ferreira de Lucena Junior
aus São Paulo - Brasilien

Hauptberichter:	Prof. Dr.-Ing. Dr. h. c. Peter Göhner
Mitberichter:	Prof. Dr.-Ing. Dr. h. c. mult. Günter Pritschow
Tag der Einreichung:	19.06.2002
Tag der mündlichen Prüfung:	18.11.2002

Institut für Automatisierungs- und Softwaretechnik
der Universität Stuttgart

2002

IAS – Forschungsberichte

Band 4/2002

Vicente Ferreira de Lucena Junior

Flexible Web-based Management of Components for Industrial Automation

D 93 (Diss. Universität Stuttgart)

Shaker Verlag
Aachen 2002

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

de Lucena Junior, Vicente Ferreira:
Flexible Web-based Management of Components for Industrial Automation /
Vicente Ferreira de Lucena Junior.
Aachen : Shaker, 2002
(IAS-Forschungsberichte ; Bd.2002,4)
Zugl.: Stuttgart, Univ., Diss., 2002
ISBN 3-8322-1011-3

ISBN 3-8322-1011-3
ISSN 1610-4781

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen
Telefon: 02407 / 95 96 – 0 • Telefax: 02407 / 95 96 –9
Internet: www.shaker.de • eMail: info@shaker.de

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik (IAS) der Universität Stuttgart.

Mein besonderer Dank gilt

Herrn Prof. Dr.-Ing. Dr. h. c. P. Göhner für die Betreuung dieser Arbeit sowie für den Freiraum zur Vollendung dieser Arbeit,

Herrn Prof. Dr.-Ing. Dr. h. c. mult. Günter Pritschow für die Übernahme des Mitberichts,

allen Kolleginnen und Kollegen am Institut für die gute Zusammenarbeit und hilfsbereite Unterstützung,

den Studenten, die im Rahmen von Studien- und Diplomarbeiten zum Gelingen dieser Arbeit beitrugen, damit die vorgestellten Konzepte nicht nur Ideen blieben, sondern auch verwirklicht werden konnten,

der brasilianischen Forschungsgesellschaft CNPq für die finanzielle Unterstützungen und Förderung des Forschungsvorhabens,

und Juliana, Paula, Mateus und meiner Familie in Brasilien für ihr Verständnis, ihre Geduld und ihren Ansporn während der Entstehung dieser Arbeit.

Stuttgart, im Juni 2002

Vicente Ferreira de Lucena Junior

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACPLT	**A**a**C**hener **P**rozess**L**eit**T**echnik (Aachen Process Control Engineering)
AIRS	**A**rtificial **I**ntelligence based **R**euse **S**ystem
API	**A**pplication **P**rogramming **I**nterface
ASP	**A**ctive **S**erver **P**ages
ASSET	**A**sset **S**ource for **S**oftware **E**ngineering
AWT	**A**bstract **W**indow **T**oolkit
BLOB	**B**inary **L**arge **OB**ject
CDL	**C**omponent **D**escription **L**anguage
CE	**C**onsumer **E**lectronics
CID	**C**omponent **ID**
CLR	**C**ommon **L**anguage **R**untime
COM	**C**omponent **O**bject **M**odel
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture
COTS	**C**ommercial **O**ff-**T**he-**S**helf software
DARPA	**D**efense **A**dvanced **R**esearch **P**roject **A**gency
DBMS	**D**ata**B**ase **M**anagement **S**ystems
DCOM	**D**istributed **C**omponent **O**bject **M**odel
DIO	**D**istributed **I**ntelligent **O**bjects
DIRECT	**D**igital **R**esource **C**atalogue
DNS	**D**omain **N**ame **S**ystem
DOM	**D**ocument **O**bject **M**odel
DTD	**D**ocument **T**ype **D**efinition
ESDL	**E**mbedded **S**oftware **D**escription **L**anguage
FAQ	**F**requent **A**sked **Q**uestions
FOCS	**F**eature **O**riented **C**lassification **S**ystem

FRR	**F**ederal **R**euse **R**epository
FTP	**F**ile **T**ransfer **P**rotocol
GAMS	**G**uide to **A**vailable **M**athematical **S**oftware
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper**T**ext **M**arkup **L**anguage
HTTP	**H**yper**T**ext **T**ransport **P**rotocol
IAC_d	**I**ndustrial **A**utomation **C**omponents **D**escriptor
IAS	**I**nstitut für **A**utomatisierungs- und **S**oftwaretechnik – Universität Stuttgart (Institute of Industrial Automation and Software Engineering – University of Stuttgart)
IB	**I**nter**B**ase
IDL	**I**nterface **D**escription **L**anguage
IFA	**I**nstitut **F**ür **A**utomatisierungstechnik – Technische Universität Dresden (Institute of Automation - University of Technology – Dresden)
IIOP	**I**nternet **I**nter-**O**RB **P**rotocol
JAXP	**J**ava **A**PI for **X**ML **P**rocessing
JDBC	**J**ava **D**ata**B**ase **C**onnectivity
JDK	**J**ava **D**evelopment **K**it
JNLP	**J**ava **N**etwork **L**aunch **P**rotocol
JRE	**J**ava **R**untime **E**nvironment
JSP	**J**ava **S**erver **P**ages
JWS	**J**ava **W**eb **S**tart
KIS	**K**omponenten**I**nformations**S**ystem
LaSSIE	**La**rge **S**oftware **S**ystem **I**nformation **E**nvironment
MOREplus	**M**ultimedia **O**riented **R**epository **E**nvironment **P**lus
NAS	**N**etwork **A**rchitecture **S**ervice

ODBC	**O**pen **D**ata**B**ase **C**onnectivity
OID	**O**bject **Id**entifiers
OMG	**O**bject **M**anagement **G**roup
OQL	**O**bject **Q**uery **L**anguage
ORB	**O**bject **R**equest **B**roker
PLT	Institut für **P**rozess**L**eit**T**echnik – Technische Universität Aachen (Process Control Engineering Institute – University of Technology – Aachen)
RBSE	**R**epository **B**ased **S**oftware **E**ngineering
RMI	**R**emote **M**ethod **I**nvocation
RSL	**R**eusable **S**oftware **L**ibrary
SALMS	**S**oftware **A**sset **L**ibrary **M**anagement **S**ystem
SAX	**S**imple **A**PI for **X**ML
SEL	**S**oftware **E**ngineering **L**ibrary
SIM	**S**tandard **I**nterface **M**echanism
SMTP	**S**imple **M**ail **T**ransport **P**rotocol
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
SOORLS	**S**ummary-based **O**bject **O**riented **R**euse **L**ibrary **S**ystem
SQL	**S**tructured English **Q**uery **L**anguage
TCP/IP	**T**ransmission **C**ontrol **P**rotocol / **I**nternet **P**rotocol
TON	**T**imer-**ON** Component
UDDI	**U**niversal **D**escription, **D**iscovery and **I**ntegration
ViPER	**Vi**sual **P**rogramming Environment for **E**mbedded **R**eal-Time Systems
W3C	**W**orld **W**ide **W**eb **C**onsortium
WAIS	**W**ide **A**rea **I**nformation **S**earch
WSDL	**W**eb **S**ervices **D**escription **L**anguage

WIMP	**W**indow, **I**cons, **M**enus, and **P**ointer principle
XML	E**X**tensible **M**arkup **L**anguage
XSL	E**X**tensible **S**tylesheet **L**anguage
XSLT	**XSL T**ransformation

Glossary

API: Application Programming Interface. An API is an interface to a specific environment. For example: The Windows API exposes interfaces to the Windows environment, allowing developers to access and control Windows functions. The Java API provides similar interfaces to a Java Virtual Machine.

Applet: An applet is a self contained Java programme designed for Web applications. Applet is downloaded and run on the client to provide specific functionality.

Application: Any stand-alone programme that accomplishes a pre-determined task.

Application Server: An application server is a network server that hosts and runs applications or components.

AWT: The Abstract Window Toolkit is an API for providing Graphical User Interfaces (GUI) to Java applications. It includes user interface components and graphics and imaging tools.

Embedded System Software: Software used to control specialized hardware attached to the computer system. Embedded systems have normally limited computation power and small memory capacity.

Executable: An executable is essentially a programme. It can be run independently of a host application. The only requirements for an executable are a compatible operating system and any associated runtime library.

Hard Real-Time System: Systems where failures to meet response time constraints leads to system failure.

HTML: The Hypertext Markup Language is a "tagged" language for transferring data using HTTP. It allows users to format text, include pictures, and insert hyperlinks to other data. The Web browser reads the "tags" and displays the data accordingly.

HTTP: The Hypertext Transmission Protocol is an Internet standard protocol for exchanging files (text, images, sound, video, etc.) over the Internet.

IDL: An Interface Definition Language file is a file that contains definitions of interfaces to components, consisting of an interface header and interface body. The header contains attributes that apply to the interface as a whole. The body contains individual interface definitions such as data types used in remote procedure calls and prototypes for the remote procedures. While not required, an IDL makes it easier for a developer to define and query information about a component's interfaces.

Java (Programming Language): Java is a development language used to build components and applications.

Java (Virtual Machine): A Java VM is required on any computer where you intend to run a Java application. Sun Microsystems, who owns Java, has licensed the specifications for the virtual machine so that manufacturers of operating systems can build Java VMs for their environment. As a result, the Java language is compatible with a wide range of platforms.

JDBC: Java Database Connectivity is an API for connecting Java applications to databases. JDBC is similar to ODBC.

ODBC: Open Database Connectivity is an Application Programming Interface (API) that provides access to a variety of data sources. It is an industry standard for exchanging data. As such, it allows computers in a multi-platform environment to access data on a SQL (or any ODBC-compliant) database.

OMG: The Object Management Group is a non-profit corporation founded by eleven companies including 3Com, American Airlines, Hewlett-Packard, Sun Microsystems, and Unisys. It supports a component-based software marketplace through industry standards.

OOP: Object Oriented Programming is the process of building applications by encapsulating functionality into individual objects. These objects feature polymorphism and inheritance.

Reactive System: System that has some on-going interaction with its environment.

Real-Time System: Systems that must satisfy explicit (bounded) response time constraints or it will fail.

Servlet: A Servlet is a self contained Java programme designed for Web applications. Servlets are run on the Web server and data is sent to the client via HTML or XML.

SOAP: Simple Object Access Protocol is a message-based protocol based on XML for accessing services on the Web. Initiated by Microsoft, IBM and others, it employs XML syntax to send text commands across the Internet using HTTP.

Soft Real-Time System: Systems where performance is degraded but not destroyed by failure to meet response time constraints.

SQL: The Structured Query Language is an industry standard programming language for accessing and updating a database. SQL is also used to refer to the database management system (DBMS) that stores this data. SQL queries allow you to select, insert, update, and find data in the SQL database.

Swing: The Swing classes provide support for forms-based Java programming. It provides the ability to include trees, tabbed panes, splitter panes, and other user interface enhancements to Java applications, giving them the look-and-feel expected in today's application marketplace.

TCP/IP: The Transmission Control Protocol/Internet Protocol is the base protocol used for communication over the Internet. It allows both connection-based and non connection-based (fire and forget) transmission of data over the Internet. Essentially, TCP/IP is the language spoken by the Internet Protocols/Services such as HTTP, SMTP, FTP, etc.

UDDI: Universal Description, Discovery and Integration is an industry initiative for a universal business registry of Web services. Led by Ariba, IBM, Microsoft and others, UDDI is designed to enable software to automatically discover and integrate with services on the Web.

UML: The Unified Modeling Language is designed to specify and document the structure of system under development. It is a graphical language for expressing programme design in a standard way.

Web Services: Web-based applications that dynamically interact with other Web applications using open standards that include XML, UDDI and SOAP. Microsoft's .NET and Sun's Sun ONE (J2EE) are the major development platforms that natively support these standards.

WSDL: Web Services Description Language is a protocol for a Web service to describe its capabilities. Co-developed by Microsoft and IBM, WSDL describes the protocols and formats used by the service.

XML: The Extensible Markup Language (XML) is an industry standard method for using "tags" to describe data for exchange between different platforms, languages and applications.

XSL: The Extensible Stylesheet Language is used to describe XML data that is sent over the Web to be presented to the user. It is an industry standard language that gives the Web page author control over how XML data is displayed, which fields are presented, where they are displayed on the page and in what order.

XSLT: XSL Transformations is an industry standard that describes how to change an XML document from one structure to another structure. It is used to transform the source tree of one XML document into a result tree for a new XML document.

Abstract

The development of applications based on elementary components is one of the oldest known engineering approaches. Such a procedure is well practiced, for example, in the hardware electronics industry and has been applied more and more in other domains. In the last years software components have been adopted as a meaningful solution for the construction of larger software applications improving the development productivity and improving the quality of the final products. But the more components that are available the more difficult is the proper management of those components and consequently more difficult is the proper choice of the most suitable one for solving a specific task.

This thesis deals with this problem and proposes a systematic way of managing components. How to store components properly is investigated and later on how to find, understand and decide about the reuse of the most appropriate component. This thesis was centered in the industrial automation domain and a study of the most relevant characteristics of that domain was done. There is no common standardized model available for representing software components used in the industrial automation domain mainly because of the great amount of quite different component technologies being used in that domain. In fact, it is not possible to summarize all of them in only one model. The problem faced was to propose a way of constructing a repository able to deal with those technologies together and be able to represent the technical details of each one of them properly. The proposed solution involves a flexible representation for the components that consists of two parts, the first one with the common information for every component technology involved, and the second one where the specificity of each component technology is summarized. Additionally, through the systematic management process proposed, recently developed components are published before being used, and component users are conducted during the search for desired components.

A prototypic tool for the management of components was constructed using technologies associated with the Internet. The necessary flexibility of the component management system was obtained through the representation of the specificity of each component technology with the eXtended Markup Language (XML) and associated technologies. In the example presented, different component technologies like JavaBeans and ViPER are successfully managed together.

Zusammenfassung:

Flexibles netzbasiertes Management von Komponenten für die Automatisierungstechnik

Die Entwicklung von Komponenten für den Einsatz in unterschiedlichen Anwendungsbereichen, z.B. im Bauingenieurwesen oder in der Hardwareentwicklung ist eine der ältesten Ingenieurtätigkeiten. In den letzten Jahren wurde die Anwendung von Komponenten auch für die Entwicklung großer Softwareapplikationen eingeführt, wodurch die Produktivität erhöht und die Qualität des Endproduktes verbessert werden konnte. Aber je mehr Softwarekomponenten verfügbar sind, desto schwieriger ist die Verwaltung der Komponenten und desto schwieriger ist auch die Auswahl einer passenden Komponente, die es erlaubt, eine spezifische Aufgabe zu lösen.

Der Anwender von Komponenten benötigt bei der Suche nach geeigneten Komponenten als Grundlage eine Bibliothek mit Softwarekomponenten. In dieser Bibliothek versucht er eine Komponente zu finden, die seinen Wünschen und Anforderungen genügt. Diese Entscheidung wird er auf der Basis von Informationen über die Schnittstellen und das Verhalten der gespeicherten Komponenten treffen. Die Komponenten und die dazugehörigen Informationen stammen von den Entwicklern der Komponenten. Der Umfang, die Qualität und auch die Terminologie der Dokumente kann deshalb sehr stark variieren, was die Effizienz bei der Suche stark beeinträchtigt. Diese Arbeit befasst sich mit dieser Problematik und macht einen Vorschlag, wie die Informationen über Komponenten, die speziell für den Einsatz in der Automatisierungstechnik entwickelt wurden, einheitlich dargestellt und verwaltet werden können, um dadurch die Qualität der gespeicherten Informationen zu erhöhen.

Hat ein Anwender eine Komponente gefunden, welche das von ihm gewünschte Verhalten aufweist und eine geeignete Schnittstelle besitzt, muss geprüft werden, ob sich diese Komponente in seine Applikation integrieren lässt. Das ist von verschiedenen Rahmenbedingungen abhängig, die bei Komponente und Applikation übereinstimmen müssen. Rahmenbedingungen können beispielsweise die verwendete Komponententechnologie, das Betriebssystem, Echtzeitbedingungen usw. sein. Zudem werden für verschiedene Rahmenbedingungen unterschiedliche Informationen benötigt, um entscheiden zu können, ob die Komponente geeignet ist. Ein Ziel dieser Arbeit war es, eine Möglichkeit zur Darstellung von Komponenten vorzuschlagen, welche die Rahmenbedingungen integriert. Dies erfordert natürlich eine gewisse Flexibilität bei der Ablage der Informationen, da diese nicht nur aktuelle Technologien umfassen darf, sondern auch zukünftige Entwicklungen berücksichtigen muss.

Um die Suche von Komponenten zu erleichtern, wird ein Klassifikationsschema vorgeschlagen. Dieses ist spezifisch für die Automatisierungstechnik und berücksichtigt verschiedene

Rahmenbedingungen, die für diesen Anwendungsbereich typisch sind. Zusätzlich werden für das Verwalten von Komponenten Vorgehen definiert, die durch ein Werkzeug unterstützt werden.

Ein Klassifizierungsschema bestimmt, wie die Einordnung von Komponenten in eine vorgegebene Struktur durchzuführen ist. Die Anbieter können die vorgeschlagenen Regeln und Kriterien übernehmen und haben die Freiheit neue einzuführen. Beim Suchen wird der Anwender von Komponenten durch die Struktur und die damit verbundene Reduzierung der Auswahl unterstützt. Das vorgeschlagene Klassifizierungsschema für Komponenten der Automatisierungstechnik besteht aus elf Klassen. Diese Klassen decken die Hauptaspekte ab, die bei der Softwareentwicklung für Automatisierungssysteme auftreten.

Der Prozess zur Verwaltung der Komponenten umfasst die Prozesse für das Ablegen und das Suchen. Sie werden als Veröffentlichungsprozess und Auswahlprozess bezeichnet. Der Veröffentlichungsprozess garantiert, dass alle für ein gutes Verständnis einer Komponente notwendigen Informationen beim Ablegen einer Komponente in die Bibliothek aufgenommen werden. Dazu muss der Anbieter einer Komponente eine Reihe von Informationen bereitstellen, die für das spätere Auffinden, Verstehen der Komponente und für die Entscheidung über den Einsatz der Komponente wichtig sind. Dieser Veröffentlichungsprozess wird in drei Schritten durchgeführt: Die Klassifizierung der Komponente, ihre vollständige Dokumentation und schließlich die Ablage aller Informationen in einer Bibliothek. Der Auswahlprozess stellt sicher, dass ein Anwender bei der Suche nach einer Komponente effizient unterstützt wird und alle für seine Entscheidung notwendigen Informationen bereitstehen. Der Auswahlprozess besteht aus drei Schritten: Der Suche nach Komponenten in einer Bibliothek durch die Einschränkung von Kriterien und die Auswahl von Merkmalen, der Analyse des Funktionsumfangs der gefundenen Komponenten und der Entscheidung über die Anwendung einer gefundenen Komponente.

Es ist nicht möglich eine einheitliche Beschreibung für die Ablage von Komponenten zu finden, da die Beschreibungen für verschiedene Rahmenbedingungen unterschiedlich sein müssen. Deshalb wurde die Beschreibung von Komponenten in dieser Arbeit in einen invarianten und einen flexiblen Teil aufgetrennt. Der invariante Teil enthält grundlegende Informationen, die für jede Komponententechnologie einheitlich beschrieben werden müssen. Der flexible Teil enthält Informationen, die spezifisch für eine Komponententechnologie sind. Er wird von Technologie zu Technologie angepasst und enthält Daten, die unterschiedlich aufgebaut sein können. Das vorgeschlagene System für das Management von Komponenten benutzt für jede Komponententechnologie die Summe aus invarianten und flexiblen Daten.

Auf der Basis von Internet-Technologien wurde ein Werkzeug für das Management von Komponenten einer Bibliothek prototypisch realisiert, dessen Flexibilität bei der Repräsentation der verschiedenen Komponententechnologien durch den Einsatz der eXtended Markup Language (XML) und ihr verwandter Technologien erreicht wird. Für die Repräsentation der

invarianten Daten wurde eine relationale Datenbank gewählt. Die graphische Anwendungsschnittstelle (Graphical User Interface - GUI) wurde ebenfalls in zwei Teile, einer statischen GUI und einer dynamischen GUI, aufgeteilt. In einem Beispiel wird gezeigt, wie verschiedene Technologien mit dem erstellten Werkzeug gemeinsam in der selben Komponentenbibliothek verwaltet werden können.

Als Ergebnis steht ein System zur Verfügung, das das Management von Komponenten für die Automatisierungstechnik unterstützt. Die flexible Charakteristik des Konzepts ermöglicht das Einfügen von Komponenten mit neuen Rahmenbedingungen.

1 Introduction and Motivation

Classical software development weaknesses such as inaccurate schedules and cost estimates, inadequate productivity of software developers, and unsatisfactory level of quality of developed systems, are still problems to be solved. In the last decades, many different development approaches have been tried in order to suppress such difficulties as for example the object-oriented analysis and design [Booc94, Balz96]. Nevertheless, it is not possible to affirm that such methodologies were able to attend all necessities and wishes of software development people [Udel94, BrSi02].

Component-based software development is one of today's hottest discussed topics in the software community. The creation of reusable components has been seen as the most appropriate way of increasing the production of reliable software systems. After their construction, components are stored in libraries in order to be re-utilized in future projects. As the cost of reusable software components are normally many times greater than the equivalent conventional software artifacts [KAO91, BCE+97], developers have to make their reuse as efficient as possible by compensating those additional costs through the increase of the reusability level of the components.

Although many authors have written about software components and the advantages of developing such software pieces in the last years, it was not possible to find a consensual technical definition about components themselves that could be accepted by the majority of experts. Some authors define software components as an evolution of the object-oriented approach and the concepts used for components and classes (or objects) are the same [BaSc92, Sugi95, Yu97, SPH98]. Some others authors presented their own definitions for software components [Udel94, Dell97b, Trac97, Szyp98], some of them well accepted by the software community. In more general approaches, all reusable parts of a software system (e.g.: routines of C programming language, specific modules, set of functions) or even part of the software development process (e.g. requirements specifications, tests results) are presented and treated as components themselves [ZaWi95a, ZaWi95b, KeSc98].

Components have also been used in the industrial automation domain in the last years [Göhn98, HoRi99, Dujm02]. This domain has very specific characteristics that were not sufficiently investigated by the community of researchers and practitioners working with components [Kope00]. Nevertheless, component technologies' proposals with use at industrial automation applications have been presented recently by the process automation industry, by the embedded systems industry, and by university research centers. These facts show the potential of such component-based approaches for the industrial automation domain.

An additional problem is that the most common representation of components suggested in publications consists of one interface and one behavior description part and this representation is

not able to express all necessary information about components being used in the industrial automation domain. For example, important real-time characteristics are often not being represented in the behavioral description documentation part. In fact, one main difficulty observed is that the information needed to correctly understand one component varies from user to user, and more importantly this information varies among different component technologies.

1.1 Reasons for the Management of Components

Component based software development has as objective the implementation of software systems by the assembly of prefabricated software building blocks. These pieces of software must be generalized enough in order to be easily utilized in future projects. Nevertheless, it is not enough to design software components to assure a great level of reuse. The developed component itself must contain some properties that encourage and facilitate its reuse. The most important ones are summarized below:

- A good component should be easily understood. Its interface (inputs and outputs), adjustment parameters, and the way one connects it to another component should be clear. That is not an easy characteristic to achieve as the specifications of a component depend on the description of the technology in which the component was developed, demanding the intervention of specialists to be well understood.
- A good component should also be useful. As previously stated in [JSW92] even if a component is functionally rich, the totality of its functionality may not be useful to all applications where it could be inserted. For the case of a user searching for a specific functionality, a good component is not the one that can do many different tasks, but the one that can do exactly what is needed by the user.
- A last characteristic of a good component is its applicability. A component reaches a high level of applicability only if the effort to accomplish its integration is low. That means, that the way components are integrated should be made clear enough to reduce the time of adaptation rework on the desired component.

Even if all these features are achieved, it is not possible to assure that a large margin of reuse will be reached. There are some organizational problems that complicate the reutilization of components [Luce00]. New mechanisms or tools shall be introduced in the development process to try to guarantee that a useful, functional, and (re-) usable component is found and will eventually be reused in new projects.

These organizational elements shall be created in order to enable developers to take advantage of reusable software components. A component must have sufficiently many uses and therefore many clients to be economically viable. Additionally, for clients to use a component instead of a specialized solution, the component needs to have substantial advantages [Szyp98]. One advantage could be technological superiority, but other advantages are more likely to help, as

for example, being the first solution of an open problem, having a broad support base and a brand name, and so on. In other words, it is necessary to expose a developed component in an attractive form, covering all relevant technical and commercial details.

Software components stored in a library must be readily accessible. A user supporting tool must be devised to search and retrieve the closest match that meets the programmer's needs. One technical problem causing poor reuse of existing software components is that the information about the data structures, interface of each component and behavior description is implicitly rather than explicitly specified [Dai95]. Some authors recommend, in spite of additional costs, the standardization of the component's representation in terms of an internal specification (the functionality) and an external specification (interface and associated documentation) as a way to improve the component's understandability [SuBa97]. But this procedure alone is not enough to reach the proposed goals because even where the documentation is fully created and stored, the necessary information may be neither quick to find nor easy to understand. This thesis deals with this problem, making a suggestion on how to organize the information of different component technologies used in the industrial automation domain in the most appropriate way.

1.2 Requirements and Goals of the Management of Components

The management of components must support the people involved in the development of component-based applications, i.e. component developers and component users. In fact such a system will only be successful if it is well accepted by component developers (who create, classify, and store components in a proper form), and by component users, i.e. system developers (who search for, try to understand, and try to integrate diverse software components together). Based on this global scenario the desired requirements and goals may be formulated.

Requirements of this thesis

- This thesis should attend the needs of component developers, helping them to document their components properly and to store their components in a common electronic repository.
- The proposed management concept should support component users to find components previously stored in an electronic repository and to correctly understand their usage.
- Users considered by the management concept must be conducted along each desired management procedure.
- Components managed by the concept presented in this thesis are supposed to be components with use in industrial automation applications.
- The management concept should be widely accessible.

Goals of this thesis

The concept presented in this thesis proposes a systematic management approach able to help component developers to offer their components with a meaningful strategy, and able to support component users to find a stored component based on the desired functionality, to access the right information about one component, and to help component users to decide properly about the reuse of a component. The idealized management of components has three major features, it must be easy to use, it shall work with components dedicated to the industrial automation domain, and it shall be widely accessible.

- Easy to use means that the global concept must be user-friendly, with different interfaces for different users, and shall also conduct the actions of all users involved in the management process. It is expected that the component management system provides guidance on how to obtain the information desired by a specific user. The exchange of information between component developers and component users may be enabled in a convenient way. The introduction of new data or the access to existing data shall be done in a decentralized form. The confrontation of the desired functionality against the one of an existing component shall also be supported. Nevertheless, the decision about using a component will be done by the potential component user.

- The concept is dedicated to the industrial automation community. It is not a goal of this thesis to propose a new formal component model for the industrial automation domain. The goal is to propose a common flexible model for the representation of component technologies that actually have any application in industrial automation systems. Such a management concept shall be used by customers with diverse backgrounds. It is assumed that most of these users will be experts on industrial automation, or software engineers. In spite of that, no user expertise shall be admitted as a pre-requisite for the usage of the concept. Moreover, any potential user shall be able to verify the contents of potential component candidates. Nevertheless, the storage of new components shall respect some rules and someone storing a component must be previously known by the system.

- Lastly, the desired wide accessibility may be achieved easier by an Internet-based system. The goal is to propose a system where users will be able to access it from any computer connected to the Internet. For that purpose the most relevant characteristics of Internet-based systems and the respective Internet technologies will be studied.

Components shall be represented considering the particularities of their original construction technology. In that sense, the terms used to represent each part of the component representation must be respected. Diverse component technologies are also supposed to share the same repository and to be managed by the same procedures.

1.3 Overview of the Next Chapters

In the next chapter, the most relevant works about management of components reported in publications are presented. The reports about these approaches help to properly understand the problem involved and to learn from the experience of other groups of researchers having similar goals as the ones of this thesis. A comparison of the studied approaches and some fundamental conclusions are also presented in this chapter.

As said before, the goal of this thesis is to manage components with any use in the industrial automation domain. Many of the most successful component models proposed by market leaders have been used to construct industrial automation applications. Moreover, some other proposals come from specialized industry areas and from universities. The most relevant component proposals are discussed in chapter 3.

But those components must be represented in a meaningful manner in order to be properly managed later on. In chapter 4, the necessary knowledge about components developed particularly for the industrial automation domain is represented using semantic networks. The most relevant aspects of components designed to be used at those domains are considered in the proposed representation model.

The concept of a systematic management of components procedure is presented in chapter 5. An application development process composed of two main parts is proposed, the publication of components and the selection of components. The ideal way of creating a component management system, flexible enough to work with diverse technologies is also presented.

The technologies available to convert the theoretical study of the previous chapter in a meaningful Internet-based application are discussed in chapter 6. There, some architectural and technological decisions developed during this thesis are presented in order to sustain the practicality of the study done.

The flexible Web-based management of components for industrial automation was implemented in a prototype. Three main tasks are developed by this prototype, the storage of components, the search for components, and the understanding and comparison of components. The description of the implementation details of the elements composing the component management system and an exemplary usage are presented in chapter 7.

The conclusions of this thesis are presented in chapter 8. A summary of the most important considerations and about the obtained results are presented in that chapter. Additionally some insights of possible future works in this amazing research area are presented. This thesis is then closed with the Appendix A where an example of the documentation of a ViPER synchronous component is presented. This component is called TON and it is used to illustrate the generation of the interfaces to the users of the component management system.

2 State of the Art of the Management of Components

In this chapter a summary of the most relevant approaches for the management of components is presented. The selection of the commented works was based on their contribution to the state of the art and on their similarities with the system to be proposed later. The presentation is made in a chronological way. All of them had a very important role at the instant of their publication and still give helpful insights about the requirements of future works. The approaches are presented by trying to answer the following questions:

a) What is a software component at this approach?
b) How is a component represented here?
c) How is the information that represents a component obtained?
d) How can somebody find a desired component?
e) Is there any additional support to facilitate the understandability and reuse of a pre-selected component?

At the end of this chapter a table summarizing all approaches is presented. The items in the table should enable a comparison of the presented works at a common basis and should make it easier to understand where efforts are still needed to construct a meaningful component management system.

2.1 Most Relevant Component Management Approaches

The management approaches presented in this section are shown in Figure 2.1. A time scale is used to locate the works when they were proposed.

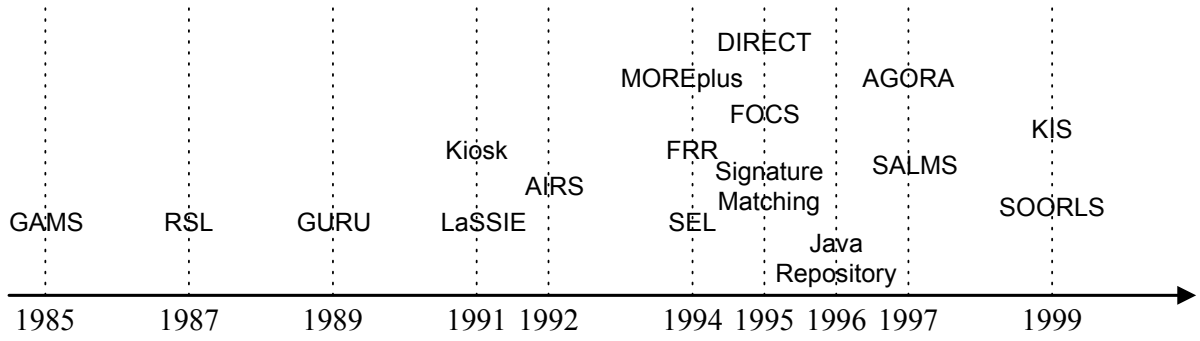


Figure 2.1: Chronological Presentation of Component Management Approaches.

It is possible to see that this area of interest is the same as some researching groups world wide since the middle of the 80's when the first suggestions appeared. With the advent of new information processing technologies at the beginning of the 90's, mainly the internet and related

technologies, a large amount of new proposals were presented again. In spite of the great number of interesting proposals, the evolution of those theories proved that a definitive answer was not yet obtained. Recent works, as for example the ones dated 1999, pointed out some difficulties that they themselves were not able to solve. That means that a significant contribution to this amazing area is still possible to be proposed. Particularly, it was not possible to find a well suited proposal for the management of components with use in the industrial automation domain.

2.1.1 Guide to Available Mathematical Software

This system, also called GAMS, was proposed in 1985 and provided a conceptual framework for scientist-end-users and librarian-maintainers to deal with large quantities of mathematical and statistical software. It was developed in the Scientific Computing Division of the National Bureau of Standards (NBS) in Maryland [BHK95]. The main idea was to offer a classification scheme for mathematical and statistical software, a database system to manage information about this software, and both an on-line interactive consulting system and a printed catalogue for providing users with one overview of the available information. A goal of the system was to improve the level of reusability of FORTRAN subprogrammes available for the NBS' researchers. It used a subject-oriented classification scheme organizing the programmes in pre-defined areas of interest (e.g. integral calculation, Fourier series, etc). The description of the reusable artifacts was implemented through a dictionary containing a set of basic information about each one. This information (variables types, parameters involved, etc) was also introduced manually by the storing of the component. The searching and finding of components were done based on this description through the browsing of the desired functionality. Only elementary help was offered for the reuse of available components, because the system was supposed to be used by experts with some confidence with computational systems and with the mathematical and statistical applications involved. This help facility was based only on the documentation of the FORTRAN subprogrammes. It also assumed the employment of someone to manage the whole system doing the job of a database administrator.

2.1.2 Reusable Software Library

The prototype called RSL was developed by Intermetrics Inc. in 1987 and coupled a passive database with interactive tools designed to make reuse of integral parts of the software development process [BAB+87]. The original intention was to accept and support reusable software written in any programming language, but the most successful examples were demonstrated using Ada. The complete concept was composed of a library management, a user query, a component retrieval and evaluation, and a software computer aided design tool. The library management included an automated data collection, based on specially commented lines. The special characters limiting the comment lines were inserted manually in the source code. It

also offered a standardized data entry. The RSL's software classification strategy was based upon the combination of two mechanisms. The first was the assignment of a hierarchical category code to each component added to the library, specifying the type of component (e.g. math functions, data structures, etc). The second mechanism permitted the insertion of up to five descriptive keywords for each component. The query subsystems provided users with a menu-driven interface to search for components with specific attributes and to generate reports about their attributes. The search could be targeted on a particular attribute (author, unit name, category code) or be expressed in English with the RSL's natural language searching feature. The system was able to evaluate the most suitable component through a score subsystem. Based on the information entered by the user, candidates were presented with their rating scores. The end selection could then be done. At last, the RSL had an integrated component design tool. Although it was said that components written in any programming language could make use of the RSL, the developed system worked only with Ada programmes. The system was not able to cooperate with different partners and was developed in a single user mode. No special advisory functionality was presented.

2.1.3 GURU

This system was constructed in 1989 and considered that, due to the costs of redoing, a manual classification of reusable software components could not be commercially viable. The authors proposed an automatic mechanism for recovering and saving a description of the components [MaSm89, Maar91, MBK91]. A full text indexing was generated using the existing electronic documentation for each component. This data was than organized through a cluster technique generating a specific vocabulary. The search for components was done based on this self generated vocabulary and on natural language. In this second case, the proposed phrases should be reworked in order to obtain keywords that could fit to the pre-existent vocabulary. For the case of an unsuccessful search, the system offered a basic navigation mechanism enabling the consultation of similar components. The system was developed to work with proprietary software programmes. The tests presented were done with executable software components written for a Unix-like operating system called AIX. No help, but only the original component's documentation, was offered for the users. No comparison between components, or additional information about their applicability, was available.

2.1.4 Kiosk

Kiosk was a set of hypertext-based tools developed in the Hewlett Packard Laboratories in California in 1991 [CFG91]. The main goal was to improve the selection process of components. The term component was understood as all related software work-products, comprising for example source code, tests documentation, and design notes. No specific programming language was specified as compulsory in the component construction. A key

element of Kiosk was a simple, general purpose hypertext system that allowed the generation of non-intrusive links between nodes. The hypertext structure for the libraries was automatically generated from a batch tool called Cost++, which used a description of the components and the desired classification to create links. Nodes were understood as Unix text files plus all links that pointed into or out of those files. Editing mechanisms were also provided allowing standard operations such as inserting and deleting text and new links. Nodes and links were represented using a proper model with an in-memory representation. Kiosk offered limited browsing and navigation mechanisms, and a full-text pattern matching procedure used for selecting reuse candidates. The insertion of new elements was not allowed for every user, demanding special rights and additional help. Mechanisms for the evaluation, understandability and quality measures of components were not particularly considered and were based only at the original component's documentation.

2.1.5 Large Software System Information Environment

The system called LaSSIE was developed by Devanu [DBSB91] at the AT&T Bell Labs in 1991. This was an attempt to build software information systems that integrated architectural, conceptual, and code views of a large software system into a knowledge base. This knowledge base was built using a classification based knowledge representation language, called KANDOR, to provide the retrieval semantics. The knowledge base was constructed with three different views of the reusable object: the code, the architecture, and the application domain. It provided a user interface with a graphical browser and a natural language query processing system. The data is automatically classified during its insertion in the database. Nevertheless, a manual insertion of the component itself and all related data was necessary. All limitations of the KANDOR were inherited by LaSSIE. The classification algorithm, well suited for telecommunication components, was not fast and was difficult to be adequate enough for new application domains. No help for understanding the pre-selected component was given.

2.1.6 Artificial Intelligence based Reuse System

The AIRS knowledge based software library was developed at the University of Maryland in 1992 [OHPB92]. That system allowed developers to browse a software library, searching components that best met some stated requirement. A component was described by a set of pairs (feature, term), that could represent all available software artifacts, such as sub-programmes, procedures, and functions. A feature represented a classification criterion, and was defined by a set of related terms. The system allowed the representation of packages (logical units that group a set of components), which were also described in terms of features. Candidates for reuse, components or packages, were selected from the library based on the degree of similarity between their descriptions and a given target description. Similarity was quantified by a nonnegative magnitude (distance) proportional to the effort required to obtain a candidate.

Based on this distance, the user could decide on the direct reuse of a component. This distance also gave an idea of the amount of work necessary for a possible modification on the component for reuse. The functionality of a prototype implementation of the AIRS system was illustrated by its usage at two different software libraries, a set of Ada packages for data structure manipulation, and a set of C components for use in command, control, and information systems. The feature-term pair could not be modified and the way a distance value was obtained was not precise. As the authors said, “sometimes this value expressed the intuition of the component developer” [OHPB92]. The system was neither scaleable nor easy to adapt to other domains. No additional help for the understanding of reuse candidates was offered.

2.1.7 Software Engineering Library

SEL is a hypertext based tool for the presentation of dependencies among components through hyperlink structures and for the search of these structures firstly presented in 1994. This library, used for the management of already developed components, was proposed by Freitag [Frei94] in a cooperation work with BMW Germany. The basic idea was to introduce a separation of the software component from its functional description. A component was every reusable software artifact. Every single component was manually described by an additional document that could be linked to a document net, composing an information network. The relationship among components could be generated semi-automatically based on their own description. In order to obtain the complete relationship among components it was also necessary to manually introduce a relationship description document. The search of components could not be done based on a semantic point of view. That means that users could not search for a specific application domain, or for a particular characteristic of a component. The possibility of navigating through the whole system was also not considered. The search was implemented based on expected related keywords resulting in links to similar components able to fulfil the desired search criterion. Besides the general description and the links, no additional information about the component itself, its usability or quality evaluation was given.

2.1.8 Multimedia Oriented Repository Environment Plus

This commercial tool also called MOREplus was proposed in 1994 and was used to create web-based software libraries. It was commercialized by MountainNet Inc., and it was an evolution of the MORE environment [EMD94], a tool resulted from the NASA’s RBSE-Project (Repository Based Software Engineering). The RBSE was a public repository of thousands of software engineering information and application source artifacts with over one thousand remote users [Eich94]. MOREplus was supposed to work with self-developed artifacts and with components from other manufactures. The component representation was based on a metadata structure which should be filled in manually. The information stored in its underlying database was not the artifacts themselves, but rather information concerning those stored artifacts which were

obtained by other mechanisms. The users-interfaces were based on HTML pages developed with TCL scripts. Diverse users or user groups had some limitation of privileges, what enabled the implementation of different views based on this grouping classification. The search was implemented by single navigation on the data or by natural language query on the metadata. There was no information about the usage of the components, no active help or notes about the best way of reuse.

2.1.9 Federal Reuse Repository

The FRR system was a web-based Reusable Software Library (RSL) interface and searching tool implemented using the web-browser Mosaic presented in 1994. Mosaic provided a simple, easy-to-use method to find and extract reusable assets from a RSL, allowed distributed access to assets from a variety of platforms, and could support most of the features of other formal RSLs without any modification [PoWe94, PoWe95]. At the FRR, components represented abstract data types, system utilities, Application Programming Interfaces (APIs), and other general-purpose functions. The search and the data presentation mechanisms were implemented through a structured abstract that was generated and stored manually. The main efforts were done in the generation of a friendly interface for the search of software components and for the respective documentation. A user could search for a component using a hierarchical view, arranged by language or library, by subject listing, or by keywords. Through the use of HTML formularies, functions normally found in commercial-grade RSLs, such as component searching, user registration, and problem reporting, were also implemented. Automatic generation of HTML pages and the use of command scripts were further allowed to provide different views of the RSL, such as searching by subject. The integration of the RSL with a Wide Area Information Search (WAIS) provided a searching functionality based on keywords. Nothing else but the original documentation was added, resulting in no additional help, nor comments about the usability of the stored software components.

2.1.10 Specification and Signature Matching

Specification matching is a way to compare two software components, based on descriptions of the component's behavior presented in 1995. It was supposed to help determine whether one component could be substituted by another or how it could be modified to fit to new requirements. Signature matching is a method for organizing, navigating through, and retrieving from software libraries based on formal specifications [MoWi95a, MoWi95b]. A formal specification has an interface described by a signature, and a behavior specified through pre- and post- conditions. Those systems worked only with components that obey these specifications. Two kinds of software items were considered as components, functions and modules, and hence two kinds of matching, function matching and module matching. The signature of a function was simply its type, and the signature of a module was a multi-set of user-defined types and a

multi-set of function signatures. For both, functions and modules, were not considered an exact match but also various flavors of relaxed matches. As the tools were based on the formal definition of a component model, the collection of necessary information could be done automatically. A good level of knowledge by the user was also presumed because no facilities for searching or deciding about the reuse of a component was provided.

2.1.11 Feature Oriented Classification System

This approach was also known as FOCS and was a graphical prototype developed by Börstler in 1995 [Börs95] based on an extension of the facet classification [PrFr87] with additional functionalities for the finding of similar components, and for the entry and query through textual description. Components here were any reusable piece of software that could be classified by the facet approach. The feature oriented classification and the component description were graphically modeled and stored at the GRAS graphical database [KSW92] in a manual way. A comparison scheme for the stored components was not offered, which made any kind of evaluation of similar components impossible. The entry of components' description and the later search for stored components were done through a syntax oriented text editor. The help mechanism was limited. No information about the usability, nor additional links to the developers of the stored components were presented.

2.1.12 Asset Source for Software Engineering

The Asset Source for Software Engineering tool was an internet library designed to work with different software-related reusable products also called DIRECT-ASSET and firstly presented in 1995. The Digital Resource Catalogue (DIRECT) was an internet catalogue developed by the Science Applications International Corporation commercially offered to the public through the ASSET-Server [DRC02]. The development of ASSET started in 1991 at DARPA - Defense Advanced Research Project Agency, as part of the programme STARS. That was simply an electronic catalogue with direct query procedures. The result of such a query, was a list with components that fulfiled the proposed requirements. The requirements could be called up based on some valid keywords. This catalogue worked not only with software components but also with products related to the reuse of software. That included for example, technical papers and proceedings of conferences that treated the theme "reuse". This model was developed to enable the data transfer among different software libraries. The Assets were classified using a pre-defined set of keywords and also based on their programming language. The insertion of components was done manually and the system was planned to work with components from any developers wanting to publish their work. A commercial version including electronic commerce functions was planned. The introduction of documents and papers related to the components was a very interesting feature of this tool. The search and finding of desired components had no built-in help mechanism.

2.1.13 Java Repository

The Java Repository approach was an electronic marketplace for Java classes and Java related artifacts based on a fixed description scheme proposed in 1996 [BKR96]. The main objective was the exchange of free Java resources between developers and users. The resources were described by some obligatory fields such as product category and pre-defined keywords. Some of the stored information was the internet address of the developer, a reference to pre-defined categories, and some relevant keywords. All information about the components was introduced manually by their developers. The user of a resource could evaluate it through a post-defined quality grade (a judgement scale from 1 to 10). This grade contained no quantitative criteria, making its usage sometimes dubious. The grade was presented to other potential users in order to support their decisions about reusing a component. The selection was done based on the searching of terms already stored in the description text or based on the browsing of the available categories. A measurement for the system's usage and a registration of products were part of this system. No additional help for understanding the components was given. Some additional characteristics such as e-commerce were announced to be integrated in the future.

2.1.14 Software Asset Library Management System

This system also called SALMS was a client/server application with graphical user interface for the classification and exact finding of components, and for the collection of related components proposed in 1997 [Soda97]. SALMS was developed by Sodalia, a joint venture of Bell Atlantics and Telecom Italia. Its goal was the classification, description, and searching of reusable software components. A component alone was denominated an Artifact while a set of logically related components had the name Asset. Assets were classified into SALMS using a so-called faceted scheme. For each Asset, the library administrator could manually select a particular keyword (or term) which best characterized the asset in a given perspective (or facet). Facets were organized as hierarchies of terms, from the most general down to the most specific term. Facets and terms were editable in SALMS in order to customize the tool to a particular environment. A set of information about the asset and its artifacts were provided in the descriptors. Indeed, information such as an overall description of the asset, its author, size, level of reusability, level of testing (hence of quality), data about who maintained it and under which conditions, previous reuse experience reports, etc, were added. Some retrieving mechanisms were offered by SALMS such as, direct access by specifying the unique identifier, filtered access by specifying a filtering criteria on asset attributes, text based access, navigation in the asset relationship network, and facet access by specifying a term for each facet or relaxing the search towards similar keywords. No help or understandability mechanisms were offered.

2.1.15 A Search Engine for Software Components

The code-name AGORA was attributed to this system which was a specialized search engine able to generate and automatically index a world-wide data base of software products. This prototype system was developed by the Software Engineering Institute at Carnegie Mellon University in 1997 [SHW98]. After searching for components, a database is built by AGORA where the found components are classified by their construction types. Users of this system could search for components in the created database describing specific properties of a component's interface. The system combined Web searching engines with an introspection process. Introspection describes the capability of components to provide information about their own interfaces. AGORA was specially successful when working with JavaBeans and CORBA components, and supported two basic processes, data collection, and data search and retrieval. Data collection consists of two new sub-processes, location and indexing. Location involves finding components on the internet while indexing means collecting interface information about the already found components, and recording this information in a local database. Based on this information a component index was constructed. The data collection process was primarily an automated background task, while search and retrieve were typically performed by humans. There was no feature to help users to evaluate the real applicability of a component. The definition of a component was also strongly attached to the JavaBeans and CORBA models.

2.1.16 Summary-based Object Oriented Reuse Library System

This tool also called SOORLS was developed at the Department of Computer Science and Engineering, at the Arizona State University in 1999 [LeUr99]. Its main goal was to support librarians, who managed data bases of object oriented reusable components, and software developers who intended to reuse these components in the development of new software applications. In this approach classes and interfaces of the Java programming language were considered as reusable components. By parsing the declaration of such a component, SOORLS automatically extracted information such as the access modifier of the component, its return type, name, and structural related information acquiring knowledge about the relationship among other components. This information was stored in the form of a descriptor and a declaration for each component. An automatic indexing process that encoded the properties of the components was applied to the descriptors and declarations of each available component. Similar components were then grouped in a classification process. Reusable candidates were retrieved based on this classification, allowing the future browsing by category of the same purpose components. This query was based on input data given by the user. The system offered also a post-comprehension procedure in order to enable a better understandability of pre-select components. It gave users more detailed information about the selected components. The automatic retrieval of component information proved to work with Java-based software artifacts, but it was not extended to other types of components.

2.1.17 Komponenteninformationsystem

This system also known as KIS was a cooperative information system for software components developed at the Department of Computer Science III at the University of Aachen, Germany, in 1999 [Behl99]. This decentralized system used the Internet as intercommunication media. The original project was a cooperation work among diverse local industries and the University of Aachen [Behl98]. Its goal was the increase of the reuse of components offered by diverse developers through the distribution of a comprehensive amount of information about each software artifact. Software components, also called reusable assets, were understood as commercial off-the-shelf software (COTS) and public domain software or shareware, which could be provided by any interested developer. The representation of components was based on a pre-defined metaschema where all necessary information about the component was determined previously. The functionality of the system was divided into three main parts according to the users' needs. Users are system developers searching for reusable components, component developers offering their components, and someone with the role of the manager maintaining the system accordingly. The search for components could be done by the selection of pre-defined categories through a browsing system, or could be based on the search of keywords. The insertion of new components was supported by a man-machine interface that demanded the introduction of a series of compulsory data. Every single interaction with the system was done via the Internet. An automatic search for new components in the Web and an automatic retrieval of the related data was also integrated. The system offered the possibility of communication between a developer and a user of components through electronic media.

2.1.18 Some Other Approaches

Although the most relevant works were already presented above, some other approaches are described in this section in order to increase the completeness of this text. One of the most frequent definition of software component originated with the Ada programming language. Many authors proposed ways to improve the readability and understandability of Ada libraries. In 1987 Dausmann proposed a new library structure for reusable Ada components [Daus87]. His work made the access to components stored in different libraries possible. He also considered the possibility of adapting a component in order to apply it to diverse new needs. The proposed work failed to solve the problem of retrieval of a desired functionality among diverse components. In 1988 Ouwen proposed a set of tools for the storage and retrieval of reusable software components which consisted of Ada packages and procedures. He introduced a functional specification document for each component and used a faceted classification scheme [OGH88]. Royce related his experience at the Defense System Group with the Network Architecture Service (NAS), trying to improve the reusability level of over 120,000 Ada sources lines of safety critical software programmes [Royc89]. Jensen reported more successful research with the so called AdaSAGE [JSW92] which was a library that worked with Ada reusable

components trying to help to identify and to create new reusable components. One problem was that the known attempts to create a reuse environment often resulted in the increase of development time and cost. AdaSAGE tried to support the whole production and maintenance process. It also enabled better communication among project workers.

Some other authors have expressed very particular views of software management. In 1988, Damier and Defude described ESTRELLA a multimedia object-oriented data model based upon objects, classes and functions [DaDe88]. They proposed a document model capable of managing structured documents and to index them with a superimposed code method. The search and retrieval of software artifacts were implemented based on an Oracle database. Multimedia here was very far from the actual concept and meant only the access of different text files. Beckman described his Encyclopedia of Software Components in 1991. That was one of the first attempts to integrate a structured search by interactive browsing, a goal directed search based on keywords query, and an automated component insertion as hypermedia data, in one unique system [BBJ+91]. In 1992 Swanson and Samadzadeh identified that one of the most frequent problems in the reuse of components was the inefficient use of the library [SwSa92]. That led programmers not to use such new systems either as customers or as contributors. With their catalogue interface, they tried to solve this problem for a library of C functions. A set of additional files were organized in order to facilitate the man-machine interaction.

Another line of research tried to propose ways of explaining the meaning of management of components. In 1993 Podgurski and Pierce proposed a method called behavior sampling for automated retrieval of reusable components [PoPi93]. Behavior sampling was able to identify relevant routines by executing reusable candidates on a search procedure. The candidates were selected based on a sample of operational inputs and outputs given by the user, and the comparison of the correspondent obtained results. Dellarocas believes that in component based software development the identification and proper management of interconnections among the participating pieces is the central concern for improving the reusability of components [Dell97a, Dell97b]. With his environment called SYNTHESIS, he tried to support software development through the representation and management of interdependencies among software components. He proposed a design handbook of software component interconnections, which catalogued common software interconnection dependencies and sets of alternative protocols for managing them.

2.2 Comparison of the Approaches Presented

In the last two decades many researchers tried to increase the reusability of diverse software artifacts (programmes, modules, functions, objects, and components) introducing libraries or other management approaches. The concept of what was really a component and the ways one could improve its reusability varied in a very wide range. A summary of the main characteristics

of the most relevant commented approaches is presented in Table 2.1. Five points are analyzed, the type of each component, the way components are represented, how the necessary component's data is acquired, the way somebody can find desired information, and what kind of support is given for the reuse.

As seen in Table 2.1, some approaches were dedicated for components that fulfil a particular technology specification, for example the Java Repository system worked only with software components written in Java. A great number of systems were able to treat only self-developed components, on the other hand only a few did not create their own components, and three approaches were able to work both with self-developed and third-party components. Almost all presented systems tried to be as general as possible, dealing with software components independently from a specific domain.

The second sub-title of the table deals with the form a software component was represented with at each system. This representation form enables diverse ways of divulging later information about the stored software artifacts. It also influences the process of storing components. The most used representation scheme was based on a classification structure where the components were related to specific pre-defined items. Some approaches accepted a post insertion of new categories, expanding the original classification scheme. The indexing of components, creating links among functional related pieces, was also a very frequent feature. Some systems were able to work with these two ways of representation. The formal specification of components was avoided by almost all presented systems, probably due to the difficulties in proposing a formal method that could be accepted by a large number of researchers and practitioners. Two other ways of representing a component, by a knowledge based model and by the introduction of a quality measurement were also not used overall.

Another important item is the way someone can insert new components in a component management system, or how the system obtains new elements. Here basically two different approaches exist, manually at the very first moment a component is created, or automatically making a search of new elements on third-party databases. This search for new elements is done based on pre-defined characteristics. The first approach is said to be an additional cost factor as it implies in some pre-work of specialists. The automatic retrieval of data implies the definition of a desired model (although this model might not necessarily be a formal one), able to contain all necessary information. The component itself must obey this model, otherwise it would not be included in the system.

The way users obtained information about a desired component also varied. The most popular form was the browsing through a classification scheme. This browsing returned a list of probable items to be reused. As the browsing was usually based only on general points, the list of candidates could be longer than desired, making the reuse process harder and sometimes not viable. Direct retrieval of candidates based on some desired characteristics was not easy to

implement and very inefficient as it implied previous knowledge about the stored items. Nevertheless, this way of recovering components can be improved by the creation of advisory support for the users. That can be implemented, for example, by using a set of pre-defined keywords, already presented in many systems. These keywords must be carefully defined as they are supposed to describe the essential characteristics of a component. The presentation of components that attended only partially the original requirements of the user was straightforward. This option led to the identification of components that through modification was able to be successfully reused.

Table 2.1: Comparison of Management of Components Approaches.

	Type of Components				**Representation**					**Data Acquisition**		**Searching and Retrieving**				**Reuse Support**			
Technical Characteristics / **Presented Approaches**	Domain Specific	Technology Specific	Proprietary	Third Party	Classification	Indexing	Formal Specification	Knowledge Based	Quality Measure	Manual	Automatic	Browsing	Direct Retrieve	Pre-defined Keywords	Similarity	General Description	Quality Evaluation	Active Help	Understandability
1- GAMS	X	X			X					X		X				X			
2- RSL			X		X					X	X			X	X	X	X		
3- GURU		X	X			X					X			X	X	X			
4- Kiosk			X	X	X						X	X			X	X			
5- LaSSIE			X		X			X		X	X	X			X	X			
6- AIRS		X	X					X		X		X	X		X				
7- SEL			X			X				X	X			X	X	X			
8- MOREplus			X	X	X					X		X		X		X			
9- FRR			X		X	X				X		X	X	X		X			
10- Sig. Matching			X				X				X		X		X				
11- FOCS			X		X					X			X		X				
12- DIRECT				X	X					X		X		X		X			X
13- Java Repository		X		X	X					X		X		X		X	X		
14- SALMS			X		X				X	X		X		X		X	X		
15- AGORA		X		X		X					X	X			X				
16- SOORLS		X	X		X	X					X	X			X	X			X
17- KIS			X	X	X	X				X	X	X		X		X		X	

The last characteristic shown in Table 2.1 deals with the form developers tried to support users to understand and reuse existent components. That is a decisive factor on the quality of the component management system. Almost the total of the presented approaches offered only a general description of the software component. This description was arbitrarily imposed and sometimes did not fulfil the users' expectations. Proposals to evaluate the quality of the components, or component presentation, in the aspect of reuse, were rarely presented. Proposals to improve the understandability of components were also rare. The idea of including an active help facility, based on the exchange of electronic mails, was only recently suggested.

2.3 Management of Hardware Components

The characteristics of the hardware components industry, where great companies control the market, make the approaches for the management of these components quite different from the one proposed for software components. Moreover, it was not possible to find any specific reference in publications about the topic management of hardware components. Nevertheless, it was possible to verify that there are two approaches in the market that have similar characteristics to the management of software components. They are the hardware components catalogues and the hardware components simulators, the main characteristics of both approaches are described below.

- Hardware Components Catalogues: In these applications a complete list of electronic hardware ships of a specific producer is presented. Examples of well designed catalogues may be found in the Web-pages of Philips Semiconductors [Phil02] and in the Web-page of Intel Company [Inte02]. In general, the hardware components presented in these catalogues are divided in terms of well defined application domains. They also are grouped according to their basic technology (digital or analog electronic components) and their implemented functionality. For example, it is possible to select all digital electronic components that implement one OR gate in the catalogue of Philips. Users can receive the complete information about one hardware component in the form of a data sheet. These electronic data sheets reproduce their own paper version exactly. The insertion of new components is restricted to the maintainers of the catalogue and, consequently, normal users have no access to this insertion mechanism. As the hardware component catalogues are developed to attend the necessities of a particular company they do not compare similar components of third party companies.

- Hardware Components Simulators: The electronic simulators available on the market contain a basic library of components simulations which may be used during the design of electronic circuits. Examples may be found at the following Web-pages [EWB02, DLS02]. It is normally possible to design larger components using the ones available on the simulation environment and to store these new constructed components into self defined

directories. The functionality of the resulting component may be then tested having their behavior registered in the form of signal waves. These new simulation circuits can be stored later. Similar procedures such as the ones used by the hardware electronic catalogues, where components are grouped in terms of their application domain, technology, and functionality, may be adopted by the developer of new simulation components. Nevertheless, only simple mechanisms to document these new components are available. The creation of the directories and the correspondent maintenance of new components is the responsibility of the users themselves. It is also usual to buy a new set of components (called libraries) designed to solve problems of a particular application domain, but normally the libraries designed for a specific simulator do not work in other simulators offered by third parties. In contrast to the design of software components, the design of a simulation of a hardware component on simulation environments is far from the construction of the real hardware elements.

The study of catalogues and simulators of hardware components available on the market done during this thesis was a profitable source of ideas. The hardware electronic industry has been working for a long time based mainly on components. Many concepts applied in the software components approaches were inspired by this industry. The flexible management of components proposed in this thesis will also be able to deal with the information about hardware components with application in the industrial automation domain.

2.4 Lessons Learned

None of the presented approaches were able to completely fulfil the requirements of a component management system specially designed for industrial automation components. In spite of it, they have many good characteristics that may be incorporated in the concept of such a system. Some relevant technical conclusions about those approaches are presented in this section, as well as a description of the previously exposed ideas that might be reused and the kind of mistakes that might be avoided in future research.

Almost all presented researches have tried to deal with a large range of different components. There is no problem with this consideration whenever the components can be represented by a common model. The absence of a common specific model proved to be one serious drawback, because it made it impossible to work under a common platform. The correct retrieval of a component based on a wide range of models is not easily done. On the other hand, the adoption of only a specific technology of components limits the usability of any management concept. Additionally, it is nowadays not possible to affirm that a unique component technology will ever be able to accomplish all requirements of industrial automation applications. It is necessary to be able to work with different technologies. One specific characteristic of the concept presented in

this thesis is the use of a flexible common presentation model where all important characteristics of an industrial automation component must be considered.

The classification of components based on pre-defined categories was very well used and it is still an interesting way of retrieving components from a repository. These classification schemes made the indexing of components and the respective finding of similar ones possible. Approaches using a formal specification model were less successful mainly due to their intrinsic difficulties to propose a model that could be accepted by a large number of users. Approaches based on formal knowledge representation models were difficult to generate because they normally demanded an automatic retrieval of information that is not always possible.

Every single component must be classified based on a domain specific classification scheme. Some authors have avoided this procedure attributing an insupportable cost increase. The same authors tried to develop sophisticated data acquisition algorithms, but the lack of clarity made the later finding and reusing of components a hard task. In the assumptions of this thesis, it is considered that the cost of classifying a new component is not a deciding factor. Indeed, a meaningful classification is a key concern on the later reuse of components. The system must help users to classify their components using an active support tool based on the common domain model.

The finding of components based on the navigation of existent descriptions of components was used overall. The advantages of this approach was that users could learn from the navigation and as much as they searched for components as easy was to use the system and to find appropriate components. This characteristic shall be maintained in future works, but should not be the only way of finding a component. It is interesting to include the possibility of retrieving a component directly based on any known characteristics. Pre-defined keywords closely related to the domain are another very interesting way of finding one desired functionality.

A very important factor increasing the possibility of finding and reusing components is the support to understand their correct application. As a matter of fact, the presentation of a general usage description is not enough to guarantee a good level of comprehension and further reuse of a component. The available documentation must be organized in a well structured way facilitating its access. The introduction of mechanisms of exchanging information between component developers and component users is very interesting as it permits an active interaction between them. A concrete example of such interaction can be represented by the introduction of a component’s quality evaluation generated by users. Lastly, the component management system must be easy to use.

As seen in this chapter, the general approaches for the management of components have as a main goal the increase of reusability of already developed software components. Although many different attempts were proposed, none of them could fulfil all requirements of a meaningful component management system for industrial automation components. Mainly two factors were

decisive for this undesired level of acceptance: firstly, developers tried to offer systems so general and complex that they could never really foresee all involved problems, and secondly, only recently the support for users was implemented in an acceptable level. These two problems are considered in this thesis where a particular management solution for components used in industrial automation applications will be presented. Additionally, the potentiality of internet-based systems are investigated in order to make the accessibility and usability of the final component management system easier.

The conception of the flexible web-based component management system for industrial automation presented here will have some common points with the researches cited in this chapter. The main objective of this thesis is to investigate how to efficiently store, find, understand, evaluate, and finally decide about the reuse of components. This research was concentrated on a specific domain, the industrial automation applications, leading to a flexible common model able to describe components used in this domain. All users involved in the reuse process must also be supported, component developers by the insertion of new components, component users by the searching, finding and evaluating of components. The idea of an active advisory system, where users are conducted all over the process, is also considered.

In order to propose a meaningful component management system it is necessary to know precisely the characteristics of the elements being managed. One question that remains to be answered is if there are components being used in the industrial automation domain and, if so, what these components are like. The research developed along the years of this thesis shown that many different component approaches currently available have particular usage in industrial automation systems. For example, many of the most successful component models designed by market leaders as Microsoft and Sun Microsystems have been used to realize industrial automation applications. Other component approaches were designed exclusively for that domain and were proposed by the electronic devices industry or by research groups inside universities. The characteristics of the most relevant component approaches with use in industrial automation and examples of already developed applications for the industrial automation domain are the main subject of the next chapter.

3 Component Technologies used in the Industrial Automation Domain

The main idea behind components is to construct any useful product based on the composition of smaller and elementary (or even complex) building blocks. The definition of software components is not easy at all, and as stated in [HeCo01], in order to properly understand software components technologies it is necessary to define three concepts: software component itself, component model, and software component infrastructure. Those definitions are shown below:

A **software component** is a software element that conforms to a component model and can be independently deployed and composed without modifications according to a composition standard.

A **component model** defines specific interaction and composition standards. A component model implementation is the dedicated set of executable software elements required to support the execution of components that conform to the model.

A **software component infrastructure** is a set of interacting software components designed to ensure that a software system or sub-system constructed using those components and interfaces will satisfy clearly defined performance specifications.

In order to properly manage components it is necessary to understand those three aspects mentioned above. That is the reason why during this thesis research about some technologies with any use in the industrial automation was done. The results of this research are related in this chapter where the most relevant component technologies (software component itself, component model, and software component infrastructure) with use at the desired domain are presented. Those component approaches are grouped in accordance with their origin, i.e. technologies proposed by the software industry, technologies used solely by the automation industry, and technologies proposed by university research centers. The main characteristics of each technology is discussed below, followed by some examples of usage at industrial automation applications.

3.1 Most Common Commercial Components Approaches

The most successful software companies and software organizations worldwide have their own concept about components and their usage. Some technologies were also proposed as standards but it was impossible to obtain any consensus about only one technology being recognized as standard for any possible application case. In fact many component technologies live together

sharing their insertion on the market. In the following sections the most relevant commercial technologies are presented.

3.1.1 CORBA

The Common Object Request Broker Architecture (CORBA) was developed as a standard distributed object architecture by Object Management Group (OMG). CORBA objects differ from typical programming language objects because they can be located anywhere on a network, and can operate with objects written to run on other platforms [Szyp98, HeCo01]. CORBA objects can also be written in any programming language whenever there is a mapping from the Interface Description Language (IDL) to the desired language platform. This technology works as the object-oriented middleware architecture in heterogeneous distributed object systems. The server provides a remote interface and the client calls this remote interface. The architecture is built around three key building blocks:

- OMG Interface Definition Language (OMG IDL), which is used to define object types by specifying their interfaces. An interface consists of a set of operations and the parameters of those operations. This interface definition is independent of the programming language used, but maps to all of the popular programming languages via a set of OMG standards, for instance C, C++, Java, COBOL, Smalltalk, Ada, Lisp, and IDLscript were all mapped by the OMG standards.
- The Object Request Broker (ORB) is responsible for all mechanisms required to find the requested object implementation, to prepare the object implementation to receive the request, and to communicate the data completing the request.
- The Internet Inter-ORB Protocol (IIOP) is the standard communication protocol that allows vendors of ORBs to interact over the Internet successfully. It is a representation to specify the target objects, operations, and all parameters of every type that client's ORB and object's ORB may use.

Figure 3.1 shows how the CORBA architecture is implemented and works.

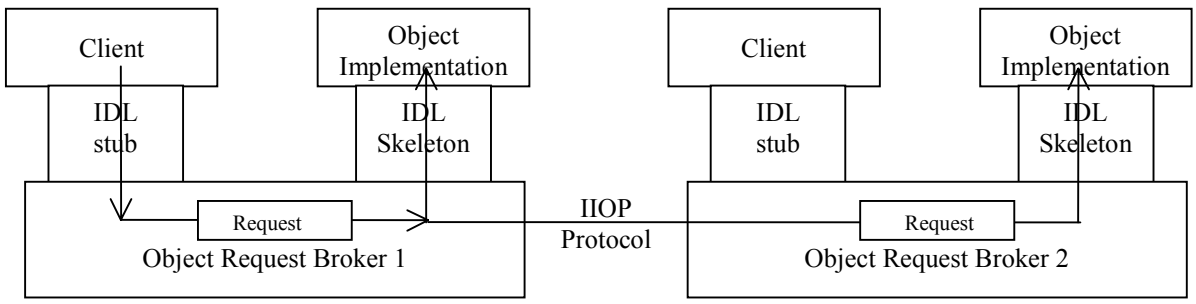


Figure 3.1: Request from Client to Object and from ORB to ORB Communication.

Stubs and skeletons work as proxies for clients and servers, respectively. IDL defines interfaces so strictly that the stub on the client side has no trouble meshing with the skeleton on the server side. In CORBA, every object instance has its own unique object reference, which is the information needed to specify an object within an ORB. Clients use the object references to direct their invocations, identifying to the ORB the exact instance they are supposed to invoke. The object implementation receives a request through the IDL skeleton. The object implementation provides the semantics of the object, usually by defining data for the object instance and code for the object's methods. In case of a remote invocation, the client first obtains its object reference. When the ORB examines the object reference and discovers that the target object is remote, it marshals the arguments and routes the invocation out over the network to the remote object's ORB. In this case, the client's ORB and object's ORB must agree on the standard protocol IIOP.

CORBA has been used in automation systems and electronic devices in many situations. Such applications generally comprise of many software objects that interact with each other. Specialized versions of CORBA run real-time systems, and small embedded systems. For example, Foliage Software Systems, an American software company [Foli02], developed an upgrade package for an old generation of a commercial electron-beam lithography system used to make semiconductor masks based on CORBA. The older system was written in FORTRAN and used a 1980's minicomputer as its control system. The upgrade package reused the material handling, vacuum control, and mechanical-drive stage components from the original system, while replacing the electron-beam column, control system, and pasteurization engine. This upgrade package more than doubled the placement accuracy, while increasing the system speed by a factor of eight. This is a high-precision, high-speed system with placement accuracy measured in billionths of meters, and throughput measured in tens of millions of images processed per second. The system was developed using C++ and CORBA in a distributed, heterogeneous environment incorporating systems running Windows-NT, Unix, as well as other platforms.

3.1.2 COM and DCOM

COM stands for Component Object Model, whereas DCOM stands for Distributed COM which is an extension of COM, both were developed by Microsoft Corporation. COM and DCOM provide a framework for supporting communication and interaction among programmes across processes and machine boundaries. COM allows software components to interact via interfaces which are described by an Interface Description Language (IDL), this description enables the interfaces to be used by various programming languages [Stal98]. An object that implements an interface and provides COM binary-compliant pointers to a table of function is shown in Figure 3.2.

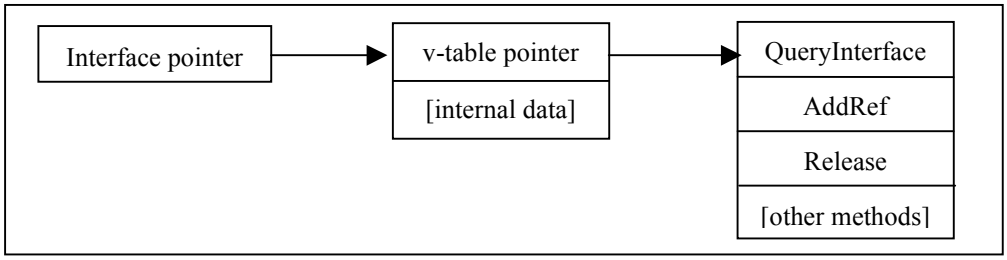


Figure 3.2: COM Interface Representation.

COM makes those functions available to any client who asks for a pointer to the interface, whether the client is inside or outside of the process that implements those functions [GHL+98]. Objects with multiple interfaces can provide pointers to more than one function table. The three elements of the v-table have the following functionalities. The QueryInterface takes the name of an interface, checks whether the current COM object supports the interface and if so, returns pointers to the supported interfaces. AddRef increments reference count and lastly, Release decrements reference count to enable the server to know when all clients have finished using the interface. Distributed COM transparently expands the concepts and services of COM. COM defines how components and their clients interact. DCOM supports communication among objects on different computers or machines [Carn97]:

Several features in COM and DCOM satisfy and simplify some constraints in distributed computing system. DCOM completely hides the location of a component, whether it is in the same process as the client or on a machine halfway around the world. In all cases, the way a client connects to a component and calls the component's methods is identical. Not only does DCOM require no changes to the source code, it does not even require that the programme be recompiled. A simple reconfiguration changes the way components connect to each other. DCOM is completely language-independent, application developers can choose the tools and languages that they are most familiar with. These technologies manage connections to components by maintaining a reference count on each component. DCOM increments the components reference count whenever a client connects to a component and decrements reference count whenever the client releases its connection. If it detects a broken connection, DCOM decrements the reference count and releases the component if the counter has reached zero. DCOM can also use any transport protocol, including TCP/IP, UDP, IPX/SPX and NetBIOS. It provides a security framework on all of these protocols, including connectionless and connection-oriented protocols.

COM and DCOM have been used to build control applications in automation systems, as for example the “Distributed Control System for Semiconductor Processing Tool” [Foli02]. It is a distributed control system architecture for a precise manufacturing tool used in the semiconductor manufacturing process. The tool has more than 1,500 analog and digital I/O points, as well as intelligent motion controllers, embedded DSPs, a variety of serial I/O devices and multiple CPUs, and requires five to twenty millisecond sustained response servicing. The

architecture is designed around DCOM and supports multi-threading, distributed I/O, high-speed processing, client-server data storage, and an internationalized, ActiveX-based, touch-screen user interface. This application was designed primarily in C++ for Windows NT and incorporated an assortment of development tools, libraries, databases, and embeddable components.

3.1.3 JavaBeans and Enterprise JavaBeans

The Java platform has been used worldwide in the industry for some years, and today more than one million developers around the world have already deployed a Java application [DeDe99]. The component technology JavaBeans is pointed as one meaningful solution to deliver highly reusable components due to its interoperability standard. The JavaBeans component architecture is a portable and platform-independent architecture for the Java application environment. JavaBeans component is written just the same way as any other Java class [Trac97]. Moreover, the existing components, or even Java classes can be turned into JavaBeans.

In fact, a JavaBean is a component that allows developers to reap the benefits of rapid application development in Java by assembling predefined software components to create powerful applications and applets with use in many different areas. Graphical programming and design environments that support beans provide programmers with flexibility by allowing programmers to reuse and integrate existing disparate components that in many cases were never intended to be used together. These components can be linked together to create applets, applications, or even new Beans being reused in new applications.

A Bean is comprised of two primary elements: data and methods. The data part of a Bean completely describes the state of the Bean, whereas the methods act on this data. A JavaBeans component may have methods with different types of access. For instance, private methods are accessible only within the internals of a Bean, whereas protected methods are accessible both internally and in derived Beans. The methods with the most accessibility are public methods, which are accessible internally, from derived Beans, and from outside parties such as applications and other components. These public methods are grouped to form the component's interface. The Bean interacts with the outside world through such interfaces using specific protocols. Therefore, programmers just need to know the interfaces of a Bean to be able to manipulate and interact with it. They do not need to know any detail of the internal implementation of the Bean itself. Figure 3.3 represents this architecture.

A powerful advantage of JavaBeans is that they are able to integrate existing component models to work with the other component architectures, for example, CORBA, ActiveX, etc. JavaBeans were originally designed for user-interface and client-side tasks but their functionalities have been extended to support distributed computing and server-side features by the introduction of the Enterprise JavaBeans (EJB) [Dail01, Zona99]. EJB components allow applications to

communicate across multitiered client and server environments, and across Internet and Intranet structures.

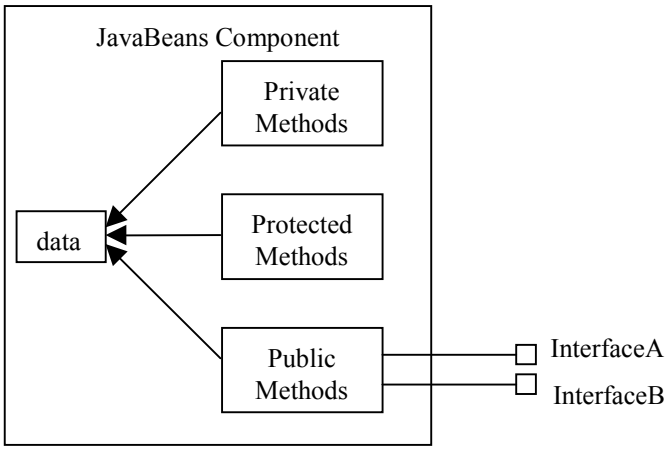


Figure 3.3: Relationship Between Interfaces and Methods in a JavaBeans Component.

JavaBeans and Enterprise JavaBeans have been used to develop software components with applications in industrial automation systems for some years. For example, ErgoTech Systems, Inc., a leading independent developer of component software for manufacturing, factory automation and machine control in the USA, announced the release of its Professional Edition Virtual Instrumentation Beans™ product in 1998 [Ergo98]. VIB™ is a suite of JavaBeans used as components for building Java graphical user interfaces for real-time data access and display. The VIB products are used to develop graphical user interfaces for industrial, laboratory and building monitoring, control and automation. Their Professional Edition eases the creation of client-server solutions, the user interfaces can be distributed over a network and viewed remotely. This suite composed of JavaBeans components creates meters, strip charts, active input devices (buttons, knobs, sliders), bar charts, seven-segment displays, among other technical devices. Additional components for building automation applications offered by this company include components such as switches, smoke alarms and motion detectors.

3.1.4 Microsoft .NET

The Microsoft .NET technology is based on the Extensible Markup Language (XML) Web services platform. It enables users to interact with a broad range of smart devices via the Web, while ensuring that the user, rather than the application, controls the interaction [Micr01a]. This platform consists of the following core elements [Micr01b]:

- .NET infrastructure and tools which are used to build and operate this new generation of services. It is composed of four main parts: the Microsoft Visual Studio.NET, the Microsoft .NET Enterprise Servers, the .NET Framework, and the Microsoft Windows.NET.

- .NET services are an integrated set of building block services for Internet operating systems, including Passport.NET (for user authentication and services for file storage), user preference management, calendar management, and many other functions.
- .NET user experience network that is a broad and adaptive user experience platform, where information is delivered in a variety of ways on a variety of different devices.
- .NET device software that enables a new breed of smart Internet devices that can leverage Web services.

The core architecture of the Microsoft .NET platform is the so-called .NET framework, which is an environment for developing, deploying, and running Web services and other applications. It consists of three main parts: the Common Language Runtime, the Framework Classes, and the ASP.NET [Micr01c]. The Common Language Runtime (CLR) allows multi-languages components to work together. CLR is able to execute programmes written in many different languages, therefore it can run applications written using multiple languages within a single application. The second part of .NET framework is a common set of class libraries that works with any language in order to facilitate interoperability between languages. With Visual Studio.NET, developers are able to debug applications written across languages, many third party companies intend to provide languages and tools to work with it and the .NET framework architecture. The third element is the extended Active Server Pages, called ASP+, which provides a high-level application model for building Web applications and services.

This technology shall be important in the near future being widely used in embedded systems. The Microsoft's .NET vision entails a world of mobile users, with PDAs and mobile phones as standard access devices [Micr00, Micr01d]. Someone using Visual Studio.NET to develop mobile solutions may receive a derivative of the Web forms technologies (called mobile forms) from Microsoft. Mobile forms are designed to allow device-independent application development, so for example, the same mobile form will work with a Nokia 7110 mobile phone and with a Palm PDA, as it will deliver content based on the type of the device accessing an application.

3.2 Proprietary Industrial Components

Sometimes the characteristics of a business area forces industries to adopt particular production methodologies and tools better suited for their needs. The embedded electronic industry is one of these areas with very specific needs. In this section two component approaches used for embedded systems are presented. The first has been used in consumer electronic devices and the second in automotive electronic systems.

3.2.1 Koala

Koala is a component model developed by the Philips Research Laboratories specially to be used in embedded systems which is composed of units of design, development, and reuse, each of them designed independently of each other [OLKM00]. A Koala component communicates with its environment and with other components through interfaces, in the same way as COM and Java, i.e. a Koala interface is a small set of semantically related functions. A simple IDL is used which contains functions for initializing a component, and a Component Description Language (CDL) used to describe the boundaries of each component. Interfaces are labelled with two names, a globally unique name which refers to a particular description interface repository, and a local name to refer to the particular interface instance. A configuration is a set of components connected together to form a product, where a typical configuration contains tens of components, and a typical component may contain 10 interfaces. Modules are declared within a component and connect to its interfaces or to the interfaces of its sub-components. Modules have access to any interface whose base is bound to the module itself, working like an interfaceless component that can be used to glue other interfaces.

The connection between components in Koala are implemented by the compiler at compilation time. Koala obtains a directed graph of modules, interfaces, and bindings, and it generates a header file by renaming macros. The following features of the Koala approach show how Koala can deal with diversity in the development of software:

- Interface Compatibility: It is possible to create a new interface type that contains all the functions of the previous interface plus some additional ones.
- Function Binding: This feature is used to bind functions of different names or even different interfaces by allowing a function to be bound to an expression in CDL. The generated macro will contain calls to functions of interfaces bound to that module.
- Partial Evaluation: Koala understands a subset of the C expression language and partially evaluates certain expressions.
- Diversity Interfaces: Components may be parameterized, they might not contain configuration-specific information in order to be reusable in many different platforms. In Koala, a Standard Interface Mechanism (SIM) was developed to reach this goal.
- Diversity Spreadsheets: Each component is allowed to provide an interface that contains the number of threads required in real-time kernel usage. It is also possible to use Koala to add all the threads at compilation time.
- Switches: This feature is used to handle the structural diversity in the connections between components. Koala also supports a limited form of dynamic binding in real-time kernel usage.

System developers within Philips Consumer Electronics (CE) are currently using Koala for developing embedded software in TV sets, set-top boxes, video recorders and DVD players

[Omme00]. Due to the large variety of products in the product population of Philips, many of their architecture concepts have a scope that is not global. This fact, let them create a family of components that may be easily adaptable to regional requirements through the Koala component approach [Omme01].

3.2.2 ASCET-SD

ASCET-SD is a component-based development tool for embedded control systems, developed by ETAS GmbH [ETAS99] hosted in Stuttgart, Germany. The tool is a graphical development environment that allows target-in-dependent functional specifications, where components are represented by block diagrams. Control software for embedded systems is generated automatically from those diagrams. Target-identical prototyping is provided at early tests phases in the development cycle enabling the designer to concentrate on the physics of the embedded control system, and design the control unit in an abstract environment in order to finally generate the desired target code.

In ASCET-SD components and their functionality are described by block diagrams, state machines, text specifications, and C-interfaces all together providing design engineers with some description options for control algorithms [Flei00]. With the ASCET-SD tool, the specification of developing project is translated into C-code automatically. Afterwards, this code may be transformed into executable binary code using target-specific tools (compilers, linkers, debuggers), and loaded on the target platform. Code generation is supported in the scenarios of physics experiments, implementation probes, and controller tests.

The block description of control systems is based on interfaces in the control process. ASCET-SD blocks represent objects which encapsulate items of information and are interconnected by interfaces. The object-based presentation of ASCET-SD ensures that those blocks can be reused reliably. The block diagrams contain a purely physical presentation of control algorithms. Information about the implementation is added to the block diagram by means of editors, where the order of computation is defined. In addition to the typical data flow elements, block diagrams in ASCET-SD contain also control flow elements such as branches. The control flow is shown in separated line and link types. It is also possible to specify the processing order of block operations directly by assigning block attributes to the graphic. Complex algorithm can be presented in graphic form by means of sequencing of elementary blocks and respective control flow elements.

ASCET-SD makes use of state machines, a widespread modeling technique in use by the control systems developers community. This modeling method is of special interest for systems where different control strategies are required depending on the working point. Trigger conditions and state methods are also specified by block diagrams or even by texts. Text specification of control algorithms are implemented in the Embedded Software Description Language (ESDL),

which contains a portable physical description of the component. Instances are encapsulated in classes and modules. In ASCET-SD, all objects are fixed at compilation time. C sources can be integrated in two ways, internal C-code is managed in the ASCET-SD database in the same way as ESDL classes, whereas external C-code is stored in the user's final system being used directly for other applications. Binary code can also be integrated if programme sources are not available. In the ASCET-SD tool set, the operating system is configured visually in an editor.

The main application area of this tool are embedded systems for automotive industry. Vehicle electronics have already penetrated nearly all parts of those systems. Historically, development started with engine and transmission control, but nowadays electronic control units also includes the brakes and the suspension as well as comfort systems (cockpit features, navigation systems, etc). Knorr-Bremse [HSW01], a manufacturer of brake systems for rail and commercial vehicles, has already relied on ASCET-SD for brake systems for rail vehicles since 1998. The main reason for that decision was because ASCET-SD ensures consistency from analysis to the finished code and optimizes the interface between project engineering and software development. At the same time, it improves the quality of the code and it is compatible to the internal operating system. Knorr-Bremse was also able to save on resources by replacing certain development steps with standard ASCET-SD modules.

3.3 Component Approaches Proposed by Universities

In spite of the above mentioned approaches, component based development is one of the hottest topics of interest in many university research groups nowadays. In the next sections three component proposals developed at German universities are described. They were investigated in the scope of this thesis because they have applications in the industrial automation domain.

3.3.1 ViPER – Synchronous Components

The concept of synchronous components has been applied at the Institute of Industrial Automation and Software Engineering (IAS) at the University of Stuttgart for developing software for embedded real-time systems since 1997 [LuGu98, Gunz02]. This approach utilizes the fact that embedded real-time systems are reactive systems [HaPn85] which are computer systems interacting continuously with their environment. Synchronous languages have been developed to simplify the programming of reactive systems. They are based on the synchrony hypothesis [BeGo92] which makes the following abstractions:

- The computer is infinitely fast.
- Each reaction is instantaneous and atomic, dividing time into a sequence of discrete instants. Different reactions cannot interfere with one another.
- A system's reaction to an input appears at the same instant as the input.

A real system behaves synchronously if it is fast enough. It must always finish its computations before more events arrive from the environment. The synchrony hypothesis is a generalization of the synchronous model used for digital circuits where each reaction must be finished in one clock cycle. The synchronous model of time simplifies the design of correct systems. Temporal details are hidden during specification and so the behavior of the system is also simplified. Non-deterministic behavior caused by the interference of parallel actions cannot occur. Deterministic systems are one order of magnitude easier to specify, analyze and test than non-deterministic ones. The most important languages based upon the synchrony hypothesis are ESTEREL, LUSTRE, SIGNAL, STATECHARTS and ARGOS [HLR92, Halb93, Hare87]. ESTEREL is an imperative language while LUSTRE and SIGNAL are declarative and STATECHARTS and ARGOS are graphical languages.

On the basis of the synchrony hypothesis it is possible to define components which can be easily composed to form larger systems. Since the components communicate through signals being sent by broadcast, the components are not required to make any assumptions about each other. They work independently like software ICs. A synchronous software component consists of a reactive part and a transformational part. The reactive part is specified in a synchronous language. The transformational part is optional and consists of data-type specifications and several data-handling functions written in the host language (Figure 3.4). At the IAS' synchronous component approach, ESTEREL is used to implement the reactive part and C to implement the transformational part of the components. The interface of a component consists of input and output signals [GuNä99] as presented in the figure below.

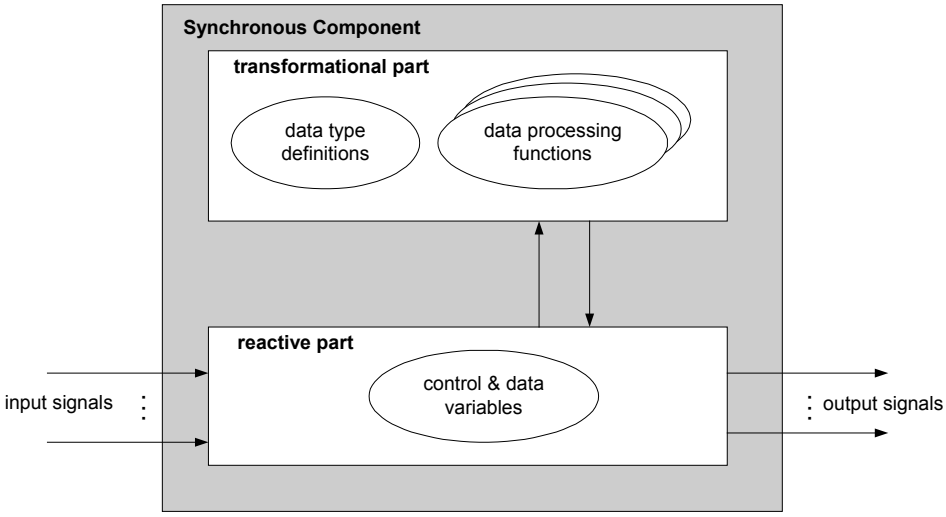


Figure 3.4: Synchronous Software Components Architecture.

ESTEREL is a synchronous language based on a mathematically precise defined semantics developed specifically for deterministic reactive systems. The programming model of ESTEREL is the specification of modules which are assumed to be executed in parallel. The modules communicate with one another and with their environment through signals. The signals

are broadcasted and can be received from each module. A signal has a state and can optionally have a value of an arbitrary type in each instant of time. Sending and receiving signals is performed instantaneously according to the synchrony hypothesis. ESTEREL only allows the specification of deterministic behavior. The input signals for each reaction step determine the output signals uniquely (and their values) to be sent in this instant as well as the input output behavior of the programme. If reactions are not instantaneous, they are as fast as they can be, because they include only actions that must be done during one cycle time. Process-handling and synchronization are done at compilation time, therefore produce no actions. ESTEREL only supports some basic data types (boolean, integer, float, double and string) and operations. Any other necessary data types, their possible operations, and external implementations must be provided by additional code written in the host language.

The synchronous components developed at IAS are used under the ViPER concept. The name ViPER stands for **Vi**sual **P**rogramming Environment for **E**mbedded **R**eal-Time Systems and summarizes both a method and a tool environment for visual component-based development of embedded real-time systems [Gunz02]. The concept consists of the design model, the generation model and the execution model as shown in Figure 3.5. The three models are described in the following paragraphs.

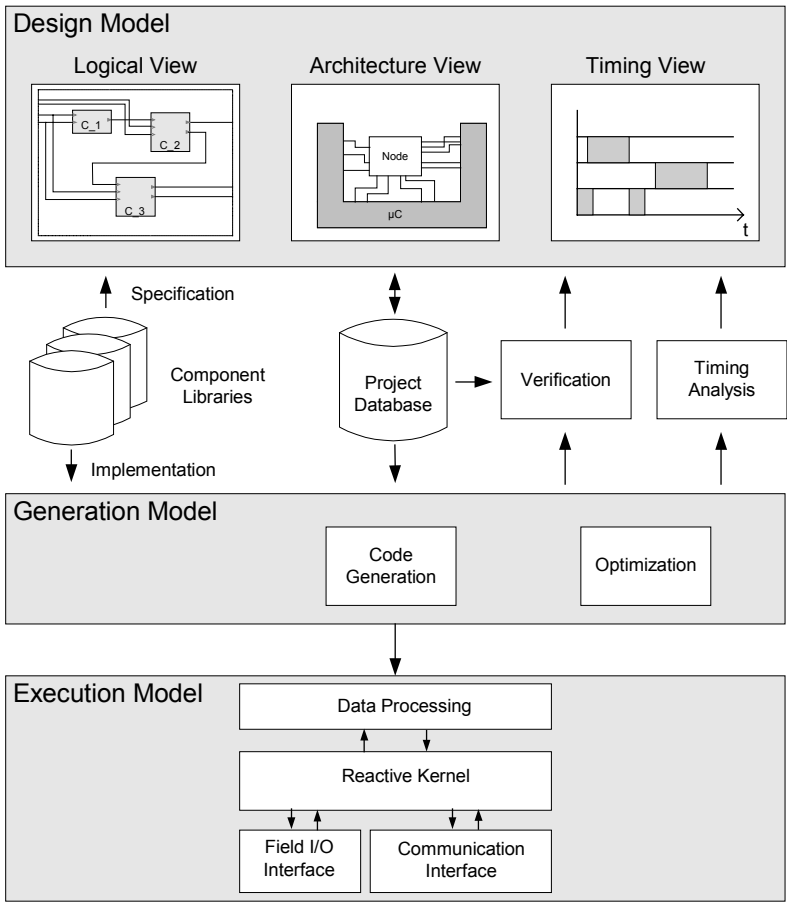


Figure 3.5: Elements of the ViPER Concept.

- In the *design model*, application software can be constructed from components provided in several libraries. At this level of abstraction only high-level descriptions of components are used. Components are specified with their interfaces and parameters. The implementation of the components is not visible to the user. For the development of embedded systems the hardware platform must also be considered. For each platform a hardware library has to be provided. This library includes the specification of the hardware components which can be used from the software. The design model itself has three different views – the *logical view*, the *architecture view* and the *timing view*. In the *logical view*, the logical structure of the system is specified, components are connected and data and control flow are defined. For this purpose block diagrams are used which can be hierarchically composed. In the *architecture view* software components can be connected with platform-specific hardware components. This includes connections with hardware ports to which sensors or actuators are connected as well as communication connections between the nodes of a distributed system. The *timing view* specifies the temporal behavior of the system based on the division of time in time-slots in which certain actions are performed.
- The *generation model* describes the process of generating executable code from the high-level graphical specifications. In order to meet the requirements of a specific application, the code produced must have minimal overhead. This is achieved using the synchronous model, generative techniques and code optimization. Moreover, it is also possible to generate code for simulation and verification purposes. The generation process consists of several stages where ESTEREL code, C-code and makefiles are generated. The intermediate code is a finite state machine driving action tables that call the data processing functions. For a specific hardware platform, a platform-specific C-compiler and a development environment has to be provided to compile, link and download the generated code to the targets.
- The *execution model* is an execution framework for a single node in a time-triggered system. It provides a generalized view on a distributed time-triggered application. For a concrete application, the component-based design is mapped into this framework by the generation model automatically where the overall structure does not need to be changed. This can be reached by clearly separating platform-specific and application-specific parts from platform- or application-independent parts.

Some examples of successful usage of this component approach were the development of the control systems for a steer-by-wire application [GuNä99] and the construction of a library of synchronous components based on the IEC norm number 1131-3 for the usage at programmable controllers applications [Luce99].

3.3.2 Distributed Intelligent Objects

The component model Distributed Intelligent Objects (DIO) permits the utilization of standardized object technologies and frameworks to configure a particular, custom-made

solution for distributed, heterogeneous automation systems [RiHo99]. This component approach was developed at the Dresden University of Technology at the Institute of Automation (IFA). In order to create or adapt an application, the DIO software model, offers a set of libraries containing pluggable components that are manipulated through a graphical environment. Within this environment, available components are inserted into projects by mouse-click and may also be connected to other components graphically having their parameters adjusted afterwards in order to generate the desired application.

A DIO component encases certain functionalities and data of a system unit (hard and software) in a unique identifiable object and offers their services, or their service dependencies, over pre-defined software interfaces with help from typed ports. At the run time, components exchange events and data over established connections among themselves, without having to possess mutual knowledge of their concrete static interfaces. DIO components implement a special interface, which is specified with CORBA-IDL or COM-IDL, depending on which technology is to be used in the complete specification of a particular project. DIO components are produced and destroyed by a DIO server which is normally associated with a task of an object and, where the actual component implementations are located. DIO ports represent a concept for event-oriented and strictly typed object interactions, working well at run-time and in real-time conditions. This technique is used for the access and for the exchange of variables such as process signals, control data, and events (timer, interrupts, alarms, etc.). Figure 3.6 presents the basic elements of a DIO component and the connection between two components.

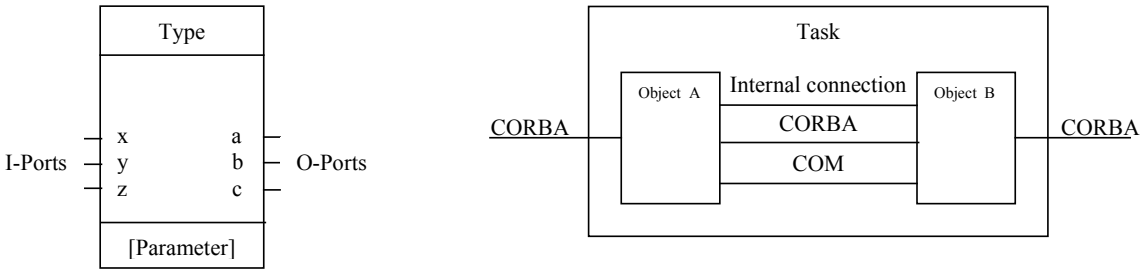


Figure 3.6: Basic Elements of DIO Components and their External and Internal Connections.

An I-Port represents an offered service of a component (similar as a method), an O-Port is used to communicate with a service (similar as an event or a method call). The implementation of ports is made through normal CORBA objects or CORBA object references. This decision depends on whether a port is to be exported for inter-process communication with other components or whether it is to be used internally in the actual process. DIO defines three primitive communication modes. The first is denominated NORMAL_MODE and implements the simple exchange of objects, whose processing does not depend on another event. The second communication mode is called ONEWAY_MODE where an object can be transmitted, without acknowledgement and can be returned immediately. The last mode is the BLOCKING_MODE,

a port in this mode implements the exchange of objects, whose processing depends on certain events. Every DIO component may have parameters which are accessed from outside via their names and the desired values. DIO accepts parameter modifications during run-time of a component. Parameters are useful for the design phase of an application permitting the adjustment of different levels but will only be effective at run-time when they really receive a value. An operation mode specification is introduced to the component model in order to achieve safer run-time behavior and optimal performance. In DIO, operation mode is defined as a type of enumeration. DIO prescribes itself, in two possible models, the configuring mode if they do not transmit or receive objects over their ports, and the remark mode if it can communicate with other objects over their ports.

There are features of DIO-applications that proved to be useful for automatic controller design programmes, real time simulation programmes, control systems programmes, process visualizations as related by the University of Dresden [RiHo99]. Moreover, DIO has frameworks for CORBA and COM so that development and application of user-specific components are substantially facilitated.

3.3.3 ACPLT Components

ACPLT stands for **A**a**C**hener **P**rozess**L**eit**T**echnik (Aachen Process Control Engineering) and it was developed and maintained by the Aachen University of Technology at the Process Control Engineering Institute (PLT). Its main goal is to give a platform for the component-based development of tasks used in process control systems. ACPLT works seamlessly in heterogeneous systems, because its message transfer mechanism is based on existing standards like TCP/IP. Moreover, ACPLT offers an application layer with a degree of abstraction more suited to the purpose of process control engineering [Albr00]. The structure of the ACPLT product family is presented in Figure 3.7 [AMU00].

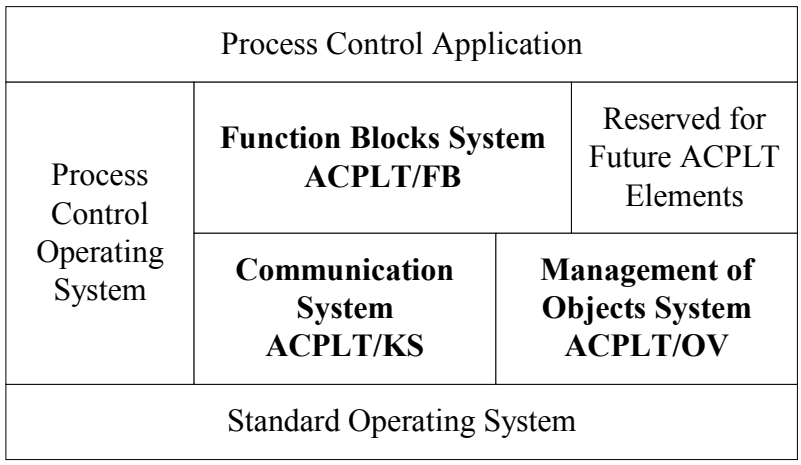


Figure 3.7: Elements of the ACPLT Family.

The lowest functional level of ACPLT is composed of a communication system (ACPLT/KS) and a management of objects system (ACPLT/OV). Potential users of these middleware components are developers of new functional units (also called components) implementing new communication features, or end users wanting to access information available at this architectural level. The upper level of the architecture is denominated ACPLT/FB and is responsible for the realization of classical process control software structures using the services offered by the middleware components. Potential users of this layer are end users developing process control applications.

These process control applications may be developed by the deployment of already stored components or by the generation of new components not yet available on the library. Such a new component does not need always to be developed from scratch, rather it may be constructed upon other components that partially fulfil the requirements. The open structure of ACPLT components make this adaptation procedure easy to be implemented. The main characteristics of the three elements of the ACPLT family are described in the following paragraphs.

ACPLT/KS is a platform-independent communication system, and network-capable communication environment. It supports various flavors of Unix operating systems (for instance HP-UX, IRIX, Linux, Solaris) as well as Microsoft's Windows NT/95/98, and COMPAQ's/Digital's OpenVMS (on Alpha hardware). Communication within ACPLT/KS is based on the client-server model and happens by synchronous message exchange implemented through the Remote Procedure Call mechanism. The ACPLT/OV is responsible for the realization of the system objects in accordance with the desired platform for the end application [PLT99]. ACPLT/OV manages not only the application objects but also the model of the objects in a persistent data base. Classes and respective associations are stored in libraries and can be called during run-time execution. The final code is automatically obtained by the usage of a code generator able to read the model representation and convert it to the C programming language. ACPLT/OV is completely integrated to ACPLT/KS which enables the access to the network environment permitting transparent communication among distributed objects.

The function block system ACPLT/FB offers users the possibility of constructing their own process control applications by the choice of available blocks [PLT99]. The basic function blocks of ACPLT/FB can be seen as representatives for the well-known single hardware modules. They combine the advantages of encapsulated functionality known from single hardware modules with the flexibility of modular software realizations. The ACPLT/FB is based on a static object model that describes the most important classes and their associations. Engineering using function blocks is straightforward since no programming is necessary, users need only to set the desired structure and parameterize the function blocks properly. The only situation where users may need to programme themselves is when the desired functionality is not yet available. In these cases they need to programme their own function block and load it dynamically into the system.

Some software packages based on ACPLT are already available as commercial products of many companies, as for example Endress+Hauser, Commutec and Foxboro [Eppl02]. At EC Erdölchemie GmbH, ACPLT is used in an industrial communication system providing access in a heterogeneous environment to about 100,000 technological objects, which are rendered during run-time into approximately 1.4 million communication objects. In another application at Cologne-Stammheim, a large-scale wastewater treatment plant, ACPLT is responsible for a system to handle the messages and alarm archives, trend information, and process data. Some other publications relate about the successful usage of this component technology at laboratory automation applications.

3.4 Summary of the Presented Component Approaches

All technologies discussed in this chapter are grouped in Table 3.1, where the main characteristics of each one is summarized. The table shows how different may be the technologies with any application in the industrial automation domain.

Table 3.1: Component Technologies used in Industrial Automation Applications.

	Developed by	Coded in	Platform	Interoperability	Interface
			Commercial Approaches		
CORBA	OMG	Mapped to: C, C++, Java, Ada	Any platform if there is CORBA ORB Compatible	ORB in IIOP	Defined IDL, Supports multiple inheritance at interface level, Binary standard
COM DCOM	Microsoft Corporation	Specification in: Binary level, C++, Java, COBOL, VB	Any platform if there is COM Service	RPC in Object Remote Procedure Call, in Binary Standard	Defined IDL in MIDL data type, Support multiple interfaces for objects, Binary standard
JavaBeans EJB	Sun Microsystem	Java	JVM	Java Remote Method Protocol	Defined in Java, Support multiple inheritance at interface level, Byte code standard
.NET	Microsoft Corporation	C# and others With support of Common Language Runtime	.NET platform based on XML service platform	XML and Simple Object Access Protocol (SOAP)	IDL standard
			Industry Approaches		
KOALA	Philips	C	Windows	Through interfaces	Defined IDL and CDL
ASCET-SD	ETAS GmbH	C	ETAS-OS ERCOSEK	Messages	ASAP standard
			Academic Approaches		
ViPER	University Stuttgart (IAS)	C and Esterel	Unix (extensible to other OS)	Signals and Sensors	Defined in Esterel
DIO	TU Dresden (IFA)	C++	Windows	RPC like CORBA or DCOM	Defined in IDL standard
ACPLT	RWTH Aachen (PLT)	C++	Windows like, Unix like	TCP/IP and ONC/RPC	Defined in IDL standard

It is possible to conclude, based on the study presented in this chapter, that the diversity among the component technologies used in industrial automation applications really is one important aspect to be considered by any management proposal. A meaningful component management system must be able to work with many different component specifications, meaning that it must deal with different representations simultaneously. One goal of this thesis is to propose a way of representing components able to respect the specificity of each component technology without limiting the usage of the management approach only to this technology. In other words, the way of representing the component technologies adopted by the desired component management system must be flexible being able to be adapted to other technologies and even to receive new representation models of future component technologies. This flexibility was not present in the management of components proposals found in publications (see chapter 2) and will increase the final value of this thesis.

In this chapter the most relevant component approaches with any usage in the industrial automation domain were presented. Not only the software industry proposes and produces software components, there are also proposals of component technologies coming from the hardware industry. Developers of consumer electronics and embedded systems need efficient software artifacts and also proposed their own components and component-based development approaches. Moreover many university research centers have been studying and developing components specially designed to the industrial automation applications. One meaningful component management system shall be able to deal with such variety of component approaches. More than that it should be able to accept new approaches whenever these new technologies have any usage in the industrial automation domain.

But what differentiates components developed to be used in the industrial automation domain from other components? Which kind of information must a user receive about such components in order to properly decide about their reuse? How is it possible to represent this knowledge and how is it possible to model it? These questions are very important for this thesis and are the main theme of chapter 4. In this chapter a study of how the knowledge about components can be represented is presented. Afterwards, the most important knowledge about the industrial automation domain is summarized in a graphical manner. All this information will be used in the formulation of a flexible model able to represent different component technologies simultaneously which will be the basis of the component management system proposed in chapter 5.

4 Knowledge Representation for Industrial Automation Components

The usage of large collections of reusable, domain specific software components poses new problems for software engineers. The best way to locate and retrieve appropriate components and how to meaningfully browse and navigate among functionally related components are some of these open problems, that have been tried to be solved through the introduction of new tools to support the whole component-based development process. Central to such tools is one component repository, which contains the actual reusable components, and, more importantly, it must contain information about these components enabling users to retrieve and discriminate between functionally related components.

But what is the right information to be represented about one component and how should this information be organized? The effective choice of a component requires a deep and detailed understanding of diverse knowledge about the component's functionality [Fisc87, HeMa91]. In order to properly use a component in the way it was intended to be used, users must understand performance characteristics, function call sequences, and even pathological behavior [Kotu98, CiRo99]. In other words, system developers need to know what to expect from a software component, they must have access to documentation about the component's interface, behavior, and respective operational conditions.

As seen in last chapter, there are many technologies that have applications in the industrial automation domain. The differences among those technologies make it impossible to adopt a unique formal model for representing components able to cover them all. Nevertheless, the representation model to be proposed in this thesis must be well suited to all those technologies and must constitute a collection of information capable of well describing a component facilitating its understanding. Industrial automation applications are essentially real-time systems that are constrained by hardware limitations, what makes it generally more challenging to develop than general applications. Real-time systems frequently execute under tight time requirements in low memory using low computing power. A well designed representation model must put in evidence the most important real-time characteristics of the components used in this domain.

In the following sections a study about the usual ways of representing components are presented. In section 4.2 one of the most important aspect of the presented concept is discussed, i.e. the particular knowledge someone needs to deal with in order to understand the particularities of the industrial automation applications. Following this section, a discussion about the technical and non-technical information needed about components is done. The

concept for a classification scheme for the domain of interest for this thesis is presented in section 4.4.

4.1 Representing Information about Components

The question proposed in this section is how the available information about reusable components must be represented in such a way that the management of these components can be done better later. Different approaches are found in various publications as the index based, knowledge-based and hypertext-based. Additionally, components were represented through formal specification methods [FrGa90]. After the study of these approaches it is possible to define which one may fit better to the problem presented, i.e. the management of components used in the industrial automation domain.

4.1.1 Usual Representation of Components

As stated in [Szyp98]: "For a component to be composable with other components it needs to be sufficiently self-contained. Also, it needs to come with clear specification of what it requires and provides. In other words, a component needs to encapsulate its implementation and interact with its environment through well defined interfaces". That is actually the most acceptable way to represent components nowadays. Here the functional view of components is emphasized and only two aspects are represented:

- What the component can do, its functionality or in other words its behavior (what it requires and provides), and
- How the component can communicate with other components and systems, or its interaction description (well defined interfaces).

The definition of components as independently deliverable packages of software functionalities leads to a representation where the technical information aspect is the most important one. More precisely, the only technical information needed for the reuse of a component is a description of its behavior and a description of its interface. This way of representing components is summarized in Figure 4.1.

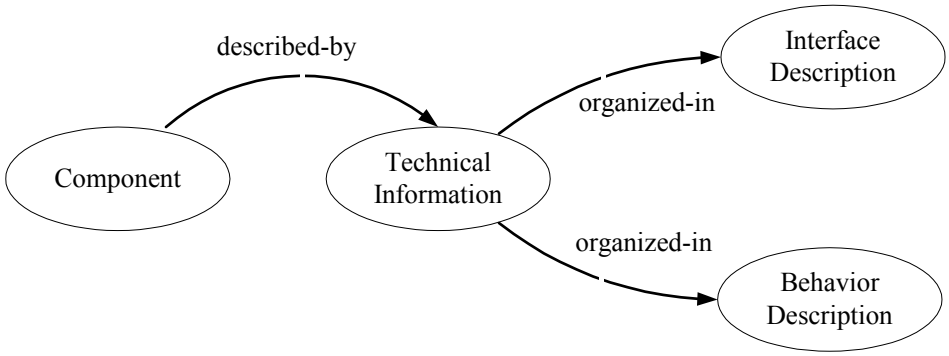


Figure 4.1: Most Common Way of Representing Components.

One question to be answered is if this information is enough to achieve a good management concept, able to facilitate the searching, the finding, the understanding, and finally the decision about reusing components. Indira Kurugant from Lucent Technologies, a company of the Bell Laboratories, wrote in 1999 [Kuru99] that the component selection for reuse purposes is difficult due to the lack of clarity on what constitutes a component, due to the incomplete understanding of the functional specification, of the performance attributes, and the deployment constraints that must be met, and lastly due to the non-uniformity in vendor offerings with respect to packaging of functionality into components. None of those desired characteristics are represented in Figure 4.1.

Other aspects that must be taken into consideration are the non-functional requirements that originated in the component being managed [LeWi00]. For the goals of this thesis, important non-functional requirements are the usability, reliability, performance, and supportability offered by components. Some decisive characteristics of a component are originated from these requirements and must be presented as a possible way of selecting components for reuse. The more developers add technical and non-technical descriptions to their components the greater the chances of someone reusing them [PSB00]. Examples of such descriptions considered in this thesis are on-line documentation, demonstrations of use, evaluation procedures, sample of code and a complete set of read-me files describing for example product changes, bug fixing, systems requirements, and some other issues.

But a component may also be selected based on a different set of values that classify it on a specific aspect of interest for the domain of usage, as for example, on the desired hardware platform, on the operating system, or on programming language [DuKn93, MMM94, DaFu95, Henn95]. All these aspects are also neglected in the model of Figure 4.1. A better representation composed of technical and non-technical aspects of a component, and with a mechanism for the searching and finding of desired components proposed here is presented in Figure 4.2.

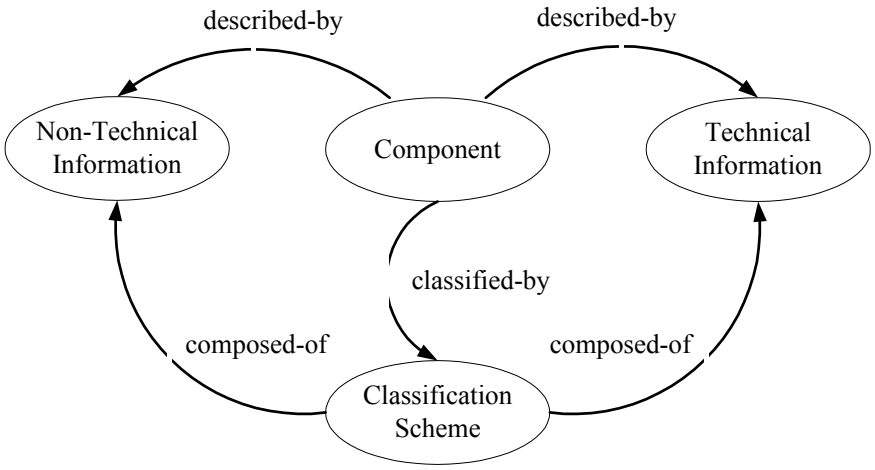


Figure 4.2: Extended Representation for Components.

This representation is a refinement of the most common way of representing the knowledge about a component more suited to the component management system described in this thesis. It is divided in two blocks of information, technical and non-technical and additionally has a classification scheme composed of information obtained from the two mentioned blocks.

4.1.2 How to Represent the Information about Components

The central goal of those developing reuse repository systems is to find some manner of searching for a component that satisfies the specific needs of an individual component user. The component must be appropriate for the user but it must also be recognized as being appropriate otherwise it would never be used. This search and find of desirable components is obtained easier through a well designed component representation. Frakes and Gandel [FrGa90] defined a representation mechanism as being a language (textual, graphical, or other), used to describe a set of objects. For example, books in a library are represented by bibliographic records in a library catalogue. The reason that a representation is created is to allow operations on it that would be more difficult or even impossible to be done on the represented object itself. Using the above mentioned example, it is much easier to sort a set of bibliographic records according to author than to sort the same number of books by the same criteria. In the rest of this section methods to represent the knowledge about a component are studied. In a first step, three non-formal approaches frequently discussed in various publications are presented. They are the indexing-based, the knowledge-based, and the hypertext-based representations. Afterwards some discussion are presented about representing components using formal specification methods.

Informal Methods for Representing Components

The indexing-based approaches are based on the proposition that there is a characteristic vocabulary, globally accepted, well known, and recognized by specialists of a specific domain. This vocabulary is then used as the basis for representing components in this domain and it is obtained in two different ways: through controlled or through uncontrolled mechanisms. A controlled vocabulary considers the semantics of the associated terms and must be developed by specialists. An uncontrolled vocabulary is developed without any previous intervention or semantic specification, the terms that appear are not previously defined. The indexing of components themselves is done in a manual procedure (implemented by a specialist) or in an automatic procedure (using specially developed tools).

The indexing-based representation used in the Information Retrieval Approach [SaMc83] extracted the characteristic vocabulary of a component domain based on their frequency of appearance in the available documentation. In this representation, the selection of terms was done based only on statistical results obtained, no semantic consideration was taken into account. In some other works this way of characterizing components was treated as automatic

classification [Luqi87, FrFo87, LoMi89, Bond97]. In accordance with the automatic indexing method, all components having the same terms are arranged together in groups. The classification is done in a mechanical procedure and no reference to the intellectual analysis of creating similar classes based on the semantic relationship is done.

Indexing representation can also be generated based on keywords or classes previously selected. Generally the terms are derived using a combination of two methods. The first method is denominated “Literary Warrant”, the index terms are derived from the examination of the subject area and a term will be used only if it occurs often enough in publications of that subject area or domain. The second method is the “User Warrant” and the index terms are included in the controlled vocabulary list if they are of interest to the user population.

In the indexing-based approach, classes that cluster the pre-defined vocabulary together are used to organize a domain specific hierarchic structure. The representation of components is then done by the association of one component to each term from the vocabulary. Additionally each term may have one specific semantic definition. In the indexing representation based on uncontrolled vocabulary no restriction is placed on what terms can be used to describe a component. Terms used in uncontrolled vocabulary approaches can be drawn from any source. Nonetheless, those terms are usually obtained from the available component documentation through a free text analysis procedure. In a second step some level of hierarchy are imposed to the terms selected.

Knowledge-based approaches deal with methods for the representation of knowledge about objects of interest, relationships among these objects, and respective rules. They were mainly developed and applied in artificial intelligence applications. Two useful factors to consider when evaluating knowledge representations are their representational adequacy and their heuristic power. Representational adequacy refers to how much one can express with the representation. A simple list of keywords has poor representational adequacy because the syntactic and semantic relationship between the keywords is missing. Heuristic power refers to the kinds of inference one can do with the representation.

One appeal of the knowledge-based approach is that the representation offers a powerful way of expressing the relationships between system components. This is important for helping users to understand the function of components. One potential problem with knowledge-based methods is that the knowledge acquisition itself has proved to be a barrier, as it was in other investigation areas of artificial intelligence. Some publications shown that it was too difficult or to expensive to acquire the knowledge needed to feed knowledge-based reuse systems [Stef95]. There are three basic flavors for knowledge-based systems: semantic networks, production rules, and frames structuring.

A semantic network is a graphic descriptive language used as network model of information. The network is presented as a directed graph where nodes correspond to conceptual objects and

which arcs correspond to relationships between those objects. An arc is labeled by the name of the relationship between objects. Several arcs can have the same label. However each object is represented by only a single node. Given a node in the network, it is assumed that the network provides direct access to all the relationships in which the node participates, independent of the direction of the arc.

The Figures 4.1. and 4.2 previously presented are examples of simple semantic networks. The syntax of the graphical representation of semantic networks here adopted considered that nodes are depicted as ovals and that arcs are depicted as arrowheads and labels. Arcs naturally have two ends, so they are most useful for representing two-part or binary relationships. Relations involving more than two parts (n-array relations) can be represented as nodes. Thus, in addition to nodes that correspond to physical objects there may exist nodes that stand for relations.

Moreover, the relationships represented in semantic networks can be expressed in the clausal form of logic. Examples of such usage were related by Deliyanni and Kowalski [DeKo79]. The graphical approach of semantic networks have good knowledge representation adequacy. It is also well known that many other representations in knowledge systems are based on graphs [Stef95] making the choice of semantic networks appropriate and well accepted by knowledge experts. In fact, it is quite easy to express knowledge in this form, nevertheless, its heuristic adequacy is quite poor. It is very difficult to do useful reasoning with this representation unless a more powerful semantics is imposed on it.

Production rules is a good knowledge representation formalism. An example of the classification of a component using this technique is shown below.

IF component needed IS control system

AND algorithm IS pid

THEN component to use IS pid10.c

Note that the knowledge representation is quite precise, but the way of acquiring these rules are not trivial, demanding much time and effort and causing an increase of the costs because human intervention is necessary to formulate the rules.

The knowledge representation approach called frames structuring is based on data structure composed of slots and fillers. Using the example of the previous paragraph the component “pid10” is represented as:

pid10

AKO:	algorithm
Operation:	controlling
Operands:	control signals

The slots here are in the left hand columns and the fillers in the right following the colons. From the representation, it is possible to conclude that “pid10” is a kind of algorithm (AKO) which implements a control task using specific control signals. The mechanism for the acquisition of knowledge of this approach has the same problems presented in the production rules approach.

Differently from the other two presented approaches, the hypertext-based approach [Frei94, GaSc90] considers that the document or its description is part of one information network. This network is then constructed based on the Hypertext-Concept [Land94] where the reusable artifacts represent the nodes and the relationships and dependencies between artifacts are represented by the arcs. This structure allows users to move from one place to another in a body of text via links. Making use of the example above, someone searching for the component “pid10” could arrive at a complete documentation on control systems. Figure 4.3 shows a hypertext representation for that component. From the representation it is assumed that “pid10” is a code written in C, which is also used in the programme “PowerControl”, and both programmes are additionally discussed in the book Automatic Control Systems. As much as the two other presented approaches, hypertexts represent knowledge in a very well described form [CFG91]. One interesting but also problematic aspect is that the information may be distributed in different locations and sources. The generation of links also needs human intervention.

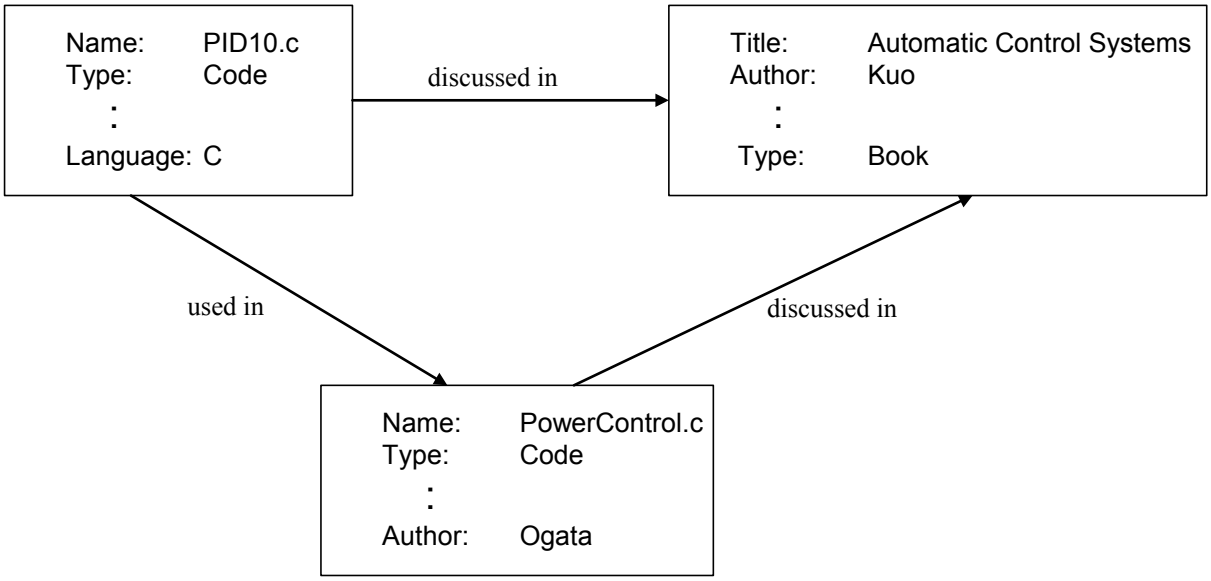


Figure 4.3: Example of a Hypertext-based Representation of a Component.

For the three above presented approaches the acquisition of knowledge about one object is done in a manual form based on the proper choice of attribute-value couples or through semantic text analyses [Nie92, NPV93]. For the end user of such a knowledge system this relation among elements and their attributes is transparent because the gathering of knowledge is a procedure done beforehand based on the expertise of the people involved. For someone developing such systems the quality of the knowledge acquired is a very important factor that may determine the

success of the final system. In the same way as by the indexing-based approaches some related systems gave support for the user by natural language approaches [DBSB91, DeJo97], some others needed the manual introduction of desired attributes [OHPB92]. A characteristic for all knowledge-based approaches is that they offer to their users not only the information acquired but also additional intrinsic knowledge about the studied objects.

Formal Methods for Representing Components

All the above mentioned methods of representing components are classified as informal approaches, nevertheless they do represent components in a correct, precise and accurate way. On the other hand, it is also interesting to investigate some formal approaches as Estelle, LOTOS and SDL [Turn93], Z [Spiv92], and CSP [Hoar78] where software artifacts were formally described. These approaches tried to precisely describe the interface and the semantics of software components using different mathematical tools. For example, through the introduction of a signature for each artifact [ZaWi95a] and its specification (behavior and semantics) [ZaWi97], or using an algebraic specification methodology. But solving the problem of matching algebraic specifications of components is not decisive for reuse purposes. Some other authors tried to add a kind of semantics to the specification matching, but could not prove any decisive advantages of their approaches [JeCh93, JeCh95, MMM95, MoGa91]. Another limitation is that the algebraic specification is applicable only to the description of components in a limited amount of application domains. In fact, no successful attempt to the industrial automation domain, the central goal of this thesis, could be found in publications.

The proved advantages of such approaches are the powerful and precise evaluation if an existing specification meets the requirements of a desired specification. A disadvantage is that the usage of mechanisms to find components that only partially meet the desired requirements are confused and difficult to implement. Nevertheless, not all aspects of a component are described by formal specification, as for example time constraints, or hardware requirements. An effective usage of formal methods that could increase the reuse of software artifacts could not yet be proven to exist in the praxis [MoBa91], nevertheless some approaches tried this way [JeCh93, JeCh95, ZaWi95b, ZaWi97, Chen93]. Meyers et al justifies the avoidance of formal methods for representing components with the following arguments [MMS98]:

- “It is too difficult to build mathematical models of the most delicate aspects of ‘real’ programmes, from floating-point computation to pointers;
- There are few powerful tools to assist this effort;
- Even more fundamentally, it is just too expensive and difficult to apply formal techniques thoroughly.”

The small contribution in practice in contradiction with the enormous effort to construct such models (specification of the reusable component model must be precise, correct, and complete) turned this approach to be not viable to be applied to the components designed for industrial

automation. Another problem is that the user must be accustomed to new formalisms applied to the component model, what makes the costs even higher.

Concluding Notes on How to Represent Components

The main goal of the concept presented in this thesis is to facilitate the finding and understanding of components for a specific domain area that has no standard model previously defined. The definition of a formal model for the components is out of the scope of this thesis mainly because such approaches have already proved to be inadequate to solve similar problems when applied at industrial approaches [MMS98, Wieg98, Seli01]. The concepts of indexing-based and hypertext-based representation are used later when talking about the possible ways of organizing and presenting the information acquired about one component.

After experimenting with the approaches mentioned above and judging their pros and cons, semantic networks proved to be the most appropriate form of describing the knowledge someone needs to have in order to meaningfully understand the functionality, usage and limitations of components for the industrial automation domain. The graphical representation of this knowledge obtained using semantic networks is much easier to understand and reason about than the one offered by the other approaches studied. It was also observed that modifications on the desired representation were easily implemented in the network. Moreover, the elements composing the necessary knowledge and the relationships among these elements offered a better global overview when represented by a network.

Semantic networks is then the approach used until the end of this chapter to organize the large amount of information necessary to correctly work with components themselves and to work in the industrial automation domain. The goal is to represent the necessary knowledge to work with components and to work in the industrial automation domain through semantic networks in order to manage this knowledge properly later on through a suitable component management system. Sources used for the compilation of this knowledge, and consequently for the construction of the semantic networks, are the findings and opinions of experts on components and experts on the industrial automation domain presented in publications and the experiences developed at the IAS during this research work. The result of this research lead to a global knowledge representation containing the most relevant information about components for that domain which is presented and explained in the next sections.

4.1.3 Representing Industrial Automation Components

The knowledge model of Figure 4.2 is a good representation for components in general but has no particular aspects from the target usage domain of this thesis. The necessary knowledge and the way of presenting this knowledge vary if the final users have different goals and backgrounds. In fact a representation of a component must consider particularities of the

application domain that are decisive to the understanding of the component and consequently to the reuse decision.

A better representation must consider how the technical and non-technical knowledge varies inside different component technologies used in the desired domain. In Figure 4.4 an improvement of the component representation of Figure 4.2 is presented. The new representation considers the above mentioned semantics relationships. In this thesis the target domain is industrial automation, and the different technologies applied at this domain must be taken into consideration. Additionally, it is necessary to develop a classification scheme able to deal with the domain particularities resulting in a better representation for industrial automation components.

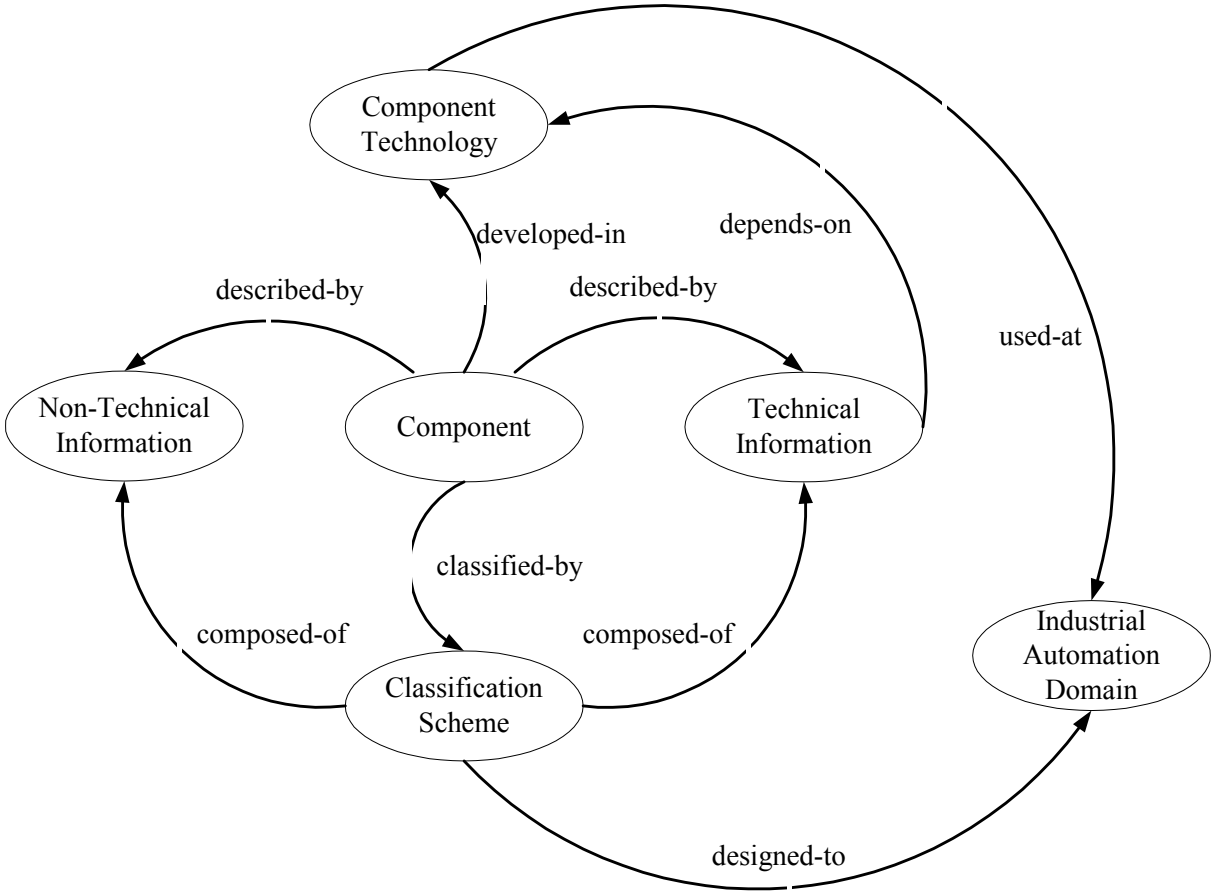


Figure 4.4: Global Knowledge for Understanding Industrial Automation Components.

The presented diagram represents only the main aspects of the knowledge needed to work with a component at the industrial automation domain. In fact, it is necessary to refine this model even more exploring the particularities and special needs that must appear in every represented conceptual object and respective relationships. This refinement is done in the next three sections specifically for the industrial automation domain, for the technical and non-technical information, and for one classification scheme designed for industrial automation applications.

4.2 Characterizing Industrial Automation Applications

Industrial automation applications and real-time applications often have the same meaning and these terms are interchangeable. The one common feature of all real-time software system is timeliness; that is, the requirement to respond correctly to inputs with acceptable time intervals. However, this simple property characterizes a vast spectrum of very different types of systems ranging from purely time-driven to purely event-driven systems, from soft real-time systems to hard real-time systems, and so on. Over time, each of these categories of systems has developed its own idioms, design patterns and modeling styles that collectively capture the distilled experience of many projects.

The software control of any real-time system and associated hardware is maintained by the software's ability to predict the next state of the system given the current state and the set of inputs. The goal is to predict how a system will behave in all possible circumstances. In other words, an industrial automation application must be deterministic. A deterministic system is defined as the one that for each possible state, and each set of input, determines a unique set of outputs and next state of the system.

Real-time systems change their states based on the occurrence of events, which are divided into two categories: synchronous and asynchronous. Synchronous events are those which occur at predictable times. Asynchronous events occur at unpredictable moments and are usually caused by external sources. Based on these facts, components for industrial automation applications are classified into two major categories based on their trigger characteristic: event-triggered or time-triggered. Components that were developed to work in synchronous platforms are said to be time driven or time-triggered. Components developed to work in asynchronous platforms are denominated event driven or event-triggered. The possible applications of such component types vary strongly making the previous knowledge of this characteristic an important factor. From the point of view of temporal performance a system can be primarily classified as real-time or non real-time. Additionally, two types of real-time systems are distinguished:

- Soft real-time systems which are systems where the failure to meet a specified deadline reduces the utility of the result, but does not lead to a significant financial loss. An example of such a system is a letter sorting machine, if a letter is dropped into the wrong box because of a timing failure of the system, no serious consequences will result.

- Hard real-time systems which are systems where the failure to meet a specified deadline can lead to catastrophic consequences. An example of a hard real-time application is a computer system controlling a railway system. If a wagon is released on the wrong track because of a timing failure of the computer, a serious accident may occur.

In a hard real-time system, a failure in the temporal domain is as critical as a failure in the value domain. Temporal properties, as for example the expected response time, are system properties

that depend on the proper cooperation of all levels of design: hardware, synchronization mechanism, operating system, etc. In a hard real-time environment, it cannot be assumed that timeliness can be tested into a system in a stand alone way, timeliness must be the consequence of a rational system and software design process. Nevertheless, even some of the more recent software specification and design technologies that are targeted specifically at real-time applications, such as UML-RT [Seli99], real-time CORBA [OMG98], or real-time Java [CaRu99], are not based on precise model of time and do not consider temporal properties as first order quantities, but rather as an addendum.

Talking about software components for real-time applications Kopetz [Kope97] wrote: “One problem with the use of software components in real-time applications is that a software component *per se* does not have any temporal properties – the temporal behavior only emerges if the software is bound to a particular hardware component. It follows that the user has to investigate the temporal performance and the fault-model of a software component on the selected hardware platform.”

In any real world system there is some delay between the presentation of the inputs and the appearance of the outputs. This time between these events is called the response time. How fast the response time needs to be depends on the desired application for the system. The real response time of a component depends on many factors as for example the utilized hardware system and operating system. In addition, there is no way to write a tool able to determine precisely the response time of a software component in the same way one can do for hardware components, where the presented values are precise or are expressed under acceptable error percentages. In spite of that, the way to represent the response time for software components is similar to the one of hardware components. Based on a set of operating conditions considered ideal, the response time is measured. A second measure is obtained for the worst imagined conditions. After validating these two values some statistical analyses are applied and a medium value, denominated as expected response time, is obtained. A component user uses these measures as reference values that are corrected under certain restricted system conditions.

Developers of components for real-time applications are expected to measure and divulge the obtained values of the worst case, best case, and expected response time. Most important is to communicate under which conditions the measures were obtained. It is also desirable to receive a well described test environment specification with the component including instructions of how to reproduce the original tests. The hardware components industry has adopted this procedure for many years. It is common to find a section in every data sheet of hardware components with a complete description of test circuitry, necessary wave forms, and expected results for certain operation conditions.

Depending on the size or complexity of the technical process and also in accordance with the way of implementing the automation procedures, a system is classified into two different ways.

The first one is called plant automation and deals with great industry plants composed of many smaller parts that work together to implement the desired functionality. One example of plant automation is the control of the complete production system of a manufacturing plant. The second way of classifying is called product automation and deals with products having small computational power where all desired functionality must be self embedded. Such an industrial automation system is also called embedded system and is used to control specialized hardware systems in which the computer system is installed. One example of those applications is the control system of fuel/air mixture in the carburetor of automobiles. Embedded systems may also be distributed. Here an individual processor executes an assigned specific task and cooperates with other processors to execute a more complex task. This type of system is widely used in areas of avionics, astronautics and robotics.

In the plant automation the costs of computational resources are not a decisive factor [LaGö99] and sometimes the automation solution involves specific hardware platforms [PrWu96]. In spite of it, some solutions involving components tend to reduce this cost [PTH97]. In embedded systems any single cost reduction may determine the viability or non-viability of the complete automation project. It is necessary to know the exact resource usage whenever talking about embedded applications. Components developed to work in such systems are expected to have their resource usage demands clearly and precisely specified in terms of memory usage.

At the hardware component's world each component has specific implemented tasks associated to it. Those components are idealized to be applied at a particular application domain. Industrial automation itself is the global domain of this thesis but it is necessary to classify the components used in this domain in sub-domains in order to determine their possible usage better. Each of the two main divisions of the industrial automation, i.e. product automation and plant automation, have many other sub-divisions. That is where the terms chosen to specify the real target domain of a specific component must be originated, for example, a component belonging to the sub-division "Product Automation" should be classified as "Home Entertainment" another one should belong to "Medicine Systems". Additionally, industrial automation applications can be distributed in a hierarchical classification in accordance with their management level [LaGö99].

The description of the task a component implements is important to help users to find possible reuse candidates. No precise description of the implemented task is supposed to be given here, but only a high level description of the component's ability and applicability. Standard terms of the industrial automation domain should express these categories of component's functionality. The considerations presented in this section summarize the knowledge about the industrial automation domain necessary to make any decision towards the reuse of components specially developed for this thesis. This knowledge is represented using a semantic network in Figure 4.5. This information is not always available, in fact no similar approach could be found in publications. The presented concept solves this problem proposing a model where this data is represented and will always be asked from the component developer.

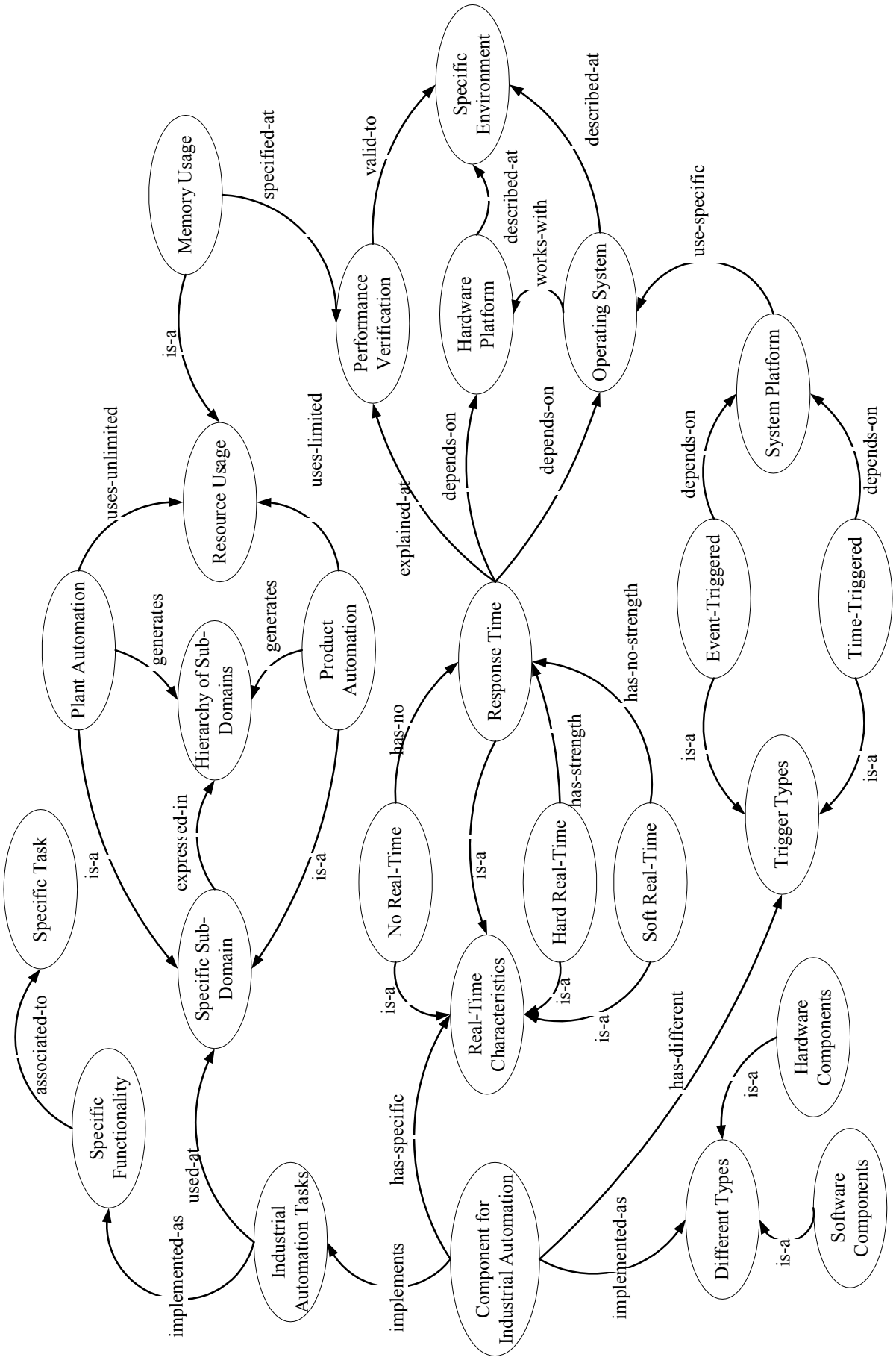


Figure 4.5: Particularities of a Component for Industrial Automation Applications.

4.3 Additional Desirable Knowledge about Components

Although the information about components already discussed covers intrinsically technical aspects there are still some other perspectives that need to be analyzed as, for example, the functional and the operational characteristics of components. Some other details that also might not be neglected are those related to commercial and organizational issues. In this section, all these aspects are worked out leading to a more sophisticated representation of the knowledge necessary for the reuse of components.

4.3.1 Non-Technical Characteristics of Components

When reasoning about the reuse of a component, users need to know some non-technical information in order to build their final decision. The choice between two technical equivalent components sometimes relies on items like the developer company or the type of support offered [PSB00, PhBr00]. Experts affirm that users want to reduce the post sales problems and need to know some details about the suppliers beforehand [PhAr01, PhBr01]. These characteristics are often treated as irrelevant but they do have a great importance and must be represented in a meaningful component management system.

With respect to the organizational perspective each component must have an identification mechanism able to make its recognition unique. Additionally, a component may have different versions with different implications on its functional and operational behavior. Each new version of a component may be presented with the respective date of purchase. The price of the component may vary within its different versions. New versions could have different prices, in some cases even cheaper than the one of the previous version. The end price of a component is supposed to be advertised whenever the policy of the company permits. Nowadays, it is common to present a scale of pricing depending on the services offered, as for example the kind of support the developer provides. The diverse ways of support the company is able to offer must also be clearly specified.

Every company has its history of good and bad services that is associated with an image of quality. Components are delivered by these companies and normally developed by employees of these companies. The publication of additional information about the employee technically responsible for the component is a way of reducing the distance between users and technical people that influence the reuse decision. Users that can solve their doubts about a component in a fast and direct way increase their confidence in the company, leading to a better overall evaluation.

In Figure 4.6 the most relevant non-technical information about components is summarized. This knowledge is fundamental to support users decisions about reusing available components and might not be omitted in a meaningful management concept. When talking about a

component management system based on a repository of components used by different component developers and by different component users, the information represented in that figure must be given whenever possible. That is the approach considered in this thesis.

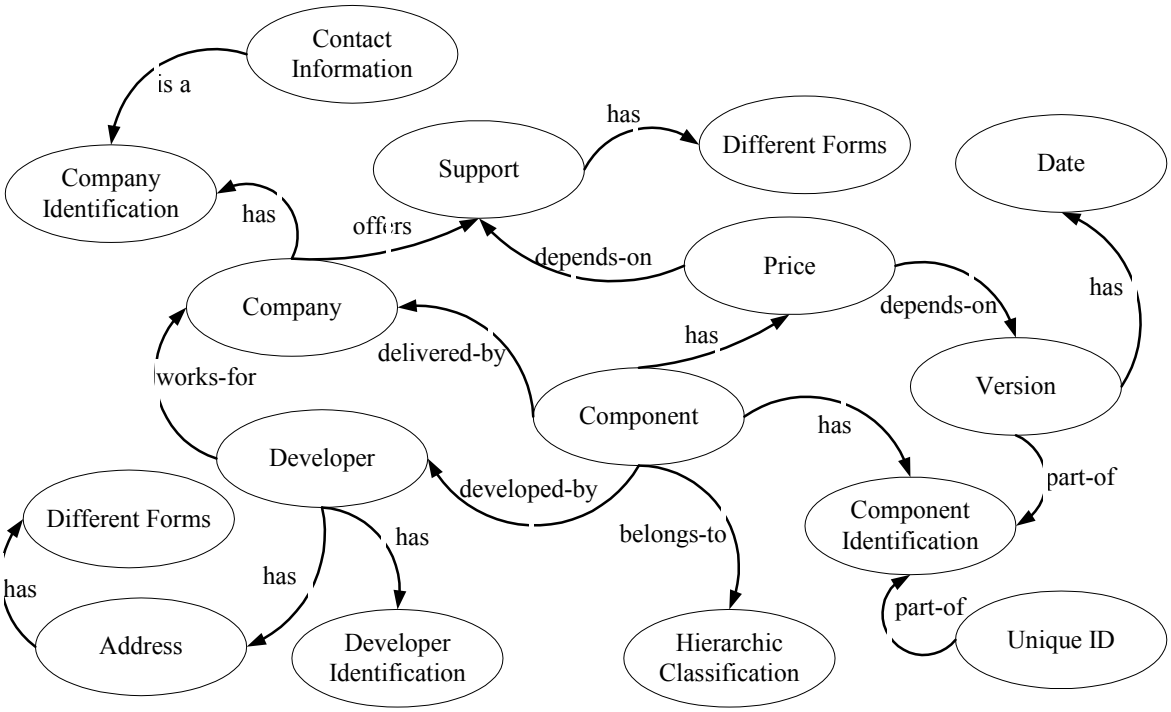


Figure 4.6: Representation of the Non-Technical Information of a Component.

Some other social and organizational factors towards component reuse were discussed in [KuBr00]. Interesting is the position that the major cause of most failures in the reuse of components process is the people involved rather than technical issues. It is not enough to show that a component fulfils every single requirement of the user, it is necessary to convince the potential user that a component based approach is good for the project and, consequently, good for the application developers themselves.

A well succeeded component management system must take facts like those into consideration making the relationship between component users and component developers as easy and convenient as possible. The component management system proposed in this thesis introduces communication mechanisms among those users and facilitates the relationship mentioned above. Nevertheless, before arriving at this level of decision, potential users should make considerations about the technical characteristics of the desired component. Even the most attractive component from the point of view of costs may not be used if it does not attend an excluding technical requirement.

4.3.2 Technical Characteristics of Components

In order to describe better what a component can do, it is necessary to precisely document three aspects:

- A functional specification, which describes the component's semantics. This explains what a component does and how a user should use it.
- An implementation description, which details the decisions of the component designers on how to construct the components and on how to store the data meeting the intended specification.
- An executable which delivers the component's capability on a designated platform.

In other words, a component is described in a set of technical information which is supposed to be organized in functional-related information, the specification of a component, and in operational-related information, the implementation design of a component. All these aspects are also represented by an executable which depends on a specific environment to precisely realize the advertised functionality. It is worth emphasizing that not all three aspects have to be presented to make a component reusable. For example, the developer of a component may be happy to publish details of the specification to maximize the component's visibility, but may not want to publish his implementation design which he may regard as his own intellectual property.

The specification of a component describes what that component does, and how it behaves. The functionalities are rendered by some designer who provides an implementation for the component, expressed in terms of code and data which is guaranteed to meet the specifications. In general, the implementation is written in different programming languages as well as it executes on different platforms, from the users' language and platform. One component may be replaced by another in an application, as long as both implement the same specification. The split between specification and implementation is the essence of the term encapsulation.

The importance of a rich component specification is critical. It is the functional specification that allows component users to understand quickly what functionalities are offered by the component. It allows users to match the component's capabilities to their specific needs. Useful components exhibit varying degrees of completeness and usefulness in their specifications, ranging from natural language description to highly formalized models and graphics. The specification of components also varies in accordance with the technology used.

A specification describes the functionality of a component in terms of an interface description complemented by a behavior description. Interfaces summarize how a user should interact with a component but still hides underlying implementation details. In fact, one interface has no detail about how the data, which is managed by the component, is organized and stored. The internal working of the component is completely hidden. Component users may only utilize them by calling the operations defined in the interface, no other access is possible. The

complementary behavior description assumes different forms. Some approaches use diagrams to explain the behavior better, others prefer a set of pre- and post-conditions. A well described component behavior must have a combination of different information about each operation contained at the interface. Among this information is the operation name, its textual description, the parameters involved, the pre- and post-conditions, and the intent of the operations.

Although the specification of a component is independent from its implementation, the most appropriate way of representing interfaces and behaviors of components depends on the technology in which the component is supposed to be implemented. That is not only a technical specific problem, it involves also the repertory used by specialists in different technologies. Someone looking for the interface description of a JavaBean component wants to receive an API description as a result. Another one looking for a synchronous component interface wants to receive a list with signals, their respective types, and textual descriptions. In fact there is not a unique way of describing interfaces and behavior of components among different component technologies [NaTa01, Ster98, HeCo01]. Even looking at a specific domain as the industrial automation applications, specialists diverge on using a unique model for such description [Seli99, Kope00]. Some operational information must be given in order to clarify implementation details of the component. The most relevant knowledge about this aspect was already discussed in section 4.2. It is important to remember that the operational information necessary to decide about the reuse of a component is understood better if the developer is able to publish some usage examples. With these examples, component users can reproduce the divulged implemented functionality making the component's understandability easier. The necessary knowledge about the technical information of components is presented in Figure 4.7.

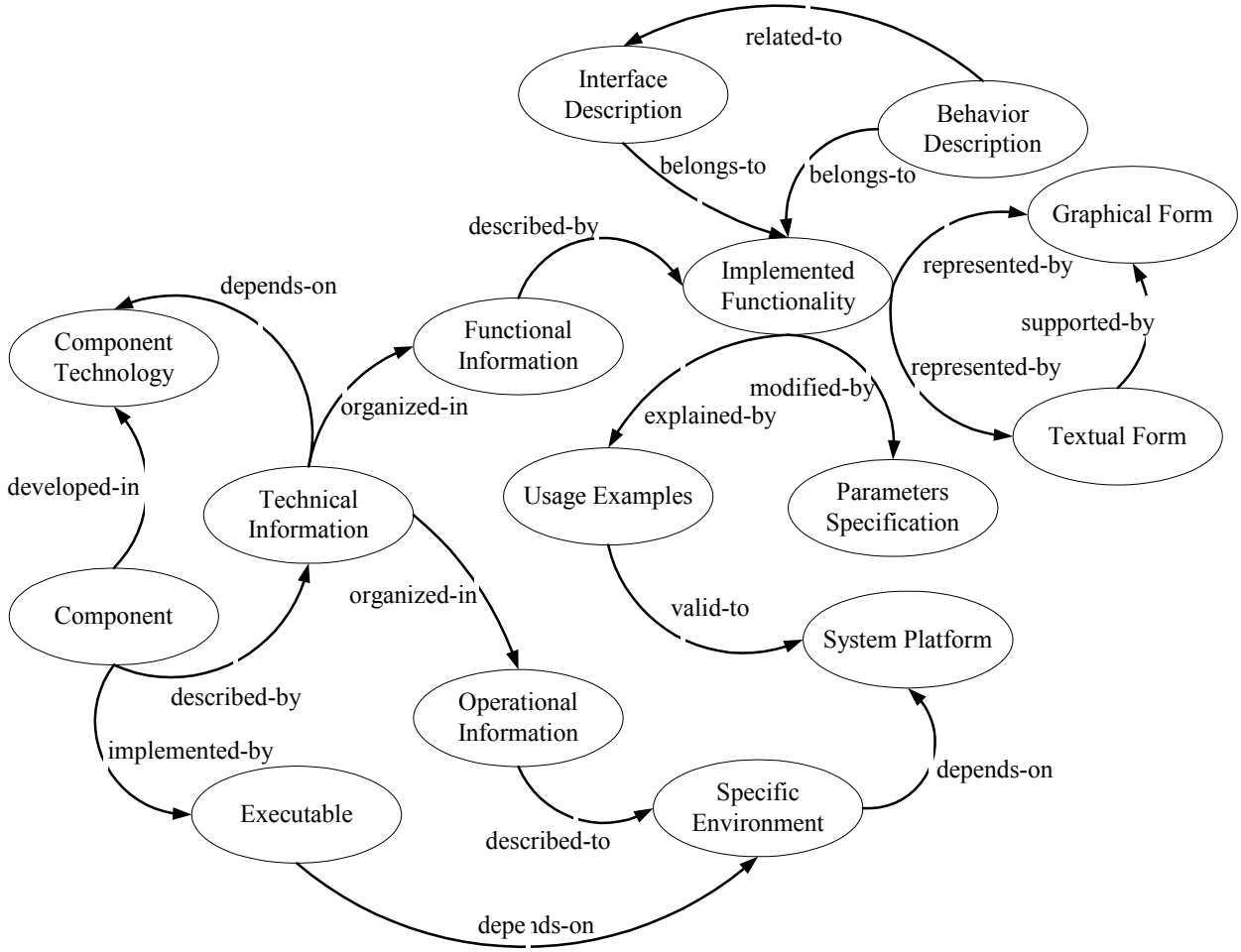


Figure 4.7: Representation of the Technical Information of a Component.

The complete information is easily obtained by the superposition of the last three figures, i.e. 4.5, 4.6 and 4.7. Note that someone investigating other aspects of a component could obtain quite a different knowledge representation. The semantic networks presented in this section were obtained exclusively for representing the knowledge necessary to understand a component well. Therefore they only represent the knowledge necessary to support the decision about reusing a component for a desired application. Additionally, the networks were designed considering the particular characteristics of the industrial automation domain. As written before, these semantic networks were based on the expertise of researchers and engineers working with components and working in the industrial automation domain related in the specialized publications and based on the experiments done along this research in the IAS. Based on this knowledge a concept to manage all necessary information about components for the industrial automation domain is proposed in the next chapter. In the following section a classification scheme specially designed for the industrial automation domain is presented. This classification scheme is based on the information available for components discussed in the last sections.

4.4 Classification Scheme for Components in the Industrial Automation Domain

Large repositories of components represent valuable assets but the larger they grow, the harder it becomes to capitalize them for reuse purposes. Among the main problems are how to extract appropriate components with minimal effort and how to do it with maximum efficiency. As presented in chapter 2, the automatic retrieval of software components has proven to work well for a specific component modeling, but approaches for general components could not reproduce the success of the first ones.

Given a set of entities (objects, concepts) represented by descriptors (keywords), the grouping of those entities into disjoint classes according to some criterion of descriptor matching is called classification. Matching may express some kind of semantic similarity. A classification scheme determines how to perform classification in a given setting, prescribing the sets of descriptors and possible internal ordering, matching criteria, and rules for class assignment.

Depending on the number of descriptors used, a classification scheme can be uni- or multi-dimensional. A classical example of uni-dimensional scheme is the Universal Decimal Classification [Robi79]. In library science, multi-dimensional (also called faceted) classification, was introduced by Ranghanathan [Rang57], breaking down information into a number of categories thus addressing corresponding aspects of the classified entities.

There are two kinds of relationships a classification scheme may express: hierarchical and syntactical. Hierarchical relationships are based on the principle of subordination or inclusion, while syntactical relationships are those created to relate two or more concepts belonging to different hierarchies [Prie91]. The hierarchical approach permits a better conceptual classification of components.

4.4.1 Faceted Classification

Prieto-Diaz and Freeman developed a faceted classification scheme for software reuse [PrFr87, Prie91] in which they use six facets to describe software: function, object, medium, system type, functional area, and setting. The last three facets describe the internal and external environment. Each facet proposed has a term space, i.e. a fixed set of values, in the same sense of a controlled vocabulary, and has also an extensible set of user terms. Concepts are organized by a directed acyclic specialization relationship, and terms are assigned as leaves of the concepts.

A variant of the scheme of Prieto-Diaz and Freeman was developed in the ESPRIT REBOOT project [KST92, SSS93]. This scheme comprises four facets, suited better for describing object-oriented components: abstraction, operations, operates on, and dependencies. The first three are analogous to subject, verb and object in a natural language sentence describing component

functionality, while the fourth is the counterpart of the three environmental facets of the Prieto-Diaz and Freeman scheme. The terms spaces are also structured by relations such as specialization and synonymy. Classification in REBOOT fails to relate to structural dependencies that exist between components [SSS93].

Faceted classification offers certain features that not only improve the search, but also support potential reuser's selection process and contribute to the development of a standard vocabulary for the component attributes. Facets must be considered as perspectives, viewpoints, or dimensions of a particular domain. As stated in the expressions (4.1) and (4.2), in principle a component description C_d may have as many classification facets as desired. Each facet F_m may also have as many terms t_{mi} as wanted. In the presented notation the term t_{m0} represents that no term was given, in other words the component has no value for a specific facet.

$$C_d \supset \{F_1, F_2, ..., F_n\}_{\ n \geq 1} \tag{4.1}$$

$$F_{m \text{ where } m=1,...,n} \supset \{t_{m1}, t_{m2}, ..., t_{mp}\}_{\ p \geq 0 \text{ where } t_{m0} = 0} \tag{4.2}$$

The greater the value of n at (4.1) the harder it is to obtain a classification for the components. On the other hand, the choice of a small n leads to a restrictive set of facets that makes it impossible to implement any meaningful distinction among components. Nevertheless, it is important to emphasize that the choice of the quantity of facets that will compose the classification scheme is better to be done in a qualitative way, i.e. the chosen facets must properly describe the components in their specific domain.

4.4.2 Industrial Automation Classification Scheme

Faceted classification of elements has been already applied to many different engineering domains [MMM95, MAGM97]. Common sense shows that the conditions to use facets successfully by the classification of an element are the following:

- A facet must be composed of terms, that meet a meaningful description of the possible users' requirements
- A facet must be oriented for reuse
- A facet must be organized in accordance with the relevance for the user
- The amount of facets that will compose the classification scheme shall be limited
- The classified element may be associated to as many sets of facets as needed.

The facets adopted for the component's taxonomy used in this thesis are based on the theoretic knowledge already acquired about components and about the industrial automation domain. Three main aspects are covered by the chosen set of facets: the functionality implemented by the components, the way they perform it together with implementation details, and finally the

description of the domain where the components should be used. In other words, with this set of facets the participating components may be classified towards what they do, how they do it, and where they do it.

Looking back to Figures 4.5, 4.6 and 4.7, it is possible to infer that some of the presented information will be decisive when reasoning about reusing components. Some other points help to decide about the viability of some components but do not imply in the acceptance or rejection of them. In order to specify what a component does, it is important to know the implemented industrial automation task and the associated functionality. How the component implements its functionality is expressed by the choice of a technology, i.e. hardware or software, and among the software components in any of the technologies that are used in industrial automation applications. The system platform is also a decisive factor and must be represented by the operating system and associated hardware platform. The real-time environment must be expressed by the expected component's time behavior and the possible trigger type. Finally, in order to know where a component is supposed to be used, someone needs to know to which sub-domain and to which hierarchy level it belongs.

In this thesis a classification scheme, from here on referred to as "Industrial Automation Component Descriptor – IAC_d", is defined which is composed of eleven facets covering the main aspects and terms used by industrial automation engineers [Luce01a]. The expression (4.3) represents the descriptor. Following, a textual description of each term is presented.

$$IAC_d \supset \{F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10}, F_{11}\} \qquad (4.3)$$

- F_1: Application Domain – Industrial automation itself is a large domain that has many sub-domains. In order to determine the possible usage of a component better, developers need to specify as near as possible where the component is supposed to be used. For example, a component belonging to the domain "Product Automation" should be classified as "Home Entertainment" another one should belong to the "Medicine Systems" sub-domain. One "Plant Automation" component may belong to the sub-domain "Building Systems".

- F_2: Specialization of the Domain – This facet is used to specify the component's area of usage in more detail. The examples mentioned above could receive terms like "Image Adjust" as specialization for "Home Entertainment", or "Fire Protection" as specialization for "Building Systems".

- F_3: Hierarchic Classification – Industrial automation applications are distributed in a hierarchical classification in accordance with their management level. The adopted terms for this facet were originated from the classification discussed in [LaGö99], where three different aspects are presented. These aspects are the levels of hierarchy, levels in distribution, and levels of a company. The adopted terms are Business Level, Production Level, Process Control Level, Process Variable Level, and Field Level.

- F_4: Industrial Automation Task – Here standard terms of the industrial automation domain express the category of the component's functionality. As far as possible, terms used in the ACM's Computing Classification System [Coul97, ACM98] must be used. Some examples of usual terms are: Input Device (Sensor), Output Device (Actuator), Supervision, Control, Command, Communication. No precise description of the implemented functionality is given here, but only a high level description of the component's usage.

- F_5: Implemented Functionality – This facet contains the implemented functionality of one component. As the goal here is to publish the component's characteristic many terms may be needed to summarize all capabilities of one component. A complete textual description of the component's functional characteristics is part of the complete documentation given later.

- F_6: Trigger Type – As seen before, event-triggered and time-triggered components are of great importance at industrial automation applications and have quite distinguished usage. Users interested in components for event-triggered applications will probably have no interest in searching among time-triggered components.

- F_7: Real-Time Characteristic – Three standard values are adopted for this facet, i.e. Hard Real-Time, Soft Real-Time and No Real-Time. Deciding which approach needs to be taken is part of the earlier stages of analysis when working on such industrial automation applications.

- F_8: Component Technology – A specific software functionality is implemented using different component technologies, this kind of information is expressed at this facet. In some design processes the decision about the usage of specific technologies is already done and must be respected. When talking about hardware components this facet receives values as for example TTL or MOS technology, which give the user a precise insight into the component capabilities and limitations.

- F_9: Hardware Platform – The originally planned hardware platform is expressed in this facet. In some cases different components implement the same functionality for different hardware systems presenting different response-time in accordance with where they will be used. On the other hand, it is possible that the decision on where the application will run is already taken and the searched component must work under this limitation.

- F_{10}: Operating System – As seen before many decisions are taken just by choosing the target operating system. Nobody should wish to obtain any real-time characteristic from a component if it was designed to be implemented under a non real-time operating system.

- F_{11}: Type of Component – The proposed classification scheme is supposed to work with hardware and software components. Which kind of component will be used in any application is a fundamental decision that should be taken at the beginning of the project.

A formal representation of the classification scheme developed specifically for this thesis is presented in Figure 4.8 using the UML notation. The facets Hierarchic Classification, Trigger Type, Real-Time Characteristic, and Type of Component have pre-defined values that must be respected. During the classification process some value suggestions for each facet may be offered based on a set of values already recognized as important for the domain. Developers may adopt one of these values but may also have the freedom of introducing new terms whenever they believe these new terms should describe their components better.

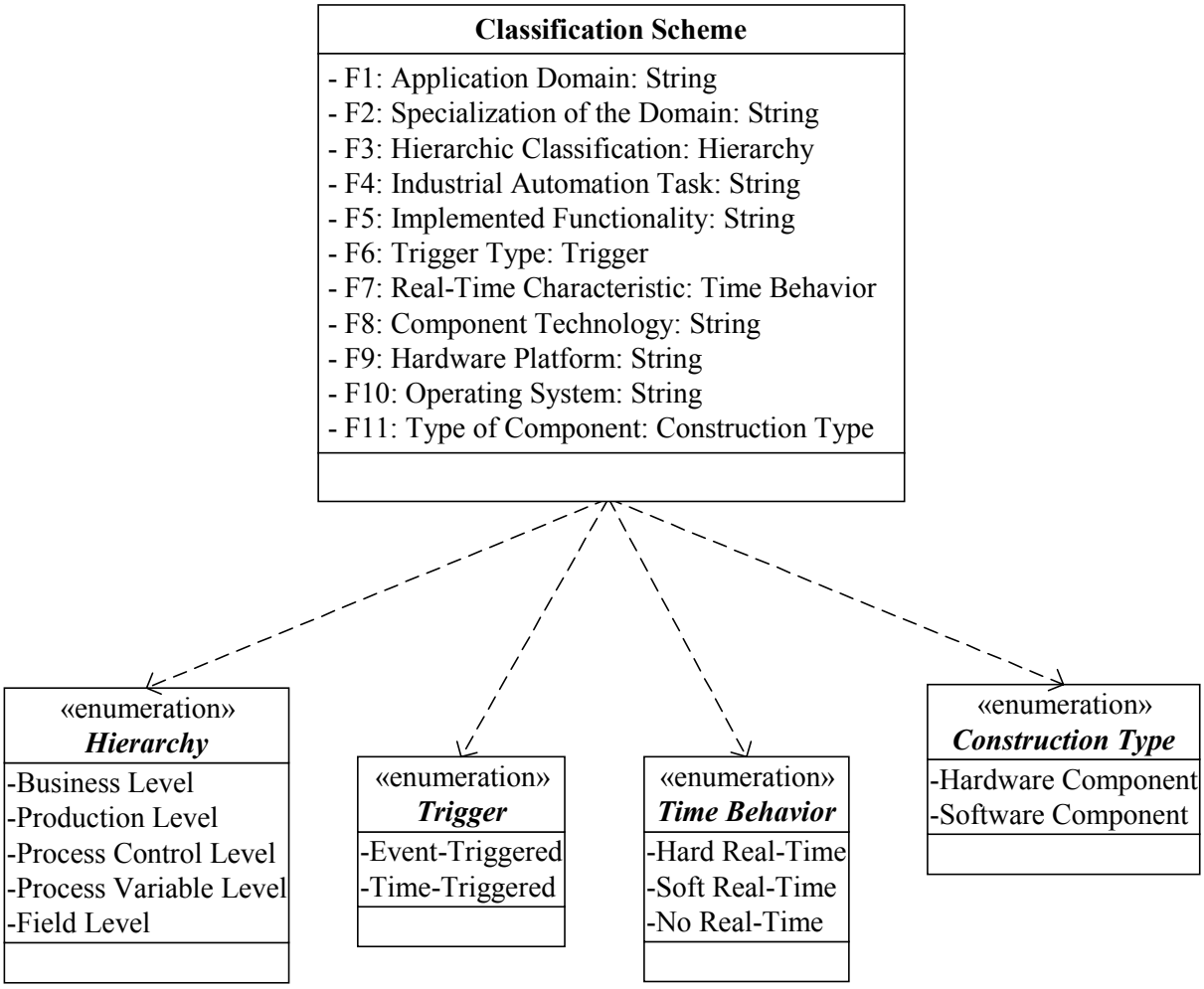


Figure 4.8: UML Representation of the Industrial Automation Component Descriptor

Note that the facets of the classification scheme derived from the semantic networks proposed in the previous sections of this chapter. The facet F3 comes from Figure 4.6 where the non-technical information of a component is represented. F8 comes from the representation of the technical knowledge about one component shown in Figure 4.7. The other nine facets were derived from the knowledge representation shown in the semantic network of Figure 4.5. That network contains the particularities of components used in industrial automation applications which confirms the specificity of the classification scheme proposed in this thesis.

4.4.3 Classifying Industrial Automation Components

Classification is an interactive process that must be used to help users wanting to find a component later. Users understand the functionality of a component by studying documentation, by static analysis data, or by studying the code itself. Term understanding is supported by the linguistic form of the term, by its position in a hierarchic structure, through text comments on its meaning, or by its usage in the classification of similar code already known. Users must decide if a given term matches the component's characteristics and if the term is specific enough to describe its real objectives. With use and experience the term space becomes more and more stable and complete. A user must be aware not only of the meaning of terms, but also of their expected quality. The following criteria are proposed for selecting terms for the industrial automation:

- Terms must be well-known words, usually technical terms or expressions, widely accepted in the domain community, or at least by experts in the particular domain of interest.
- Terms must have clear meanings, be relative and easily associated with the concepts covered by their specialization, in the classification structure. Moreover, they must be distinct and precise, in order to facilitate the direct linking of the component to the corresponding classification term.
- Terms must also be general enough, in the sense that a term may encompass more than one specialized term in the classification structure. In other words, every term may be used to address more than one component, or a specific set of components. Keeping a set of general terms, therefore small enough, and expressive at the same time, thus useful for the reuse process, is one of the basic and most difficult tasks in classification. On the contrary, keeping a large term space usually means confusion for suppliers and reusers of components, inconvenient browsing, and poor search performance.
- Redundancy must be avoided, in the sense that there might not be two terms with very close meanings in the same classification hierarchy. If this happens, then a synonym relationship shall be created, with only the most representative terms presented in the classification hierarchy.

In order to improve the quality of the terms used, component developers are supposed to be supported during the classification of their components, which is the last phase considered at the development process developed for this thesis. As it is explained in the next chapter, this support is done presenting a well organized assistance, which is responsible for describing the facets used and to offer taxonomic values for each facet of the scheme. A list with pre-defined values are used as a base for the classification of new components. These values shall be presented dynamically, i.e. if a user decides to classify a component as belonging to the application domain "Automobile" (the facet Application Domain will receive the value

"Automobile"), a list with suggestions for the facet Specialization of the Domain will only contain meaningful values in reference to the first choice (e.g. ESP, ABS, etc).

Nevertheless, the responsibility of a meaningful classification of components belongs to the developers, the ones who know the exact usage of their components best. Developers may adopt one of the suggested values but may also have the freedom of introducing new terms whenever they believe these new terms describe their component better. These new values must be automatically introduced at the repository and may be used later as a suggestion during the classification of new components. In Figure 4.9 a classification example for one software component is presented.

	Component X
F_1**: Application Domain**	Medicine System
F_2**: Specialization of the Domain**	Patient Observer
F_3**: Hierarchic Classification**	Field Level
F_4**: Industrial Automation Task**	Sensor
F_5**: Implemented Functionality**	Pulse Frequency Measurement
F_6**: Trigger Type**	Events-Triggered
F_7**: Real-Time Characteristic**	Hard Real-Time
F_8**: Component Technology**	C++
F_9**: Hardware Platform**	PC Based
F_{10}**: Operating System**	Windows NT
F_{11}**: Type of the Component**	Software

Figure 4.9: Example for the Classification of one Software Component using the IAC_d.

In this chapter the most relevant ways of representing knowledge about objects were discussed. Afterwards the knowledge necessary for the understanding of a component developed to be used in the industrial automation domain was modeled using the semantic networks technique. This representation was obtained after a discussion of the main characteristics of industrial automation systems. One classification scheme is a meaningful issue used to index any object permitting its later finding and grouping. In the last section of this chapter a classification scheme composed of facets specially chosen for the industrial automation domain was proposed.

The concept of a flexible component management system able to work with components designed for the industrial automation domain is the main theme of the next chapter. There, an optimized development process based on components is presented. The elements of the proposed component management system are detailed and lastly the complete architecture of the desired system is presented.

5 Conception of a Flexible Component Management System

The representation of the knowledge about components used in industrial automation applications presented in the last chapter needs to be meaningfully organized and manipulated in order to permit that users accessing already developed components have a good chance of understanding them and deciding about their reuse. In this chapter, how to deal with that representation in order to construct a flexible component management system is presented. First of all, how somebody can manage components considering the usual application development process is discussed. The necessary steps for a meaningful development of applications based on components is divided into two procedures, one involving the component developer and another one involving the component user. The second section treats how to properly manage the components involved on that development procedure. The most current way of developing component management systems is presented and, afterwards, an ideal conception for a component management system specially developed for this thesis is discussed. This system is able to work with different component technologies and present an adaptable interface to the users and a variable data structure.

5.1 Application Development Based on Components

The main objective of component based development is the implementation of application systems by the assembly of prefabricated building blocks. Over the years stabilized industries developed their own ways of producing products based on already available components. In the construction industry for example, it is well known that it is impractical to build everything from scratch every time. Doors, windows, and bricks are all built off-site, often by different companies. In most cases, standard construction components will fit the needs of designers. At other times, a new component will be specified and built to suit the needs of a single building. The hardware electronics industry proceed in a similar way. Components that may have diverse usage are designed and built in series in order to be reused later in specific hardware projects. In this industry the usage of available standard components is already planned at the very early phases of the development. Very often, designers adapt their work to fit to the standard components they were able to find on the market. The goal of software development based on software components is not different from the objective of the two examples mentioned above [BrWa98, RNJ99].

In order to be successful, the development process based on reusable components depends on two main factors, the component developer and the application developer (component user)

[McCl95]. The main job of component developers is firstly to investigate what kind of functionality is needed and secondly to develop the final artifact. Component developers also receive the requirements of desired components from system analysis teams or obtain these components from legacy code [CAP95, GWS96, KeSc98, Trau98]. After the necessary development work, the component is tested in order to verify its functionality and, if approved, it must be stored in a repository to be used later [BSST93, Henn96]. Indeed, since the beginning of the development process it is necessary to invest as much as possible in the construction of standard software components. These pieces of software shall be generalized enough to be easily utilized in diverse future projects [Frei00, LuFr00]. On the other hand, components designed to implement very specific tasks tend not to be used in many different applications but may be used many times in a same application. In other words, software components must be designed for reuse being used later on real application development as many times as possible [Radd98, Weyu98]. By their side, application developers are interested in building their work based on components. The development of applications based on reusable components starts with the reception of the systems requirements. Application developers realize their analysis based on the components available, i.e. during the analysis they search for existing components in order to guide their work.

The development process described above have not solved some problems yet. Components are not like Lego bricks, where anyone with a little experience may obtain the requirements to meet the standard size for plugs and sockets. But lacking the Lego users' ability to guess, discover and experiment with different pieces during construction, how should component developers convey the functionality of their work to prospective customers? How should application developers address business problems given that the solutions are intended to comprise a number of assembled components? Now consider the task of finding a suitable component. How good are the components described in the repository? When given the possibility of using a component, how will application developers know what the component does? How will they know whether it is a good fit for the requirements? Given the prospect of reusing many well described components from a repository, which one shall be the best fit for the requirements of the current job?

In fact, in order to solve the questions mentioned above properly it is necessary to propose a systematic procedure for the management of components. The systematic management procedures proposed in this thesis are able to guarantee that all information necessary for a good understanding of a component will be previously stored in a common repository. Additionally, this information will be easily found and presented for interested users making use of modern internet technologies. These management procedures are presented in Figure 5.1 and must be inserted on the development process in two separated parts covering the side of the component developer and the side of the application developer.

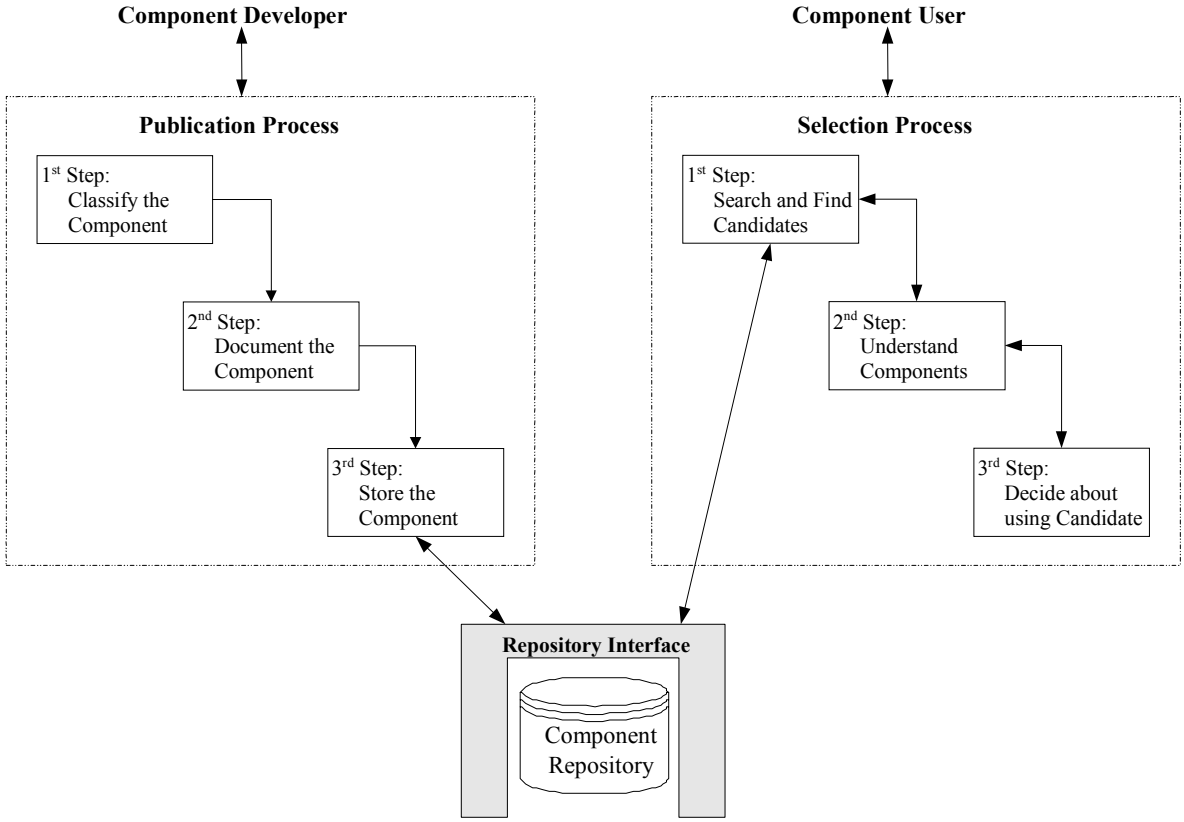


Figure 5.1: Main Steps of the Publication and of the Selection of Components Processes.

Just after concluding the development of a component the one responsible for this component must store it in the repository [SKS93, FrFo95, BUSF98]. This developer is supposed to insert a series of information that is important for the later finding, understanding, and deciding about the use of the newly developed component. This procedure is done in three steps, a classification of the component, its complete documentation, and lastly the saving of all information in the repository [Poul94, Gree99]. The denomination adopted here for this procedure is "Publication Process". On the component users side a procedure denominated "Selection Process" is proposed which involves the search of a desired functionality, the technical understanding of the found component candidates, and a final decision about their utilization [Luce01a, Luce01b]. As a result of the search of components, application developers must receive a list with all available components that fulfil the functionality they are searching for. As a matter of fact, there is not yet a way of assuring a minimal quality measure for the components or for the information divulged about these components [CTW98], in other words there is no guarantee that the components attend their requirements completely. That is why in the proposed procedure, the next step must be a detailed understanding of the suggested components. Some of those selected components may not exactly fulfil the applications requirements and constraints but, on the other hand, some components may be exactly what users need. Lastly, developers are able to decide, among the components that work properly which one suits their application better.

The contents of the component information (documentation and classification) are based on the discussion presented in chapter 4, their organization is discussed later on in this chapter. The simplified representation of the publication and selection processes is presented in Figure 5.1 and is improved in Figure 5.2. In fact, intern to each step presented in that figure there are some important procedures that must be taken into consideration in order to guarantee a meaningful management of components which are discussed in the next paragraphs.

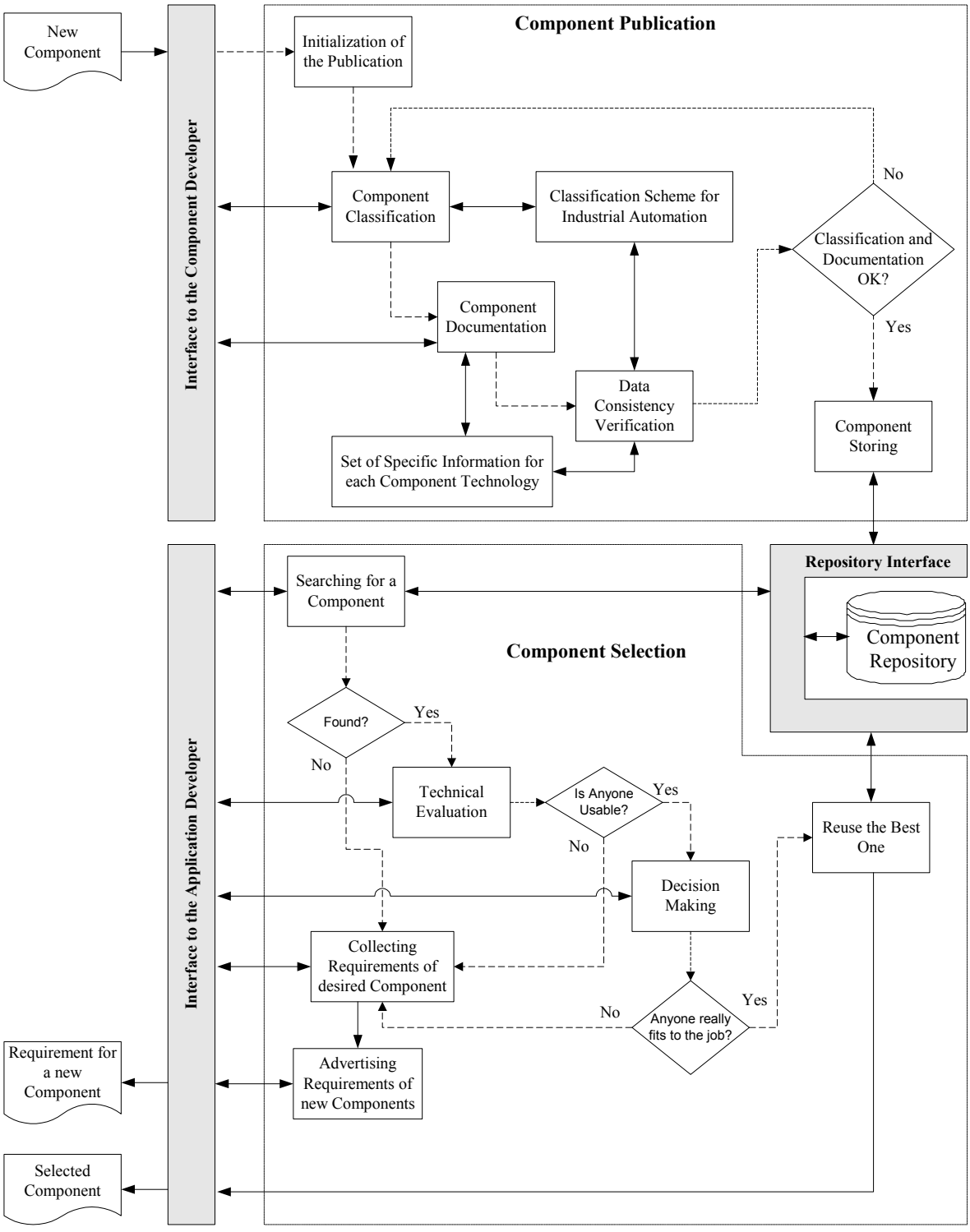


Figure 5.2: Details of the Systematic Management Concept proposed in this thesis.

5.1.1 Selection of Components

The first step towards the selection of a component consists of searching candidates that fulfil some desired characteristics. The goal is to provide system developers with a list of components which, in accordance with their needs, may fulfil expected requirements [O'De89, Cham96]. The second step, the technical evaluation, must offer a set of information that helps to certify that the presented components meet the desired technical requirements completely. At last, during the decision making step, another group of information will be studied by developers in order to support their decisions if a component shall be used in their applications in accordance with commercial aspects.

The way someone searchs for a component varies, but has some well identified common points [MiNo99]. For example, in order to perform the identification of possible candidates, users start searching for information related to the functional characteristics as well as related to the operational characteristics of the desired component. In other words, they search using terms that may limit the number of possible candidates and they take into consideration some previous technical decisions. Someone who has a good insight into the desired object is able to retrieve it easily by choosing convenient keywords in a free text search engine. But in the case of newly users or developers with less knowledge about the taxonomy involved at the domain, better results are obtained through some kind of guided search or even through a complete browsing on the objects available in the repository [Belk00]. Information like the application domain name and some kind of specialization of this domain, the automation task to be fulfiled, the operating system, or the hardware platform are good examples of searching criteria. Note that these are some of the items that compose the classification scheme of the last chapter.

In the second step of the selection process, the number of components are expected to be reduced to a small set of potential candidates. Component users must now study the candidates, understand them, and lately verify if these candidates really meet their needs. The main work here is to check the divulged characteristics of the component, eventually through some test situations, either in the actual development environment (or deployment environment) or in closely-related environments, to confirm that each component runs as advertised, and meets the desired requirements [SBP99]. Typical information required at this level are functionality textual description, parameters and interface description, usage examples, response-time and resource usage constraints, and performance verification among others. At this point some support is desired in order to facilitate the user's evaluation process. The use of checklists, where the most important original requirements are presented, facilitates a confrontation with the divulged component's characteristics. This may help to conclude if a component is applicable or not. The technical evaluation is an excluding step but it is not enough to conclude which component is the better one for a specific application. In fact, more than one component may have the same characteristics and equivalent performance, and shall be eligible for the next step where the final conclusion will be met.

At this final step, component users work only with the components that were approved in the previous ones. The goal is to determine which component may be integrated into their application. In this thesis, a commercial evaluation is proposed in order to compare all previously selected components enabling users to formulate their final decision. It is assumed that the amount of possible components that could arrive at this step will be small. But the final evaluation procedure must be executed even if only one component could be pre-selected, because items like the developer marketplace position and the component's price may be an exclusion factor. Some other comparison points that users may be interested in are the kind of support offered by the component developers, the quality measure of the component itself, or one more specific technical aspect like the expected memory usage. In this thesis, an adaptable comparison table is proposed to compare components that arrive at this last phase.

The desired final result of the selection process is the choice of one component that may be used in the application under consideration. In the most successful situation this component will be the best one of a set of components with similar characteristics and performances. Nevertheless, this situation will not occur every time as in some other cases it will not be possible to select any component that could fit to the desired requirements. Sources for this situation are the facts that sometimes the desired component does not exist, or a probable component does not work as advertised, or even a usable component does not meet the desired non-technical requirements as, for example, the available budget. In the work related here, whenever a desired component is not available, an individual list containing the user requirements are generated, being used later in the possible development of new components.

5.1.2 Publication of Components

The main goal of the publication process is to guarantee that component developers will feed the repository with the most relevant information for a later selection. In accordance with chapter 4 that means that a domain specific classification and a complete documentation of the component must be done during this process. The whole work starts after the completion of the component development. This new component must be classified using the previously presented classification scheme for industrial automation applications. If a component is "well classified", the probability that it will be found later increases [BrBu00]. In order to improve the conformity of the classification of new components some help and guidance must be offered to developers, making clear the meaning of the facets of the classification scheme and giving hints on the possible taxonomy value for each one of them. This approach generates a semi-controlled vocabulary increasing the probability of possible hits later on during the search of components [LHG+96]. It increases the knowledge acquisition creating a list of terms chosen by the experts.

The second important aspect is the documentation of new components. As the way of documenting a component is different among different developers and certainly very different

among different component technologies, it is necessary to support developers to feed the repository with the information considered relevant for the later understanding of this component [MLP+01]. A common set of information listing the necessary data for each component technology must be available and should guide developers during this step of the publication process. A separation among mandatory and desired information is also a useful mechanism that must be supported.

The responsibility of classifying a new component and to document it accordingly belongs to component developers themselves as they know the characteristics and potential usage of their products better. This procedure enhances the quality of the data stored at the repository [BaTa99]. Nevertheless, before storing a component it is necessary to verify if the data introduced is consistent with the data needed. In the procedure proposed here, components are stored only after the confirmation that their classification and their documentation were done in accordance with the classification scheme and to the data needed for their respective technology.

5.1.3 Interfaces to the Users and to the Repository

As presented in Figure 5.2 two interfaces must be considered, the interface between users and the component management system, and the interface between the component management system and the repository. One deciding factor for the success of such systems is the good acceptance of the interfaces to the users [BMJH96]. It is important to remember that component developers interact with the system in order to classify and document their work, and also component users are supposed to be guided during the whole selection process. These users' interfaces must be intuitive and convenient otherwise the system will not be well accepted [LaMy01]. The proposed component management system is planned to work with diverse component technologies and their specificity must be taken into consideration. This fact will reflect on the interface design that must respect the characteristics of the component technologies leading to better communication with the users.

Additionally, the insertion of new component models representing future relevant component technologies shall be done without great efforts. Nevertheless, this insertion will ever demand the specialized work of a system administrator able to deal with the chosen technology. In this thesis no specific interface to this user will be offered. The interface system-repository is responsible for the checking of the consistency and of the integrity of the data. It must also be robust enough to avoid loss of information. The data captured by the component developer interface shall be processed and stored on the repository in such a way that the component could be found and understood later. Different component technologies are supposed to receive different processing. Similar considerations are valid for component users wishing to receive information adapted to the technology they are looking for. The way of storing the data about components must take these requirements into consideration.

5.2 Development of Component Management Systems

In this section a concept for a component management system for industrial automation components is presented. As seen in chapter 2, many other component management systems were already proposed in previously related works. It is important to study, at a conceptual level, how such systems were originated. During this study, some pros and cons are discussed. Afterwards, a management concept that takes the particular characteristics of components used in the industrial automation applications into consideration is presented.

5.2.1 Usual Conception of Component Management Systems

Components are developed to be used later as many times as possible. In the last years many component technologies proposals and many component-based development approaches were related in publications [Szyp98, HeCo01]. Component management systems were also generated to care about the most relevant component's information trying to present it as clear as possible and making it accessible for a great number of users. A common way of maintaining this information readable and understandable is to adopt a model able to describe the component's technology in a simple but complete form. That is the point where the conception of a component management system starts. Figure 5.3 shows the conventional design path that leads to the implementation of a component management system.

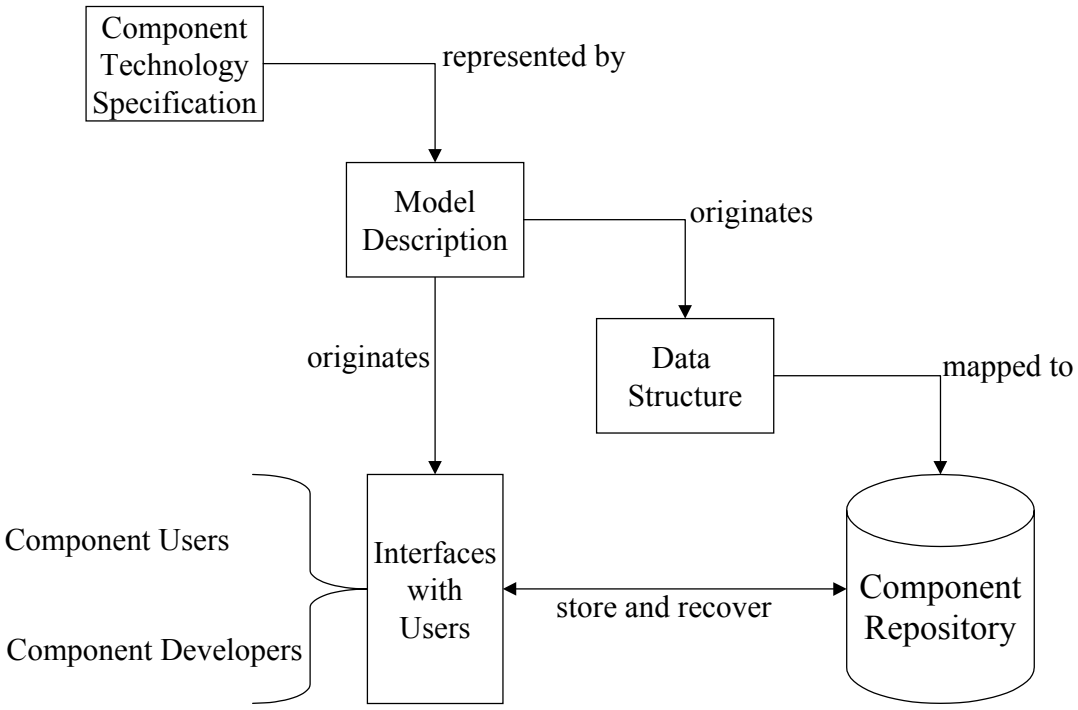


Figure 5.3: Conventional Design Steps of a Component Management System.

After specifying a component technology in detail, experts are able to propose a model to represent this technology properly [FrTe96, Digr98]. Models are developed to simplify the real

world and are supposed to communicate, as far as possible, all the relevant characteristics of the modeled elements. Using a common model is the most appropriate way of communication among diverse users of a same element [BRJ99]. From the inherent complexity of component technologies nowadays, it is possible to conclude that a model representing any of these technologies will suffer some modifications and corrections along the time [Wieg98], and it will take some time until such a model reaches a mature standard.

A mature and well accepted model is the basis for a meaningful design of the repository where the component will be managed. Firstly the model, containing all necessary elements for the understanding and later reuse of a component, must be used to originate a data structure mapping all its elements. Every single component's information, as for example the graphical way of documenting a component's behavior, must have a corresponding element on the designed data structure. In a second step, designers using the obtained structure are able to project the component repository itself.

Those components and the information about them must be stored at the repository as soon as they are developed. The communication among potential users and the repository is done through user's interfaces that support component developers to correctly introduce new components on the repository and also support component users to find already stored components. These interfaces (it is expected to have at least two interfaces one for each kind of users) are strongly influenced by the model description in use. Clearly, the interfaces to component developers should ask them to introduce the information that compose the component's model, and the interfaces to component users can not offer information that was not previously stored in the repository.

This design of repositories and associated management mechanisms proved to succeed for systems working with only one technology of components. Even though, problems may occur if the system was designed based on an inaccurate model. The introduction of modifications on the model demands changes on the data structure specification and consequently on the repository itself [Poul96]. The interfaces will also need to be adapted to the new characteristics of the model. Depending on the technology used for the implementation of such a system those modifications could cost to much effort [BCE+97]. It is important to note that models are in their essence not precise enough to represent all aspects of real objects and even mature models may have mistakes.

Another undesired situation happens whenever a new component technology is supposed to be inserted on an existent repository. The same amount of effort and additional costs considered for the case of single modifications on an old model would be also applicable here [SuBa97, MFC+00]. A new technology will necessarily be represented by a new model description that will generate new data structures and interfaces. These two situations, the modification of an actual component model and the introduction of new models, are problems still to be solved

because the data structures usually adopted are not flexible enough to deal with such modifications. Moreover, interfaces are often designed using static approaches making them dependent from the model adopted.

Lastly, component repositories are normally implemented using relational database management systems what theoretically should not pose big problems to process possible modifications on its table structure [ElNa94]. But if those modifications are frequent, they may not constitute a desirable situation, because such modifications are supposed to be implemented by experts with special access rights. Otherwise, some security inadequacies may cause improper functioning of the system and eventually lost of information.

5.2.2 Ideal Conception of Component Management Systems

Some component management systems implemented using the sequence explained above, presented good results when applied to a single component technology (see chapter 2). In those cases, a well defined model was responsible for the repository mapping and for the interface design [Ning97]. Attempts to include more than one technology by adopting a model specification general enough to cover all desired component technologies fails because such a global model will never be specific enough to give the users all necessary information about one component technology and shall limit the possibility of understanding these components properly.

This kind of reduction of the representative quality of a model to increase its range of applicability results in a lack of precision on the component's information that reduces the chances of this component to be reused [Sate95]. In an ideal component management system, the model (or models) used to generate the repository data structure and the interfaces to the users must be as rich and precise as possible. Figure 5.4 summarizes the idealized way for the generation of a flexible component management system.

In this new way of managing components proposed here many different technologies may live together at the same repository. First of all, an accurate model description must be developed for each desired component technology. This model must represent all important aspects of the technology giving the necessary information for the understanding and for the reusing of components. Nevertheless, not all information contained on the model description should be considered mandatory. Component developers may want to omit some information they judge as business secrets. In other words, the model description must be complete but must also be flexible to attend individual user's wishes.

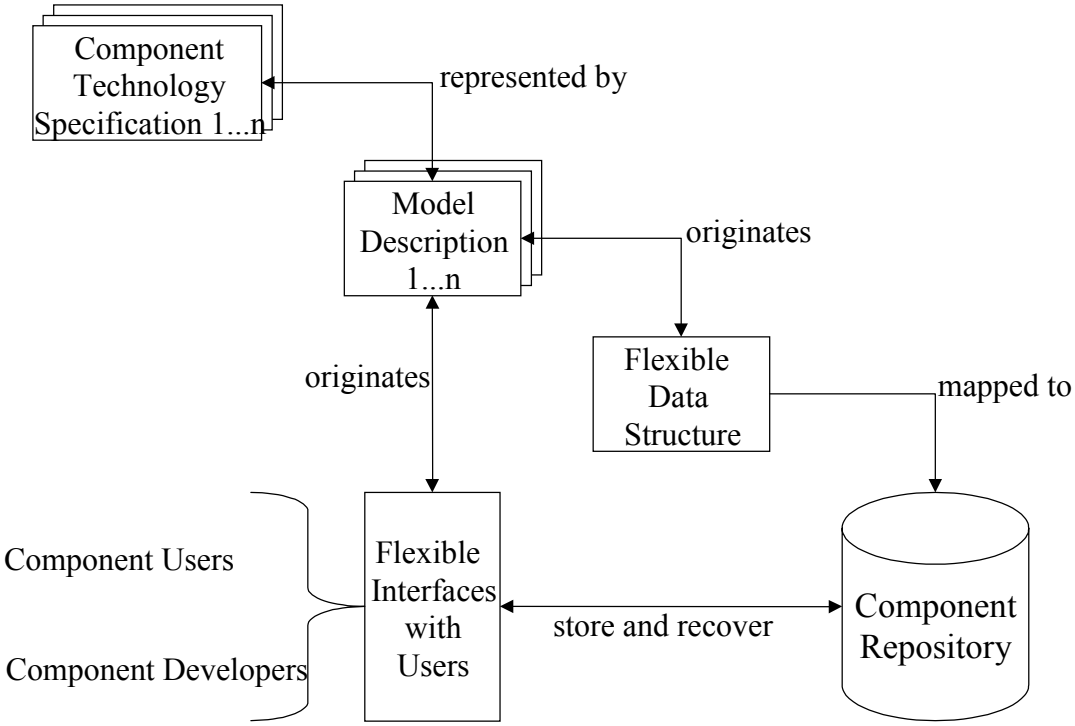


Figure 5.4: Ideal Generation Path of a Component Management System.

All available component models must be part of a set of model descriptions that will be used to generate the interfaces to the users and the data structure of the repository. The interfaces to the users must be able to adapt themselves to the type of user because component developers and component users may wish to receive different information [HBD00]. Additionally, these interfaces must be able to adapt themselves to the particular technology of the component being evaluated. The repository is expected to be flexible enough to accept different data structures, which are also derived from the models available at the set of descriptions. The insertion of new models or the modification of an existent model must generate an automatic actualization of both interfaces.

Observe that only two interfaces are considered in this component management system, one to the component developer and another one to the component user. It would be possible to foresee the necessity of a third interface designed to help a system administrator. This administrator will be responsible, for example, to develop model descriptions and to insert these descriptions into the component management system. Nevertheless, it was considered that the main focus of this thesis should be oriented to component users and to component developers. The insertion of new models shall be as easy as possible but it will demand the work of an administrator, who will have specialized access rights over the system and, more importantly, who needs to pursue specialized knowledge about the technology used to describe the component model and about the technologies used to integrate these new models on the component management system. Based on these considerations, a graphical interface to such an administrator is left as a suggestion for future works.

5.2.3 Modeling Industrial Automation Components Information

As seen in chapter 4, the knowledge necessary for the practical use of a component designed for industrial automation applications is very complex. Talking about the organization and presentation of this knowledge must result in models able to contain all the diversity of information required for each technology involved [Dai95, Pedn00]. The analysis of this information leads to a model structure divided into two blocks, one composed of information that may help users to easily find a desired component and another one composed of technical related information that may help users to understand the functionalities of a component and decide about its applicability. In summary, the desired model must help users to search and find, understand, and finally decide about the reuse of previously developed components [ShSh99], being able to contain the complex information needed to properly execute those tasks.

The data necessary for the search and find of a component is designed better for machine consumption, i.e. information characterized by a fairly regular structure, fine grained data, and little or no mixed content. The main objective is to be able to query fast and efficiently. This kind of information is denominated in technical publications as data-centric information. Such a set of data is well represented by a relational model. An example for such data is the information about the developer or about the company responsible for the component that consists of standard data as the address or telephone number. The classification scheme presented in chapter 4 is also suitable for search purposes.

On the other hand, the technical data about one component may be represented better as a set of information that varies in accordance with the component technology. This kind of information, may be characterized as document-centric information, and is typically designed for human consumption. The documents involved are composed of less regular or even irregular structures, larger grained data, and lot of mixed contents [PSB00, PhBr00, PhAr01, PhBr01]. This data is well represented as a set of document files with free content. In the case of components for industrial automation these data must fit to one specific component technology containing the information needed to well understand this technology. Note that the technical information varies strongly among the technologies applied to industrial automation systems (see chapter 3). This variation not only occurs in the content of this documentation but also in the form it should be presented in order to guarantee a good level of understanding by the users. For example, a user searching for a synchronous component wishes to receive the information about the component's interface, another user with interest for JavaBeans expects to receive information about the API reference of the component, and a third user searching for hardware components wants to receive the component's ports and pins description. Although all of them want to study the interface of the components, these users will understand the interfaces better if they are presented using the terms the users are accustomed to. This kind of variation is considered in the component management system proposed in this thesis.

Examples of the entities and respective relations involved for representing a component are shown in Figure 5.5 where a simplified representation of the two blocks of information mentioned above is presented. A meaningful component management system needs to help component developers to properly feed the repository with the data required by each model representation during the publication process, and will help component users to receive this information properly formatted and presented during the selection process. Note that the general information is the same for whatever technology involved. For that purpose, any kind of information that may be pertinent only to a specific technology type must be located at the technical information part. This technical part must be as detailed as possible for each technology used, but as said before the technical information is supposed to contain mandatory data and is also supposed to accept optional data that may be omitted whenever the users desire.

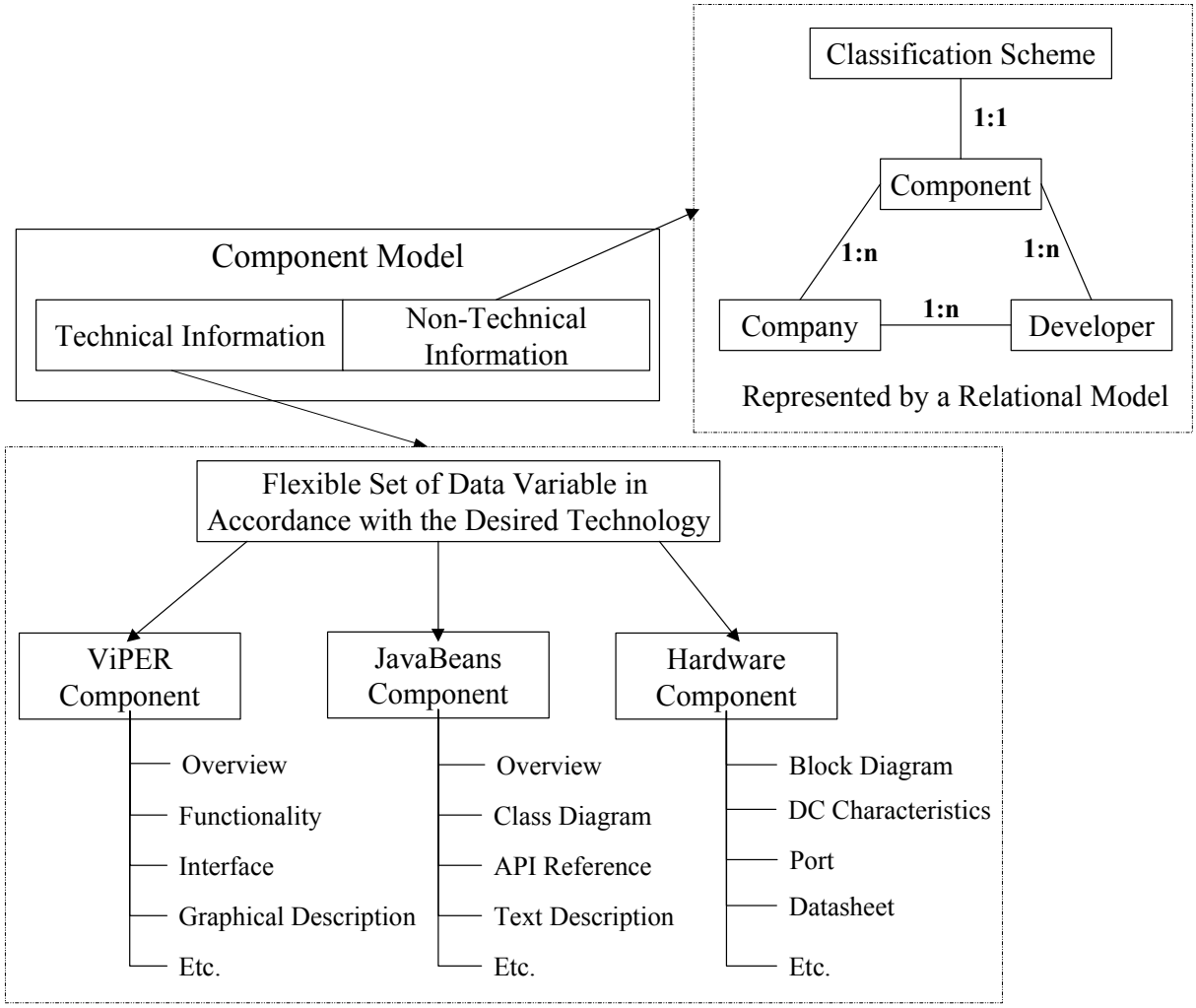


Figure 5.5: Example of the Information needed to Find, Understand and Use a Component.

5.2.4 Desired Architecture of the Flexible Component Management System

If the desired system was designed in the conventional way of developing component management systems presented in Figure 5.3 it would have the same undesirable problems cited in section 5.2.1 when applied to the management of components with any use at the industrial automation domain. The components' technologies used in that domain tend to be constantly improved because they have not reached a good level of maturation yet [Doug99, Kope00]. For example, experts have been working hard towards a meaningful and well accepted way of representing real-time characteristics components without much success in the last years [Seli99]. Additionally, from time to time new component technologies that may also be used in industrial automation applications come up on the market and should be incorporated to the component management system. As discussed before, a meaningful component management system must permit the introduction of such modifications in a simple and convenient way.

As seen before, it is not viable to propose only one model able to contain all the necessary information about every component technology used in industrial automation applications. A general model would not contain the level of information needed for a correct understanding of such components making it impossible to reach an acceptable level of reuse. A very specific model, based mainly on one technology, would solve the understandability problem for that technology but would certainly contain some information details that would not make sense for the other technologies. Using the abstraction presented in Figure 5.5, it is possible to conclude that part of the information about components is generally enough to be present on every desired technology. On the other hand, the specificity of each technology located at the technical information part must contain the desired level of details and must be unique for each component technology managed by the system.

In fact, information such as the address records of the company which has developed the component and some similar information are always independent from the component's technology description. It is true that the conventional way of describing a company address may vary, it certainly depends on the designer's particular way of working, but the content of this information will not vary and will have similar elements for whatever component technology discussed.

Based on these facts, in this thesis the information about the components is divided into two basic groups, one containing the same type of information independently from the technology, and a second one containing specific data for the particular information about the component's technology. The invariant part of the component's model shall be as small as possible belonging to every component technology stored at the repository. The denomination for this part is "Core Data", meaning that it contains basic information about the component that may help users to precisely identify possible candidates and that it is as small as possible. The technological part

must be variable, being adapted from technology to technology and containing different types of data. It must be able to accept not only text and graphics, but also modern ways of describing technical elements such as videos and graphical animations among others. This part must also be able to accept the inclusion of optional fields inside one similar component technology. The most usual terms adopted at each component technology must be respected, i.e. the terms used must receive the most accepted nomenclature by the experts of that technology. The name given for this part of the model is “Flexible Data”, with flexible here meaning that it varies among technologies and that it is composed of mandatory and optional parts.

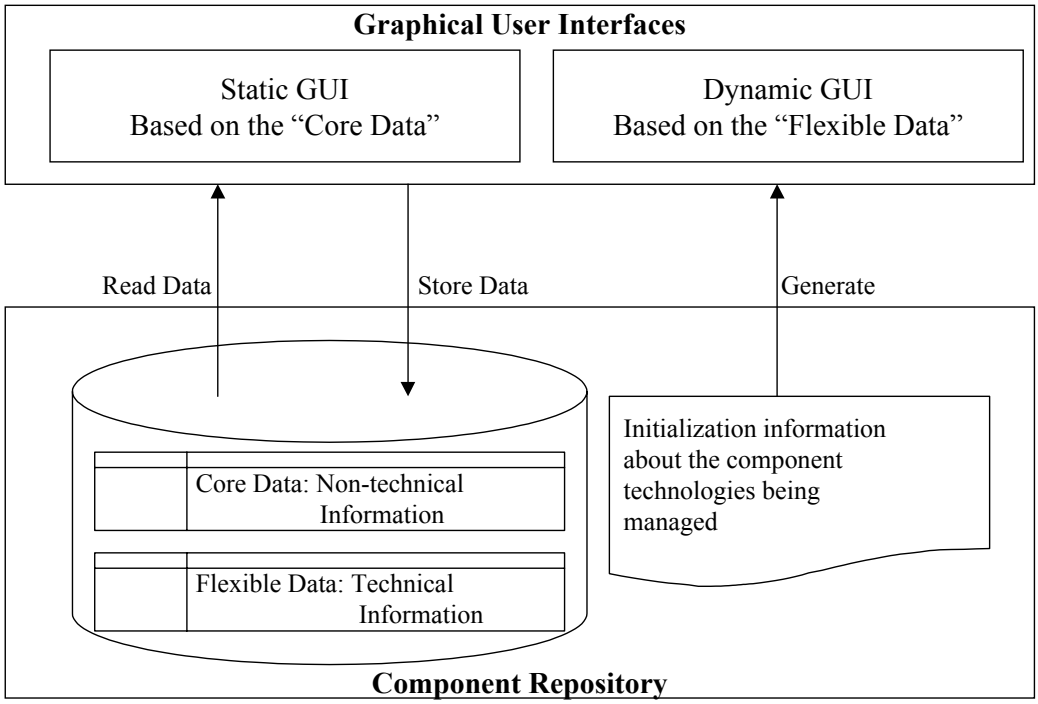


Figure 5.6: Architecture of a Component Management System able to deal with Diverse Component Technologies.

The ideal architecture for managing components is presented in Figure 5.6 and it shall work with those core and flexible data together for each technology. The graphical user interface (GUI) is composed of two parts, the static GUI and the dynamic GUI. The first one is constructed based on the core data part of the model and is supposed to always be the same. The second part is adapted dynamically to the technology being managed at the moment. These GUIs are responsible for collecting the necessary data for each technology and for presenting the already stored data in a meaningful way in accordance with the technology being currently managed. Similar conclusions are possible for the data structure of the component repository where part of the data is based on a static data structure derived from the core data, and the other part is generated dynamically in order to contain the flexible data. Core and flexible data must be stored on the repository independently because the core data may be mapped to a static data structure, while the flexible data may be represented in a variant form in the repository.

Additionally, as they will be used in different moments, one basically for the search and find of desired components, and the other for the understanding and final decision about using a component, core and flexible data may be stored independently only keeping the necessary linkage between them current.

The flexibility or variation desired at the graphical interfaces and at the repository data structure must be dynamically generated based on some initialization information previously available. This initialization part is expected to contain the specific information needed for each component technology managed by the system. In such an ideal system any modification done at the original component's model is automatically implemented at the GUIs and data structure only by modifying this initialization part. This part is also specific for each component's technology and it is necessary to have as many descriptors as technologies being currently managed. The insertion of new technologies is implemented only by the generation of a new initialization description instruction.

In summary, it was shown in this chapter that a component management system shall be dynamically adapted to the diverse component technologies with any use in industrial automation applications and shall also be able to receive new technologies in the future. One optimized development process based on components was presented. The component model was divided into two basic parts, one composed of fundamental information about components and developers here named core data, and a second one composed of variable contents that may be adapted to the component technology named flexible data. This structure is the basis for a component management system able to deal with the complexity of the components with any usage at the industrial automation domain. Such a system could be designed and implemented at many different electronic forms, nevertheless the main goal of producing components is to make them available and above all trying to increase their possibility of being reused.

A way of reaching many potential users and a very popular work platform nowadays is the Internet, which has proved to be an excellent platform for exchanging information and divulging products and services. Additionally, it is possible to implement very efficient ways of communication among users using this technology making it a natural choice as a basis technology for designing such a component management system. That introduces some specific requirements that will be presented in the next chapter. There, the main characteristics of Web-based applications are discussed. The most appropriate communication architecture for a component management system able to fulfil the requirements of the ideal management process suggested in this chapter is also presented. Later on, the technologies able to realize the desired system are studied and compared among themselves. Based on this comparison the most appropriate ones are selected generating the specification of the desired flexible Web-based component management system.

6 Web-Based Realization of the Component Management System

A Web-based application is basically an Internet service that links documents locally and remotely (Intranet vs. Internet). Documents are stored on the Internet in Web-servers that store and disseminate Web-pages. The Web-pages are accessed by users with software applications called Web-browsers, and those pages contain text, graphics, animations and videos as well as hypertext links. The links in the pages let users jump from page to page (hypertext), whether the pages are stored on the same server or on servers around the world. In the last half of the 1990s, the Web became the center of Internet activity mainly because the Web-browsers provided an easy, point and click interface to the largest collection of online information in the world. Ever since the Web became the focal point of the Internet, the amount of information available has increased at a staggering rate [Cail95].

In this chapter, the most relevant characteristics of Web-based systems are discussed. The goal of this discussion is to identify the necessary Web abilities that must be present at the desired component management system. Afterwards, the most appropriate architecture to realize such management of components able to deal with the knowledge about components for industrial automation described in chapter 4 is presented. And the desired management characteristics related in chapter 5 will also be considered. This architecture is composed of three independent layers, also called tiers, that will be implemented by Internet-enable technologies currently available. These technologies are briefly analyzed and compared, and the most appropriate are chosen to realize the complete system.

6.1 Web-based Systems

The final goal of a component management system is to satisfy the necessity of transmitting and receiving different information from all users involved. The two potential users of such systems have specific needs and requirements that must be attended to. On the one hand, the component management system must receive data, verify if this data is consistent with pre-defined structures, and decide about storing it or asking for further human intervention before storing the data. On the other hand, the system must help somebody to find the desired information, interpret it in accordance with previous knowledge, and help users to properly decide about using a component.

Most recently the usage of the Internet and associated technologies expanded the potential usage and the probability of success for industrial information systems [SeMi98]. These Web-based information systems are not more than information systems that work under the World Wide

Web [GaMy01]. Their functionalities are similar to the traditional ones but they may be reached through Web-based interfaces. Such systems make use of dedicated communication protocols, specially designed for the Web, and present the same advantages and problems of other Web-based systems [RFPG96, RRW00, ZSN01]. In order to access the desired information, users of a Web-based system only need to make use of a Web-Browser, which is able to communicate using those internet protocols.

Characterizing Web-based Systems

Web-based information systems have basically three different application usage:

- The first one is the in-house information system, whose goal is to enable the communication among business partners using local area networks. These applications are also called business-internal communication systems and are designed specifically for the internal public of a single company.
- The second usage type deals with electronic business systems among the external public and companies. These systems help users to proceed with complete business procedures and implement the denominated business-to-consumer process.
- Finally, the third usage form helps companies to implement electronic business among each other. They are the so called business-to-business communication systems.

The component management system developed in this thesis is able to support the necessities of the first two categories mentioned above. Moreover, it is possible to foresee the introduction of new modules able to attend specific business-to-business requirements.

A growing number of information systems based on the Internet and associated technologies is currently available on the market covering the three usages described above. But the most popular systems implement the role of a mediator in a sale-buy business process [LoSp98]. That is the case of business contacts obtained through electronic catalogues. The expectation is that such systems will divulge products in a significantly wider area than the conventional client-server applications could do. The Web-based management of components studied here has the same expectation willing to reach as many potential users as possible. A common characteristic of systems implementing sale-buy processes is that they do not work exclusively with text and pictures as other applications, but deal with data stored in some kind of database.

Among the most successful Web-applications having data storage are the electronic address books, on-line news, informative catalogues of products some of them with ordering and paying facilities, and on-line banking [Bent99]. Although a Web-based component management system does not fit 100% to only one of the listed applications it may inherit the main concepts of some of them. The specific requirements of the most relevant Web-applications and their characteristics were summarized in a criteria catalogue presented by Löser [Loes98]. The eleven criteria presented there are listed and briefly described in Table 6.1. An analysis using this

catalogue to characterize the requirements of a Web-based component management system is done in the sequence.

Table 6.1: Criteria for Evaluating Web-based Applications with Data Storage.

Criteria	**Description**
1- Type of Access	Most all of the Web-applications may permit only the reading of information from the database. That is the case of a catalogue of products. Nevertheless, if the catalogue should have ordering facilities, it may be necessary to write some information as for example the part number of the selected items associated with the buyer information.
2- Data Actualization	Some applications may actualize their data in a very fast pace, some others may have non-variable data. For the second case, it should be desirable to introduce buffers able to temporarily keep the results of frequent queries speeding up the access to those data.
3- Number of Simultaneous Accesses	Different systems will present different reaction times depending on the simultaneous number of accesses they are designed to support. It is meaningful to design the architecture and select the technology involved in accordance with this expectation avoiding an undesired bottleneck slowing down the global performance.
4- Accesses Superposition	This kind of superposition occurs on applications that present frequent repetitions of the same query resulting in the same data presentation. Some actual techniques may increase the speed and reduce the number of queries buffering the results properly.
5- Type of Data	If the desired interface of a Web-based system is purely based on text and pictures, it may be easy to be designed and implemented using popular technologies. If it is supposed to present sophisticated data, as for example geometric data, it should be necessary to use some specific implementation technique.
6- Data Sensibility	Sensibility deals here with the publicity of a set of data. In that sense the credit card number of a user is a very sensible data and should be protected. This protection is obtained with encryption techniques that may have different security levels.
7- Security Requirements	These requirements refer to the integrity of the database and to the protection developed against possible damage of this data. The main goal is to protect the data from possible misuse resulting in the destruction of the database.
8- User Authentication	Some applications may restrict the access to their databases to a limited number of users. Normally, this is implemented by the introduction of user categories recognized by user names and corresponding passwords.
9- User Identification	Differently from the previously described criteria, the identification of users may be implemented not to limit their access, but to customize them. Some applications have a kind of tracking system designed to identify the users profile and to offer them the most convenient services.
10- Session Duration Time	Some Web techniques create a context that will be valid for a pre-defined amount of time. That is very useful for example, for the case of the fulfiling of formularies composed of many steps. Someone designing a Web-application should estimate what kind of interactions will exist and how long a user's action should be active.
11- Response Time	There is a relationship between this criterion and the last one. For the case of applications where the user must navigate through many pages, case of a long section time, it is desired to obtain response times as small as possible for each page. The worst situation would be to have an application with many pages, each of them with a long response time.

Specific Needs of the Web-based Component Management System

As component users and component developers have quite different interests, it is desired to implement an authentication of users given them the exact rights they need to execute their job. The additional identification of users may permit the introduction of features customizing the application. The nature of the work of both users varies strongly, the duration of the usage sessions for component users and component developers tends to have different dimensions. The time necessary to fill up the system with the information contained in the classification and documentation of a component is expected to be longer than the time necessary for searching and understanding one single component candidate. Finally, it is desired to offer a small response time between pages for both users, and the pages that compose the unitary steps of the publication and selection processes must be present as fast as possible leading to a convenient man-machine interaction.

From the technical conception of components considered in this thesis, it is possible to affirm that a typical component will neither suffer actualization nor modification in short periods of time. It is also not expected to have one component that will be accessed frequently by potential users simultaneously. This reduces the necessity buffering data, limiting the buffer usage to general information pages and avoiding to buffer component's specific data. Additionally, it is not supposed to have many users searching for components at the same time, meaning that performance should not be a problem for such systems.

The interface with the users must support textual information as well as graphical ones. The system is planned to support modern ways of presenting information about components including the use of videos and other animation techniques that facilitate the comprehension of the technical aspects of a component. As the concept of component itself hides the implementation details of the functionalities, a small degree of sensibility of the data involved is assumed. The major amount of information consists of public information about components and developers. A pre-requirement for this thesis is that no kind of commercial transaction procedure will be implemented. Details containing any kind of financial secret information is not part of this component management system. Nevertheless, security considerations about the component's data itself must be taken whenever one developer is willing to save one new component on the repository. In these cases, the interfaces to the users must be able to interact with developers, asking them to properly feed the system with the desired information and to keep the necessary security level.

Table 6.2 summarizes the considerations presented above. In order to make a general comparison with other kind of Web-applications, the values for the applications Address Book, Ordering Catalogue, and On-line Banking presented in [Loes98] are repeated in this table. It is possible to verify that the Web-based component management system really does not fit to any

applications previously analyzed. The particular requirements and characteristics of this new system guided this thesis to a unique proposal to be described in the course of this chapter.

Table 6.2: Characterization of a Web-based Component Management System.

	Type of Access	Data Actualization	Simultaneous Access	Access Superposition	Type of Data	Data Sensibility	Security	Authentication	Identification	Session Duration	Response Time
Web-based Component Management	R/W	0-3	0-3	0-3	M	N/Y	1-3	Y	D	CU 5-7 CD 7-9	1-3
Address Book	W	0	0-1	0	A	N	1-3	N	N	0	0
Ordering Catalogue	R/W	3-9	6-9	0-3	A	N/Y	7-9	N/Y	Y	5-9	4-9
On-line Banking	R/W	3-9	6-9	0-3	A	Y	9	Y	Y	5-9	4-9
R – Read; W – Write; M – Multimedia; A – Alphanumeric; N- No; Y – Yes; D – Desirable; Scale: 0 – Very Small, 9 – Very High; CU – Component User; CD – Component Developer											

Ergonomic Aspects of the desired Web-based Component Management System

Another very important aspect from Web-based applications deals with ergonomic concepts applied to software systems. The norm ISO 9241 part 11 discussed this subject and established some directions on how to measure the usefulness of software applications [ISO98]. Three main criteria are considered in this thesis, the effectiveness, the efficiency, and the user satisfaction by the software system:

- Effectiveness characterizes how effectively a tool may be considered, or in other words, how precise and complete an application fulfils its proposed goals in comparison with the original requirements.
- The efficiency of an application will be evaluated better if users are easily able to work with this tool and, more important than that, if they are able to reach their objectives easily incrementing their productivity.
- The user satisfaction measures how good the user-application relationship is. It is important to note that even applications with good levels of effectiveness and efficiency can lead to a poor acceptance level if they are not able to adapt themselves to the users way of working.

Here again the interface to the users plays a decisive role, being one of the most important factors when talking about ergonomic aspects [Heil99]. Nowadays, the implementation of well

designed graphical user interfaces capable of being customized to different users is a preeminent necessity. Additionally, users expect to work with interfaces that fit to the WIMP principle (Window, Icons, Menus, and Pointer) leading to a better navigation platform and reducing the learning effort. The desired component management system must be effective and efficient enough to permit its usage as a support tool for the component-based development process. Additionally, it must be able to offer different interface designs for users in order to increase the global satisfaction.

Insertion of the Web-based Component Management Systems in the Development Process

Two other factors, as important as the user interface design, are the business context where the application is supposed to be used and the final objective of the people using it. Depending on these two points software applications are used in three different forms [Wöhe96]:

- The first one is to use them as support applications where the software tools are designed to help users to develop their work, but the work itself does not depend on the tools.
- In the second form, applications are considered as factory tools, for such cases no work can be done without the usage of these software applications. More than a support tool it is an essential part of the production process.
- Finally, in the third approach, software applications are used as strategic weapons where they are used as a fundamental stone of the global business process of the company. No work can be done without the support of those tools.

The Web-based component management system proposed in this thesis is classified better as a support tool for component users and at least as a factory tool for component developers. Someone searching for components may find what they want in third-party repositories. Additionally, if they work with an ill-designed development process, they may construct their own components whenever they need a new one. Nevertheless, considering the concepts of component-based software development from chapter 5, where any new application is supposed to be developed by the assembly of already existing components, it is possible to affirm that the Web-based component management system must be used during the whole development process. Such a system is able to improve the productivity and, if well designed, increase the satisfaction of users. From the view of component developers, a component management system must be used after every single component development. For those professionals, such a system works as a display window where their products are advertised, evaluated, and, more importantly, selected for reuse.

6.2 Architectural Conception of the Component Management System

The basic architectural concept for internet applications is the client/server architecture. This is an architecture in which the user's computer (the client) is the requesting machine and the server is the supplying machine, both of them are connected via a local area network or via a wide area network. In client/server applications, the client processes the user interface and can perform some or all of the application processing. Servers range in capacity from high-end personal computers to mainframes. A database server maintains the databases and processes requests from the client to extract data or to update the database. An application server provides additional business processing for the clients.

Because of the Internet, terms such as "Web-based" and "Web-enabled" have replaced the client/server nomenclature, yet the client/server architecture is conceptually the same [FiTa00]. Users' computers are still clients, and there are tens of thousands of Web-servers throughout the Internet delivering Web-pages. Nevertheless, the terms client/server is mostly used to refer to non-Web-based systems. On the Web, the clients run browsers and just like legacy client/server systems, they can perform little or a lot of processing, simple displaying of HTML pages, more processing with embedded scripts, or considerable processing with Java applets and associated technologies. A myriad of browser plug-ins provide all sorts of client processing. The server side of the Web is a multi-tier server architecture with linked Web-servers, application servers, database servers and caching servers.

As seen in section 5.2, three main tasks must be developed for a proper management of components. Firstly, it is necessary to develop a suitable interface to potential users. This means that two well designed and, as much as possible, customized interfaces must be developed because component users and component developers have very different goals. The second task involves the processing of the data someone wants to store or the data someone wants to read. In accordance with section 5.1, some processing must be implemented because some interaction will occur during the publication process as well as during the selection process. Lastly, the data itself must be manipulated and stored properly (section 5.2). That is not a trivial job as the data representing a component designed for industrial automation applications is somehow complex as studied in chapter 4.

Based on this division in three main tasks, it is possible to conclude that the most appropriate architecture for a Web-based component management system must contain three tiers as presented in Figure 6.1. The first tier is responsible for the data presentation, the second for the data processing, and the third for the data storage [Cruz96]. For security and privacy purposes, it is desired that almost all processing be implemented at the server side. That is also a way to guarantee a conformity for all users. Leaving the major processing tasks at the server side permits having users with less processing power which avoids non-desired running problems. A

description of the work developed in each tier followed by a study of the most suitable technologies to implement this desired work are presented in the next sections.

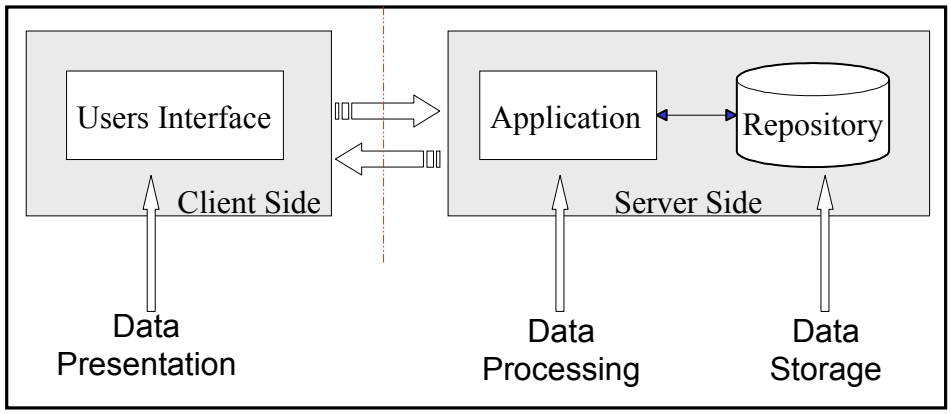


Figure 6.1: Three-tier Architecture of the Web-based Component Management System.

6.2.1 Interface to the Users – Data Presentation Tier

Component users and component developers have quite different needs, that may lead them to make use of different technologies to reach their goals. It is interesting to remember the two workflow scenarios to keep in mind how these two users are supposed to work. After finishing their development work, component developers have their components for industrial automation applications ready for publishing. The component publication process is composed of a classification and of a documentation procedure. Both procedures are divided into a series of interactions where the component management system will ask for information that developers should introduce. These procedures may take some time because the documentation can be a little complex depending on the component's technology. The interface must also guide component developers helping them to classify their components using the classification scheme for industrial automation presented in chapter 4. It involves the explanation of the facets that compose the scheme and some help to choose the values of these facets for each component classification.

A desirable feature to be included is the possibility of reusing standard information already stored for other components, as for example the company's data. It is also desirable to have the possibility of interrupting a classification or documentation section without losing the work already done. The introduction of a new component technology must be possible to do and the eventual introduction of new interactions with the component developers due to model modifications should also be supported.

Application developers wish to find components that may fit to the application they are working with. The interface of the component management system must help these users to find the desired components on the repository. But searching for an element is not a trivial task because

users may conduct a search based on wrong parameters or through wrong approaches [Henn94]. The interface to the component users must cover the most conventional ways of searching for elements in an electronic platform [LRS99, WaWa98, Blah01]. In this thesis, three searching approaches are considered: the searching based on free-text values that might be present somewhere on the documentation of the elements, the searching based on previously known classification characteristics, and lastly the browsing over the complete repository in order to get accustomed with the elements stored there. These approaches will be called free-text search, guided search, and browsing respectively.

After finding some potential candidates, component users must be supported to evaluate if any of these candidates are appropriate for the purpose they are looking for. This involves a detailed presentation of the component's information helping users to really understand what the component can do and what it can not do. This presentation must be well organized including separated information about the functional characteristics of the component as well as about operation characteristics and non-functional information. Additionally, it is desirable that the interface permits users to start an electronic contact with component developers or someone else responsible for this component in order to elucidate eventual questions about the component itself.

From the technical evaluation it may result that more than one component implements exactly what users are searching for. In such cases, the component management system must offer the users a way of comparing candidates, helping users to decide properly about the real potential of candidates [Aunv01, Paiv02]. The interface is expected to give support to pre-select the most appropriate components and to choose some desired comparison criteria. The result of such procedure may be a table where the pre-selected components and the desired criteria are presented simultaneously facilitating the user's decision. After deciding on the use of one component, the users must receive help to acquire this component.

It is also possible that none of the already available components fulfil the desired requirements of the user. This is detected during the searching procedure whenever receiving an empty list as a result of the search, or afterwards during the technical evaluation when the probable candidates prove not to fulfil the desired characteristics, or lastly, during the decision making when the resulting components may be considered non-viable due to their cost, for example. For each of these cases the system proposed here must give support for component users to generate an order publishing the requirements of the new component they need. The interface must be able to guide these users during the preparation of their request for new components.

These desired features for the interface between the component management system and the component users and the component developers may be implemented using a variety of Web-based technologies. All these technologies have positive points and disadvantages that must be pondered before deciding about the use of one of them. In order to fulfil all the above mentioned

requirements of the desired component management system, three technologies used for the generation of Web-based interfaces were investigated. In the next paragraphs the results of this research are presented for the interfaces designed using pure HTML, using dedicated client-side applications, and using Java applets.

Pure HTML Interfaces

The Hypertext Markup Language is a standard language for describing documents that uses a Document Type Definition (DTD) for specifying the structure of the document itself [Grah96]. Regarding the design of user interfaces, HTML is able to send information from the client-side to the server-side using the HTML-formulary. With formularies it is possible to implement some information exchange between users and systems permitting the design of communication features. A HTML-formulary is divided into fields identified by names and contains values to represent their contents. Different communication elements are available at the standard HTML as for example checkboxes, radio buttons, and pull-down lists that are able to generate a convenient way of collecting the desired information.

An interface based only on the HTML language is very easy to install because it only needs a conventional browser to work properly. As it is a standard accepted overall, interfaces designed with HTML are able to work with the most popular browsers and do not present any communication problems. Modifications are easily implemented using a simple text-editor. During the tests done for this thesis, some incompatible problems occurred whenever wanting to introduce graphical elements to the interface. Features like drag-and-drop, point-and-click, and other popular interface elements can not be designed using only the HTML language. Moreover, it is not possible to introduce any pre-processing over the input data which limits the possibility of programming interactions before sending the data to be stored.

Client-side Interface Applications

The design of graphical applications to implement Web interfaces may be also done when using Java as a programming language [RFPG96, AyBe99]. Nevertheless, Java programmes need a run-time interpreter at the client-side otherwise they do not run, but this problem is easily solved through the installation of the Java Runtime Environment (JRE) that must be running at the client-side before starting the application. This installation is not a problem nowadays due to the popularity of the Java platform and because the JRE may be downloaded without any additional cost. This interesting fact makes Java programmes platform independent, which guarantees that Java applications will run overall. Moreover, the Swing library of components, which was introduced by the Java Development Kit (JDK) version 1.2, greatly increased the number of resources available to design graphical interfaces. This library is an extension of the Abstract Window Toolkit (AWT) from JDK 1.1 and offers common elements of modern GUIs permitting

an easy design of interfaces with the look and feel of Windows compatible systems [WiKo99, TaWo00].

Additional problems occurred for the cases where the applications were developed based on a different version of the JRE from the one installed at the platform where they are supposed to run. That is a common situation because developers usually work with the most actual development tools while normal users may take more time to actualize their systems. In order to minimize this problem, the tool Java Web Start (JWS) [JWS01a, JWS01b] from the Sun Microsystems company was investigated and proved to be a successful approach because after installing the JWS at the client system and associating it to the desired application it automatically takes care of all necessary updates (inclusive support programmes) available for the application in question. This is done through the generation of an XML document describing the actual configuration of the application itself. Whenever a new version of the application is available on the Web, JWS detects it by comparing the local description file with the one available on the network, an actualization is commanded and automatically installed. This concept guarantees an easy installation and maintenance of client-side Java applications.

The combination JDK, Swing components, and JWS proved to be a very powerful platform for developing well designed users interface. The installation of such an application demands some time and effort only at the very first moment. From this point on the application itself is able to command updates automatically. Such interfaces are easily modified or extended to cover new requirements, the only skill necessary for those modifications is fluency in the Java programming language.

Interfaces Based on Java Applets

Similar to the Java applications, applets are computer programmes written in the Java language that may use the classes and tools available at the JDK platform (including the Swing components). On the other hand, applets do not reside originally on the client-side but on the Web-side. They are downloaded whenever the user needs and run only under browsers able to deal with Java applets [Schu99]. An advantage of this technique is that the applet is always in the most actual version, i.e. users do not need to worry about actualizing their applications.

An applet developed to implement a well designed graphical interface is large, what causes some problems, for example is its great downloading time. Additionally, the usage of the Swing components is absolutely necessary to develop a nice interface. As a consequence, none of the Swing components used by the application shall be missing at the client-side, otherwise it would not work properly. For these cases, it is necessary to order the additional installation of plug-ins leading to even greater download periods. Applet developers must pursue a good level of expertise on the JDK platform. Later modifications on the interface are possible, but it is important to remember that these modifications will be restricted to the user’s browser capability of dealing with Java applets.

6.2.2 Manipulating Component's Information – Data Processing Tier

In the Web-based component management system proposed here, the data processing tier is responsible for the preparation of the data introduced by component developers to be stored on the repository from one side, and accesses the desired data, preparing it for its presentation whenever asked by component users from the other side. Some processing power is necessary to complete these two tasks successfully. During the component publication, component developers are asked to give some information in order to classify and document their components properly. This information must be verified, firstly to prove if it is in accordance with the classification scheme for industrial automation systems proposed in chapter 4, and secondly to verify if all important data about the component technology was given accordingly. Whenever the classification or the documentation fails, the system must ask for complementary information and must insist on it until the inserted data is consistent with the pre-defined model. This tier must also be able to deal with different models simultaneously, and the insertion of new models should be as easy as possible.

Component users should not worry about how they can reach the information stored on the repository. These data must be worked out for the searching of components, offering to users meaningful ways of searching and browsing the repository in a transparent approach. Information about one specific component is expected to be prepared during the technical evaluation facilitating the user's comprehension of the most relevant information in a very convenient way. Lastly, the decision process involves processing of specific information about some components that may be retrieved and processed after being sent to the interface to the users at a real-time pace. The contents of the interfaces are generated dynamically in accordance with the current information demand and is composed of data stored at the repository. The most relevant technologies investigated that are able to implement such communication and processing tasks via Web are briefly presented in the next paragraphs. They are the Servlets, the Java Server Pages, and the Remote Method Invocation.

Processing Data with Servlets

Servlets are platform-independent, 100% pure Java server-side modules that fit seamlessly into a Web-server framework and that can be used to extend the capabilities of a Web-server with little overhead, maintenance, and support. These programmes are accessed through pre-defined set of interfaces distributed by Sun Microsystems called Servlet-API, where all possible interfaces are declared and some of them are also implemented. The connection of a Web-server to a servlet is done using the so called Servlet Engine, which is a container where the servlets really run. During the installation procedure, a set of configuration data must be adjusted at the Web-server and at the Servlet Engine in order to enable a proper communication between both of them. After this adjustment, each HTTP request is checked out to see if it includes access to the servlets. And if it does so the request is redirected to the Servlet engine in order to be processed.

The communication between servlets and users interfaces can be implemented by a simple HTML-formulary using the GET and POST commands. The desired servlet is accessed through the methods *doPost* and *doGet*. The real functionality of these methods must be programmed in accordance with the desired processing task. The response of the servlet is then sent back to the client who must understand the data and present it to the user. But the interface can also be implemented through an applet or a real Java application. For these cases better communication is obtained, starting a direct connection between client and server and sending the data in binary format (or ASCII). Using this approach, the proper presentation of the data to the users relies on the interface designers, that must interpret the received data and present it meaningfully.

Dynamical Data Manipulation with Java Server Pages

One problem detected during the construction of interface logic based on servlets is that each single modification on the interface design requires a complete regeneration of this interface. Additionally, the programming of HTML-pages inside the source code of a servlet requires a lot of work. The Java Server Pages (JSP) technology separates the user interface design from content generation enabling designers to change the overall page layout without altering the underlying dynamic content [Tura00, Hall01]. The processing of data using this technology is divided into two parts. Firstly it is necessary to generate HTML documents representing the interface to the user, and afterwards the necessary logic is introduced through special tags that are responsible for calling the desired programmes.

This technology uses XML-like tags and scriptlets written in the Java programming language to encapsulate the logic that generates the content of the page. Additionally, the application logic resides in server-based resources that the page accesses with those tags and scriptlets. By separating the page logic from its design and display, JSP proved to be a suitable technology to build Web-based applications. In fact, JSP is an extension of the Java-Servlet technology. Similar to the servlets, JSP pages need a run-time environment here called JSP Engine. Most all new versions of Servlet Engines come nowadays with a JSP Engine integrated, and JSP pages must reside on the Web-server containing this engine. Together, JSP technology and servlets provided an attractive alternative to other types of dynamic Web scripting or programming that offers platform independence, enhanced performance, and separation of logic from display.

Data Processing Using the Remote Method Invocation

The Remote Method Invocation (RMI) technique is the Java approach for the Remote Procedure Call concept [WiKo99, RMI00]. The main goal here is to give developers the possibility of using objects remotely located as they were on the client-side computer. RMI allows parameters, return values and exceptions passed in RMI calls to be any object that is serializable. It uses the object serialization mechanism to transmit data from one virtual machine to another and also records the call stream with the appropriate location information so that the class definition files

can be loaded at the receiver. Parameters and return values for a remote method invocation become live objects in the receiving Java Virtual Machine.

Some additional work must be done in order to implement an application able to use the RMI architecture. First of all, it is necessary to design the classes interfaces, what is done in the same form of a normal Java class plus the declaration of a pre-defined method called *Remote*. The services offered by this class must then be implemented respecting the previous declaration of classes and methods. These classes must finally be registered at the client and at the server sides following specific procedures. Of course, the collection of the necessary data and the desired processing over this data must be programmed as client and server-side applications respectively. The RMI solution for the communication between user interfaces and processing tier is more suitably applicable for the cases where no HTML are used as client side front end. Additionally, specific knowledge about the RMI architecture is necessary making the programming of such applications a little bit more difficult than the other two technologies investigated to implement this tier of the component management system.

6.2.3 Storing the Component's Information – Data Storage Tier

The right ways of formatting and of accessing data are the most important questions to be answered when talking about data storage. Decisions about these two points must be based on the desired information structure and on how potential users are supposed to manipulate the stored data. Additionally, the communication through networks, and in particular via the Web, poses some other requirements to this thesis that must be obeyed. Final goals are to keep the data consistent and to avoid the lost of information previously stored at the repository. Many tools specially designed to deal with these questions have been available on the market for a long time and were examined for the development of this thesis.

Database Management Systems (DBMS) are software systems developed to control the organization, the storage, the retrieval, the security, and the integrity of data in a database system [HeSa00]. They accept requests from the application and instruct the operating system to transfer the appropriate data. DBMSs work with traditional programming languages, but some of them have their own programming language. They let information systems be changed as the organization's requirements change and new categories of data can be added to the database without disrupting to the existing system. Additionally, DBMSs help users to formulate queries over the data stored on them and from time to time stabilized query techniques are improved and adjusted to new technologies available on the market [SeMi99, SWM01]. Mitschang [Mits01].classified the possible usage of different database systems in accordance with their ability to deal with the data type complexity and with the available query complexity as reproduced in Figure 6.2

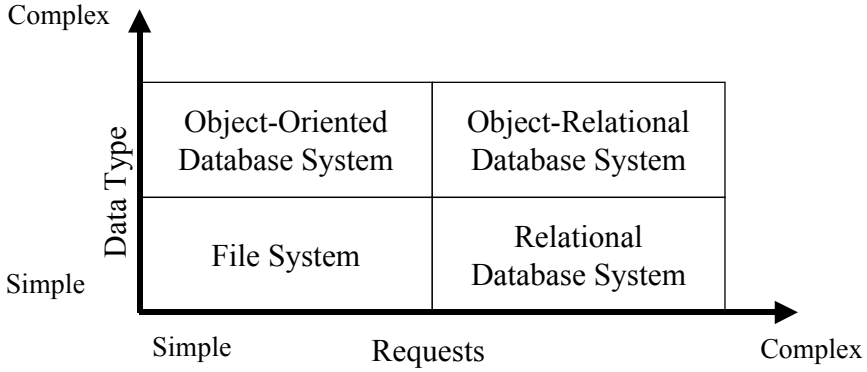


Figure 6.2: Classification of Database Systems Applicability.

Among the major attractive features of DBMSs for the component management system are their ability to deal with data security. They can prevent unauthorized users from viewing or updating the database. Using passwords, users are allowed to access the entire database or a subset of it. The DBMSs can also ensure that no more than one user can update the same record at the same time, which is a way of assuring the data integrity. With DBMSs, the details of the data structure are not stated in each application programme. Without a DBMS, the programmer must reserve space for the full structure of the record in the programme and any change in data structure requires changing all application programmes.

In the next paragraphs some important characteristics of the relational DBMS, the object-oriented DBMS, and about the eXtensible Markup Language (XML) are presented. The goal is to verify which one is more suitable to work with the data structure of a component for industrial automation. Such data base systems must be able to deal with a well structured data originated from the core data, and simultaneously able to deal with an unstructured and variable data originated from the flexible data.

Characterizing Relational Database Systems

The main criterion normally used to classify database management systems is the data model on which the DBMS is based [ElNa94]. The data model used most often in current commercial DBMSs is the relational one, which represents a database as a collection of tables, where each table can be stored as a separate file. The first task to design such a relational database is to obtain a relation model able to properly represent the elements to be inserted on the database. The translation from such a relational model to a set of tables representing it is then done in a straightforward manner. Relationships among objects are represented by associations among entities on the relational model using three available types of associations, the 1-to-1, the 1-to-n, or the n-to-m association. The numbers and the variables represent the possible cardinality among the entities.

DBMS systems are able to deal with the Structured English Query Language (SQL) and the desired queries on a database are designed using this language. Routine queries often involve more than one data file. A relational DBMS is designed to handle numeric data types like integers and floating point numbers, alphanumeric characters, fixed length strings, date, time, and money data type. Enumeration composed of a set of possible values explicitly listed are also available. Additionally, relational DBMSs support a Binary Large OBject (BLOB) field, which holds any binary data (image, video, etc.), but the database programme does not manipulate the BLOB directly. Another application has to be written or some middleware has to be used to process the BLOB file.

The communication between a data storage tier implemented by a relational DBMS and the data processing tier is easily realized using the interfaces Open DataBase Connectivity (ODBC) or Java DataBase Connectivity (JDBC) [Dehn98]. ODBC is a database programming interface from Microsoft that provides a common language for Windows applications to access databases on a network. It is made up of the function calls and the ODBC drivers themselves. JDBC is the Java counterpart of Microsoft's ODBC, and is a programming interface that lets Java applications access a database via the SQL language.

The core data of the component's representation of section 5.2 is well suited to be represented by a relational model and consequently to be managed by a relational DBMS. Positive points in adopting this approach are the easy implementation of queries over the data which are based on standard SQL, and the direct incorporation of the features presented in common commercial relational DBMSs, for example, the data security and integrity will be under the responsibility of the chosen database management system. On the other hand, the implementation of the flexible data part of components into a relational DBMS constitutes a problem as the variant characteristic of this part does not fit completely to a relational model. Additionally, although relational DBMS are able to work with BLOB (large object) fields that hold anything, extensive use of these fields can overload the processing on the data base. Storing such kind of data as part of a file system proved to be a more suitable alternative [Aunv01].

Using Object-Oriented Database Management Systems

One goal of OO databases is to maintain a direct correspondence between real-world and database objects so that objects do not lose their integrity and identity and can easily be identified and operated upon [ElNa94]. OO databases provide a unique system-generated object identifiers (OID) for each object, which is similar to the primary key attribute of the relational model. A real-world object possesses different keys in different relations, making it difficult to ascertain that the keys do indeed stand for the same object. Another interesting feature of OO databases is that they are able to deal with object structures of arbitrary complexity, containing all significant information that describes those objects. In contrast, in relational database systems, information about a complex object is often scattered over many relations or records,

leading to loss of direct correspondence between a real-world object and its database representation.

Object databases handle one-to-many relationships combined with many-to-one relationships. Using the object model, an object-oriented DBMS can store anything or refer to anything. Some OODBMSs allow access via an SQL-like language, the so called Object Query Language (OQL). Increasingly, OODBMSs are being merged with relational databases, providing a single environment for traditional business transactions, multimedia data and complex structures. Most recently the use of parallel data object-relational DBMS have been tried to improve the performance of database systems even more [JaMi98]. In an object database, a picture or video clip object can include the routine to display it, which is dynamically invoked by the DBMS. Communications features with other systems can also be implemented using appropriate ODBC or JDBC interfaces.

As presented in Figure 6.2 object-oriented database systems and object-relational database systems are supposed to be used better for applications dealing with complex data types. The first is more suitable for working with simple requests while the second is able to deal with complex requests. The representation of components for industrial automation is composed of a part that can be certainly classified as having a simple data structure, the core data, and it is over this part that the search requests will be processed. These requests may reach a certain level of complexity. The other part of the representation, the flexible data, has a complex structure, having for example multimedia elements, but no request shall be implemented over it.

Storing Data Modeled by the Extensible Markup Language

The eXtensible Markup Language (XML) is an open standard for describing data from the W3C – World Wide Web Consortium [W3C, W3C01]. It is used for defining data elements on a Web page and business-to-business documents. It uses a similar tag structure like HTML, however whereas HTML defines how elements are displayed, XML defines what those elements contain [BeMi00]. Unlike HTML, which uses a rather loose coding style and which is tolerant of coding errors, XML files have to be well formed, which means they must comply with rigid rules. XML has been quoted as one of the most promising technologies for sharing data over the Web [CaCa01, SeRo01].

One important feature of XML that proved to be well adapted for this thesis is that the validation of an XML file is done comparing its content to a pre-defined file structure. Two techniques have been used to create this structure definition, the DTD and the XML Schema. DTD stands for Document Type Definition and is the standardized schema of the Standard Generalized Markup Language. DTDs act as rulebooks that allow authors to create new documents of the same type and with the same characteristics [LaCe99, GoPr01]. They are used to define contents as well as determine the elements allowed in one file and which elements can be contained in other elements. The XML Schema is a superset of DTD also used to define the contents

characteristics of an XML file. Unlike DTD, XML Schemas are written in XML syntax and are created with any XML tool. XML Schema is flexible and simple to use because it contains user-defined data types what makes the global comprehension of the Schema easier than others based on standard tags. Using Schemas designers are able to specify their own meaning, usage, and function of elements of an XML document.

As a matter of fact, an XML file is not more than a set of characters divided into tags and contents that must be processed to gain some valued significance. This processing is normally done by XML parsers. A parser is a computer programme able to rip apart the textual representation of a document and turn it into a set of conceptual objects to be used later. Parsers are designed to identify XML tags and to work properly with the contents under these tags. There are many XML parsers on the market for use with many different programming languages, some of them suitable to be used with the Java programming language [JXML01]. These parsers will be compatible among themselves if they are able to work with one same standard parser API. The most usable parser APIs nowadays are the DOM – Document Object Model, the SAX – Simple API for XML, and the JAXP – Java API for XML [JDOM00, McLa00].

Another very important aspect of the XML technology appropriate to the component management system under consideration, is its ability to carry instructions about the desired presentation style of a document in an independent form. Through the use of stylesheets, the visual style of a document is reported without any modification on the document itself. The eXtensible Stylesheet Language (XSL) is a specification of the W3C for applying formatting to XML documents in a standard way. Over time, a part of XSL called the XSL Transformation (XSLT) has evolved into an independently useful language for transforming one XML document to another.

More recently, some proposals appeared talking about the XML databases [BeFe01, Bour01a, Bour01b]. That is a database able to manipulate and to store XML documents. There are two possible approaches for these databases. The first is the XML-enabled database, which is a relational or object-oriented database that has been extended to hold XML data. In this method, there is always a conversion to and from the XML document to the underlying structure such as rows and columns in the relational model. In addition, the XML-enabled database may only store part of the XML document. The second is the native XML database, which indexes XML documents directly and stores the entire XML document and related elements.

From the research developed, it was possible to conclude that the XML technology offers tools able to deal with any data type, and these tools are flexible enough to accept user specified data structure. Changes over previously specified data structures are easily modified and simultaneously the work with diverse structures is also possible and easy to be implemented. These characteristics are very well suited to process the flexible data part of the component's

model of section 5.2. The possibility of inserting meaning for each element of a file from one side, and additionally communicating presentation formatting specifications for those elements from another side, is a powerful characteristic of XML with potential usage to manage the flexible data part of the component's model, and to develop adaptable interfaces to the users respectively.

6.3 Choices among the Presented Technologies

The decision about the most appropriate technologies for realizing each tier took into consideration the requirements of the component management system and is presented in this section. More than one technology will be adopted at each tier, because the requirements of the system vary in accordance with the type of user, with the amount of work to be done over the data, and with the data itself.

Technologies Selected for the Data Presentation Tier

The judgement criteria for the data presentation tier involved four points:

- The first one is the necessary installation work done by the client whenever wanting to use the component management systems. Important here is to compare how much effort is necessary to properly install and work with the interfaces using the presented technologies.
- The second criterion is the interface usability, i.e. how easy it is to use and how convenient the interfaces constructed with each technology are.
- Then the extensibility offered by the technologies is judged, verifying if it is possible to introduce new functionalities on already implemented interfaces.
- Lastly, the alteration capability of the interfaces is compared or in other words, how easy it is to modify already designed and implemented interfaces issues.

The alteration capability is a decisive factor, because the system is supposed to work with different component models, which may lead to different interfaces to the users. This tier must also consider the special necessities of component users and component developers. Developers want to publish their components, storing their information on the component repository. These users are expected to have a professional relation with the component management system using this system as a strategic weapon for their business. In spite of that, the interaction between component developers and component management system might not abdicate features that facilitate the communication between the system and the users. For example, component developers must be able to stop their actual work at any point of the documentation procedure without losing the classification already done. Additionally, the possibility of working off-line in a local machine must be considered in order to avoid long connection periods and to reduce costs. The above mentioned scenario is realized better by a Java application programme [Serh01, Paiv02]. Whenever this application is associated to the Java Web Start (JWS)

technology, it is guaranteed that the application will be flexible and will always be at the most actual version. This feature is obtained because JWS keeps a configuration file adapting the application to what is described in this file. Any desired modification on the interface application is implemented directly only by modifying the initialization file once for all users. This technique enables a synchronization between server and clients applications not presented in any other technology evaluated during this thesis. On the other hand, component users are expected to proceed with simple tasks and may have less compromise with the component management system. They may interrupt their working sections in any stage of their job without saving the actual situation. Avoiding any installation procedure at the component user side is also desired, which would reduce the acceptance of the component management system. Component users may utilize the component management system selecting components without great effort which means they need a powerful interface. Such interfaces are obtained using HTML associated to any scripting language. For conformity and portability purposes Java Script is the most appropriate choice here.

Technologies Selected for the Data Processing Tier

As a matter of fact, a comparison between servlets and JSPs is redundant as JSP was developed based upon the Java servlets technology. Both present the same potentials and problems. While servlets are used to work better with Java applications, JSPs are well suited to be embedded in HTML code. Three comparison points were judged for this tier:

- The first comparison criterion is the internet capability of each technology, i.e. how easy is to apply each technology in the construction of Inter- and/or Intranet applications.
- In the sequence the system's interfaces characteristics that are obtained using those technologies is compared.
- The last criterion is the amount of work necessary to implement any processing task using each technology.

The data processing tier is implemented better with the Java programming language and associated technologies (servlets or JSPs). All interactions with users need data that will be processed at this tier reducing at most the necessary computational power from the users side. The communication with users must be constructed in such a way that no special features from the users side is needed. This tier must also communicate with the data storage tier through technologies related to the way of storing the data itself. In this thesis the communication occurs through JDBC technology and through XML parsers [Aunv01, Horn01, Paul01]. Although RMI configures a relatively easy and very powerful alternative to implement distributed applications, it is not the best choice for the component management system under consideration. One reason is that the complete component management system will be constructed on a client server architecture composed of only one central server. The system does not need to use the powerful but complex mechanisms offered by the RMI technology to deal with distributed applications.

Additionally, RMI applications need to implement connections to computer ports that may be blocked by firewalls in some systems. This configures a problem leading to malfunction of the application. Such problems are avoided using servlets or Java Server Pages.

Technologies Selected for the Data Storage Tier

Four aspects were used as criteria for judging the technologies investigated for implementing the data storage tier of the flexible Web-based component management system:

- The available storage mechanisms where the ways of storing data offered by each technology were tested and evaluated considering the characteristics of the components that will be manipulated by this tier.
- The flexibility grade offered by each technology where it is mainly investigated how easy the introduction of new data structure elements is or the modification of existing ones on already developed data structures.
- The queries mechanisms comparing the search alternatives available for each technology.
- The last item deals with the presentation and distribution of the search results, here it is investigated which data formats are used and how someone can work it out for the above mentioned technologies used in the data storage tier.

In accordance with the characteristics of the component models, where a very important part may vary strongly depending on the technology of the component being currently handled, it is possible to affirm that flexibility is a decisive factor for choosing the technology to be used for the realization of this tier of the component management system. Moreover, the presentation of the flexible data, containing the technical information of a component, shall be more accepted by users if the system is able to adapt the contents to a desired presentation style in accordance with the component technology. This separation of content from style is easily implemented using XML and associated technologies [Aunv01, Paiv02]. Nevertheless, the component model also contains a core data that must be always the same and that is used as a searching source. The query mechanisms offered by RDBMSs and OODBMSs are much easier to use than any similar implementations applied to XML data. Additionally, the storage and consistence of data is managed better by mature DBMSs. Finally, the great advantage of the object-oriented approach over the relational one is its ability to deal with flexible structures. But the flexibility obtained using XML is much more promising and easy to deal with. Whenever ignoring this characteristic of OO-approaches, the choice of the DBMS technology relies on a relational system, which is the easier and less expensive alternative to work with the core data. In most cases, the maintenance of a heterogeneous data format is undesirable. Nevertheless, the characteristics of the component model for industrial automation, composed of a core data and a flexible data part, is managed better by such a heterogeneous approach. Core data is well suited to be represented by a relational model and consequently better managed by a RDBMS. Flexible data is represented better as an XML document fulfiling all requirements towards the desired

flexibility and modifiability for this part of the model. A link between the core data and the flexible data must be introduced and maintained properly.

Resulting System based on the Evaluation of the Technologies

The resulting architecture for the flexible Web-based management of components of industrial applications is presented in Figure 6.3. It is not represented in the figure, but it was already mentioned, that the Data Processing and the Data Storage tiers are implemented better on the same computer at the server-side. The desired flexibility is obtained using the set of component models and every component technologies that will be managed by the system will have a representation stored.

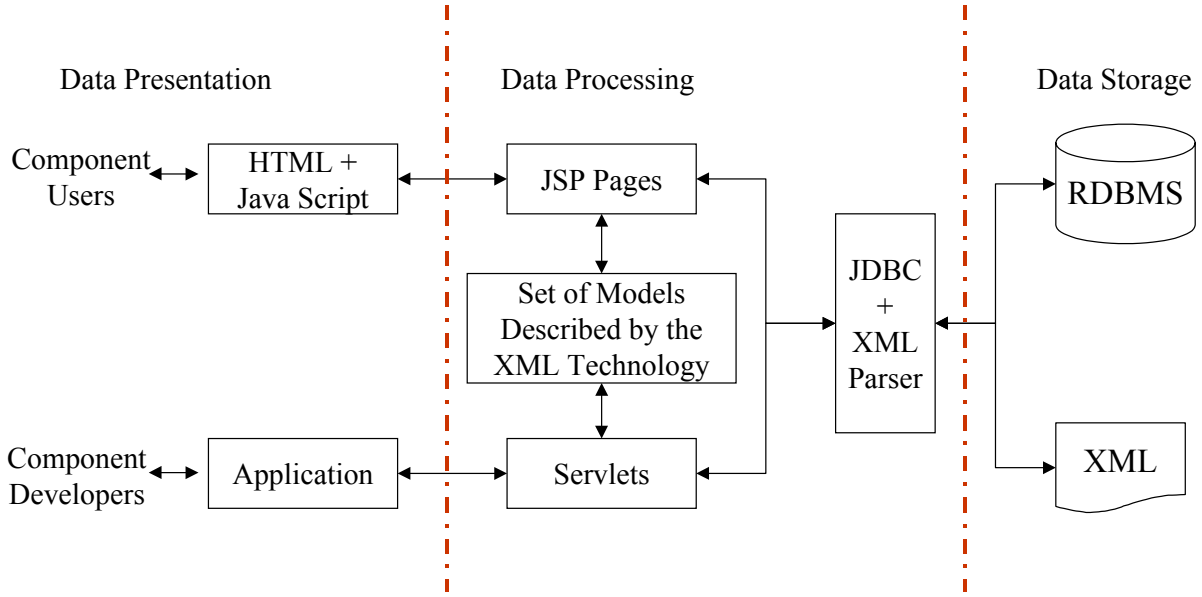


Figure 6.3: Selected Technologies for Each Tier of the Component Management System.

The Web-based component management system proposed in this chapter was originated from the analyses of the most relevant technologies currently available for constructing those kind of applications. First of all, the technological aspects of Web-based systems were presented. After analyzing the characteristics of desired application, it was possible to conclude that a three tier architecture was more suited for the system to be realized. As many different technologies available nowadays are able to implement the necessary tasks for each tier, comparisons were made permitting the formulation of proper decisions about each one. Afterwards, the most appropriate technologies to fulfil the desired requirements of the system were selected, which resulted in the system realization proposition presented in the end of this chapter.

But the characteristics of the component management system proposed in this thesis must be verified. In order to proceed with this verification, prototypes were developed using the above selected technologies for each tier. The complete description of each prototype realization and the exemplary usage of the complete system is the subject of next chapter.

7 Description of the Flexible Web-based Component Management System Prototype

A prototype of the flexible Web-based component management system proposed in this thesis was implemented using the technologies suggested in section 6.4. (see Figure 6.3). The complete system is divided into three smaller prototypes each of them covering tasks that could be meaningfully grouped together. One first prototype is responsible for implementing the insertion of new components at the repository. It covers the interface to component developers, the necessary data processing before the storage, and the storage itself. The second prototype implements the necessary tasks for the search and find of components, and it is designed to deal with users wanting to search at the repository for a component with particular characteristics and in different ways of searching. The last prototype covers the presentation of the information about previously selected components. Its responsibility starts after the finding of possible candidates, presenting the technical information about these candidates dynamically and in accordance with their particular technology. A common part of the global system is the repository where the data about components is stored and from where users receive the desired information. This part is composed of a relational data base system containing the core data, and a file system containing the flexible data of the component's model written as an XML document (see chapter 5). Additional information about the models themselves and about the proper way of presenting the component's information to users are also stored in this common repository. Implementation details of the data storage part as well as the description of the three above mentioned prototypes are presented in this chapter, which is closed with a usage example based on the synchronous component technology.

7.1 Implementation of the Data Storage Tier

As stated in chapters 5 and 6, the core-data part of the component's representation model contains information shared by all component's technologies being managed by this system. In this prototype, the core-data was reduced to a minimum amount of data chosen to help users to search and find desired components. Basically, this part contains data about developers and companies responsible for the components, the facets of the classification scheme presented in chapter 4, and a representation for the possible dependencies among components. Figure 7.1 presents the table structure of the relational model representing the core data.

The facets F1 and F2 of the classification scheme for industrial automation components (IAC_d), Application Domain and Specialization of the Domain respectively, are modeled through two tables where domains and sub-domains names are stored. This is supported by the specification of the actual level of the domain associated to the identification of the immediately upper domain level.

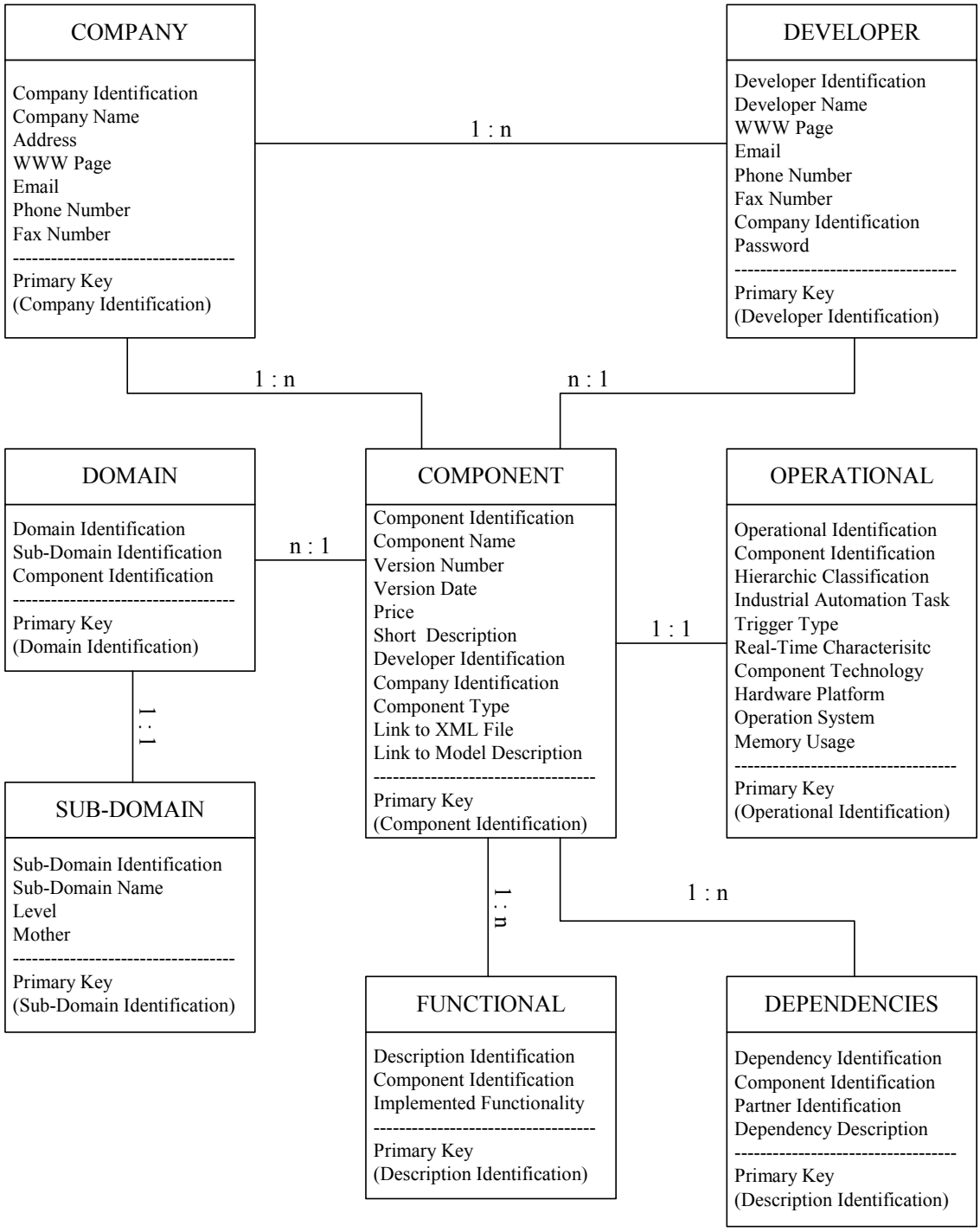


Figure 7.1: Table Structures and Relations of the Core-Data Part of the Component's Model.

As a component may implement many different functions, it may have many different textual descriptions explaining those functions. This fact is represented by the table called FUNCTIONAL. The other facets of the classification scheme were represented in the table OPERATIONAL and the facet Component Type is contained in the table COMPONENT. The connection between core-data and flexible data, where the technical information about each component is stored, is implemented through the information about where the XML file of the component itself and its model description is stored. These two fields are also represented in the

table called COMPONENT. The relational data base system was realized by the Interbase 6.0 (IB) data base management system. The communication between this RDBMS and the elements of the data processing tier was implemented through the respective JDBC drivers.

All other information about components was modeled and stored in the flexible data part. A model description for each technology is represented by XML Schemas and the corresponding presentation style by XML stylesheets. An example of the desired contents of the flexible data for the synchronous component model is presented later in this chapter. In the next sections, each prototype implementation is described in detail.

7.2 Flexible Storage of Components

The main architectural elements of the prototype system responsible for the storage of components are presented in Figure 7.2. It contains the global data flow, the division of the data storage in a relational database system and in a file system, all servlets installed at the server side, and the classes and elements of the Application Component Developer at the client side.

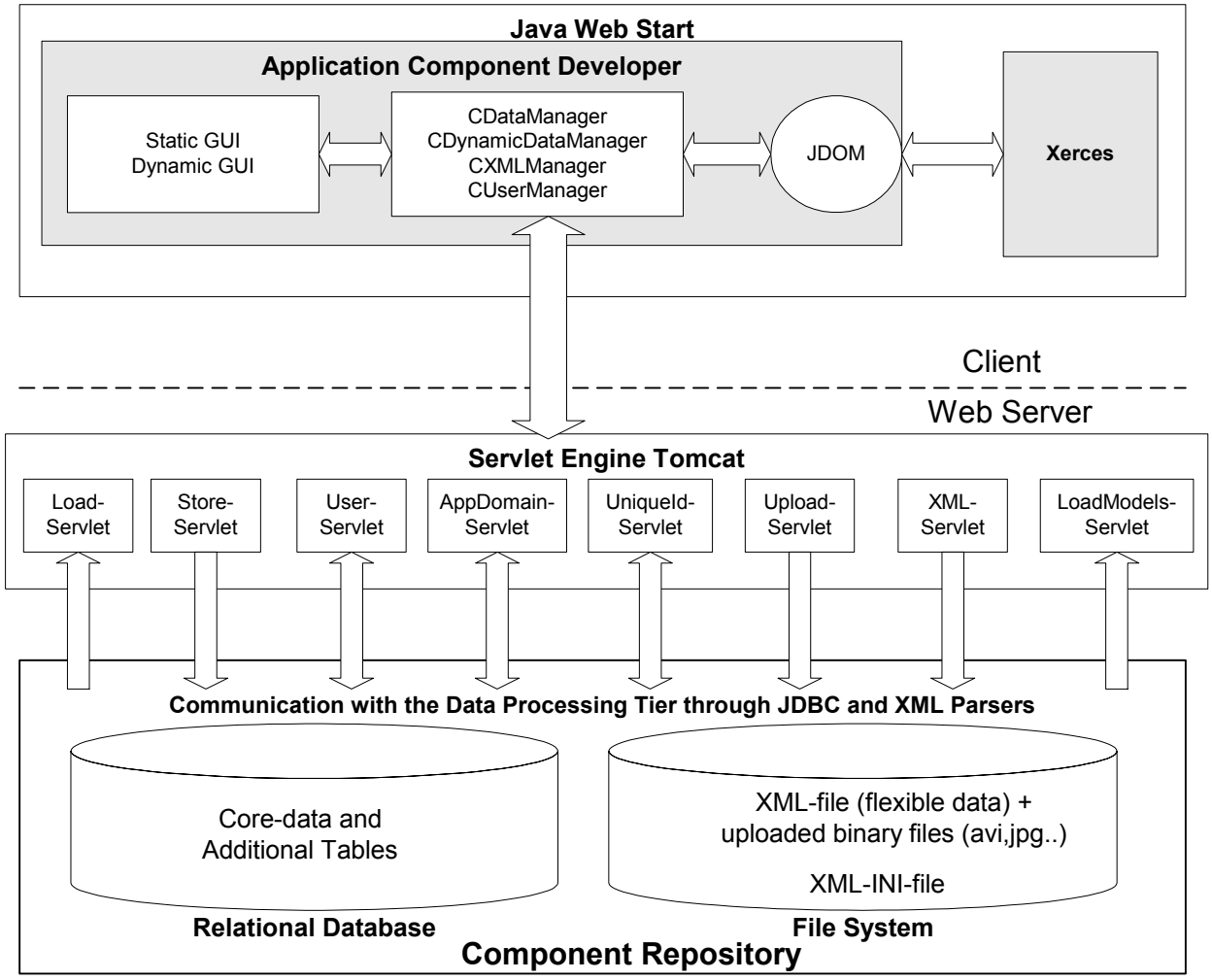


Figure 7.2: Architecture of the Flexible Storage of Components Prototype.

7.2.1 Objectives of the XML-INI File

The initialization file contains a description of the desired GUI and is used whenever a client starts the component developer application programme [Serh01]. The main goal is to generate a flexible interaction acquiring different information based on the actual component technology in use. That is why the initialization file not only contains parameters to configure a proper GUI presentation style of the user interface (adaptable background colors, fonts types, etc), but also contains descriptions of the desired input elements and instructions on how to verify the conformity of this data.

In principle, a initialization file could be written in any electronic form or data format as it is always possible to develop a particular parser to read this file, understand its commands and parameters, and convert it into the desired programme application. In the construction of this prototype, the XML language was chosen to compose the initialization file because XML has a standard and well accepted syntax and because there are currently many tools available on the market that can manage XML files properly. The choice of such tools reduce the effort by constructing a parser for a private usage.

Nevertheless, a description language still needs to be developed, with valid tags and parameters, able to represent all elements that must be present at a component developer interface for this component management system. The XML-INI file developed for this prototype contains these elements describing the desired fields and data types used to format the interface to component developers, whose main goal is to acquire data about components being classified and documented. Some rules about the desired interactions between users and system were previously defined and are listed below:

- The input data must contain text fields (only one line of text), memo fields (many lines of text), lists with pre-defined values (enumeration), and dialogue fields to represent the elements of the desired interface to the user.
- The input fields must contain integer, double, string, binary, large binary and date data types.
- One item field is able to receive more than one value.
- Every element of the interface is expected to contain a description of the element itself that may be used as help for users.
- The pre-defined data must contain information about the accepted range represented by the insertion of default, maximum, and minimum values. These values are supposed to be interpreted in the context of the particular data type.

The above mentioned characteristics of the initialization file are implemented with the help of the XML tags presented in Table 7.1.

Table 7.1: XML Tags used in the XML-INI File.

Tag	Description
<IASCGUI>	Denotes the root-element
<Item>	Tag specifying that this is a new graphical element. Item elements are identified through an attribute called ID, which is unique inside an XML-INI-file
<Type>	Specify the description of data types
<Caption>	Denotes the headlines of one graphical element
<Name>	That is the name of one element, the way it is referenced inside the document
<Min>	Symbolize the minimal valid value of a field element and varies in accordance with the data type of this element. For non-numeric data types the comparison is implemented using the Java method "equal"
<Max>	This is the maximal value of a field element
<DefaultValues>	Denotes the standard value of a field. The input data contains this value if no action contrary to this occurs
<Presentation>	Designates a desired input element
<MultiAllowed>	Boolean variable communicating if one item accepts more than one input value
<Description>	Denotes textual explanation about the items to be inserted

The next question after having decided about the initialization file format is how to parse this file properly. As cited in chapter 6, there are many different parsers available on the market able to deal with XML files, most of them appropriate to work with the SAX and DOM APIs. SAX is an event-based interface that generates events during the parsing of a file whenever a tag is identified. These events are captured and processed by the application. Users have no access to the whole document but only to the actual element being processed. DOM offers the possibility of manipulating the whole document as well as representing this document in a tree structure. In this prototype it is necessary to execute some elaborate processing tasks over the whole XML file as for example reading the whole file and comparing it with the desired model. This fact leads to the decision of working with a parser compatible with DOM and able to present a good performance over the concurrency. The chosen parser was the JDOM which offers the access to the whole XML document, supports write and read operations, has a close relation to the Java programming language, and has efficient methods that results in concrete Java objects. The used version of JDOM was the Beta version number seven.

7.2.2 Description of the Client-Side Elements

The interface to the component developers is implemented as an application that must be distributed with the support of the Java Web Start, which is the reference implementation of the Java Network Launch Protocol (JNLP). The client side application is divided into three smaller functional elements, the application component developer, the Java Web Start, and JDOM together with Xerces that are described below.

The task of the application Component Developer is to offer a convenient interface for the classification, for the documentation, and later on for the storage of components. It additionally

supports the actualization or modification of already stored data. This application interface was developed using the SWING library of the JDK 1.3 and contains the following features:

- Possible usage in off-line mode
- Run-time dynamic generation of GUI elements using the XML-INI file contents
- Efficient presentation and manipulation of information about the components with support of GUI elements like lists, tree structures, and other dialog fields
- Support of different component models
- Data upload from the Web-server side.

The features mentioned above were implemented by some Java classes, and the most significant classes used are presented in Figure 7.2. The class *CUserManager* is responsible for the management of the data about users and companies. *CdataManager* deals with the core-data of the component model implementing read and storage mechanisms. *CDynamicDataManager* manages the flexible-data part of the component and also covers the generation of GUI elements based on the XML-INI file contents. Lastly, the class *CXMLManager* is responsible for the manipulation and generation of XML data files containing the technical information about the inserted components.

In this prototype the combination of JDOM and the parser Xerces is basically responsible for three activities. The first is the parser of the XML-INI files where the dynamic GUI to the users is described permitting a run-time adaptation to users and to component models. The second is the conversion of the core-data stored on the relational data base system to an XML string whenever it is used, for example to present the classification values already stored as suggestions for new classifications. And lastly, they are responsible for the generation of complete document files containing a combination of the core-data and the flexible-data, usable for example, for data exchange with different systems. The installation of JDOM is done directly. It is only one file distributed together with the application, while the Xerces is sent to clients as a separate data with help from JWS.

7.2.3 Server-Side Elements Description

As seen in Figure 7.2, eight servlets were developed for the realization of this prototype whose functionalities are briefly described in the next paragraphs.

a) *LoadServlet* – This servlet is used when component developers select one already stored component as a basis for the classification or documentation of a new component. It is also used for the modification of existing information. The information stored in the data base and the respective XML file are loaded and sent to the client side application in the form of an object that will be manipulated by the class *CDataManager*.

b) *StoreServlet* – Whenever someone commands the storage of a component at the client side, this servlets receives the data stored in the memory at this moment and stores the core-data part in the RDBS and the flexible-data in the file system.
c) *UserServlet* – This servlet is responsible for verifying the kind of user who is trying to access the component management system. Depending on this category, the user will be asked to introduce password and will receive the corresponding access rights.
d) *AppDomainServlet* – The classification structure of a component considering its domain and sub-domain is handled by this servlet. It reads the actual domain tree from the data base and presents this structure as a suggestion for the classification of a new component. Users are supposed to select one of the available values, but they are also free to introduce new specifications. Whenever detecting that a new specification for the domain was suggested, this servlet stores it in the repository making those new values available for further usage.
e) *UniqueIdServlet* – The main task of this servlet is to generate a unique identification for any new component managed by this system. This is done with help from the RDBMS using the primary keys created for each component.
f) *UploadServlet* – Representation models may be configured to contain large binary data as, for example, video or audio files explaining the correct usage of components. The proper transfer of this data from its original location to the repository is done by this servlet. Such procedure results in the creation of new sub-directories and links written at the file system.
g) *XMLServlet* – This servlet is responsible for verifying the consistency of a recently generated XML file containing the information about one component and the respective model representing the desired elements of this technology. After this verification, *XMLServlet* generates all necessary subdirectories and commands the storage of the component's XML file.
h) *LoadModelsServlet* – This is the servlet that loads one or more different component models available, preparing the system to deal with these models. It reads the information of the desired model from the file system and sends the necessary information to the client side application, that works this data properly.

7.3 Searching on the Repository Dynamically

The way someone searches for a desired object in an electronic library or any other similar application may have many variations [Horn01]. In this prototype three alternatives were considered:

- Users may search based on pre-defined and well known keywords or keywords categories. This approach is well suited for the usage of the IAC_d, its facets and respective values as search criteria.
- The second alternative consists of searching all over the repository towards any desired textual formulation.

- The last search alternative is based on the browsing of the existing structure of the data and their classification categories, which enables a complete overview of all elements stored at the repository.

But the repository may not contain the desired artifact resulting in empty lists of candidates for any of the searching alternatives mentioned above. For these cases, some helping procedures were added to this prototype in order to permit a later request of a desired component. Additionally, a procedure to notify users whenever a stored component does not fulfil the published characteristics was implemented. This feature was developed through an evaluation system where users may give a grade for components they have already used. Figure 7.3 presents the main elements of this prototype.

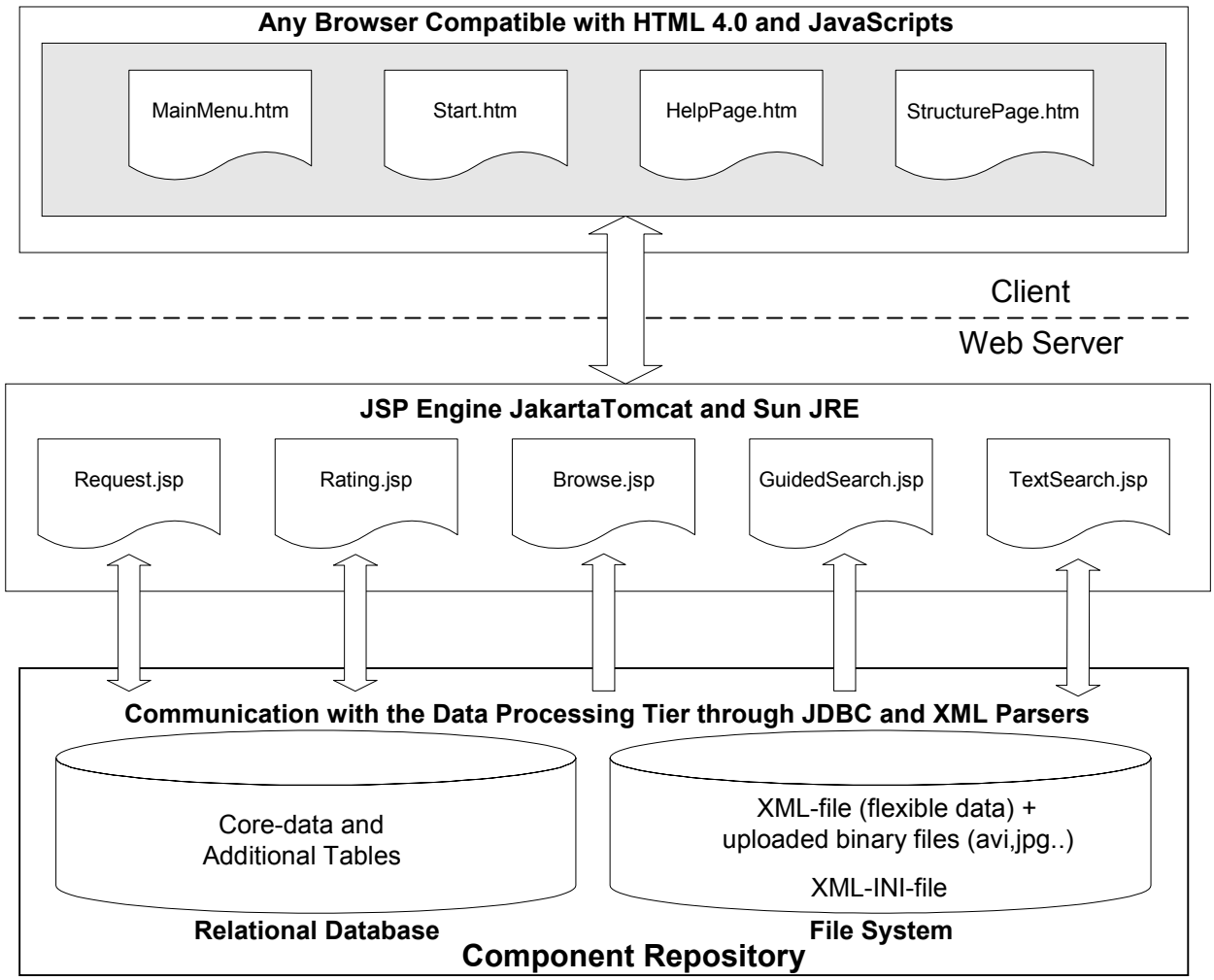


Figure 7.3: Architecture of the Dynamical Search of Components Prototype.

Note that the client side was implemented as a set of HTML documents embedded with Java scripts. These pages call the necessary Java Server Pages generating the dynamical searching in accordance with the contents stored at the repository. The data processing tier, also hosted at the server side, consists of a JSP engine able to process the desired functionalities. The Web-server

used has the Sun JRE version 1.3.1 running on it. The communication between the processing tier and the data storage tier was also implemented via JDBC drivers, and via an XML parser. In the next sections some details of the implemented Java Server Pages are presented.

7.3.1 Guided Search and Free-Text Search over the Repository

One problem of searching approaches using controlled vocabulary is the proper way of communicating the search criteria and corresponding valid values of the vocabulary to potential users. In this prototype, a mask presenting all search criteria was developed as a front page for users researching the repository. The searching criteria of the mask were derived from the IAC_d plus the name of the developer company. A screenshot of this mask is presented in Figure 7.4.

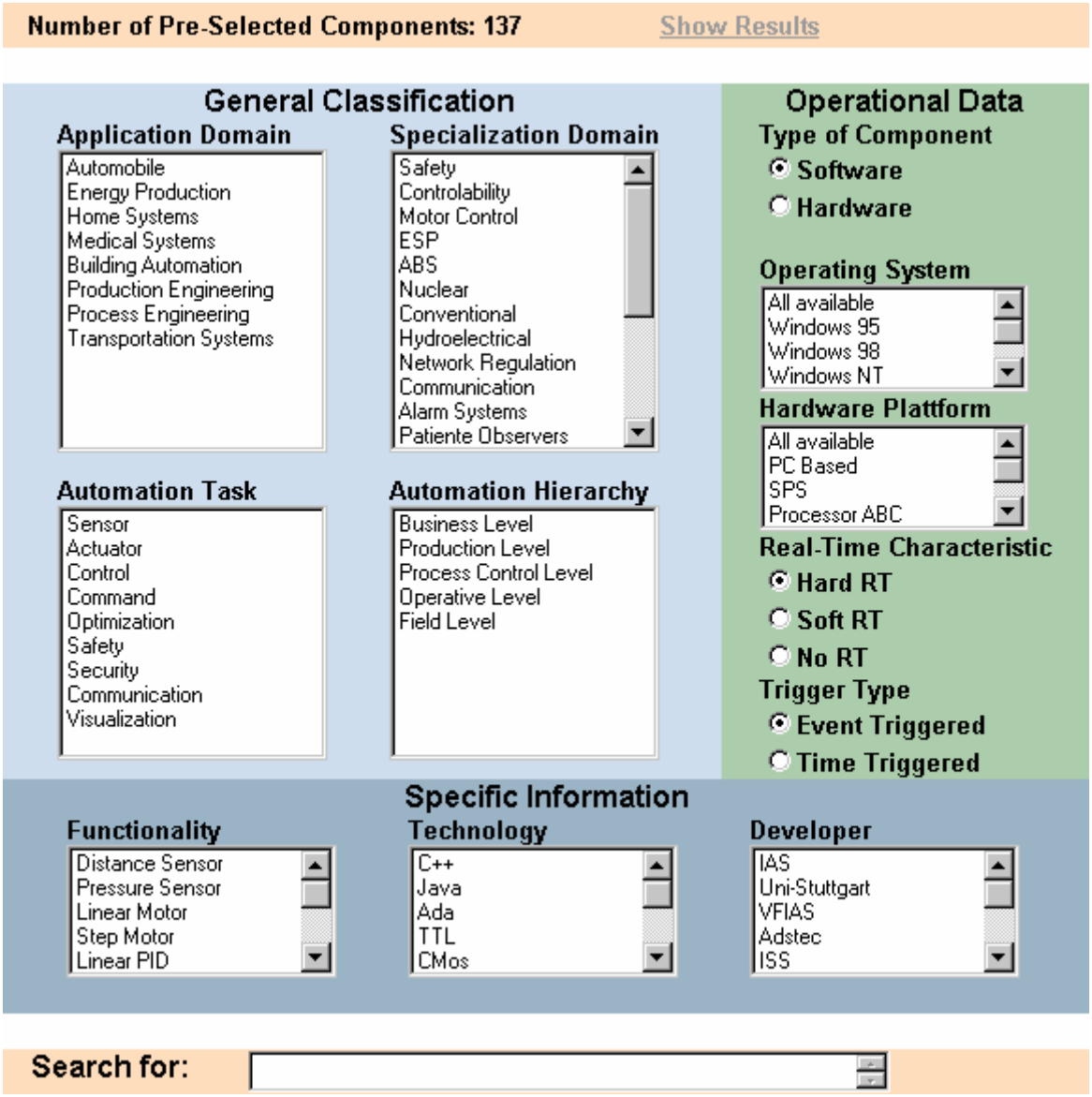


Figure 7.4: Front-end Mask presented for the Guided-Search Option.

This Guided Search alternative is implemented by the page *GuidedSearch.jsp*. The presentation is dynamically adapted in accordance with the repository content. Users are supposed to select one of the presented values by clicking on it. This selection generates a SQL query over the data base whose results are meaningfully presented with the actualized contents of all other selection criteria for further interactions. Every time users refine their selections, the number of available components decrease. Whenever users believe the number of pre-selected components is small enough for a detailed evaluation, they may request to see the results which are presented as a list with the name of the available components.

In case users believe that the offered search criteria or the presented values are not enough to attend to their necessities they may use the Free-Text search implemented by the page *TextSearch.jsp*. Here a string comparison is implemented over the files containing information about components stored in the file system. Additionally, a query is generated to search the data base for the desired value. This procedure was implemented to cover the cases where users prefer the Free-Text approach and just ignore the values presented in the guided search.

7.3.2 Browsing the Complete Repository Structure

The above mentioned search options were developed to serve users that know what they are searching for. That is the case of application developers looking for functionalities they need based on their top-down analyses of the application being constructed. Nevertheless, the component based development sometimes starts from already existent components being used as the base of a desired application that will be completed by some additional work. In this approach, requirements which are not completely fulfiled are programmed from scratch being integrated to the components previously selected.

Users wanting to proceed in this second way need to receive a wide view of the components stored at the repository, which is acquired easier through the browsing over the whole content of the system. The page *Browse.jsp* implements this browsing mechanism. The IAC_d is used as a tree structure where each leaf is one of its facets. Users navigate on this structure receiving a list with all components available at the last level of one branch. The facets Application Domain and Specialization of the Domain were combined to generate a browsing tree with a dynamical number of levels. This number depends on how detailed the original classification of a component is. The other facets of the scheme always result on a browse tree with only one level.

The browsing items are always obtained from the values stored at the data base system. For that purpose a SQL query is generated for each click of the users on a desired topic. This permits a dynamical generation of navigation items containing the values stored at the repository at this moment. New values recently stored, i.e. the most recent values used for the classification of a brand new component, are automatically incorporated as browsing items the next time a user clicks on the upper level of this classification values.

7.3.3 Additional Implemented Features to Find a Component

Three additional features were implemented in order to increase the interaction options between users of this prototype. The first was a thesaurus functionality applied to discover possible synonyms not yet considered in the component management system. The second is an optional rating procedure for users that have already tried to use a component. And the last feature is the support for the generation of a request for new components. All three features needed additional tables at the DBMS in order to store data from users. No modification was necessary to be implemented at the core data structure, but each new table has one reference to the components. These three alternatives are described below:

- Thesaurus: The thesaurus functionality was developed to cover cases where users of the Free-Text search are not able to find any component based on the string they used. In such situations, a page is presented where the values available for the facet Implemented Functionality are listed. Users are asked to verify if any of these values may be used as a synonym for the string they were originally searching for. If this is the case the original string and the selected value are stored in a new table in the data base. The next time, someone searches for the first string, all components classified using this string and the values stored in the table as synonyms of this string will be presented. This functionality was implemented as part of the page *TextSearch.jsp*. It was considered that a term may have at maximum of five synonyms.

- Rating Procedure: It is known that many times the published characteristics of components does not conform exactly with what they really implement. Component users must trust component developers but may also claim against wrong documented components. This feature is implemented by a rating procedure where users are able to register their satisfaction or disappointment with a component previously used. In this rating procedure component users introduce their opinion and emit a grade for one specific component. The comments are stored as textual fields and can receive many lines of text, while the grade is a numeric value based on a scale of integers from 1 to 10 (1 is the worst grade). Authors of commentaries must identify themselves through the insertion of their email addresses. All these data are stored at an additional table that has also a field to receive the calculation of the grade average of each component. The page *Rating.jsp* is responsible for this feature. Here a kind of forum is generated where dissatisfied component users are able to register their claims and, on the other hand, component developers may detect and eventually correct failures of their products.

- Request Generation: As already mentioned, there may be occasions when no component could be found to fulfil the user requirements. The solution proposed by this prototype, implemented by the page *Request.jsp*, is to give support to component users to generate a request based on their original requirements. Someone using this feature is asked to fill a

formular describing the possible application area of the component, and the correspondent functional and non-functional requirements. Users must introduce their contact address. Component developers are able to browse all requests available in a dynamically generated page. This procedure facilitates the communication between component developers and component users working like an opportunity table for the first and like a wanted announcement for the second category of users.

7.4 Dynamical Presentation of the Component's Information

The system architecture of the dynamic presentation of component's information prototype is shown in Figure 7.5 and consists of four main parts, which are the Dynamic Interface Content, HTML Generator, E-Mail Module, and Comparison Module [Aunv01]. As seen in the architecture below, the client part of this presentation system only needs a web browser to communicate with and to request for services from the Web server side. The server side system elements are described in the following sections.

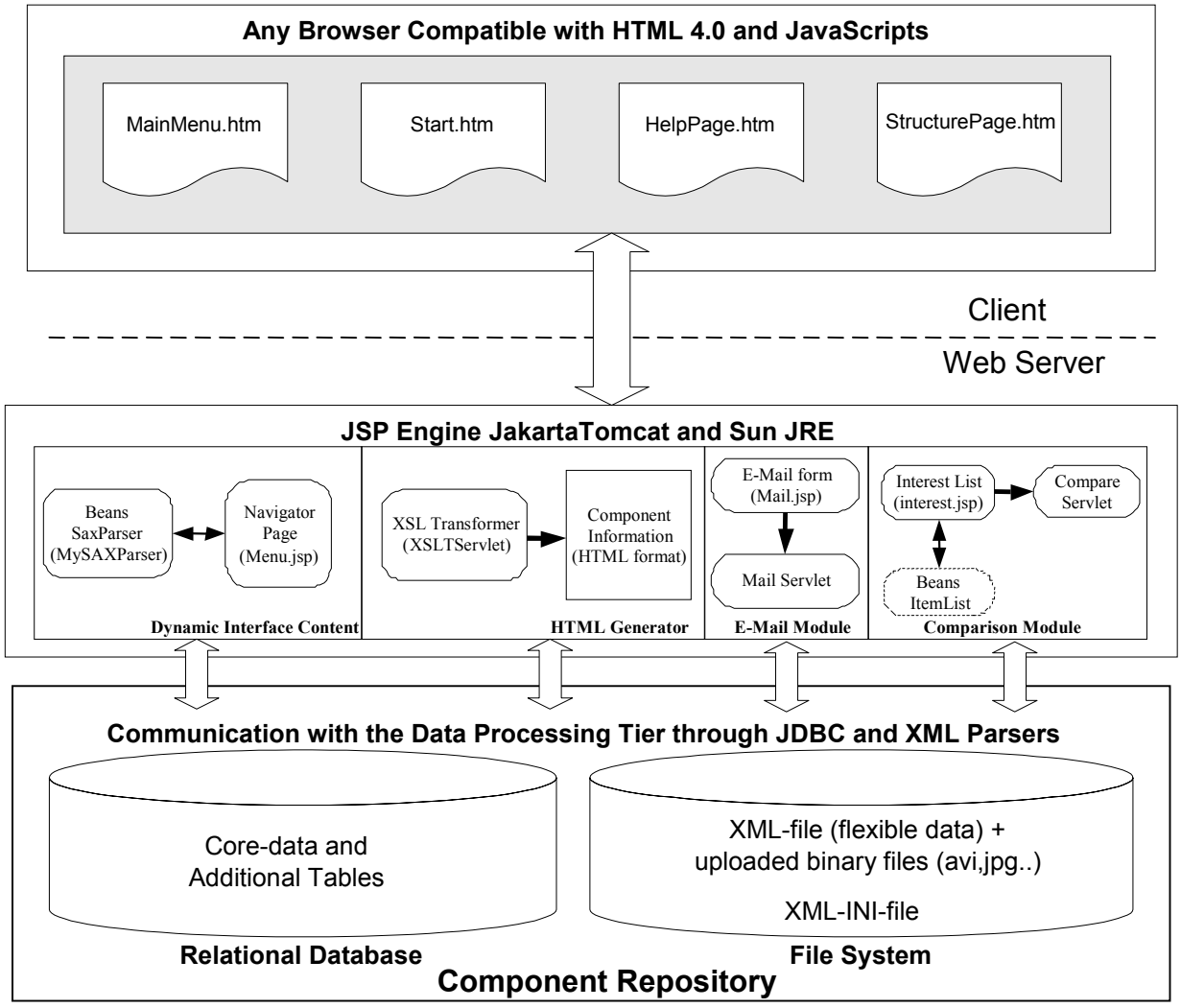


Figure 7.5: Architecture of the Dynamical Presentation Prototype.

7.4.1 Dynamic Interface Content Generation Part

There are two elements within this sub-system, a Navigator Page and an XML Parser. The Navigator Page generates the static HTML web page sent to component user's web-browser whereas its contents are dynamically adapted depending on the component in interest. The system part *Menu.jsp* receives the identification of the desired component, here called Component ID (CID), from the searching part of the complete component management systems. The CID number is enough to identify the name of the XML file where the flexible-data of the desired component is stored. From this point on, the XML parser knows which XML document must be read, and the desired element values are returned to the JSP page composing the desired content dynamically.

Differently from the flexible storage prototype, in the dynamic presentation, the component information documented in XML is read by a SAX parser. The main reason for this choice is that in this prototype no global insight of the structure of the document is needed. There are several Java APIs for XML offered on the market for parsing XML documents compatible with SAX. Here the so-called *MySAXParser* was used, which is implemented using the JavaBeans technology and has all the advantages of this software component technology being reusable as well as compatible to the Java environment as applets, servlets, standalone Java applications, and especially JSP pages. The Java class *MySAXParser* is used by *Menu.jsp* to parse through XML document in order to get particular element values to configure the navigator page for component users studying a specific component.

7.4.2 HTML Generator Details

The HTML Generator part of the dynamic presentation prototype is used for solving the lack of XML supporting browsers in the meantime. The most important system component in this part is the XSL Transformer which was implemented using Servlets, and plays an important role in the presentation of component information making this information viewable in the current browsers available in the market. The Extensible Stylesheet Language (XSL), is the part of the XML family which is used to describe the presentation rules that apply to XML documents. This is an important task because XML is not suitable for displaying information directly on Web pages as it does not include any information about what the visual presentation should be. XSL enables the reusability of styling templates as well as forces a clean separation of contents from presentation. XSL includes both a transformation language, called XSLT, and a formatting language, which are independent of each other. XSLT is an XML-based language and W3C specification that describes how to transform an input XML document into another XML document, or even into HTML, PDF or some other document format.

In this prototype, several XSL stylesheets were prepared for the sake of representing the information within XML documents appropriate to particular component technologies.

Therefore, whenever an XML document is requested to be read, the type of the component is considered selecting the proper XSL Stylesheet permitting that the XML data be rendered suitably and correctly in the desired format. This procedure starts with users selecting their interested component. The system searches for the parameter value that defines the path to get the XML document in the file system within the web server. For each technology considered there is a respective stylesheet that must be used. Finally, the XSL Transformer is called and the HTML document is generated and sent to the users.

Recently, only the Internet Explorer version five or later contains XSL display engines enabling XML data to be applied to XSL Stylesheet on the client browser to produce HTML pages. However, in the meantime, it is more reliable to convert all desired XML files to the HTML format on the server side even considering the frequent industry announcements of future browsers able to render XML directly. Additionally, the performance of XML parsers and XSL processors is presently sufficient to be used within the server without decreasing the server performance in real world applications. The HTML Generator was implemented to do the transformation of XML documents as shown in Figure 7.6.

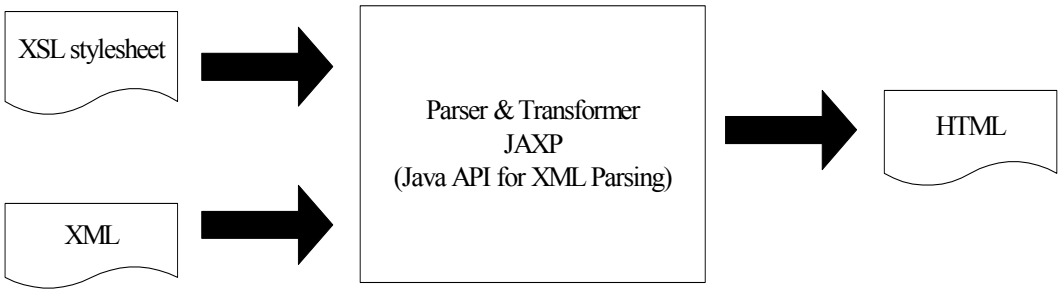


Figure 7.6: HTML Generator Overview.

This generator was implemented using Java API for XML Processing (JAXP), which was developed by Sun Microsystems and provides basic support for parsing and transforming XML documents through a standardized set of Java platform APIs. This XSL Transformer engine is called *XSLTServlet* and it is used whenever an XML document is requested to render information by component users. From the navigator web page (*Menu.jsp*), *XSLTServlet* is called and it gets the required input parameters such as XML file name and XSLT style sheet type that are stored in the file system. Component information within XML document is converted into HTML format page on the server, and then the resulting document from the transformer engine is sent to the component user's web-browser.

7.4.3 Description of the E-Mail and Comparison Modules

In this prototype, an E-Mail module is the communication tool that has been prepared for component users to communicate with developers. This E-Mail module works on the server side, therefore it does not require any special application from client side to be used. Component

users just click on an e-mail address presented on the information web page and the e-mail module is automatically started. Then component users can express their needs or requirements about components via a textual message.

There are two elements inside this e-mail module. The first one is the E-Mail form (called *Mail.jsp*) and the second one is the Mail Servlet. The E-Mail form is an interface to fill in mandatory information such as user's name, e-mail address, subject of contact, as well as complementary information that the user wants to communicate to the e-mail receiver. At the end, component users click on a "Send" button calling the Mail Servlet which receives the content of the formular sending it to the receiver. If the service is successfully done, the Mail Servlet sends feedback to the user informing them that the "Message was successfully sent".

It is assumed that there will sometimes be several components that can be used to complete one desired task. In this prototype, a comparison module implements a help tool for users to decide which component is the most suitable one for satisfying their requirements. The criteria to be compared by this module are dynamically chosen, i.e. users can select the desired criteria according to their interests. The offered criteria are Company Name, Price, Component Technology, Operating System, Hardware Platform, RAM Usage, Response Time, Trigger Type, Complete Size, and Real-Time Classification, Reusability Grade, and Quality Measure. There are three parts within this comparison module, the Interest List, the Item List, and the Compare Servlet. The Interest List is a JSP page that is used to display the interested component names already selected by users and to display the comparison criteria. If the user hits on the "add" button in the navigator web page, the Interest List (*interest.jsp*) is called, then the Item List, developed as a JavaBean, is employed by this JSP page to keep track of the user's interest. The selected components are listed and displayed to the users within the *interest.jsp* web page, which is shown whenever the "add" or "remove" button is hit. From the Interest List page, users may hit a "compare" button whenever they want to make a comparison of listed components versus selected criteria. Then, the Compare Servlet is called and executed in the server side. This Servlet connects the database system via JDBC in order to fetch information of the component regarding to the selected criteria. After getting all required information, the results are sent to the users in HTML format.

7.5 Exemplary Use of the Component Management System

The prototypes described in this chapter were tested using diverse component technologies. XML-INI files, Stylesheets and Schemas were developed to cover the JavaBeans, ViPER Synchronous, and Hardware component technologies. In order to illustrate the potential of the conception presented in this thesis better, some details about the creation of the components documentation for the synchronous technology are given in this section followed by some screenshots of the component management system.

A ViPER synchronous component, called TON (Timer-On), was chosen as a basis for the example explained here. The complete data available about this component is reproduced in Appendix A. Such a datasheet is the origin of the analysis of what shall be included as flexible-data about one component technology. In this particular case, it is possible to conclude from the general information part about TON that one technical related link, the complete documentation, and the quality measure of this component are information normally available. In this case, the technical link is a line of text, the complete documentation is a PDF file and the quality measure is a number. ViPER components are supposed to have a textual description of their functionality and whenever possible, a graphical description to help understanding this information. Some of them have parameters to adjust their behavior and the interface is described as a table with called signals followed by their direction (input or output), their type, and by a small textual description. Three keywords were chosen to classify the implemented functionality of this component better. This field shall contain as many values as desired by the component developer. In this case, three usage examples were added, one post script file, a video presentation, and a flash animation. The response-time was attributed as a percentage of the system's clock period. Three post-script documents relating the performance verifications about this component were introduced permitting users to receive documentation proving the way the operational information was obtained. Such fields may contain documents stored in different formats but may also be omitted. Lastly, the commercial information contains exemplary data about the company and the developer responsible for the component.

The XML-INI file is obtained based on such analysis of a standard data sheet of the component technology. Here it is assumed that the presented datasheet contains all standard elements composing the information about synchronous components. Figure 7.7 shows parts of the obtained XML-INI file for that technology. In the figure, three elements represent the textual description of the functionality of the component, the component's quality measure, and the performance verification of the component. These three pieces of code are described in sequence in order to clarify their meaning and the way they were obtained.

From the datasheet of the component TON, it is possible to conclude that the textual description of the component's functionality is a paragraph containing normal text characters. This information is mapped to the XML-INI file at the item number seven where it is specified that this item is a memo field. It is also a unique value as the element *MultiAllowed* is set to false. Lastly, a small description of what is expected to be inserted by component developers at this field is described under the element *Description*. The information about the quality of synchronous components, the item number thirteen of the example, was mapped as a numeric value composed of integers where the minimal value is one and the maximal is ten. It is also a unique value, it is assumed that a component may receive only one quality grade. Lastly, the performance, item seventeen, is a set of large binary fields which is the category for PDF documents and other types of large files. The default value is set to *Not Available* meaning that

this is an optional field. Users are supposed to describe the type of the verification implemented and to insert the respective file. In this example, the developer of the component TON will insert this information contained on the data sheet. The first one is called *Set of Test Cases* and is associated with the file *TON_Tests.pdf*. The other two follow the same logic. The totality of the datasheet is mapped using the same procedure of the three examples described above.

```
<?xml version="1.0" encoding="UTF-8"?>
<IASCGUI Model_ID="1">
...
<Item id="7">
  <Name>Textual_Description</Name>
  <Type>MemoField</Type>
  <Caption>Textual Description of the Functionality</Caption>
  <Presentation>MemoField</Presentation>
  <DefaultValues>  </DefaultValues>
  <MultiAllowed>false</MultiAllowed>
  <Description>
     Please enter a text describing your component
  </Description>
</Item>
...
<Item id="13">
<Name>Quality</Name>
  <Type>Integer</Type>
  <Caption>Quality Specification</Caption>
  <Min>1</Min>
  <Max>10</Max>
  <Presentation>IntegerField</Presentation>
  <DefaultValues>
    <Value>1</Value>
  </DefaultValues>
  <MultiAllowed>false</MultiAllowed>
  <Description>
    Please enter the Quality Measure of your Component
  </Description>
  <Description>The value 1 means that no Measure is available
               and 10 is the best value possible </Description>
</Item>
...
<Item id="17">
<Name>Performance</Name>
  <Type>largebinary</Type>
  <Caption>Performance Verification</Caption>
  <Presentation>TextField</Presentation>
  <Presentation>LargeBinaryField</Presentation>  <DefaultValues>
    <Value>Not Available</Value>
  </DefaultValues>
   <MultiAllowed>true</MultiAllowed>
  <Description>
      Please enter the name of the verification and your PDF file
  </Description>
</Item>
...
</IASCGUI>
```

Figure 7.7: Partial Content of the XML-INI File for the ViPER Synchronous Component Technology.

Such analysis of a component technology is done one time being modified as many times as necessary throughout the usage of the system. Based on this information, the flexible data about the component is acquired and stored on the file system and on the database. The desired way of presenting this information for component users is configured in an XML Stylesheet that is also designed specially for each component technology to be managed by the system. As said above, similar procedures were done for the JavaBeans and for the Hardware component technologies. Figure 7.8 shows a exemplary list of available components that could be obtained from the guided search or from the free-text search containing five component candidates.

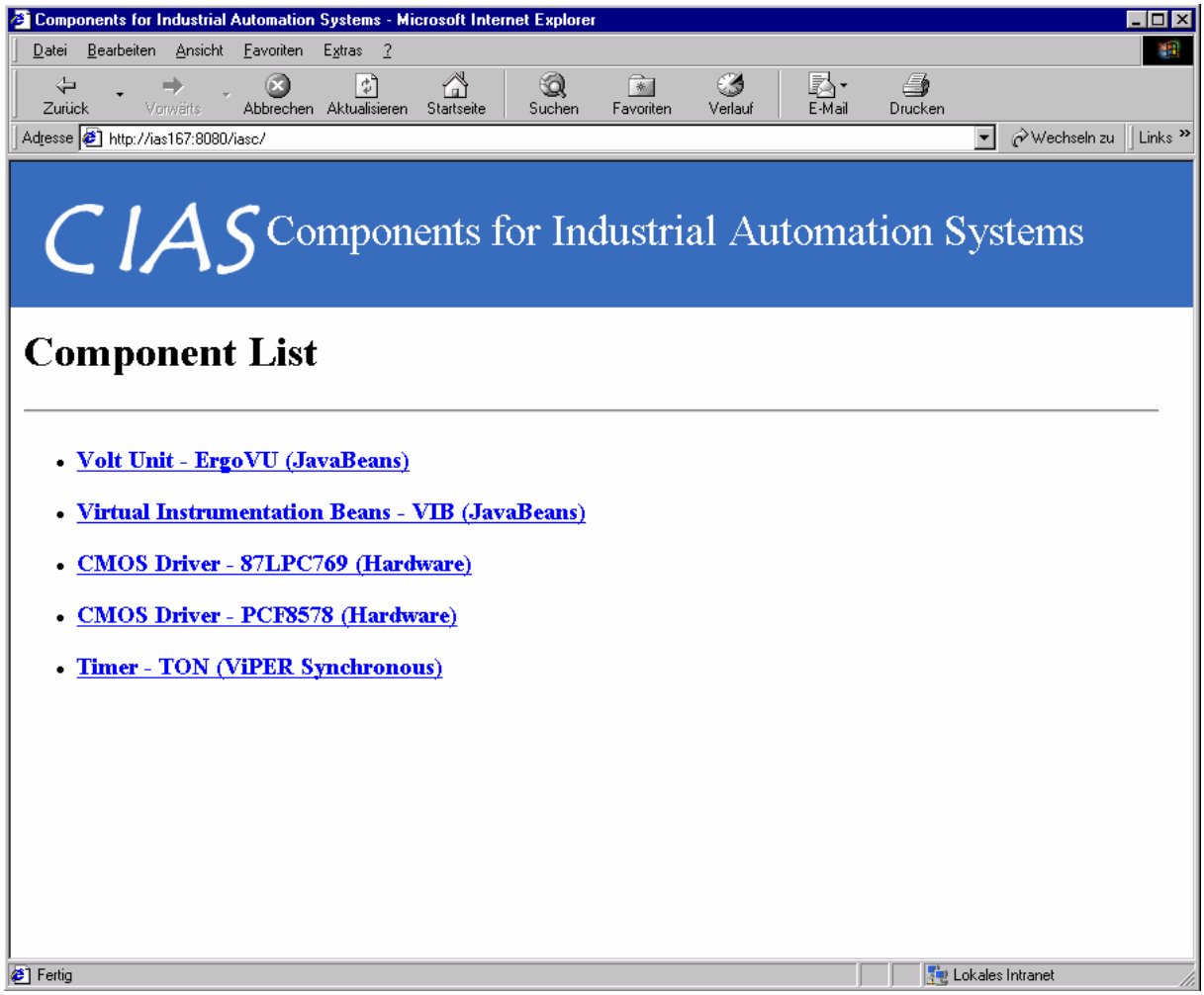


Figure 7.8: Exemplary Result of the Search Prototype.

Whenever users decide to evaluate a specific component, they click on the desired component name. The result of the choice of the ViPER component called TON from Figure 7.8 is presented in Figure 7.9. This page is the result of the transformation of the XML file containing the flexible information part of the component TON using the instructions contained in the Stylesheet developed for its component technology. The information structure is also dynamically generated being adaptable in relation to different component technologies, and inside one same technology being adaptable in relation to the information available for one specific component. For example, the item *Additional Document* presented for the ViPER

component TON in Figure 7.9 may not be presented for the another ViPER synchronous component. In the same sense some other field may appear in other components that are not presented for the component TON.

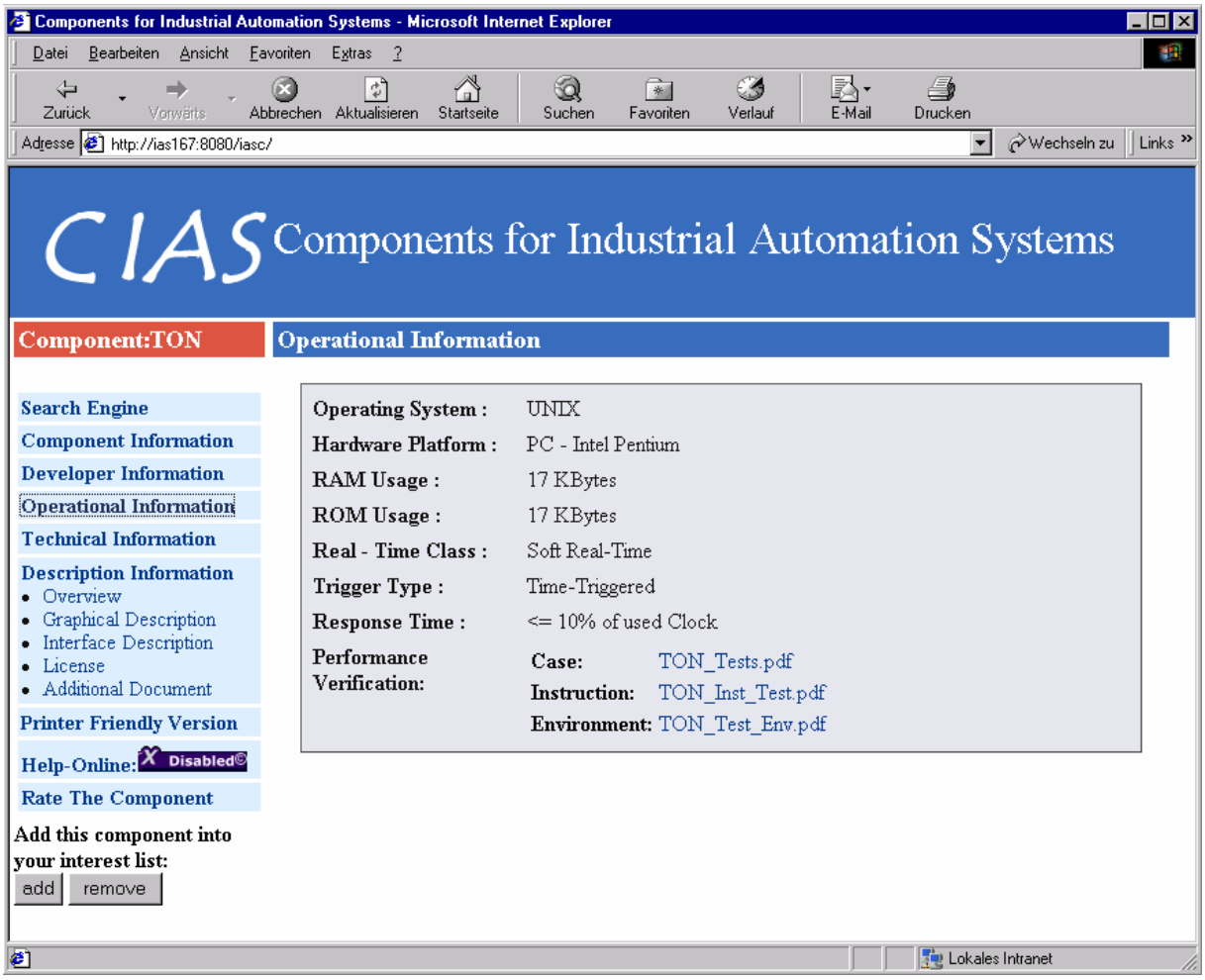


Figure 7.9: Component Information Page in HTML Format.

In Figure 7.9, it is possible to see the buttons “add” and “remove” used to generate the dynamical comparison. Whenever component users are interested in selecting one component they must activate the “add” button. This component is then inserted into a list of interest of the user currently working with the system. Wrongly inserted components may be removed from the list through the button “remove”. After selecting the desired components, users are able to command the comparison among those components. For this purpose, they must choose the comparison criteria and may also annotate their own expected requirements inserting those values in a dynamic generated table. Figure 7.10 shows a list with two selected components and the comparison criteria available to chose. The last column in this figure is where users are supposed to annotate their desired values for each criterion. In Figure 7.11 the result of the comparison commanded is presented. Note that the table is dynamically generated based on the selected components and on the chosen criteria. The desired values inserted by the user are also presented in the table.

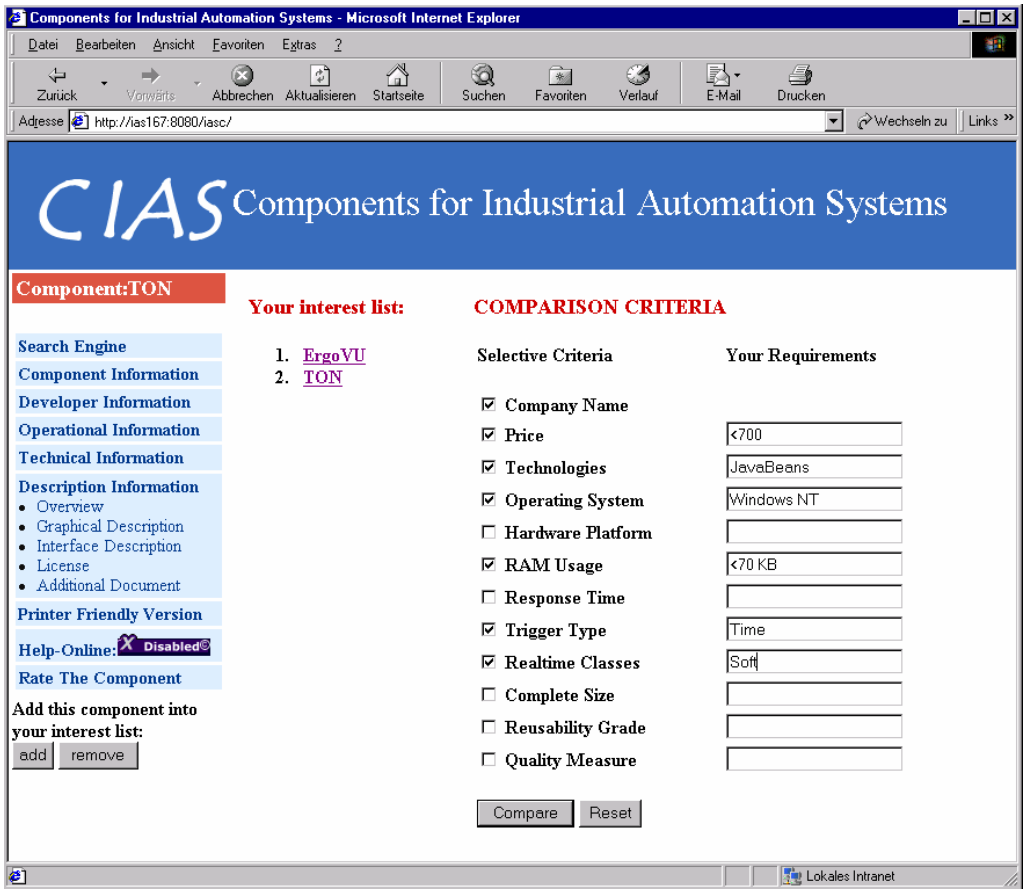


Figure 7.10: List with Pre-Selected Components and Comparison Criteria.

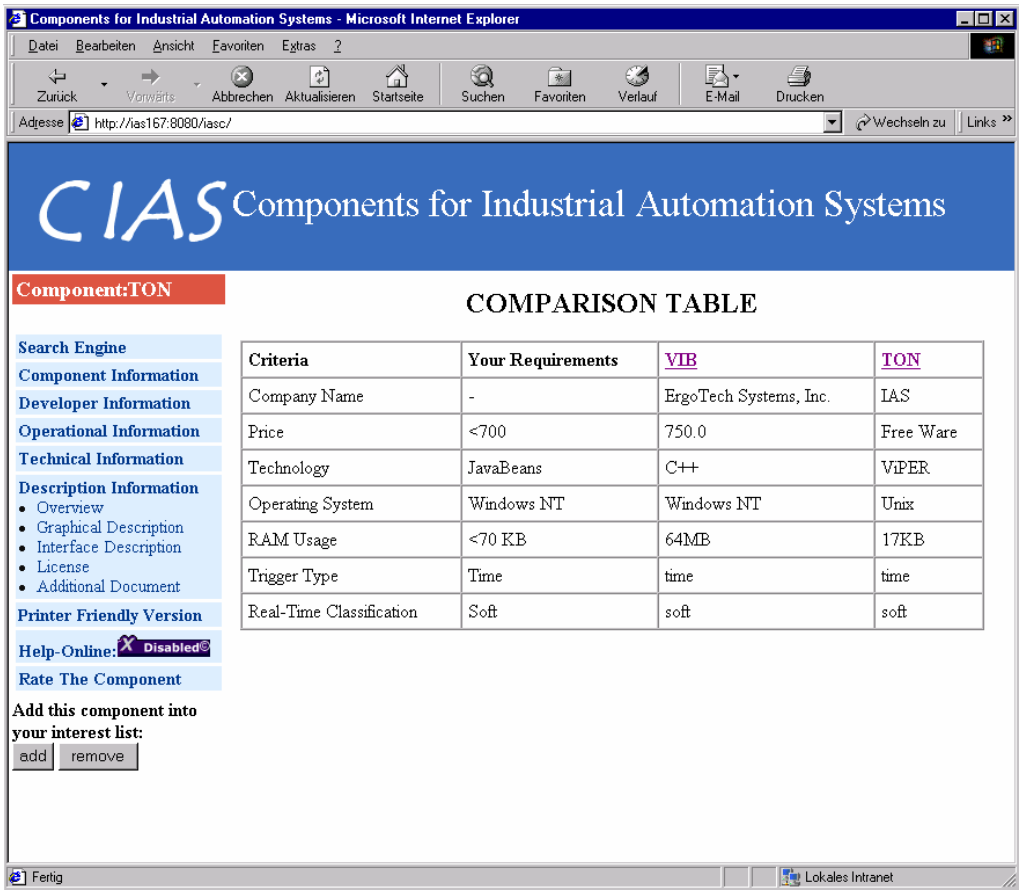


Figure 7.11: Dynamically generated Comparison Table.

7.6 Some Final Considerations about the Prototype

In this section some personal considerations about the developed prototype are formulated in order to give the reader a better insight into the work developed. The main reason is to compare some theoretical suppositions about the concept of the component management system with some real aspects observed during the construction of the prototype and later on during its tests.

The insertion of new component technologies in the repository is really a task that demands some effort. The first analysis about one component technology always resulted in a model that could not attend the necessities of all users and all developers of components of the specific technology. For example, to reach the model and documentation structure of the ViPER components presented in Appendix A of this thesis four modifications were necessary. These modifications demanded some work over the XML-INI files and XML Schemas but they did not form a big problem due to the easiness of the XML technology and due to the flexibility offered by the component management system.

The way of storing new components in the repository proposed in this thesis facilitates the work of component developers. The classification of components is easily done due to the presentation of the classification scheme itself. More interesting than that is the presentation of value suggestions for each facet of the scheme. The documentation itself demands some work but does not take much time if the component is already documented electronically. Moreover, asking component developers for specific information about one component being documented guides developers resulting in a uniform component structure inside each component technology involved.

Selecting components through the guided-search mechanism offered by this prototype is more convenient than searching through the free-text way. Through the refinement implemented during the first search mechanism mentioned, users receive the information about the available amount of components and can command the presentation of results only when these amount of components is meaningful for an effective understanding of each component separately. The dynamical comparison of components is really a helpful feature and the possibility of selecting the field to be compared is one of the more interesting features of this comparison mechanism.

Some additional work is necessary to transform the prototype in an actual product. Nevertheless, the global documentation of the prototype was completely implemented which will make this work easier. Some points that were not implemented, as for example security and privacy features, will certainly demand more effort. A meaningful future step is to register the component management system as a Web service. In fact, even the selection part or the publication part of the work here presented could be registered as such a service. Web services enable software systems to interact with each other around the world. In the past, this has only occasionally been realized within private networks using the industry standard CORBA and

Microsoft's DCOM distributed component platforms. Web services still require cooperation and agreement among people to define business transactions and processes. Web services standards only define the format and transport architectures, but the meaning of each element of data exchanged also has to be defined ahead of time by industry consensus.

The registration of the developed system as a Web service may be done by the WSDL (Web Services Description Language) which is a protocol for a Web service to describe its capabilities. WSDL describes the protocols and formats used by the service and can be housed in a UDDI (Universal Description, Discovery and Integration) directory, and the combination is expected to promote the use of Web services worldwide. UDDI is an industry initiative for a universal business catalogue of Web services and it is designed to enable software to automatically discover and integrate with services on the Web. Using a UDDI browser, users can also review the information contained in the registry, which is a network of servers on the Internet similar to the Domain Name System (DNS). UDDI contains white pages (addresses and contacts), yellow pages (industry classification) and green pages (description of services). The green pages include the XML version, type of encryption and a Document Type Definition (DTD) of the standard. UDDI messages ride on top of the SOAP protocol, which invokes services on the Web.

In this chapter the prototyping of the Flexible Web-based Component Management System for Industrial Automation was presented. The global system was divided into three small functional related prototypes described here. An exemplary usage, and some screenshots of the system were added to improve the illustration of the work realized. The prototypes run currently with three different component technologies, the Synchronous Component, the JavaBeans Component, the Hardware Component. In the next chapter some conclusions about the complete work and some insights into possible future works related to this one are presented.

8 Conclusions and Related Future Work

In this chapter, the original scenario of the start-up situation of this research work is reviewed. Afterwards, the evaluation of the work presented is done. This chapter and the global work is closed with some insights about possible future works about this research area.

8.1 Start-up Scenario of this Research

Component-based development is a standard methodology of producing artifacts in many engineering areas of actuation. Dividing a bigger problem into many smaller ones, more suitable for being manipulated, is the everyday practice of engineers all around the world. This successful history led people interested in software development to try similar procedures. One of the most current solutions proposed for these problems is the creation and use of the denominated software components which are one central point of this thesis.

These components have been used in many application areas. Libraries of components designed to be used in the development of graphical users interfaces, for example, are easily found in many common tools on the market. Moreover, the advantages of using software components to develop end-users applications were also tried by other specific application domains. In particular, the component-based approach has been used in the industrial automation domain which is the domain of interest of this thesis.

As the amount of components increases, some problems occur because it is difficult to keep the necessary overview of all components available. Trying to solve these problems, some research has been carried out in order to properly manage such sets of components. At the beginning of this thesis, the most important approaches for the management of components were studied. This literature research proved that many problems were still to be solved. Moreover, solutions specifically proposed for the industrial automation domain could not be found.

8.2 Evaluation of the Presented Concept

Based on the scenario mentioned above, a study about the most relevant components technologies with any application in the industrial automation domain was developed. This study was presented in chapter 3 of this thesis. It was proved that, in fact, components have been used to implement many different tasks in the industrial automation domain. Some existing approaches are specially designed for embedded systems, while other ones are specific for the plant automation. The diversity of these technologies proved that one component management system designed based on only one component model would not be able to contain the most relevant technologies used in industrial automation domain. In spite of that, the goal of this thesis was to deal with all that diversity simultaneously.

The final concept started to be formulated by the analysis of the necessary knowledge to work with components in the industrial automation domain. It was shown, that this domain contains many particular aspects, very relevant for the selection, understanding and reuse of components, which are normally not considered by those developing components. Moreover, the way this information is communicated among practitioners must respect nomenclatures and usage of the original component technologies.

In order to guarantee the observation of these aspects, a model composed of a classification scheme and the technical and non-technical characteristics of components was proposed. For the purposes of this thesis, a classification scheme specially designed for the industrial automation domain was developed. The scheme is composed of eleven facets selected based on the most relevant aspects that users of the industrial automation domain are interested in when searching for components. The facets may receive the values desired by users classifying their components, but a suggestion mechanism based on the values currently stored by each facet was implemented later. This approach leads to a semi-controlled vocabulary for the classification of components.

The model is the central part of the component-based development process described in this thesis. In this process, component users are supposed to classify and document their products in accordance with a specific set of data. This classification and documentation was denominated as publication process. Component users will search, evaluate, and decide about the use of already developed and published components. The complete data of a component was divided into two blocks, one denominated core-data, where the general information was organized and stored, and another one called flexible data, responsible for containing the technological specific data of the components being managed by the system. This separation between technical and general data was the key point to propose one concept able to work with all diverse technologies used in the industrial automation domain.

The main idea of such a concept is to reach as many potential users as possible. That is why, it was researched how the concept presented could be implemented by an Internet-able platform. The desired system was divided into three functionally related tiers. The most appropriate technologies able to implement those aspects were verified. After choosing the most successful technologies for each tier, a prototype was developed. In the prototype, the technical characteristics of a component are represented using the XML and related technologies. The core-data of components is stored as an XML document, the flexible interface to the users is represented by a self developed XML-INI file and by XML Schemas. The generation of this INI file was exemplary shown for the synchronous software component. The prototype proved to be able to receive different technologies respecting their specificity. Currently, it is able to manage the above mentioned synchronous components, the JavaBeans components, and a representation of hardware components.

8.3 Possible Future Works

An interesting feature to be studied in the future is how to translate existing datasheets of specific component technologies automatically into the XML-INI files and the XML Schemas used in this thesis. Currently this transformation needs human intervention, because it was considered that this job must be done by an administrator with specific knowledge about the component management system. Nevertheless, it is possible to foresee that standard datasheets stored in electronic form could be parsed by specially developed applications and translated automatically into the desired initialization files.

Another aspect that could be enhanced in this thesis is the control of the reusability level of components. Much more research must be done in order to achieve a scientific approved reusability grade. The one suggested here is based on empirical considerations and depends strongly on the collaboration of the component users. The automatic acquisition of statistical information about the level of reuse of the components stored in the repository may introduce a better overview of what kind of components are being used more regularly. More that that, it may give important advice about the most accepted component technologies. This statistical information may be divulged in order to guide new component developers towards the necessities of the component users market.

A third point that could be originated from this thesis is the development of a quality certification procedure for components being managed by this system. In the system proposed here, the information given by component developers is not verified. The adoption of quality certification for components is not a consensus yet. Many researchers affirm that this certification is necessary but could not propose a meaningful procedure to achieve it. As soon as one acceptable standard appears on the research community it shall be inserted and tested on this component management system. Such a quality certification could enhance the confidence of users over the components stored in the component management system.

A last point would be to expand the prototype presented here to receive electronic commerce features. Such a platform would work as a real marketplace for component users and component developers with special interest on industrial automation applications. For that purposes many special considerations about privacy and security, not originally considered during this thesis, should be incorporated. Nevertheless, the absence of such a specialized forum in the current marketplace makes this suggestion an interesting point to be discussed in the future.

Appendix A: Complete Documentation of a ViPER Synchronous Component

In this appendix an example of the documentation available for a ViPER synchronous component is presented. The component has the name TON (Timer-ON) and implements the functionality of one of the standard function blocks defined at the norm IEC 1131-3 for the usage at programmable controllers.

A.1 General Information about the Component Timer-ON (TON)

- Type of Component: *Software*
- Identification:
  - General Name: *TON*
  - Short Description: *Timer detector with positive output*
  - Version Number and Date: *Version 1.0 30/01/2002*
- Technical Related Links: *ANSI Organization (http://www.ansi.org/)*
- Complete Documentation: *TON.pdf*
- License: *Free of charge for academic purposes*
- Classification Tag:
  - Application Domain: *Assembly System*
  - Specialization of the Domain: *SPS Systems; SPS Programming; Control Systems*
  - Industrial Automation Task: *Actuator; Timer; Alarm*
  - Automation Hierarchy: *Field Level*

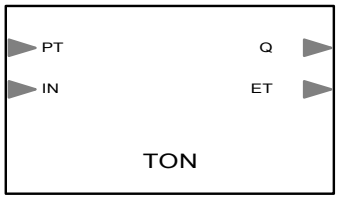
A.2 Functional Information about the Component TON

- Textual Description of the Functionality:

  The numerical input PT will be read and internally stored. The binary output Q will follow a false-to-true jump of the binary input IN after a waiting time of PT, if and only if IN is still true. The time after the jump of the input IN is presented in the numerical output ET until it reaches the value of PT. If IN = false, Q = false and ET = false.

- Graphical Description of the Functionality:

Graphic symbol of TON:



Time diagram of the component TON:

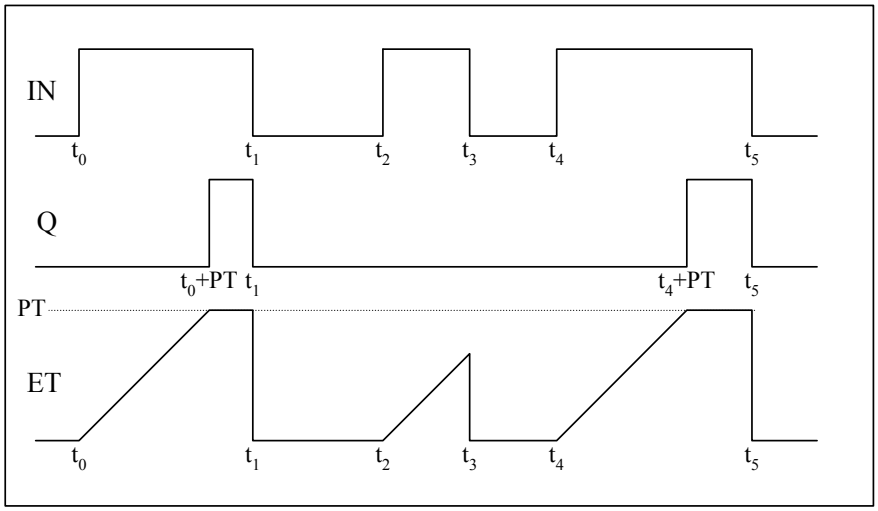


Diagram description:

If the step occur at the input IN:

While IN = 1, ET increase at each clock pulse and:

If ET = PT, Q = 1.

End while

If IN = 0, Q = 0 and ET = 0.

Figure A.1: Graphical Description of the Functionality of the Component TON.

- Parameters: *No parameters available*
- Interface Description:

Interface of the component TON			
Name	**Direction**	**Type**	**Description**
PT	Input	Integer	Numerical input value: Waiting time in seconds.
IN	Input	Boolean	Binary input value.
ET	Output	Integer	Numerical input value: Actual waiting time in seconds.
Q	Output	Boolean	Binary output value.

- Implemented Functionality:
  1) *Timer*
  2) *Positive Output Timer*
  3) *Input Defined Counter*
- Differences from Previous Version: *None*
- Usage Examples:
  - Description of Examples: *TON_Example.pdf*
  - Usage Demos: *TON_Video.avi*
  - Simulations Example: *TON_Simul.swf*
- On-line Help link: *not available*
- References and Experiences: *http://www.esterel.org/*
- Additional Documents: *Consult the Norm IEC 1131-3*

A.3 Operational Information about the Component TON

- Run-Time Environment:
  - Operating System: *UNIX*
  - Hardware Platform: *PC – Intel Pentium*
  - Response Time: *≤ 10% of used Clock*
  - Resource Usage: *ROM = 17 KBytes*
    RAM = 17 KBytes
  - Real-Time Classification: *Soft Real-Time*
  - Trigger Type: *Time Triggered*
- Component Technology: *ViPER Synchronous Component*
- Performance Verification:
  - Set of Test Cases: *TON_Tests.pdf*
  - Instructions for Test: *TON_Inst_Test.pdf*
  - Test Environment Description: *TON_Test_Env.pdf*

A.4 Commercial Information about the Component TON

- Developer Name: *IAS – Uni-Stuttgart; Dipl.-Ing Lucena*
- Developer Contact:
  - Conventional Address: *IAS – Uni Stuttgart, Pfaffenwaldring 47*
  - Electronic Address: *lucena@ias.uni-stuttgart.de*
- Price: *not available*

Bibliography

[ACM98] ACM – Association for Computing Machinery: The ACM Computing Classification System (1998), http://www.acm.org/class/1998/ccs1998.html, January 1998

[Albr00] Albrecht, H.: Object Model of a Process Control Engineering Internet, ISA Expo 200, New Orleans, August 2000

[AMU00] Albrecht, H., Meyer, D., Uecker, F.: ACPLT – Komponenten für die Prozessleittechnik, White paper, Lehrstuhl für Prozessleittechnik, RWTH Aachen, http://www.plt.rwth-aachen.de, 2000

[AyBe99] Ayers, D., Bergsten, H.: Java Server Programming, Birmingham: Wrox Press Ltd., 1999

[Aunv01] Aunvichit, K.: Conception and Development of a Dynamic Presentation System for Component in Industrial Automation, INFOTECH Master Thesis No. 1821, IAS, Universität Stuttgart, November 2001

[Balz96] Balzert, H.: Lehrbuch der Software-Technik : Software-Entwicklung. Heidelberg, Berlin, Oxford; Spektrum Akad. Verl. 1996

[BAB+87] Burton, B. A., Aragon, R. W., Bailey, S. A., Koehler, K. D., Mayes, L. A.: The Reusable Software Library. IEEE Software, Special Issue on Reusability, Vol. 4, No. 4, July 1987, pp. 25 – 33

[BaSc92] Banner, B., Schonberg, E.: Assessing Ada 9X OOP Building a Reusable Components Library. Conference proceedings on TRI-Ada '92. 1992, pp. 79 – 90

[BaTa99] Ballou, D. P., Tayi, G. K.: Enhancing Data Quality in Data Warehouse Environments, Communications of the ACM Magazine,Vol. 42, No. 1, January 1999, pp. 73 – 78

[BBJ+ 91] Beckman, B., Boyd, B., Jupin, J., Shen, S., van Snyder, W., Tausworthe, R., van Warren, L.: Encyclopedia of Software Components. Third annual ACM Conference Proceedings on HYPERTEXT '91, 1991, pp. 425–426

[BCE+97] Basili, V. R., Condon, S. E., Emam, K. E, Hendrick, R. B., Melo, W.: Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components, Proceeding of the International Conference on Software Engineering 1997, Boston, USA, 1997, pp. 282 – 291

[BeFe01] Bertino, E., Ferrari, E.: XML and Data Integration, IEEE Internet Computing Magazine, Vol. 5, No. 6, November/December 2001, pp. 75 – 79

[BeGo92] Berry, G., Gonthier, G.: The ESTEREL Synchronous Programming Language – Design, Semantics, Implementation, Science of Computer Programming, Vol. 19, No. 2, 1992 pp. 87 –152

[Behl98] Behle, A.: An Internet-based Information System for Cooperative Software Reuse. Proceedings of the 5th International Conference on Software Reuse, June 1998, pp. 236 – 245

[Behl99] Behle, A.: Wiederverwendung von Softwarekomponenten im Internet. Dissertation RWTH Aachen D 82, June 1999

[Belk00] Belkin, N. J.: Helping People find What They Don't Know, Communications of the ACM, Vol. 43, No. 8, August 2000, pp. 58 – 61

[BeMi00] Behme, H., Mintert, S.: XML in der Praxis, München: Addison-Wesley 2000

[Bent99] Benton, W.: Interfacing Relational Databases to the Web, Linux Journal, Vol. 1999, Issue 67es, November 1999

[BHK85] Boisvert, R. F., Howe, S. E., Kahaner, D. K.: GAMS - A Framework for the Management of Scientific Software. ACM Transactions on Mathematical Software, Vol. 11, No. 4, 1985, pp. 313 – 355

[BKR96] Buxmann, P., König, W., Rose, F.: The Java Repository – An Electronic Intermediary Java Resource. Proceedings of the 7th Annual Conference of the International Information Management Association (IIMA), Colorado, USA, December 1996

[Blah01] Blaha, M.: Data Warehouses and Decision Support Systems, IEEE Computer Magazine, Vol. 34, No. 12, December 2001, pp. 38 – 39

[BMJH96] Bruckhaus, T., Madhavji, N. H., Janssen, I., Henshaw, J.: The Impact of Tools on Software Productivity, IEEE Software, Vol. 13, No. 5, September1996; pp. 29 – 38

[Bond97] Bondeli. P.: Developing Reusable Multi-Tasking Components Using Object-Oriented Techniques in Ada 95, Proceedings of the 8th International Workshop on Real-Time Ada, 1997, pp. 33 – 34

[Booc94] Booch, G.: Object Analysis and Design with Applications. The Benjamin / Cummings Publishing Company, Inc. 1994

[Börs95] Börstler, J.: Feature-Oriented Classification for Software Reuse. Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering (SEKE '95), Rockville, USA, June 1995, pp. 204 – 211

[Bour01a] Bourret, R.: XML Database Products, http://www.rpbourret.com/xml/XMLDatabaseProds.html, 2001

[Bour01b] Bourret, R.: XML and Databases, http://www.rpbourret.com/xml/XMLandDatabases.html, 2001

[BrBu00] Brereton, P., Budgen, D.: Component-Based Systems: A Classification of Issues, IEEE Computer Magazine, Vol. 33, No. 11, November 2000, pp. 54 – 62

[BRJ99] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide, Addison-Wesley, 1999

[BrSi02] Broy, M., Siedersleben, J.: Objektorientierte Programmierung und Softwareentwicklung – Eine Kritische Einschätzung, Iformatik Spektrum Zeittschrift, February 2002, pp. 3 – 11

[BrWa98] Brown, A. W., Wallnau, K. C.: The Current State of CBSE. – IEEE Software Magazine September/October 1998; pp. 37 – 46

[BSST93] Batory, D., Singhal, V., Sirkin, M., Thomas, J.: Scalable Software Libraries, Proceedings of the First ACM Symposium on Foundations of Software Engineering, 1993, pp. 191 – 199

[BUSF98] Balci, O., Ulusarac, C., Shah, P., Fox, E. A.: A Library of Reusable Model Components for Visual Simulation of the NCSTRL System, Proceedings of the 1998 Winter Simulation Conference, 1998, pp. 1451 – 1459

[CaCa01] Cafarro, M. A. G., Caffaro, M. G.: The Challenges that XML Faces, IEEE Computer Magazine, Vol. 34, No. 10, October 2001, pp. 15 – 18

[Cail95] Cailliau, R.: A Little History of the World Wide Web, http://www.w3.org/History.html, 1995

[CAP95] Castano, S., Antonellis, V., Pernici, B.: Building Reusable Components in the Public Administration Domain, Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability, 1995, pp. 81 – 87

[Carn97] Carnegie Mellon Software Engineering Institute. Software Technology Review : Component Object Model (COM), DCOM, and Related Capabilities (http://www.sei.cmu.edu/str/descriptions/com_body.html#1457065), 1997

[CaRu99] Carnahan, L., Ruark, M.: Report from the Requirements Group for Real-time Extensions for the Java™ Platform, National Institute of Standards and Technology, http://www.nist.gov/rt-java, September 1999

[CFG91] Creech, M. L., Freeze, D. F., Griss, M. L.: Using Hypertext in Selecting Reusable Software Components, 3rd Annual ACM Conference Proceedings on HYPERTEXT '91 , 1991, pp. 25 – 38

[Cham96] Chambers, C.: Towards Reusable, Extensible Components, ACM Computing Surveys 28, December 1996

[Chen93] Chen, S.: Retrieval of Reusable Components in a Deductive, Object-Oriented Database Environment, Ph.D. Thesis, RWTH Aachen, 1993

[CiRo99] Cicalese, C. D. T., Rotenstreich, S.: Behavioral Specification of Distributed Software Component Interfaces, IEEE Computer, July 1999, pp. 46 – 53

[Coul97] Coult, N.: ACM's Computing Classification System Reflects Changing Times, Communications of the ACM, Vol. 40, No. 12, December 1997

[Cruz96] Cruz, I. F.: Tailorable Information Visualization, ACM Computing Surveys 28 (4es), December 1996

[CTW98] Chávez, A., Tornabene, C., Wiederhold, G.: Software Component Licensing: A Primer, IEEE Software Magazine, September-October 1998, pp. 47 – 53

[DaDe88] Damier, C., Defude, B.: The Document Management Component of a Multimedia Data Model. ACM ISBN 0-89791-274-8, 1988, pp. 451 – 464

[DaFu95] Damiani, E., Fugini, M. G.: Automatic Thesaurus Construction Supporting Fuzzy Retrieval of Reusable Components, Proceedings of the 1995 ACM Symposium on Applied Computing, 1995, pp. 542 – 547

[Dai95] Dai, W.: Development of Reusable Expert System Components: Preliminary Experience, Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability, 1995, pp. 238 – 246

[Dail01] Dail, H.: Working Smarter – Building with J2EE Components, Java – Sun Homepage, http://java.sun.com/features/2001/06/j2eesmrt.p.html, 2001

[Daus87] Dausmann, M.: Library Structures for Reusable Components. Proceedings of the 1987 ACM SIGAda International Conference on Using Ada, 1987, pp. 226 – 235

[DBSB91] Devanbu, P., Brachmann, R. J., Selfridge, P. G., Ballard, B. W.: LaSSIE: A Knowledge-based Software Information System, CACM, Vol. 34, No. 5, May 1991, pp. 35 – 49

[DeDe99] Deitel, H. M., Deitel, P. J.: Java – How to Programme, Prentice Hall, New Jersey, 1999

[Dehn98] Dehnert, W.: Anwendungsprogrammierung mit JDBC, München: Hanser Verlag 1998

[DeJo97] Devanbu, P., Jones, M. A.: The Use of Description Logics in KBSE Systems, ACM Transactions on Software Engineering and Knowledge Engineering, Vol. 6, No. 2, April 1997, pp. 141 – 172

[DeKo79] Deliyanni, A., Kowalski, R. A.: Logic and Semantic Networks, Communications of the ACM, Vol. 22, No. 3, March 1979, pp. 184 – 192

[Dell97a] Dellarocas, C.: The SYNTHESIS Environment for Component-Based Software Development. Proceedings of the 8th IEEE International Workshop on Software Technology and Engineering Practice, 1997, pp. 434 – 443

[Dell97b] Dellarocas, C.: Towards a Design Handbook for Integrating Software Components. Proceedings of the 5th International Symposium on Assessment of Software Tools and Technologies, 1997, pp. 3 – 13

[Digr98] Digre, T.: Business Object Component Architecture, IEEE Software Magazine, September / October 1998, pp. 60 – 69

[DLS02] Digital Logic Simulator, Web-Page of the Research Systems Company, DLS Simulator, http://www.research-systems.com/easysim/, 2002

[DRC02] The Digital Resource Catalogue, http://direct.asset.com

[Doug99] Douglas, B. P.: Creating Executable Models, Software Development Magazine, September 1999

[Dujm02] Dujmovic, S.: Anwendungsentwicklung mit Komponenten-Frameworks in der Automatisierungstechnik, Ph.D. Thesis (under evaluation at the University of Stuttgart), IAS, Universität Stuttgart, 2002

[DuKn93] Dunn, M. F., Knight, J. C.: Automating the Detection of Reusable Parts in Existing Software, Proceedings of the 15th International Conference on Software Engineering, 1993, pp. 381 – 390

[Eich94] Eichmann, D.: Ethical Web Agents. Electronic Proceedings of the 2nd World Wide Web Conference, October 1994, http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings

[ElNa94] Elmasri, R. A., Navathe, S. B.: Fundamentals of Database Systems, 2nd Edition, The Benjamin / Cummings Publishing Company, 1994

[EMD94] Eichmann, D., McGregor, T., Danley, D.: Integrating Structured Databases into the Web: the MORE System. Proceedings of the First International Conference on the World Wide Web, Geneva, May 1994, pp. 369 – 378

[Eppl02] Epple, U.: Aachener Prozessleittechnik / Kommunikationssystem (http://www.plt.rwth-aachen.de/english/welcome.html), 2002.

[Ergo98] ErgoThech: Business Wire, ErgoTech Announces New Release of JavaBeans for Virtual Instrumentation http://industry.java.sun.com/javanews/ stories/print/ 0,1797,8860,00.html, 1998

[ETAS99] ETAS GmbH&Co.KG. ASCET-SD White Paper : Development Environment for Embedded Control Systems, 1999

[EWB02] Electronic WorkBench Company, Web-Page with the Electronic Hardware Simulation Products, http://www.interactiv.com/, 2002

[Fisc87] Fischer, G.: Cognitive View of Reuse and Redesign, IEEE Software, July 87, pp. 60 – 72

[FiTa00] Fielding, R. T., Taylor, R. N.: Principled Design of the Modern Web Architecture, Proceedings of the International Conference on Software Engineering ICSE 2000, Limerick, Ireland, pp. 407 – 416

[Flei00] Fleisch, W.: Simulation and Validation of Component-Based Automotive Control Software, Proc. Of the 12th European Simulation Symposium, Hamburg, Germany, September 2000

[Foli02] Foliage Software Systems,Inc. Success Stories : DCOM/CORBA http://www.foliage.com/success/index.shtml, 2002

[Frei94] Freitag, B.: A Hypertext-Based Tool for Large Scale Software Reuse, Proceedings of the 6th International Conference on Advanced Information System Engineering, LNCS 811, 1994, pp. 283 – 296

[Frei00] Freining, C.: Reengineering zur Komponentenidentifikation einer Steuerungssoftware, Diplomarbeit No. 1749, IAS, Universität Stuttgart, May 2000

[FrFo87] Frakes, W. B., Fox, C. J.: An Approach to Integrating Expert System Components into Production Software, Proceedings of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow, 1987, pp. 50 – 56

[FrFo95] Frakes, W. B., Fox, C. J.: Sixteen Questions About Software Reuse, Communications of the ACM, Vol. 38, No. 6, June 1995, pp. 75 – 87

[FrGa90] Frakes, W. B., Gandel, P. B.: Representing Reusable Software, Information and Software Technology, Vol. 32, No. 10 December 1990, pp. 653 – 664

[FrTe96] Frakes, W. B., Terry, C.: Software Reuse: Metrics and Models, ACM Computing Surveys, Vol. 28, No. 2 June 1996, pp. 415 – 435

[GaMy01] Gal, A., Mylopoulos, J.: Toward Web-Based Application Management Systems, IEEE Transactions on Knowledge and Data Engineering, Vol. 13, No. 4, July/August 2001, pp. 683 – 702

[GaSc90] Garg, P. K., Scacchi, W.: A Hypertext System to Manage Software Life-Cycle Documents, IEEE Software, Vol.7 No. 3, May 1990, pp. 90 – 98

[GHL+98] Gray, D. N., Hotchkiss, J., LaForge, S., Shalit, A., Weinberg, T.: Modern Languages and Microsoft's Component Object Model. Communication of the ACM, Vol.41, No.5, May 1998

[Göhn98] Göhner, P.: Komponentenbasierte Entwicklung von Automatisierungssystemen, VDE-GMA Kongress 1998 (VDI Berichte Nr. 1397), Ludwigsburg, 1998, S. 513-521

[GoPr01] Goldfarb, C. F., Prescod, P.: The XML Handbook, Prentice Hall PTR, 2001

[Grah96] Graham, I. S.: HTML Sourcebook, John Wiley & Sons, Inc., 1996

[Gree99] Green, R.: Component-Based Software Development: Implications for Documentation, Proceedings on the 17th Annual International Conference on Computer Documentation, 1999, pp. 159 – 164

[GuNä99] Gunzert, M., Nägele, A.: Component-Based Development and Verification of Safety Critical Software for a Brake-by-Wire System with Synchronous Software Components, Proc. International Symposium on Parallel and Distributed Systems Engineering PDSE 99, Los Angeles

[Gunz02] Gunzert, M.: Komponentenbasierte Softwareentwicklung für sicherheitskritische eingebettete Systeme, Ph.D. Thesis (under evaluation at the University of Stuttgart), IAS, Universität Stuttgart, 2002

[GWS96] Greenwood, R. M., Warboys, B. C., Sa, J.: Cooperating Evolving Components a Rigorous Approach to Evolving Large Software Systems, Proceedings of the 18th International Conference on Software Engineering, 1996, pp. 428 – 437

[Halb93] Halbwachs, N.: Synchronous Programming of Reactive Systems, Kluwer Academic Publishers, 1993

[Hall01] Hall, M.: Servlets und Java Server Pages, München: Markt+Technik Verlag 2001

[HaPn85] Harel, D., Pnueli, A.: On the Development of Reactive Systems: Logic and Models of Concurrent Systems, Proc. NATO Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, NATO ASI Series F, vol. 13, Springer-Verlag, 1985, pp. 477 – 498

[Hare87] Harel, D.: Statecharts - A visual Approach to Complex Systems, Science of Computer Programming, Vol. 8-3, 1987, pp. 231 – 275

[HBD00] Hirsh, H., Basu, C., Davison, B. D.: Learning to Personalize, Communications of the ACM, Vol. 43, No. 8, August 2000, pp. 102 – 106

[HeCo01] Heineman, G. T., Councill, W. T.: Component-Based Software Engineering Putting the Pieces Together, Addison-Wesley, 2001

[Heil99] Heilmann, H.: Skript zur VorlesungWirtschaftsinformatik, BWI, Stuttgart 1999

[HeMa91] Helm, R., Maarek, Y. S.: Integrating Information Retrieval and Domain Specific Approaches for Browsing and Retrieval in Object-Oriented Class Libraries, Proceedings of the OOPSLA 1991, pp. 47 – 61

[Henn94] Henninger, S.: Using Iterative Refinement to Find Reusable Software, IEEE Software Magazine, September 1994, pp. 48 – 59

[Henn95] Henninger, S.: Supporting the Process of Satisfying Information Needs with Reusable Software Libraries: an Empirical Study, Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability, 1995, pp. 267 – 270

[Henn96] Henninger, S.: Supporting the Construction and Evolution of Component Repositories, Proceedings of the International Conference on Software Engineering 1996, pp. 279 – 288

[HeSa00] Heuer, A., Saake, G.: Datenbanken: Konzepte und Sprachen, Bonn: MITP-Verlag 2000

[HLR92] Halbwachs, N., Lagnier, F., Ratel, C.: Programming and Verifying Real-Time Systems by means of the Synchronous Data-Flow Language LUSTRE, Transactions on Software Engineering, Vol. 18, No. 9, 1992, pp. 785 – 793

[Hoar78] Hoare, C. A. R.: Communicating Sequential Processes, CACM, Vol. 21, No. 8, August 1978, pp. 666 – 677

[HoRi99] Hoang, M., Rieger, P.: Komponentenbasierte Automatisierungssoftware: Objektorientiert – anwendungsnah, München, Hanser Verlag, 1999

[Horn01] Horn, H.: Konzeption eines dynamischen, internetfähigen Beratersystems für die Auswahl von Komponenten aus einer Bibliothek, Diplomarbeit No. 1810, IAS, Universität Stuttgart, December 2001

[HSW01] Heintz, P., Siller, A., Waldmann, P.: ASCET-SD Traffics Trade : Safe rail vehicles thanks to ETAS tools, Knorr-Bremse Systems for Rail Vehicles GmbH. Real Times, January 2001, pp. 16 – 17

[Inte02] Web-Page of the Intel Corporation, Intel Products, http://www.intel.com/products/, 2002

[ISO98] ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability, 1998

[JaMi98] Jaedicke, M., Mitschang, B.: On Parallel Processing of Aggregate and Scalar Functions in Object-Relational DBMS, Proc. of the ACM SIGMOD Int. Conference on Management of Data, Vol. 27, Issue 2, 1998, pp. 379 – 389

[JDOM00] JDOM.ORG : JDOM Homepage, www.jdom.org, 2000

[JeCh93] Jeng, J. J., Cheng, B. H. C.: Using Formal Methods to Construct a Software Component Library, Proceedings of the 4th European Software Conference, Springer-Verlag, LNCS 717, September 1993, pp. 397 – 417

[JeCh95] Jeng, J. J., Cheng, B. H. C.: Specification Matching for Software Reuse: A Foundation, ACM SIGSOFT Software Engineering Notes , Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability, Volume 20, Issue SI, August 1995, pp. 97 – 105

[JSW92] Jensen, J. S., Stewart, H. D., Whittington, P. H.: Successful Experience with AdaSAGE Reusable Component Library. Conference Proceedings on TRI-Ada '92, 1992, pp. 276 – 280

[JWS01a] Sun Microsystems Inc.: Java Web Start, http://java.sun.com/products/javawebstart/index.html, 2001

[JWS01b] Java Web Start: Support Readiness Document Java Web Start 1.0, http://java.sun.com/products/javawebstart/index.html, 2001

[JXML01] Developing XML Solutions with Java Server Pages Technology, http://java.sun.com/products/jsp/jspxml.html, 2001

[KAO91] Krutz, W. K., Allen, K., Olivier, D. P.: The Costs Related to Making Software Reusable: Experience from a Real Project, Proceedings of the Conference on Ada: Today's Accomplishments – Tomorrow's Expectations, 1991, pp. 437 – 443

[KeSc98] Keller, R. K., Schauer, R.: Design Components: Toward Software Composition at the Design Level, Proceedings of the 1998 International Conference on Software Engineering, 1998, pp. 302 – 311

[Kope97] Kopetz, H.: Real-Time Systems, Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers, 1997

[Kope00] Kopetz, H.: Software Engineering for Real-Time: A Roadmap, 22nd ICSE, The Future of Software Engineering, 2000, pp. 201 – 211

[Kotu98] Kotula, J.: Using Patterns to Create Component Documentation, IEEE Software, March/April 1998, pp. 84 – 92

[KST92] Karlsson, E. A., Sorumgard, S., Tryggeseth, E.: Classification of Object-Oriented Components for Reuse, Proceeding of the TOOLS Europe, Dortmund, 1992

[KSW92] Kiesel, N., Schürr, A., Westfechtel, B.: GRAS - A Graph-Oriented (Software) Engineering Database System. Information System, Vol. 20, No. 1, Oxford: Pergamon Press, 1995, pp. 21 – 52

[KuBr00] Kunda, D., Brooks, L.: Assessing Organizational Obstacles to Component-Based Developement: A Case Study Approach, Information and Software Technology, Vol. 42, 2000, pp. 715-725

[Kuru99] Kuruganti, I.: A Component Selection Methodology with Application to the Internet Telephony Domain, Slides of the Software Technology Center, Lucent Technologies – Bell Laboratories – Advanced Technologies, 1999

[LaCe99] Laurent, S. S., Cerami, E.: Building XML Applications, McGraw-Hill, 1999

[LaGö99] Lauber, R., Göhner, P.: Prozessautomatisierung I, Third Edition, Springer – Verlag Berlin – Heidelberg, 1999

[LaMy01] Landay, J. A., Myers, B. A.: Sketching Interfaces: Towards More Human Interface Design, IEEE Computer Magazine, Vol. 34, No. 3, March 2001, pp. 56 – 64

[Land94] Landow, G. P.: Hypertext Theory, John Hopkins University Press, 1994

[LeUr99] Lee, S. K., Urban, J. E.: SOORLS - A Software Reuse Approach on the Web. International Journal of Software Engineering and Knowledge Engineering, Vol. 9, No.3, 1999, pp. 279 – 296

[LeWi00] Leffingwell, D., Widrig, D.: Managing Software Requirements – A Unified Approach, Addison-Wesley, 2000

[LHG+96] Liu, L., Halper, M., Gu, H., Geller, J., Perl, Y.: Modeling a Vocabulary in an Object-Oriented Database, Proceedings of the 5th International Conference on Information and Knowledge Management, November 1996, pp. 179 – 188

[Loes98] Loeser, H.: Techniken für Web-basierte Datenbankanwendungen: Anforderungen, Ansätze, Architekturen, Informatik Forschung und Entwicklung, Vol. 13, 1998, pp. 196 – 216

[LoMi89] London, R. L., Milsted, K. R.: Specifying Reusable Components Using Z: Realistic Sets and Dictionaries, Proceedings of the 5th International Workshop on Software Specification and Design, 1989, pp. 120 – 127

[LoSp98] Lohse, G. L., Spiller, P.: Electronic Shopping, Communications of the ACM, Vol. 41, No. 7, July 1998, pp. 81 – 88

[LRS99] Lam, W., Ruiz, M., Srinivasan, P.: Automatic Text Categorization and Its Application to Text Retrieval, IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 6, November/December 1999, pp. 865 – 879

[Luce99] Lucena Jr, V. F.: Entwicklung einer Komponentenbibliothek mit Synchronen Softwarekomponenten für Verteilte Deterministische Echtzeitsysteme, Diplomarbeit No. 1689, Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart, 1999

[Luce00] Lucena Jr, V. F.: Towards the Use of Active Information Presentation in the Management of Software Components, Proceeding of the 45. International Wissenschaftliches Kolloquium, Ilmenau, Germany, October 2000

[Luce01a] Lucena Jr, V. F.: A Domain Specific Classification Scheme for the Management of Industrial Automation Components, ASSE 2001 International Symposium on Software Engineering, Buenos Aires, Argentina, September 2001

[Luce01b] Lucena Jr, V. F.: Facet-Based Classification Scheme for Industrial Automation Software Components, 6th International Workshop on Component-Oriented Programming at the Ecoop 2001, Budapest, Hungary, 2001

[LuFr00] Lucena Jr, V. F., Freinning, C.: Reverse Engineering Konzept zur Gewinnung von Verhaltensinformationen aus OO-Software, 2. Workshop Software-Reengineering, Bad Honnef, Germany, May 2000

[LuGu98] Lucena Jr, V. F., Gunzert, M.: Komponentenbasierte Softwareentwicklung für eingebettete Systeme mit synchronen Softwarekomponenten, TAE Kolloquium, Esslingen, Germany, May 1998

[Luqi87] Luqi, A.: Normalized Specifications for Identifying Reusable Software, Proceedings of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow, 1987, pp. 46 – 49

[Maar91] Maarek, Y. S.: Software Library Construction from IR Perspective. ACM Sigir Forum, Vol. 25, No. 2, 1991, pp. 08 – 18

[MAGM97] Mili, H., Ah-Ki, E., Godin, R., Mcheick, H.: Another Nail to the Coffin of Faceted Controlled-Vocabulary Component Classification and Retrieval, Proceedings of the 1997 Symposium on Software Reusability (SSR'97), May 1997, Boston USA, pp. 89 – 98

[MaSm89] Maarek, Y. S., Smadja, F. Z.: Full Text Indexing Based on Lexical Relations an Application: Software Libraries. Proceedings of the 12th Annual International ACMSIGIR Conference on Research and Development in Information Retrieval, 1989, pp. 198 – 206

[MBK91] Maarek, Y. S., Berry, D. M., Kaiser, G. F.: An Information Retrieval Approach for Automatically Constructing Software Libraries. IEEE Transactions on Software Engineering, Vol. 17 , No. 8, August 1991, pp. 800 – 813

[McCl95] McClure, C.: Model-Driven Software Reuse Practicing Reuse Information Engineering Style, Extended Intelligence Inc., Software Reuse Resource Center for Business, http://www.reusability.com/, 1995

[McLa00] McLaughlin, B.: Java and XML, Sebastopol: O'Reilly & Associates 2000

[MFC+00] Mili, A., Fowler, S., Gottumkkala, R., Zhang, L.: An Integrated Cost Model for Software Reuse, Proceeding of the International Conference on Software Engineering 2000, Limerick, Ireland, 2000, pp. 157 – 166

[Micr00] Microsoft Corporation. What Microsoft´s .NET Vision Means for Businesses, October 2000, http://www.microsoft.com/business/vision/netvision.asp, 2000

[Micr01a] Microsoft Corporation. XML Web Services, http://msdn.microsoft.com/netframework/, 2001

[Micr01b] Microsoft Corporation. Microsoft .NET Framework Description, http://msdn.microsoft.com/netframework/, 2001

[Micr01c] Microsoft Corporation. Microsoft .NET-Related Standards, http://msdn.microsoft.com/netframework/, 2001

[Micr01d] Microsoft Corporation. Next Generation Business Integration – The Advantages of Microsoft .NET, http://msdn.microsoft.com/netframework/, 2001

[MiNo99] Michail, A., Notkin, D.: Assessing Software Libraries by Browsing Similar Classes, Functions and Relationships, Proceedings of the 1999 International Conference on Software Engineering, 1999, pp. 463 – 472

[Mits01] Mitschang, B.: Datenbankbasierte Anwendungen, Lecture Script at the Institute of Parallel and Distributed High-Performance Systems University of Stuttgart, Summer Semester 2001

[MLP+01] Morris, J., Lee, G., Parker, K., Bundell, G. A., Lam, C. P.: Software Component Certification, IEEE Computer Magazine, Vol. 34, No. 9, September 2001, pp. 30 – 36

[MMM94] Mili, A., Mili, R., Mittermeir, R.: Storing and Retrieving Software Components a Refinement Based System, Proceedings of the 16th International Conference on Software Engineering, 1994, pp. 91 – 100

[MMM95] Mili, H., Mili, F., Mili, A.: Reusing Software: Issues and Research Directions, IEEE Transactions on Software Engineering, Vol. 21 No.6, June 1995

[MMS98] Meyers, B., Mingis, C., Schmidt, H.: Providing Trusted Components to the Industry, IEEE Computer, Vol. 31, No.5, May 1998, pp. 104 – 105

[MoBa91] Moore, J. M., Bailin, S. C.: Domain Analysis: Framework for Reuse, Domain Analysis and Software Systems Modeling, IEEE Computer Society Press Tutorial, 1991, pp. 179 – 203

[MoGa91] Moineau, T., Gandel, M. C.: Software Reusability through Formal Specifications, Proceedings of the First International Workshop on Software Reusability, Dortmund, Germany, July 1991, pp. 202 – 212

[MoWi95a] Moormann, Z. A., Wing, J. M.: Signature Matching - A Tool for Using Software Libraries. ACM Transactions on Software Engineering and Methodology, Vol. 4, No. 2, April 1995, pp. 146 – 170

[MoWi95b] Moormann, Z. A., Wing, J. M.: Specification Matching of Software Components. ACM Software Engineering Notes, Vol. 20, No. 4, October 1995, pp. 6 – 17

[NaTa01] Nakajima, S., Tamai, T.: Behavior Analysis of the Enterprise JavaBeans™ Component Architecture, SPIN 2001, LNCS 2057, 2001, pp. 163 – 182

[Nie92] Nie, J. Y.: Towards a Probabilistic Modal Logic for Semantic-Based Information Retrieval, Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, June 1992, pp. 140 – 151

[Ning97] Ning, J. Q.: ADE – An Architecture Design Environment for Component-Based Software Engineering, Proceeding of the International Conference on Software Engineering 1997, Boston, USA, 1997, pp. 614 – 615

[NPV93] Nie, J. Y., Paradis, F., Vaucher, J.: Adjusting the Performance of an Information Retrieval System, Proceedings of the Second International Conference on Information and Knowledge Management, December 1993, pp. 726 – 728

[O'De89] O'Dell, J. A.: Oh Where, Oh Where has my Software gone...New Techniques for Managing the Software Library, ACM SIGUCCS - XVII Conference on User Services, 1989, pp. 363 – 366

[OGH88] Owen, G. S., Gagliano, R., Honkanen, P.: Tools for the Storage and Retrieval of Reusable MIS Software in Ada. Proceedings of the 1988 ACM 6th Annual Conference on Computer Science, 1988, pp. 535 – 539

[OHPB92] Ostertag, E., Hendler, J., Prieto-Díaz, R., Braun, C.: Computing Similarity in a Reuse Library System: An AI-Based Approach, ACM Transactions on Software Engineering and Methodology, Vol. 1, No. 3, July 1992, pp. 205 – 228

[OLKM00] van Ommering, R., van der Linden, F., Kramer, J., Magee, J.: The Koala Component Model for Consumer Electronics Software. IEEE Computer, March 2000, pp.78 – 85

[OMG98] OMG: Real-Time CORBA, Object Management Group, www.omg.org, 1998

[Omme00] van Ommering, R.: A Composable Software Architecture for Consumer Electronics Products, Xootic Magazine, March 2001, pp. 37 – 47

[Omme01] van Ommering, R.: Techniques for Independent Deployment to Build Product Populations, Proceedings of the Conference on Software Architecture, Working Group IEEE/IFIP 2001, pp. 55 – 64

[Paiv02] Paiva, D. A.: Development of Dynamic Web-Pages for the Presentation of Software Components Technical Information, Studienarbeit No. 1822, IAS, Universität Stuttgart, Januar 2002

[Paul01] Paula, L. G. S.: Development of a Database Interface Using Active Server Pages, Studienarbeit No. 1818, IAS, Universität Stuttgart, December 2001

[Pedn00] Pednault, E. P. D.: Representing is Everything, Communications of the ACM, Vol. 43, No. 8, August 2000, pp. 80 – 83

[PhAr01] Pharoah, A., Arni, F.: Creating Commercial Components Enterprise JavaBeans™ Technology, Technical White Papers, www.componentsource.com, 2001

[PhBr00] Pharoah, A., Arni, F.: Creating Commercial Components Microsoft® COM and .NET Framework, Technical White Papers, www.componentsource.com, 2000

[PhBr01] Pharoah, A., Arni, F.: Creating Commercial Components Microsoft® .NET Framework, Technical White Papers, www.componentsource.com, 2001

[Phil02] Web-Page of the Philips Corporation, Philips Semiconductors Division, http://www.semiconductors.philips.com/products/, 2002

[PLT99] PLT – Lehrstuhl für Prozessleittechnik: ACPLT/OV Technologie-Papier No. 3: Spezifikation der Modellierungssprache, Version 1.0.0 , August 1999

[PoPi93] Podgurski, A., Pierce, L.: Retrieving Software by Sampling Behavior, ACM Transactions on Software Engineering and Methodology, Vol. 2, No. 3, July 1993, pp. 286 – 303

[Poul94] Poulin, J. S.: Balancing the Need for Large Corporate and Small Domain-Specific Reuse Libraries, Proceedings of the 1994 ACM Symposium on Applied Computing, April 1994, pp. 88 – 93

[Poul96] Poulin, J. S.: Evolution of a Software Architecture for Management Information Systems, Workshop of the SIGSOFT 1996, San Francisco, USA, 1996, pp. 134 – 137

[PoWe94] Poulin, J. S., Werkman, K. J.: Software Reuse Libraries with Mosaic. Electronic Proceedings of the 2nd World Wide Web Conference, October 1994

[PoWe95] Poulin, J. S., Werkman, K. J.: Melding Structure Abstracts and the WWW for Retrieval of Reusable Components. ACM Sigsoft Software Engineering Notes, August 1995, pp. 160 – 168

[Prie91] Prieto-Díaz, R.: Implementing Faceted Classification for Software Reuse, Communication of the ACM, Vol. 34, No. 5, May 1991

[PrFr87] Prieto-Díaz, R., Freeman, P.: Classifying Software for Reusability, IEEE Software, January 1987, pp. 6 – 16

[PrWu96] Pritschow, G., Wurst, K.-H.: Modular Robots for Flexible Assembly. Proc. of the 28th CIRP International Seminar on Manufacturing Systems, Johannesburg, 1996, pp. 153 – 158

[PSB00] Pharoah, A., Siegel, J., Brooke, C.: Creating Commercial Components CORBA™ Component Model, Technical White Papers, www.componentsource.com, 2000

[PTH97] Pritschow, G., Tran, T. L., Hohenadel, J.: Standalone PC-Controller on an Open Platform. Proc. of the 30th International Symposium on Automotive Technology & Automation, Florence, 1997

[Radd98] Radding, A.: Hidden Costs of Code Reuse, Information Week Magazine, November 1998, http://www.informationweek.com/708/08iuhid.htm

[Rang57] Ranganathan, S. R.: Prolegomena to Library Classification, The Garden City Press, Letchworth, Hertfordshire, 1957

[RFPG96] Rice, J., Farquhar, A., Piernot, P., Gruber, T.: Using the Web Instead of a Window System, Proceedings of the Conference on Human Factors in Computing Systems, April 1996, pp. 103 – 110

[RiHo99] Rieger, P., Hoang, M. S.: Komponentenbasierte Automatisierungssoftware. Hanser, München, 1999

[RMI00] Support Readiness Document Java 2 Standard Edition, Remote Method Invocation, http://java.sun.com/, 2000

[RNJ99] Rine, D., Nada, N., Jaber, K.: Using Adapters to Reduce Interaction Complexity in Reusable Component-Based Software Development, Proceedings of the 5th Symposium on Software Reusability, 1999, pp. 37 – 43

[Robi79] Robinson, G.: Universal Decimal Classification: A Brief Introduction, Technical Report, International Federation of Documentation, 1979

[Royc89] Royce, W.: Reliable, Reusable Ada Components for Constructing Large, Distributed Multi-Task Networks: Networks Architecture Services (NAS). Conference Proceedings on Ada Technology in Context: Application, Development, and Deployment, 1989, pp. 500 – 516

[RRW00] Rao, D. M., Radhakrishnan, R., Wilsey, P. A.: Web-Based Network Analysis and Design, ACM Transactions on Modeling and Computer Simulation, Vol. 10, No. 1, January 2000, pp. 18 – 38

[SaMc83] Salton, G., McGill. M.: Introduction to Modern Information Retrieval, New York, McGraw-Hill, 1983

[Sate95] Sateesh, T. K.: Conceptual Model of Real-Time Systems: A Perspective, Proceedings of the 1995 ACM Symposium on Applied Computing February 1995, pp. 206 – 209

[SBP99] Smith, I. N. C., Ballou, D. P., Pazer, H. L.: The Impact of Data Quality Information on Decision Making: An Exploratory Analysis, IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 6, November-December 1999, pp. 853 – 862

[Schu99] Schulz, K.: Java professionell programmieren, Berlin: Springer Verlag 1999

[Seli99] Selic, B.: Turning Clockwise: Using UML in the Real-Time Domain, Communications of the ACM, Vol. 42, No. 10, October 1999, pp. 46 – 54

[Seli01] Selic, B.: Specification and Modeling: An Industrial Perspective, Proc. of Int. the Conf. On Software Engineering 2001, ICSE 2001, pp. 676 – 677

[SeMi98] Selletin, J., Mitschang, B.: Data-Intensive Intra- & Internet Applications – Experiences Using Java and CORBA in the World Wide Web, Proc. of the 14th Int. Conference on Data Engineering, 1998, pp. 302 – 311

[SeMi99] Selletin, J., Mitschang, B.: Design and Implementation of a CORBA Query Service Accessing EXPRESS-Based Data, Proc. of the Int. Conference on Database Systems and Advanced Applications, 1999, pp. 273 – 282

[Serh01] Serhan, B.: Konzeption und Implementierung einer internetfähigen Informationsdatenbank für Komponenten in der Automatisierungstechnik, Diplomarbeit No. 1807, IAS, Universität Stuttgart, August 2001

[SeRo01] Seligman, L., Rosenthal, A.: XML's Impact on Databases and Data Sharing, IEEE Computer Magazine, Vol. 34, No. 6, June 2001, pp. 59 – 67

[ShSh99] Shah, K., Sheth, A.: InfoHarness: Managing Distributed, Heterogeneous Information, IEEE Internet Computing Magazine, November/December 1999; pp. 18 – 28

[SHW98] Seacord, R. C., Hissam, S. A., Wallnau, K. C.: AGORA - A Search Engine for Software Components. IEEE Internet Computing Magazine, November-December 1998, pp. 62 – 70

[SKS93] Sindre, G., Karlson, E. A., Staalhane, T.: A Method for Software Reuse through Large Component Libraries, Proceedings of the 5th International Conference on Computing and Information, 1993. pp. 464 – 468

[Soda97] Sodalia SpA: SALMS v5.1 - A System for Classifying, Describing, and Querying About Reusable Software Assets Produced Throughout the Software Life-Cycle. March 1997, http://www.gipsy.lii.unitn.it/tarsal/sodalia/flyer.html

[SPH98] Smith, R. K., Parrish, A., Hale, J.: Cost Estimation for Component Based Software Development, Proceedings of the 36th Annual Conference on Southeast Regional Conference, 1998, pp. 323 – 325

[Spiv92] Spivey, J.: The Z Notation – A Reference Manual, Second Edition, Prentice Hall, 1992

[SSS93] Sorumgard, L. S., Sindre, G., Stokke, F.: Experiences from Application of a Faceted Classification Scheme, 2nd International Workshop on Software Reusability, Lucca, Italy, 1993

[Stal98] Stal, M.: COMmunication Everywhere – Microsoft DCOM im Überblick: Objekt Spektrum Magazine, January, 1998, pp. 78 – 87

[Stef95] Stefik, M.: Introduction to Knowledge Systems, Morgan Kaufmann Publishers, 1995

[Ster98] Sterling Software Application Management Group: The CBD96 Standard Version 2.1, Standards for Specifying and Delivering Software Components Using COOL:Gen, Part number 261639-004, July 1998

[SuBa97] Succi, G., Baruchelli, F.: The Cost of Standardizing Components for Software Reuse, Standard View Magazine, Vol. 5, No. 2, June 1997, pp. 61 – 65

[Sugi95] Sugiyama, Y.: Object Make: A Tool for Constructing Software Systems from Existing Software Components. Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability. 1995, pp. 128 –136

[SWM01] Schwarz, H., Waner, R., Mitschang, B.: Improving the Processing of Decision Support Queries: The Case for a DSS Optimizer, Proc. of the Int. Symposium on Database Engineering and Applications, 2001, pp. 177 – 186

[SwSa92] Swanson, J. E., Samadzadeh, M. H.: A Reusable Software Catalogue Interface. Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing, Vol. II: Technological Challenges of the 1990's, 1992, pp. 1076 – 1082

[Szyp98] Szyperski, C.: Component Software – Beyond Object-Oriented Programming, Addison-Wesley, 1998

[TaWo00] Tabatt, P., Wolf, H.: Java programmieren mit JBuilder, Frankfurt: Software & Support Verlag 2000

[Trac97] Tracz, W.: Developing Reusable Java Components. Proceedings of the 1997 Symposium on Symposium on Software Reusability, 1997, pp. 100–103

[Trau98] Trauter, R.: Design-Related Reuse Problems - An Experience Report, Proceedings of the Fifth International Conference on Software Reuse, 1998, pp. 176 –183

[Tura00] Turau, V.: Java Server Pages, Heidelberg: dpunkt.verlag 2000

[Turn93] Turner, D.: Using Formal Description Techniques – An Introduction to Estelle, LOTOS and SDL, Wiley, 1993

[Udel94] Udel, J.: ComponentWare. BYTE Magazine. Number 19, May 1994, pp. 46–56

[Yu97] Yu, H.: Using Object-Oriented Techniques to Develop Reusable Components. Proceedings of the conference on TRI-Ada '97, 1997, pp. 117 – 124

[W3C] World Wide Web Consortium: Leading the Web to its Full Potential. http://www.w3.org/

[W3C01] XML Schema Parts 0, 1 and 2: Primer W3C Recommendation, 2 May 2001 http://www.w3.org/TR/xmlschema-0/ (/xmlschema-0/, /xmlschema-0/)

[WaWa98] Watson, I., Watson, H.: Case-based Content Navigation, Knowledge-based Systems Magazine, Vol. 11, 1998, pp. 345 – 353

[Weyu98] Weyuker, E. J.: Testing Component-Based Software: A Cautionary Tale, IEEE Software Magazine September/October 1998; pp. 54 – 59

[Wieg98] Wiegers, K. E.: Read my Lips: No New Models!, IEEE Software Magazine, September/October 1998, pp. 10 – 13

[WiKo99] Wilhelms, G., Kopp, M.: Java professionell, Bonn: MITP-Verlag 1999

[Wöhe96] Wöhe, G.: Einführung in die Allgemeine Betriebswirtschaftslehre, München: Verlag Vahlen 1996

[ZaWi95a] Zaremski, A. M., Wing, J. M.: Signature Matching: A Tool for Using Software Libraries, ACM Transactions on Software Engineering and Methodology, Vol. 4, No. 2, April 1995, pp. 146 – 170

[ZaWi95b] Zaremski, A. M., Wing, J. M.: Specification Matching of Software Components. Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, 1995, pp. 6 – 17

[ZaWi97] Zaremski, A. M., Wing, J. M.: Specification Matching of Software Components, ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 4, October 1997, pp. 333 – 369

[Zona99] Zona Research, Inc.: Enterprise JavaBeans Technology : A business benefits analysis (htttp://www.javasoft.com/products/ejb/pdf/zona.pdf), June 1999

[ZSN01] Zari, M., Saiedian, H., Naeem, M.: Understanding and Reducing Web Delays, IEEE Computer Magazine, Vol. 34, No. 12, December 2001, pp. 30 – 37

Lebenslauf

Persönliche Daten

18. 10. 1965	geboren in São Paulo - Brasilien

Schulbildung

Mär. 1972 – Dez. 1974	Grundschule „Colégio Estadual do Jardim Nordeste“ – São Paulo
Mär. 1975 – Dez. 1979	Grundschule „Márcio Nery“ - Manaus
Mär. 1980 – Dez. 1982	Gymnasium und Berufsausbildung bei der „Technischen Bundesschule von Amazonas“ Manaus - Abschluss als Elektrotechniker

Studium

Mär. 1983 – Aug. 1987	Studium der Elektrotechnik an der Bundesuniversität von Amazonas – Manaus Praktikum bei „Philips“, Abteilung für Produktentwicklung
Aug. 1987	Abschluss Diplom-Ingenieur

Weiterbildung

Aug. 1990 – Feb. 1991	Spezialisierung in Automatisierungstechnik – FUCAPI / Bundesuniversität von Paraíba – Manaus
Mär. 1991 – Aug. 1993	Magisterprogramm – Bundesuniversität von Paraíba, Campina Grande – Forschungsgebiet der Informationsverarbeitung. Abschluss „Master of Science – MSc“ im Rahmen der Automatisierungstechnik

Berufstätigkeit

Mär. 1987 – Feb. 1988	Mitarbeiter als Diplom-Ingenieur bei „Philips“
Feb. 1988 – Mär. 1991	Mitarbeiter als Diplom-Ingenieur bei „Eletronorte“
Seit Dez. 1989	Dozent für Elektrotechnik bei der „Centro Federal de Ensino Tecnológico do Amazonas – CEFET-Am“
Seit Jan. 1991	Dozent für Elektrotechnik an der Bundesuniversität von Amazonas auf dem Gebiet der Digitalen Elektronik und Regelungstechnik
Apr. 1998 – Juli 2002	Wissenschaftlicher Mitarbeiter am Institut für Automatisierungs- und Softwaretechnik der Universität Stuttgart

SHAKER
VERLAG

ISBN 3-8322-1011-3