

W1011

Knowledge-Based Product Support Systems

A Thesis submitted
to Cardiff University
for the degree of
Doctor of Philosophy

by

Nikolaos Lagos BEng MSc

Cardiff School of Engineering,
Cardiff University,
United Kingdom

2007

UMI Number: U584930

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI U584930

Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

SUMMARY OF THESIS

This research helps bridge the gap between conventional product support, where the support system is considered as a stand-alone application, and the new paradigm of responsive one, where the support system frequently communicates with its environment and reacts to stimuli. This new paradigm would enable product support knowledge to be captured, stored, processed, and updated automatically, being delivered to the users when, where and in the form they need it.

The research reported in this thesis first defines Product Support Systems (PRSSs) as electronic means that provide accurate and up-to-date information to the user in a coherent and personalised manner. Product support knowledge is then identified as the integration of product, task, user, and support documentation knowledge.

Next, the thesis focuses on an ontology-based model of the structure, relations, and attributes of product support knowledge. In that model product support virtual documentation (PSVD) is presented as an aggregation of Information Objects (IOs) and Information Object Clusters (IOCs). The description of PSVD is followed by an analysis of the relation between IOs, IOCs, and domain knowledge.

Then, the thesis builds on the ontology-based representation of product support knowledge and explores the synergy between product support, problem solving, and knowledge engineering. As a result, a structured problem solving approach is introduced that combines case-based adaptation and model-based generation

techniques. Based on that approach a knowledge engineering framework for product support systems is developed.

A conceptual model of context-aware product support systems that extends the framework is then introduced. The conceptual model includes an ontology-based representation of knowledge related to the users, their activities, the support environment, and the device being used. An approach to semi-automatically integrating design and documentation data is also proposed as part of context-aware product support systems development process.

Acknowledgements

I would like to thank my supervisor Dr. R. M. Setchi for her invaluable guidance and support throughout my work.

I would also like to thank Prof. D. T. Pham for giving me this opportunity to conduct research in his laboratory.


All my friends for being there when I needed them and my girlfriend for tolerating my long absences.

All members of the Intelligent Information Systems group which helped with their feedback and support throughout these years.

Finally, my deepest gratitude is to my family who has given continuous support and encouragement to me.

DECLARATION


This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed  (N. Lagos - Candidate)

Date 29/05/2007

STATEMENT 1

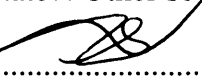
This thesis is being submitted in partial fulfilment of the requirements for the degree of PhD.

Signed  (N. Lagos - Candidate)

Date 29/05/2007

STATEMENT 2

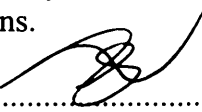
This thesis is the result of my own independent work/investigations, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed  (N. Lagos - Candidate)

Date 29/05/2007

STATEMENT 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed  (N. Lagos - Candidate)

Date 29/05/2007

Table of Contents

Chapter 1 – Introduction		
<hr/>		
1	Introduction	1
1.1	Motivation	1
1.2	Aims and Objectives	2
1.3	Outline of the Thesis	3
<hr/>		
Chapter 2 – Review of Product Support Systems, Semantic Technologies and Electronic Documentation Models		
<hr/>		
2	Review of Product Support Systems and Electronic Documentation Models	7
2.1	Product Support	7
2.1.1	Interactive Electronic Technical Manuals	7
2.1.2	Electronic Performance Support Systems	10
2.1.3	Intelligent Product Manuals	12
2.1.4	Maintenance Systems	13
2.1.5	Discussion	14
2.2	Knowledge Engineering in Product Support Systems	14
2.2.1	Knowledge Engineering vs. Knowledge Management in Product Support	14
2.2.2	Semantics-Based Technologies	16
2.2.2.1	Semantic Web	16
2.2.2.2	Ontologies	16
2.2.3	Knowledge Representation in Product Support Systems	18
2.2.4	Problem Solving and Reasoning Techniques in Product Support	19
2.2.5	Discussion	21
2.3	Content Models and Metadata in Electronic Documentation	22
2.3.1	Content Models and Components	22
2.3.1.1	Information Object	22
2.3.1.2	Information Block	23
2.3.1.3	Darwin Information Typing Architecture	24
2.3.1.4	Learning Object	25
2.3.1.5	Sharable Content Object Reference Model	26
2.3.1.6	CISCO Reusable Learning Object/Reusable Information Object	26
2.3.1.7	Lernativity Model	27
2.3.1.8	ALOCOM	27
2.3.2	Metadata	28
2.3.2.1	IEEE Learning Object Metadata	28
2.3.2.2	Dublin Core	30
2.3.3	Discussion	30
<hr/>		
Chapter 3 – Product Support Knowledge		
<hr/>		
3	Product Support Knowledge	32
3.1	Analysis of Product Support	32
3.2	Product Support Systems	36

3.3 Definition of Product Support Knowledge	38
3.3.1 Product Support Knowledge Elements	38
3.3.2 Definition of Knowledge	39
3.3.3 Product Knowledge	42
3.3.4 Task Knowledge	44
3.3.5 User Knowledge	45
3.3.6 Product Support Virtual Document	46
3.4 Summary and Conclusions	47

Chapter 4 – Semantic Modelling of Product Support Knowledge

4 Semantic Modelling of Product Support Knowledge	50
4.1 Scope	50
4.2 Product Support Knowledge Model	51
4.2.1 Architectural Model	51
4.2.2 Functional Model	54
4.3 Product Support Electronic Documentation	56
4.3.1 Product Support Electronic Documentation Components	56
4.3.1.1 Information Object	57
4.3.1.2 Information Object Cluster	59
4.3.1.3 Product Support Virtual Document	60
4.3.2 Incorporating Product Support Documentation and Knowledge	62
4.3.3 Semantics of the Product Support Documentation Model	64
4.3.3.1 Abstraction (Generalisation/Specialisation)	66
4.3.3.2 Connector	68
4.3.3.3 Arc	71
4.3.3.4 Knowledge-Specifier	71
4.4 Summary and Conclusions	73

Chapter 5 – A Knowledge Engineering Framework for Product Support Systems

5. A Knowledge Engineering Framework for Product Support Systems	75
5.1 A Problem Solving Perspective on Product Support Systems	75
5.1.1 Problem Solving and Product Support	75
5.1.2 Product Support Problems	78
5.1.2.1 Product Support Problem Definition	78
5.1.2.2 Problem Classification	80
5.1.3 Problem Solving Approach	81
5.2 A Knowledge-Level View of a Framework for Product Support Systems	85
5.2.1 General Framework Structure and Operation	85
5.2.2 Data Space	88
5.2.3 Problem Space	90
5.2.3.1 Representing Product Support Problems with Cases	90
5.2.3.2 Integrating Case-Based Product Support Problem Descriptions and Ontologies	93
5.2.4 The Hypothesis Space	96
5.2.4.1 Case Retrieval	98
5.2.4.2 Case-Based Adaptation	99
5.2.4.3 Model-Based Generation	102
5.2.5 The Solution Space	107

5.3 Summary and Conclusions	109
-----------------------------	-----

Chapter 6 – Context-Aware Product Support Systems

6. Context-Aware Product Support Systems	110
6.1 Conceptual Analysis of a Context-Aware Product Support System	110
6.1.1 Scope	110
6.1.2 Definition of a Context-Aware Product Support System	111
6.1.3 Conceptual Model of a Context-Aware PRSS	112
6.2 Modelling Context	115
6.2.1 General Ontology of Context for Product Support Systems	115
6.2.2 Activity Context	118
6.2.3 User Context	121
6.2.4 Context and Ontology-Based Representation of Knowledge in Context-Aware Product Support Systems	123
6.3 Adaptation Approach	126
6.3.1 Correlation Between Context, Tasks, and Product Support Documentation	126
6.3.2 Adaptation Method	128
6.3.2.1 Qualifications Assessment	128
6.3.2.2 Content Adaptation	130
6.4 Responding to Internal Stimuli-Integrating Product Design and Support Documentation Data	132
6.4.1 Product Data	132
6.4.2 Required Resources	134
6.4.3 Integration Approach	135
6.4.3.1 Overall Approach	135
6.4.3.2 Integration of Product Design Data with the Knowledge Base	137
6.4.3.3 Integration of Product Design Data with the Case Base	140
6.4.3.4 Supporting Facilities	144
6.5 Summary and Conclusions	144

Chapter 7 – Verification and Validation

7. Verification and Validation	146
7.1 Semantic Product Support Environment: PROSON	146
7.1.1 PROSON Environment	146
7.1.1.1 PROSON Technical Specification	146
7.1.1.2 PROSON Enabling Technologies and Tools	148
7.1.2 PROSON Development	151
7.1.2.1 Architectural Model Development	152
7.1.2.2 Functional Model Development	154
7.1.2.3 Product Support Documentation Development	156
7.2 Knowledge-Based Product Support System: PROGNOSIS	159
7.2.1 PROGNOSIS Environment	159
7.2.1.1 PROGNOSIS Technical Specification	159
7.2.1.2 PROGNOSIS Enabling Technologies and Tools	161
7.2.2 PROGNOSIS System Architecture	161
7.2.2.1 PROGNOSIS Data Space	164
7.2.2.2 PROGNOSIS Problem Space	164

7.2.2.3 PROGNOSIS Hypothesis Space	167
7.2.2.4 PROGNOSIS Solution Space	171
7.3 Context-Aware PROGNOSIS	171
7.3.1 Environment and Enabling Technologies	171
7.3.2 Context-Based Adaptation in PROGNOSIS	172
7.3.2.1 Ontology-Based Context Model	172
7.3.2.2 Adaptive Information Delivery	174
7.3.3 Responding to Internal Stimuli in PROGNOSIS	179
7.4 Conclusions	182

Chapter 8 – Contributions, Conclusions, Limitations and Future Work

8. Contributions, Conclusions, Limitations and Future Work	184
8.1 Contributions	184
8.2 Conclusions	187
8.3 Limitations	188
8.4 Future Work	189

Appendices

Appendix A – Examples of Instances of the PROSON-Based KB	191
Appendix B – Examples of Classes and Slots of the PROSON-Based KB	194
Appendix C – Example of the CLIPS Code Describing Protégé Interface	197
Appendix D – Examples of IOCs, IOs, and PSVD	198
Appendix E – The Knowledge Base	204
Appendix F – LoginHandler	207
Appendix G – ApplicationController	209
Appendix H – KnowledgeBaseWrapper	213
Appendix I – Integrated View of PROSON and Case Base	216
Appendix J – Example Cases	217
Appendix K – Other Scenarios	219
Appendix L – The Case Creation and Validation Tool	223
Appendix M – User Attributes' Values and Weights	224
Appendix N – Context Ontology (CLIPS)	226
Appendix O – Learning and Performing Activities in Traditional Systems and in Context-Aware Product Support	230
Appendix P – Content Adaptation Scenarios	231
Appendix Q – CAD Based Integration Scenarios	233
Appendix R – Part of Driving Axle Design Data in STEP	236
Appendix S – Three BOM Tested Files	238
Appendix T – Supporting Facilities for the Integration Approach	242
Appendix U – Part of Parsing Algorithm	244
Appendix V – Data Files Examples	248

References

References	251
------------	-----

List of Figures

Figure 1.1 <i>Outline of the thesis</i>	4
Figure 3.1 <i>The need for product support</i>	35
Figure 3.2 <i>Product support knowledge</i>	40
Figure 4.1 <i>A fragment of the developed knowledge base</i>	53
Figure 4.2 <i>Model of virtual documentation components and their relations</i>	58
Figure 4.3 <i>Information object cluster's model</i>	61
Figure 4.4 <i>Structure of a product support virtual document</i>	63
Figure 5.1 <i>Association between problem solving and types of problems</i>	77
Figure 5.2 <i>Solving approach for different kinds of product support problems</i>	84
Figure 5.3 <i>Knowledge-engineering framework for product support systems (knowledge-level view)</i>	86
Figure 5.4 <i>Algorithm describing the operation of the framework</i>	89
Figure 5.5 <i>Associations between ontology elements, case-based product support problem constituents, and documentation components</i>	97
Figure 5.6 <i>Specialised adjustment heuristics for parameter adjustment</i>	101
Figure 5.7 <i>Conditional statements for "No_Disks" descriptor</i>	101
Figure 5.8 <i>Example of the representation of problems and solutions</i>	103
Figure 5.9 <i>Difference in structure of a product support virtual document using case-based reasoning and model-based reasoning</i>	108
Figure 6.1 <i>The concept of context-aware PRSSs</i>	113
Figure 6.2 <i>Part of the context ontology</i>	117
Figure 6.3 <i>Relation between the context (activity model) and domain knowledge (task model)</i>	120
Figure 6.4 <i>User context concept</i>	122
Figure 6.5 <i>Part of the ontology for context-aware product support systems</i>	124

Figure 6.6 <i>Model of context-aware virtual documents and their relations</i>	124
Figure 6.7 <i>Correlation between task, context and product support documentation and comparison with adaptive systems</i>	127
Figure 6.8 <i>Overall approach</i>	136
Figure 6.9 <i>Integration with domain knowledge</i>	138
Figure 6.10 <i>Transforming a STEP file into an ontology-based product structure</i>	139
Figure 6.11 <i>Integration with case knowledge</i>	142
Figure 6.12 <i>Utilising a data file to create a new case</i>	143
Figure 7.1 <i>CLIPS code that describes small part of the architectural model for “Clutch” assembly</i>	153
Figure 7.2 <i>CLIPS code that describes small part of the functional model for “Clutch” assembly</i>	155
Figure 7.3 <i>Product support documentation model</i>	157
Figure 7.4 <i>Product support virtual documentation creation using PROSON</i>	158
Figure 7.5 <i>PROGNOSIS system architecture</i>	162
Figure 7.6 <i>Identifying goals by system usage</i>	165
Figure 7.7 <i>Connection between concepts in PROSON knowledge base and concept-level features in PROGNOSIS</i>	166
Figure 7.8 <i>Case-based adaptation</i>	168
Figure 7.9 <i>Model-based generation</i>	170
Figure 7.10 <i>Part of the context ontology in PROGNOSIS support problem constituents, and documentation components</i>	173
Figure 7.11 <i>Case 1: An inexperienced user designs a clutch (perform activity)</i>	176
Figure 7.12 <i>Case 2: An experienced user designs a clutch (perform activity)</i>	176
Figure 7.13 <i>Case 3: An inexperienced user requests more information on clutch design (learn activity)</i>	177
Figure 7.14 <i>Case 4: An experienced user requests more information on clutch design (learn activity)</i>	177

Figure 7.15. <i>Case creation in PROGNOSIS using CAD related files</i>	180
Figure E. 1 <i>The product ontology with the clutch concept selected</i>	204
Figure E. 2 <i>Part of the task ontology (actions), knowledge-specifiers (with ComplexProduct concept highlighted) and documentation ontology</i>	204
Figure E. 3 <i>Visualisation of a small part of the knowledge base (including relations and instances)</i>	205
Figure E. 4 <i>Visualisation of a part of the knowledge base subclasses (including relations)</i>	205
Figure E. 5 <i>Visualisation of the knowledge base as a tree (only is-a and instance-of relations are included)</i>	206
Figure I. 1 <i>Integrated knowledge base of PROGNOSIS</i>	216
Figure K. 1 <i>Double disk clutch</i>	219
Figure K. 2 <i>Six-disk clutch</i>	219
Figure K. 3 <i>Clutch for BMW_Z4 (Normal-racing version)</i>	220
Figure K. 4 <i>Clutch for BMW_Alpine (Lightweight version)</i>	220
Figure K. 5 <i>Inspecting a transaxle (based on behaviour model)</i>	221
Figure K. 6 <i>Concept that does not exist in the knowledge base but in a case (assembly that does not exist but the IOC for assemblies is precomposed)</i>	221
Figure K. 7 <i>Concept that does not exist in the knowledge base but in a case (assembly that does not exist but the IOC for assemblies is NOT precomposed)</i>	222
Figure L. 1 <i>Selecting the case creation and validation tool by clicking on “Add case” or “Adapt case”. “Add case” permits the modification of static and dynamic features, while the “Adapt case” only dynamic features</i>	223
Figure L. 2 <i>The values of the new case are based on previous cases</i>	223
Figure P. 1 <i>Describing a clutch for an experienced user (concise descriptions, still images, and facts compose the document in this scenario)</i>	231
Figure P. 2 <i>Describing a clutch for an inexperienced user (detailed descriptions, animations, and clarifications such as rules-of-thumb form the document)</i>	231
Figure P. 3 <i>Installing a clutch for an experienced-receptive user (a lot of text that includes warnings and facts)</i>	232

Figure P. 4 <i>Installing a clutch for an inexperienced-unreceptive user (big clear pictures including clarification types)</i>	232
Figure Q. 1 <i>Select the case</i>	233
Figure Q. 2 <i>Select the file and operation</i>	233
Figure Q. 3 <i>Validate the product structure</i>	234
Figure Q. 4 <i>Validate the values of the new case's features</i>	234
Figure Q. 5 <i>Generated product support document based on CAD data</i>	235
Figure T. 1 <i>Supporting facilities for the integration process</i>	243

List of Tables

Table 3.1 <i>Definitions related to product support knowledge</i>	48
Table 4.1 <i>Documentation and knowledge base elements</i>	65
Table 4.2 <i>Effects on availability, level of detail, type, and theme of IOCs and IOs from the semantic analysis of knowledge base constructs</i>	72
Table 5.1 <i>Structure of a case-based product support problem</i>	94
Table 5.2 <i>Content of a case-based product support problem</i>	94
Table 6.1. <i>Documentation element attributes</i>	131
Table 6.2 <i>Documentation element attributes for different contexts</i>	131
Table 7.1. <i>Clutch subassemblies and attributes considered in the case study</i>	147
Table 7.2 <i>Technical specification of the developed system components</i>	149
Table 7.3 <i>Technical specification of PROGNOSIS system components</i>	160
Table 7.4 <i>Symbols used in case study</i>	175
Table M.1 <i>Characteristics, their attributes, and values (weights)</i>	224
Table O.1 <i>Differences between performance support, e-Learning, and context-aware product support</i>	230

Abbreviations

AM	Activity Model
BOM	Bill Of Materials
CALS	Continuous Acquisition and Life Cycle Support
CBR	Case-Based Reasoning
CF	Content Fragment
CO	Content Object
CON	Context
DF	Data File
DITA	Darwin Information Typing Architecture
DoD	Department of Defence
EM	Environment Model
EPSS	Electronic Performance Support System
HYP	Hypothesis
IB	Information Block
IETM	Interactive Electronic Technical Manual
IO	Information Object
IOC	Information Object Cluster
IPM	Intelligent Product Manual
JSP	Java Server Page
KB	Knowledge Base
LO	Learning Object
LOM	Learning Object Metadata

LRU	Least Replaceable Unit
MBR	Model-Based Reasoning
MOD	Models
OBS	Observations
OWL	Web Ontology Language
PAL	Protégé Axiom Language
PCD	Performance-Centred Design
PDM	Product Data Management
PM	Physical Model
PROSON	PROduct Support ONtology
PRSS	PRoduct Support System
PS	Performance Support
PSP	Product Support Problem
PSS	Product Support Solution
PSVD	Product Support Virtual Document
RDFS	Resource Description Framework Schema
RIO	Reusable Information Object
RLO	Reusable Learning Object
SCO	Sharable Content Object
SCORM	Sharable Content Object Reference Model
SGML	Standard Generalised Markup Language
SHriMP	Simple Hierarchical Multi-Perspective
SI	Status Identifier
TM	Technical Manual
UCD	User-Centred Design

UI	Unique Identifier
UM	User Model
VD	Virtual Document
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Expanding domestic and international product competition markets have increased the competition among global manufacturing organisations. As a result many industries have chosen to adopt advanced manufacturing technologies that secure increased product and volume flexibility, superior quality and lower manufacturing costs. The introduction of these technologies requires effective training and product support (Waldeck and Lefakis 2005). For instance, a survey conducted by SEMTA (2004) shows that more than 2500 trainees are employed in the UK aerospace industry alone. In addition, because of the continuously changing manufacturing requirements and products, training materials rapidly become out of date, even just after two months of their introduction. The evolving nature of the training requirements and the need to help users to maximise different processes, actions, and strategies associated with a product, require high quality product support (Mathieu 2001, Setchi et al. 2006).

In the last few years there has been a rapid progress in product support systems because many technical barriers have been removed. Producing, storing and transporting information in large quantities are no longer significant problems. However, product support is still in its infancy in terms of information integration and sharing. That is due to the multi-disciplinary nature of product support knowledge. Stuckenschmidt and Van Harmelen (2005) state that the problem of information sharing is only solvable by giving the computer better access to the semantics of the

information. Thus *a challenge* in product support is to *define formally product support knowledge and its semantics*.

In addition, Pasantonopoulos (2005) notes that a product support framework that would formalise the development of product support systems is required. The lack of such a framework makes the process of delivering product support information much more complex and reduces the degree of automation that can be introduced within a continuously evolving Intranet/Internet environment. Another *challenge* therefore is *to develop a framework for product support*.

The advent of new mobile devices and distributed networks raises the need of providing context-specific information to the users (Setchi and Lagos 2006). As Prekop and Burnett (2003) state, computer-based services should be adapted according to the context of their use. That poses a new *challenge to product support systems that have to become more attentive, responsive and aware of user's environment and requirements*.

1.2 AIM AND OBJECTIVES

The challenges discussed in the previous section are addressed in the rest of this thesis. The overall aim is to provide consistent, up-to-date and customer-benefits driven product support that actively assists the users. This research endeavours to integrate knowledge engineering with product support and help bridge the gap between conventional product support, where the support system is considered as a stand-alone application, and the new paradigm of responsive one, where the support system frequently communicates with its environment and reacts to stimuli. This new

paradigm would enable product support knowledge to be captured, stored, processed, and updated automatically, and being delivered to the users when, where and in the form they need it.

The individual objectives of the project are:

1. To define and product support knowledge.
2. To develop a semantic model of product support knowledge.
3. To develop a knowledge engineering approach and framework for product support systems.
4. To create a conceptual model of context-aware product support systems that supports context-specific delivery of information.
5. To create an approach for integrating product design and support documentation data.

1.3 OUTLINE OF THE THESIS

The main body of the thesis comprises Chapters 2 to 7 (Fig. 1.1). Chapter 2 is a review chapter. Chapters 3-6 address the objectives listed above. Chapter 7 provides validation and verification of the research results. The final chapter, Chapter 8, summarises the contributions and conclusions of the work and makes suggestions for further research.

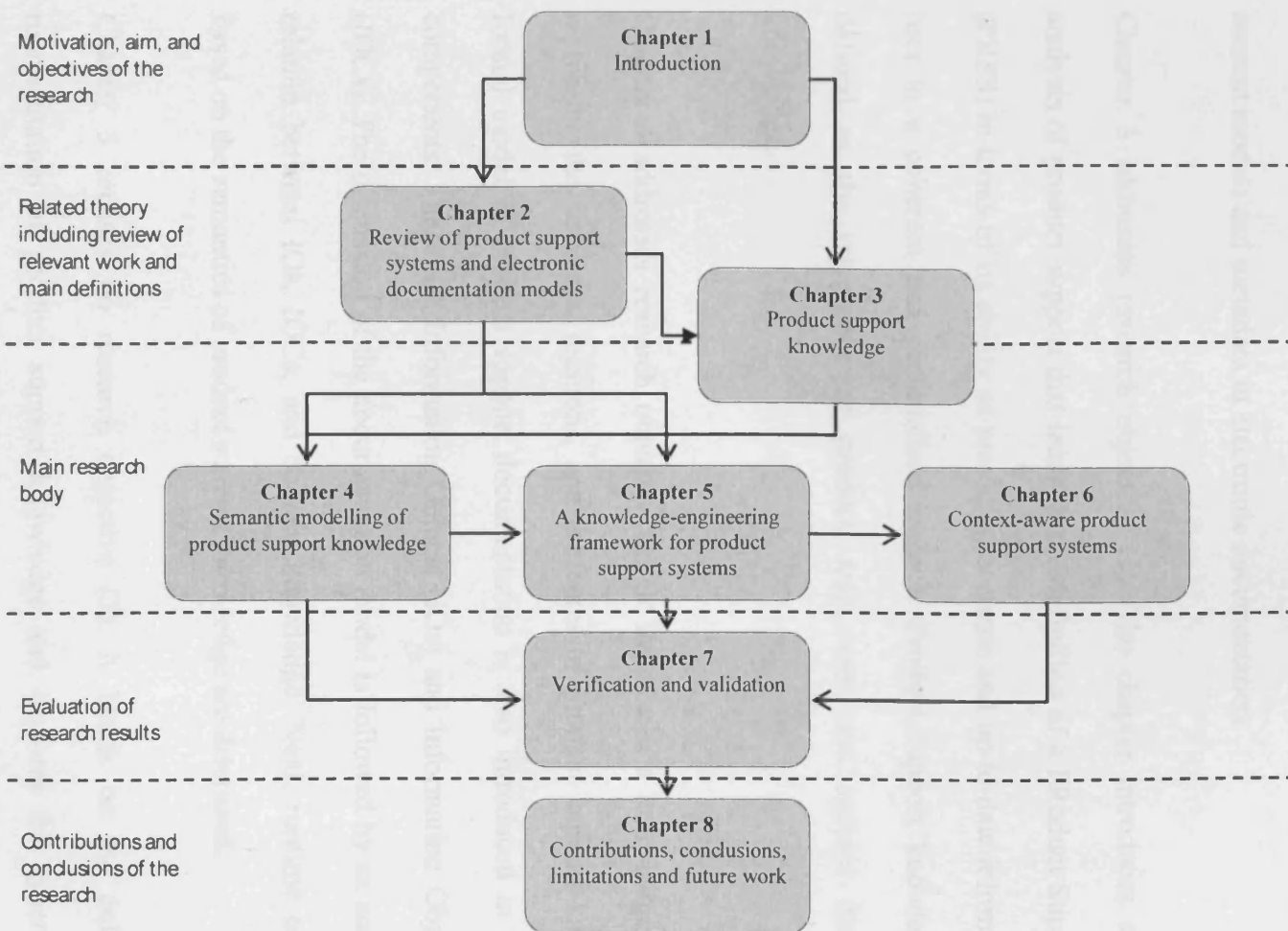


Figure 1.1. Outline of the thesis

Chapter 2 consists of three reviews that provide background knowledge for Chapters 3-6. The first review discusses product support. The second review focuses on semantics-related technologies and the application of knowledge engineering techniques in current product support systems. The third review is dedicated to content models and metadata in electronic documentation.

Chapter 3 addresses research objective (1). The chapter introduces a task-based analysis of product support that leads to the definition of a Product Support System (PRSS) in terms of its ability of providing accurate and up-to-date information to the user in a coherent and personalised manner. Product support knowledge is then defined as the integration of product, task, user, and support documentation knowledge.

Chapter 4 addresses research objective (2). It introduces a knowledge model that represents the structure, relations, and attributes of product support knowledge. A formal model of product support documentation is also introduced in terms of its components. These are Information Objects (IOs) and Information Object Clusters (IOCs). The discussion of the documentation model is followed by an analysis of the relation between IOs, IOCs, and domain knowledge. Next, runtime commitments based on the semantics of product support knowledge are discussed.

Chapter 5 focuses on research objective (3). It builds on the ontology-based representation of product support knowledge and explores the synergy between product support, problem solving, and knowledge engineering. Then, a structured problem solving approach is introduced that defines and classifies product support problems and utilises a multimodal reasoning strategy. The strategy combines case-

based adaptation and model-based generation techniques. Based on the proposed approach a knowledge engineering framework for product support systems is presented.

Chapter 6 addresses research objectives (4) and (5). It introduces a conceptual model of context-aware product support systems that extends the framework described in chapter 5. The conceptual model includes a formal representation of its context of use in terms of an ontology. The ontology contains knowledge about the users, their activities, the support environment, and device specifications and is further utilised for adapting information delivery according to context-specific features. An approach to semi-automatically integrating design and documentation data is proposed as a part of context-aware product support systems development process.

Chapter 7 verifies and validates the research presented in chapters 4-6, by including the descriptions of an integrated technology environment (PROSON) based on the semantic models introduced in chapter 4, a product support system (PROGNOSIS) developed according to the framework proposed in chapter 5, and an extended context-aware version of PROSON that illustrates the research concepts described in chapter 6.

Chapter 8 summarises the contributions made and the conclusions reached, the limitations of this research, and suggests possible directions for further investigation in related scientific areas.

CHAPTER 2

REVIEW OF PRODUCT SUPPORT SYSTEMS, SEMANTIC TECHNOLOGIES AND ELECTRONIC DOCUMENTATION MODELS

This chapter reviews technology approaches to providing electronic-based product support information. First existing systems for electronic-based product support are introduced. Particular attention is devoted to the features that differentiate them from traditional paper based support and maintenance systems. The application of knowledge engineering in product support systems is then reviewed within the context of the latest semantic technologies developed for the Semantic web, with an emphasis on the modelling and reasoning techniques being employed. The third part of the review discusses electronic documentation in terms of the content models and metadata used in e-training and e-learning applications.

2.1 PRODUCT SUPPORT

2.1.1 Interactive electronic technical manuals

Product support consists of everything necessary to allow the continued use of a product. Typical forms of support include installation, technical documentation, maintenance, user training, product manuals, help lines, servicing, and equipment upgrading (Goffin 1998; Pham et al. 2000).

One advanced approach to product support is that of Interactive Electronic Technical Manuals (IETMs). IETMs build on the notion of Technical Manuals (TMs). TM is a generic term for a document that explains how to use, maintain and handle a product from its delivery to its disposal, and in addition gives any technical information that a user is likely to need during the life of the product (BS4884-2 1993). Conventional paper-based TMs, however, have led to the problems reported below (Wills et al. 2002; Crowder et al. 2001; Su et al. 1997):

- Large volumes of documents in a wide range of formats, which are highly cross-referenced, resulting in time-consuming information acquisition.
- Widely dispersed documents with no immediate access and usability facilities.
- Difficult updating of documents leading to static, obsolete, and closed information.
- Presentation of information from the technical writers' viewpoint.
- Decontextualisation of information.

The NATO Continuous Acquisition and Life Cycle Support (CALC) initiative aimed to increase the maintenance quality and operational readiness, reduce overheads and improve the timeliness, by developing IETMs. An IETM is an interactive electronic version of a technical manual that provides information about the diagnostics and maintenance of complex mechanical systems of military or commercial nature (Boose et al. 2003; Kraidli et al. 2003).

The U.S. Department of Defence (DoD) uses a scale of five classes of IETMs to distinguish various levels of functionality offered. Classes I to III are electronically

viewable documents, ranging from page-based display (class I), to electronically scrolling documents (class II), and linearly structured IETMs (class III) (Van Amstel et al. 2000). Class IV IETMs are managed and authored directly via a database and require the use of Standard Generalised Markup Language (SGML) as source format (Boose et al. 2003). Class V IETMs support context sensitive navigation and optimise viewing through context-sensitive content delivery. They also provide a number of diagnostic services, by utilising intelligent diagnostics (Paul et al. 2003).

Su et al. (1997) state that the development process of an IETM should include requirement analysis, data collection, display design (e.g. hypermedia editing), time and resource planning, system integration analysis (e.g. integration with diagnostic and expert systems), reuse studies, rapid prototyping (i.e. creation of a prototype), multimedia selection, and testing/verifying the IETM.

IETMs utilise several technological advances in the areas of multimedia and hypermedia technologies, knowledge engineering, virtual reality, and database management (Wills et al. 2002) and have the following benefits over traditional technical manuals (Paul et al. 2003; Crowder et al. 2001; Frost 1998).

- Reduced time required to diagnose appropriate maintenance/repair actions, as the user is able to quickly locate the applicable technical information.
- Improved accuracy of maintenance diagnostics by utilising “smart” systems.
- Increased maintenance productivity, by empowering semi-skilled users and novices and allowing skilled users to focus on complex tasks.
- Increased motivation, as they advance knowledge construction.

- Increase of maintenance reliability due to the availability of updated information and exploitation of sensors.
- Reduced costs for printing, publishing, storage, and shipping.

2.1.2 Electronic performance support systems

Traditional training, which has been based on classroom instructions and treating trainees as a homogeneous group, has several weaknesses that can be summarised as follows (Ockerman et al. 1999; Raybould 1995; Bezanson 1995).

- Traditional training is geared towards increasing knowledge as opposed to improving performance and is most often evaluated on learner satisfaction instead of job performance.
- It is time-consuming and involves significant costs.
- It is done at a time other than when the employee needs to use the skill or information.
- It is trainer-centred as opposed to learner-centred.

The above realisation led to the development of the performance support concept. “Performance Support (PS) is a product or process attribute that aims to enhance user performance, through a user interface and support environment that anticipates user needs and supports them conveniently and effectively” (Bezanson 1995). The systems that materialise PS are called Performance Support Systems or Electronic Performance Support Systems (EPSSs). According to Joyce (2002), “an EPSS is an electronic device which provides information, software tools, and procedural knowledge, already available to the organisation, to an employee at their moment of

need, in order to enhance their performance of the task in hand”. The overall goal of an EPSS is to provide the right information, at the right time, in the right place, in the correct amount, and in the most efficient format (Forzese 1997).

There are different types of EPSSs that have been developed over the years. Benko and Webster (1997) categorise them according to their integration with other components and classify them as standalone EPSSs that are not integrated into the target business application, integrated non-contextual EPSSs that are integrated into the target business application but are not contextual, and integrated contextual EPSSs that are integrated into the target business application and also offer contextual support to the users.

There are three widely accepted design approaches to creating an EPSS. User-Centred Design (UCD), minimalism, and Performance-Centred Design (PCD). UCD focuses on understanding the needs of the user by involving users in every phase of the design process, from establishing requirements through iterative testing and post-release benchmark assessments (Mackenzie 2002). Minimalism aims to provide users with some basic guidance in order to start performing real tasks immediately, while allowing them to explore a system on their own (Mackenzie 2002; Forzese 1997). The goal of PCD is to increase overall productivity within organisations by enabling workers to complete meaningful tasks as soon as possible (Marion 2002). PCD should make the systems consistent, and optimise what appears on the display, the system-user interaction, the system’s behaviour, and the knowledge access and use (Gery 1995, Marion 2002).

Integral to the development of EPSSs has been the utilisation of advanced software technologies. Multimedia, virtual reality and hypertext have been used for creating user-oriented interfaces and systems (Cantando 1996). Knowledge engineering and knowledge based systems have enabled the dissemination of expert knowledge throughout the organisation (Raybould 2000). Information visualisation techniques, collaborative filtering and software agents can individually aid in the decision making and presentation processes (Quesenbery 2002).

EPSSs can provide the following benefits (Desmarais et al. 1997, Bastiaens 1999, Cantando 1996).

- Enhanced productivity.
- Reduced training cost.
- Reduced learning time.
- Increased worker autonomy.
- Increased quality due to uniform work practices and knowledge capitalisation.
- Cost reductions because of less reliance on consultants.
- Increased customer satisfaction due to better and faster service support.

2.1.3 Intelligent product manuals

Intelligent Product Manuals (IPMs) are class V IETMs. IPMs are designed to allow product users to utilise a product as easily, effectively and with as little additional care as possible while minimising support costs for manufacturers and suppliers (Pham et al. 2000). According to Pham et al. (1999a), “IPMs are computerised interactive product support systems that use product life-cycle information, expert knowledge

and hypermedia to provide just-in-time support to the user during the life of the product”.

IPMs use hypermedia, multimedia, and virtual reality for adapting the presentation to individual needs, information and communications technologies (e.g. Web), and knowledge based systems, for utilising product data as a primary source of information, and concurrent engineering for integrating product data throughout the life-cycle of the product (Pham et al. 1999b).

IPMs have all the potential advantages of IETMs and additionally have the following beneficial characteristics (Pham et al. 1999a).

- Platform independence as their distribution is realised through WWW.
- Total integration of product support with product data, product life-cycle information, expert knowledge, VR and hypermedia.

2.1.4 Maintenance systems

The trend toward increased automation has forced the managers to deploy computerised maintenance systems to maintain the complex equipment and to keep them in available state (Savsar 2006). These systems provide product support by identifying faults at real-time and/or computing operational and maintenance costs. However, the scope of these systems is different that the IETMs, EPSSs, and IPMs described in previous sections because instead of focusing on the user interaction aspects they are designed for product monitoring and fault identification (Gharbi et al. 2007). This means that user related dimensions that are in the focus of this work are not researched in maintenance systems.

2.1.5 Discussion

The technologies reviewed in section 2.1 indicate that electronic-based support has a number of advantages when compared to the paper-based one. Different approaches to product support systems have been developed, which are often described in terms of their purpose and features, enabling technology, development stages, and benefits. The following can be noted.

1. Most of the research in this area indicates the complexity of the domain, since knowledge is integrated from a number of engineering and non-engineering fields.
2. All technologies discussed share a lot of common characteristics although they have evolved independently of each other.
3. The advances achieved in each research domain can be further exploited by enabling the synergy between them. A formal approach towards developing a unified product support framework is therefore required.

2.2 KNOWLEDGE ENGINEERING IN PRODUCT SUPPORT SYSTEMS

2.2.1 Knowledge engineering vs. knowledge management in product support

Knowledge engineering can be defined as “the acquisition of knowledge in some domain from one or more non-electronic sources, and its conversion into a form that can be utilised by a computer to solve problems that, typically, can only be solved by persons extensively knowledgeable in that domain” (Gonzalez and Dankel 1993).

The importance of integrating knowledge engineering practices into the development of product support systems has been recognised by several researchers. Kabel and Kiger (1997) state that “job tasks involved in the creation of knowledge are very different than those traditionally expected from technical writers and educators” and a systematic approach towards the effective development of knowledge is needed (Setchi et al. 2005). Raybould (2000) says that knowledge and support resources become increasingly integrated while “information systems are moving from the data management age into the knowledge management age, in which...the database is being augmented by a knowledge base”.

However, research has focused on the dissemination and distribution of knowledge (knowledge management) rather than on modelling it and reasoning with it (knowledge engineering). Representative knowledge management examples include metadata-based Product Data Management (PDM) systems and languages (Ruland and Spindler 1995, Burkett 2001) and performance-centred portals (Elsbernd 2001). Only recently, the combination of structured, domain-specific and application-specific models with intelligent reasoning techniques has received significant attention (Setchi and Lagos 2005). As Stary and Stoiber (2003) note, “existing approaches lack structured representation of performance data and mechanisms for transforming that data into applicable knowledge”.

Recent interest in knowledge engineering is due to the following potential benefits to product support (Brusilovsky and Cooper 2002, Coffey et al. 2003).

- Knowledge models are a natural way of formally representing the real world in a form that can be processed by computers.

- Knowledge pertains the meaning of information and therefore decisions can be more accurately formulated.
- Knowledge engineering has widely researched problem-solving and decision-making processes and has proved to be useful in practical applications.
- Knowledge management techniques can help in disseminating knowledge throughout the organisation for faster learning and training.

Russell and Norvig (1995), however, state that the knowledge engineer must understand well the application domain before representing the important objects and relationships or deciding about the implementation of inference procedures. Sections 2.2.2 and 2.2.3 include a review of representation and inference techniques used in product support.

2.2.2 Semantics-based technologies

2.2.2.1 Semantic Web

The Semantic Web is envisioned as an extension of the current web where, in addition to being human-readable, documents are annotated with meta-information. This meta-information defines what the documents content is in a machine processable way. The explicit representation of meta-information, accompanied by domain theories, will enable a web that provides a qualitatively new level of service (Davies et al. 2003).

2.2.2.2 Ontologies

The term ontology was first used in philosophy. In that context, ontology is a theory about the nature of existence or the kinds of being (Woolf 1981); as discipline ontology studies such theories and its origin is from the Greek ontos (being). Artificial

intelligence and Web researchers have co-opted the term for their own jargon, and for them ontology is a document of file that formally defines the relations among terms. The most typical kind of ontology for the Web has taxonomy and a set of inference rules (Berners-Lee et al. 2001). Thus a piece of formally represented knowledge is based on conceptualisation. Conceptualisation means that a number of objects, concepts, relations and other entities can represent and express knowledge. An explicit specification of this conceptualisation is called ontology (Gruber 1993).

Basic element of ontology development is the representation language that is used. Most of the initial languages that were proposed, had a first order logic basis. Nowadays, the Web Ontology Language (OWL), is an ontology language standard. It is based on the following elements (W3C 2007).

- XML – provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema – is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF – is a datamodel for objects ("resources") and relations between them, provides a simple semantics for this datamodel, which can be represented in an XML syntax.
- RDF Schema – is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly

one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

2.2.3 Knowledge representation in product support systems

In knowledge system applications, domain knowledge is encoded by a wide variety of knowledge representation techniques (Yao and Etkorn 2006). Such techniques may utilise among others semantic networks, which indicate relations among concepts (Khalifa and Liu 2006), frames, which include attribute-value pairs (Cornet and Abu-Hana 2005), objects, which except for the domain knowledge represent control information (Bronson and Rosenthal 2006), and ontologies that facilitate knowledge sharing and reuse (Davies et al. 2003).

In product support the need to represent the knowledge of the domain has only recently been acknowledged. Developed models focus on user classifications, product and/or task structures, and how these are integrated. For example, the adaptive product manual developed by Pham and Setchi (2000) is based on using product, user and task models. These models are integrated within a knowledge based system which uses cases that represent previously solved situations.

A similar approach is adopted by Brusilovsky and Cooper (2002) who utilise integrated domain, task, and user models supporting the maintenance of equipment. The domain model represents the hierarchy of systems, subsystems and components, while the task model includes maintenance tasks, sub-tasks and steps. The components of the task hierarchy are connected to the components of the domain model. The three models are integrated in an expert system.

Latest research in product support indicates a trend towards semantic data modelling. For instance, Pham et al. (2003) employ a semantic data model to generate virtual documentation. The model is based on data usage analysis, which abstracts the intended purpose of the product and task data elements, and their functional characteristics. McMahon et al. (2004) also propose the use of a system model that includes concepts, constraints, and documents, and Setchi and Lagos (2006a) present a methodology for authoring technical documentation based on identified semantics.

Stary and Stoiber (2003) report a model-based approach that allows to structure and (de)compose a business process at hand into system functions and task-based user interactions for performance support. Their conceptual framework includes task, user, data, and interaction models that detail the organisation from different viewpoints.

2.2.4 Problem solving and reasoning techniques in product support

Research in AI has greatly benefited from using problem solving techniques (Dale and Weems 2005). For example, divide and conquer and building block techniques are employed in rule-based reasoning, where expert knowledge is represented using rules (Kolodner 1993). Solving by analogy is utilised in Case-Based Reasoning (CBR), which retains and reuses previous solutions generated by the system (Gonzalez and Danker 1993). Means-ends analysis is used in Model-Based Reasoning (MBR), which employs general knowledge about the application domain (Gonzalez and Danker 1993). Each technique is applicable in different circumstances. Rule-based technique is appropriate for applications that necessitate modular development and uniform knowledge representation. Case-based reasoning is preferred in situations where the domain is not well known but experiences can be easily used to enrich the

knowledge of the system. Model-based reasoning is used when the application domain is well known and especially if its causality characteristics are thoroughly understood (Kolodner 1993, Gonzalez and Danker 1993).

Studies show that the most common AI technique used is rule-based reasoning, which is primarily employed in troubleshooting. Paul et al. (2003), utilise intelligent diagnostics in combination with an IETM for supporting the operation of a radar warning receiver. The role of the diagnostic tool is carried out by an expert system that analyses faults and suggests corrective actions. Coffey et al. (2003) have also developed a system that uses an expert system for providing training to electronics technicians.

In addition, a number of researchers have used CBR for diagnosis and help-desk applications. Foo et al. (2002) utilise CBR in combination with neural networks for producing a help-desk-support environment, while Auriol et al. (1999) use a CBR system in the troubleshooting of a welding robot.

Model-based reasoning has received less attention compared to the two previous techniques. An example is the research of Brusilovsky and Cooper (2002), who employ models for adapting the interface of a performance support system and creating a problem solving engine. Gruber et al. (1997) have also used model-based reasoning to document engineered systems in support of collaborative design and simulation-based training.

Latest attempts focus on the integration of different reasoning techniques. For instance, Pham and Setchi (2003) develop adaptive product manuals by combining

CBR for interpreting user's requests and rules for adapting the generated documents. In addition, Setchi and Lagos (2005) propose the use of a model and case-based reasoning approach for creating electronic documentation.

2.2.5 Discussion

This section has reviewed knowledge engineering applications in product support in terms of the knowledge being represented and the reasoning techniques that have been utilised. The review has highlighted the following points.

1. There is a noticeable trend towards knowledge-based support delivery.
2. Knowledge engineering can be beneficial in the development of models or reasoning algorithms for advanced support systems.
3. Although previous studies address the use of knowledge engineering practices in product support, they do not follow a uniform approach towards the development of their knowledge bases. As a result, a major limitation of the previous work is the lack of design and knowledge reusability.
4. Existing approaches lack mechanisms for transforming product support data into applicable knowledge.
5. Although most recent research approaches adopt a hybrid reasoning strategy by applying combination of reasoning techniques, the application of multi-modal reasoning in product support systems has not been thoroughly researched.
6. Most of the attempts till now are application oriented, although they share similar characteristics in both modelling and inference phases.

2.3 CONTENT MODELS AND METADATA IN ELECTRONIC DOCUMENTATION

2.3.1 Content models and components

Technical documentation and e-learning are the main research areas where content modelling design has been applied. Technical documentation has adopted paradigms such as the Information Object, Information Block, and DITA, while e-learning involves the SCORM, CISCO RLO/RIO, Learnativity, and ALOCOM models, which are largely based on the notion of the Learning Objects.

2.3.1.1 Information object

Tucker and Harvey (1997) define the Information Object (IO) as “a locution of product data that describes one idea”, focusing on technical documentation. According to Ranwez and Crampes (1999) and Vercoustre et al. (1997) the notion of an IO advances reusability by segmenting the content of a virtual document into information groups that can be related to each other and combined produce a Virtual Document (VD).

Setchi (2000) extended the previous work by claiming that IO is “a data structure that represents an identifiable and meaningful instance of information in a specific presentation form”. Pham and Setchi (2003) and Lagos et al. (2005) view an IO as an entity, which has structure, presentation, and semantics (meaning) clearly separated. Pham et al. (2003) adopts a representation of the user’s purpose in terms of IO attributes. Pasantonopoulos (2005) indicates that the use of IOs has several advantages including information reuse and structuring, as well as conceptual

separation. Utilising these characteristics, Pasantonopoulos (2005) proposes that the IO should be transformed from a static entity into a dynamic object with embedded intelligence.

One of the main concerns in developing IOs has been their modular structure. Setchi (2000) organised IOs into Information Elements according to their content, while Huneiti (2004) clustered IOs based on their usage purpose. Preserving the semantic distinctiveness of any related IO structures is necessary for enabling further reuse.

2.3.1.2 Information block

Horn (1999) introduced the concept of Information Blocks (IBs) as a basic component of a structured writing approach. IBs replaced the paragraph as “basic units of subject matter” and have been developed according to the following four main principles.

- The *chunking principle*, which states that all information should be grouped into small, manageable units, called information blocks and information maps. Small (in information blocks) is defined as usually not more than 7 plus or minus 2 sentences.
- The *labelling principle*, where each label describes a number of specific criteria characterising every chunk and group of chunks.
- The *relevance principle*, which says that only information that relates to one main point, based upon that information's purpose or function for the reader, should be included in each chunk.
- The *consistency principle*, which asserts that for similar subject matters, similar words, labels, formats, organizations, and sequences, should be used.

Horn (1999) has defined about 200 types of IBs including comment, definition, description, diagram, and example, and argues that IBs in the domain of relatively stable subject matter (such as product support documentation) can be sorted into seven basic classifications, which are called information types and which are namely procedure, process, concept, structure, classification, principle, and fact.

2.3.1.3 Darwin information typing architecture

The Darwin Information Typing Architecture (DITA) is an XML-based, end-to-end architecture for authoring, producing, and delivering technical information (Day et al. 2005). Like Information Blocks, DITA consists of a set of design principles for creating “information-types”. However, DITA is topic oriented, meaning that the topic is the most abstract standard structure. Topic in DITA has no internal hierarchical nesting but rather relies on sections that define or directly support it (Hennum 2005).

Furthermore, the seven information types of IB are replaced by three other types in DITA, which are namely the concept, reference, and task ones (OASIS DITA 2005).

- The concept provides background that helps readers understand essential information about a product, interface, or task. Often, a concept is an extended definition of a major abstraction such as a process or function. Conceptual information may explain a product and how it fits into its category of products.
- Tasks are the essential building blocks for providing procedure information. A task topic provides precise step-by-step instructions for performing a task.

- Reference topics are often used to cover subjects such as the commands in a programming language and provide quick access to facts. Information needed for deeper understanding of a reference topic or to perform related procedures should be provided in a concept or task topic.

The above topics include the IB types (e.g. concept is an extended definition of a process, application, etc) and can also be specialised in order to extend the generic topics into new information types.

Verbert and Duval (2004) have compared DITA and IB and have identified the following weaknesses and strengths. DITA has a very flexible architecture as it permits any type of content structure to be defined but it is still under development and unsupported by tools. Horn's IBs are much further evolved and well documented but they are mainly an authoring environment rather than an information architecture like DITA (Namhan 2001).

2.3.1.4 Learning object

During the last few years e-learning has been recognised as an internal part of lifelong learning and training. It is defined as "the use of new multimedia technologies and the Internet to improve the quality of learning by facilitating access to resources and services, as well as remote exchange and collaboration" (COM 2001). The Learning Object (LO) has been introduced as a way of reusing and repurposing e-learning content.

According to the Learning Object Metadata standard a learning object is “an entity, digital or non-digital, that may be used for learning, education or training” (IEEE LOM 2002). Although learning objects have been often regarded as traditional documents, a number of recent approaches propose their fragmentation or composition into other entities, in order to produce electronic resources that will serve a learning purpose. As Duval and Hodgins (2003) claim, the IEEE LOM (2002) definition allows for an extremely wide variety of granularities, a problem addressed by the following content models.

2.3.1.5 Sharable content object reference model

The Sharable Content Object Reference Model (SCORM) content aggregation model supports the process of creating, discovering and gathering simple learning assets, in order to produce more complex learning resources (ADL 2004). The model has three main components; Assets, Sharable Content Objects (SCO), and Content Organisations. Assets are an electronic representation of media, such as text, images, and sound that can be rendered by a Web client and presented to a learner. A SCO is a collection of one or more assets. A Content Organization is a map that represents the intended use of the content through structured units of instruction (Activities) and relates the components to each other. The SCORM model proposes the use of meta-data (acquired from the IEEE LOM (2002) standard, which is discussed in a subsequent section) for describing each of the SCORM content model components.

2.3.1.6 CISCO reusable learning object/reusable information object

The Reusable Information Object (RIO) Strategy has been introduced by CISCO systems (CISCO 1999). An RIO is a granular, reusable chunk of information that is

media independent and 7 ± 2 RIOs compose a Reusable Learning Object (RLO). Each RLO is composed in turn of content, practice, and assessment items and is built upon a single objective, while it can be classified as either being a Concept, Fact, Process, Principle or Procedure. A set of metadata is used to characterise each level of components (e.g. Strand for the RLO denotes the major topic area).

2.3.1.7 Learnativity model

The Learnativity content model has been used to organise the content of e-Learning and knowledge management applications. It consists of the following levels (Wagner 2002); content assets that are raw media elements (e.g. a single sentence), information objects that are sets of raw media elements (such objects could be based on Horn's IBs (Verbert and Duval 2004)), learning objects that organise the components based on a single objective (more restricted sense than the definition of LOM given by the IEEE), learning components that correspond to more conventional lessons and chapters, and learning environments that refer to learning components wrapped with additional functionality such as communication tools and peer-to-peer computing. The learnativity model proposes general rules for assigning metadata to each of the levels (e.g. contextual information should be added at the learning object level).

2.3.1.8 ALOCOM

ALOCOM is a general content ontology that defines a framework for LOs and their components (Verbert and Duval 2004). The ALOCOM ontology distinguishes between Content Fragments (CFs), Content Objects (COs) and Learning Objects (LOs). Content fragments are learning content elements in their most basic form, like text, audio and video. Content objects aggregate content fragments and add

navigation. Navigation elements enable structuring of content fragments in a content object. For defining content object types, the DITA model has been used, including except for the three general types (i.e. task, concept, reference) other building blocks such as definitions, sections, paragraphs, lists and comments. Finally, LOs aggregate content objects and add a learning objective.

ALOCOM is different than previous approaches because it defines an ontology for its underlying content model. However, metadata for describing each content component have not been defined yet, while the definition of more content object types is still under development.

2.3.2 Metadata

Metadata (or data about data) is a set of elements used to describe a resource (Dublin Core 2005a). For example a book can have metadata corresponding to the author, the publication date, and the subject keywords. The more widely used metadata standards in e-Learning and electronic documentation are the IEEE LOM and Dublin Core ones.

2.3.2.1 IEEE learning object metadata

The IEEE Learning Object Metadata (LOM) standard includes simple and aggregate data elements that can be at different levels of granularity, with grade 1-4 (described by the Aggregation Level flag). The LOM has nine main categories as follows (IEEE LOM 2002).

- The general category that has general information describing the LO such as a Title and some Keywords.

- The lifecycle category that describes the history and current state of an LO with metadata such as Version, Status, and Date.
- The meta-metadata category, which explains the metadata record itself (e.g. Identifier flag describes a unique label for a metadata record).
- The technical category which identifies the Format, Size, Type and other relevant hardware/software related data.
- The educational category that defines the role of the user within the learning environment (e.g. Interactivity identifies whether the user is actively involved in the learning process (for example with questionnaires) or just passively absorbs information) and the learning Type of the LO (e.g. exercise). Other interesting metadata used to characterise an individual within this category include the Intended User Role (i.e. manager, educator, learner, etc.), the Context (i.e. higher education, school, training, etc.), and Difficulty (i.e. easy, medium, difficult, etc.).
- The rights category defines the intellectual property rights (e.g. Cost).
- The relation category, which is used to describe the links between different LOs. The most important metadata in this category are Kind (adopted from Dublin Core and including isPartOf, references, hasPart, isRequiredBy, etc.) and Resource (i.e. the target LO).
- The annotation category, that provides comments on the educational use of the LO (e.g. Description specifies the content of the annotation).
- The classification category, which describes where the learning object falls within a classification system. Some of the metadata defined in this category are Taxon (i.e. a particular term within a taxonomy), Purpose (e.g. discipline, idea, pre-requisite), and Keyword (i.e. phrases descriptive of the LO).

2.3.2.2 Dublin core

The Dublin Core Metadata is an element set for describing networked resources (Dublin Core 2005b). The Dublin Core has defined an element set used to describe the content of the resource (i.e. Title, Subject, and Description), the nature or genre of the content of the resource (i.e. Type, for example “image”), its relation to other resources (the same as Kind in the relation category in IEEE LOM), and other administrative (e.g. Creator, Publisher, and Date) and rights related information (e.g. Provenance, RightsHolder, etc.). One of the characteristics of the Dublin Core is that it proposes the use of controlled vocabularies for defining the values of the introduced metadata (no vocabulary is however presented since the standard addresses general requirements).

2.3.3 Discussion

The need to replace the paragraph with other electronic documentation constructs has initiated the development of a number of different document modelling approaches. In this section, these are distinguished according to the content models and metadata used to describe the documentation components. The following conclusions can be derived from this overview.

1. The work on IBs and structured writing affirms that the traditional paragraph is not appropriate as a construct of hypermedia documentation, especially in structured areas like that of technical writing and product support.
2. Although, IOs and IBs share some common principles like modularity and labelling, the rules in developing IBs are much more stringent than in the case of IOs.

3. To the knowledge of the author, IO is the only documentation construct which has structure, presentation, and semantics clearly separated.
4. Although a number of different content models have been proposed, there is a great diversity between granularities and encoding formats. All these need to be uniformly represented with a formal language.
5. DITA is one of the most popular content models nowadays.
6. There is a trend towards developing an ontology for electronic documents.
7. The study of the metadata models indicates that the value of metadata attributes should be based on structured vocabularies of the domain.
8. Metadata specifications define documentation objects relations on a syntactical rather than on a semantic level. This emphasises the fact that the semantics of the relations between different documentation components should be further researched.

CHAPTER 3

PRODUCT SUPPORT KNOWLEDGE

This chapter addresses the first objective of this work and focuses on defining product support knowledge. The chapter analyses product support from a task-based point of view and accordingly introduces the concept of a PProduct Support System (PRSS). Next it outlines the elements of product support knowledge and illustrates their definitions. Finally, conclusions are discussed.

3.1 ANALYSIS OF PRODUCT SUPPORT

In this chapter, product support is analysed in terms of the tasks involved. A task is defined as a strategy, which is followed in order to achieve a specific goal (Wielinga et al. 1993) and could be decomposed into subtasks, and actions. An action is a primitive operation needed for a task to take place. For example, repairing a product is considered a task that includes subtasks like ‘locate’, ‘remove’, and ‘examine’ the faulty part and actions such as ‘unscrew a bolt’. Therefore, a task can be represented in a formal way as follows.

$$G = f_T(R_T, C_T, Y) \tag{3.1}$$

where G is the goal of the task, f_T denotes the transformation that takes place within a task, R_T is the input to the task, and C_T are the constraint(s) that are encountered at

different levels of granularity Y for reaching a specific goal. Granularity is the degree to which a task can be decomposed in different subtasks, for achieving a goal.

Equation (3.1) shows that a task's goal depends on the input that the transformation process (i.e. transformation from resources to task outcomes) receives and to the constraints that control it. The input of a task includes the resources available. The optimal situation is to have all required resources available just-in-time and just-in-place. In product support, these include tools, equipment, data, information, personnel, facilities, computers, supplies, spare components, experience, skills and knowledge. The input therefore can be represented as a number of sets $\{R_1, \dots, R_p\}$, where each set stands for a different type of resource needed to achieve the goal.

$$R_T = \{R_1, \dots, R_p\} \quad (3.2)$$

where p is the number of the resources' sets, for example, $R_1 \equiv$ tools, $R_2 \equiv$ equipment, $R_3 \equiv$ knowledge, and so on. Each of these sets contains several members that are needed to complete the task. Consequently, if $r_1 \equiv$ screwdriver, $r_2 \equiv$ hammer, $r_3 \equiv$ spanner, etc., then $R_1 = \{r_1, \dots, r_i\}$, where i denotes the number of tools that are included in R_1 .

Therefore, a task can be redefined as follows.

$$G = f_T((R_1, \dots, R_p), C_T, Y) \quad (3.3)$$

where C_T is a set representing the constraints c_k .

$$C_T = \{c_1, \dots, c_k\} \quad (3.4)$$

Typical constraints that need to be satisfied in product support include requirements related to cost, quality and safety, as well as conformance to standards and regulations.

The applicability of (3.3) will be illustrated with two examples of maintenance tasks that show cases when product support is needed (Fig. 3.1).

Example 1: Corrective maintenance. Problem: flat tyre.

In order to insert a new tyre, several resources are needed like tools- R_1 (e.g. spanner), equipment - R_2 (e.g. jack) and knowledge- R_3 (skills to perform the task). If one of these resources (e.g. R_3) is missing, then the user will require support (e.g. user manual) in order to fix the car. Additional knowledge is hence needed for completing the task, which is provided by product support and according to equation 3.3 is

$$InsertTyre = f_{InsertTyre}(R_{knowledge}, C_{InsertTyre}, Y) \quad (3.5)$$

Example 2: Preventive maintenance. Problem: worn out tyre.

The user may need to change the tyre, although it is not flat, because it is worn out. The constraint in this case is the legal minimum thread depth of the main grooves of car tyres - c_1 (in the UK and the EC, it is 1.6mm). The existing resource in this case (the tyre- r_1) does not satisfy the safety requirements. The delivery of a new tyre to the user that is within the specified boundaries denotes again a product support task, which according to equation 3.3 can be represented as follows.

$$ChangeTyre = f_{ChangeTyre}(R_{components}(tyre), C_{ChangeTyre}(LawThreadDepth), Y) \quad (3.6)$$

Statement 1: Product support is needed when there is a lack of resources for completing a task and/or when the existing resources do not satisfy specific crucial requirements.

3.2 PRODUCT SUPPORT SYSTEMS

As mentioned in chapter 2, intelligent product manuals, interactive electronic technical manuals, and electronic performance support systems are some of the forms used to deliver product support to the end user. These systems utilize electronic means and technologies (e.g., computers) in order to provide information to the user.

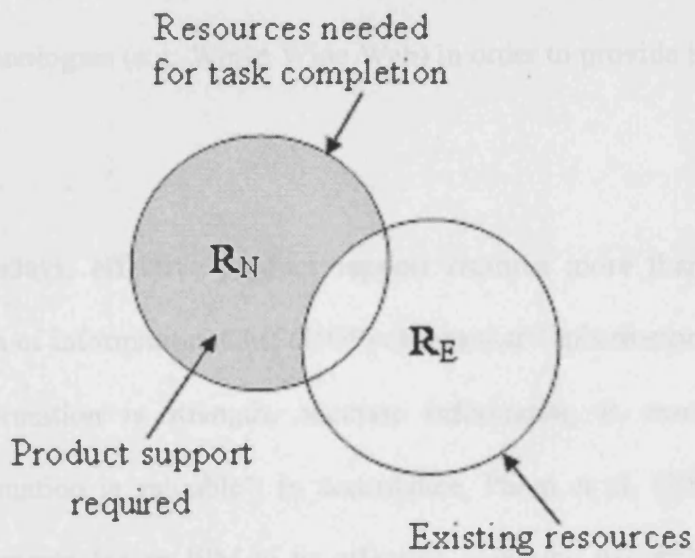


Figure 3.1. The need for product support

In addition to the aforementioned characteristics, product support systems should contain these features in order to enable the provision of accurate and up-to-date information to the user in a convenient and personalized manner. Consequently, a product support system can be defined as follows:

Statement 1: Product support is needed when there is a lack of resources for completing a task and/or when the existing resources do not satisfy specific control constraint(s).

3.2 PRODUCT SUPPORT SYSTEMS

As mentioned in chapter 2, intelligent product manuals, interactive electronic technical manuals, and electronic performance support systems are some of the forms used to deliver product support to the end users. These systems utilise electronic means and technologies (e.g. World Wide Web) in order to provide information to the user.

However nowadays, effective product support requires more than just electronic-based provision of information. Cliff (1999) claims that “information is not power but organised information is strength, accurate information is essential and up-to-date/new information is valuable”. In accordance, Pham et al. (2002) have defined several requirements for an IPM to be effective including the ability of supporting different categories of users in different activities and the availability of highly accurate information. Furthermore, in a visionary paper on EPSSs in the 21st century, Raybould (2000) argues that the convergence of knowledge engineering and performance support is essential for information overload to be avoided.

In addition to the aforementioned characteristics, product support systems should extend their features in order to enable the provision of *accurate* and *up-to-date information* to the user *in a coherent and personalised manner*. Consequently, a product support system can be defined as follows.

Definition 1. *A Product Support System (PRSS) is an electronic medium that aims to compensate the lack of knowledge of the user in a particular subject or situation related to a product by providing accurate and up-to-date information in a coherent and personalised manner.*

The implications of the last characteristic on the design of PRSSs are numerous.

1. Up-to-date information in a dynamic environment like that of product development and exploitation means that *the product support system has to be integrated into the product lifecycle.*
2. Accurate information can be provided only if the knowledge underlying PRSSs is *formally defined, rigidly structured, and semantically organised.*
3. Coherency can be achieved if the *domain knowledge is consistently represented and its relation with product support is formally described.*
4. Personalisation of delivery and presentation indicates that *knowledge about users and tasks should be modelled and included within the product support system.*

Nevertheless, as demonstrated in chapter 2, although current research addresses the use of knowledge engineering practices in product support, requirements (1), (2), and (3) have been only partially considered. It is believed that this could become an obstacle in the nearest future when a new generation of much more complex and highly customized products emerges. The rest of this chapter identifies the knowledge contained within a knowledge-based PRSS as an aggregation of product, task, user, and documentation knowledge.

3.3 DEFINITION OF PRODUCT SUPPORT KNOWLEDGE

3.3.1 Product support knowledge elements

The goal of any product support system is to deliver knowledge that is accurate, applicable, reliable and user-tailored. In order to do that, it should be able to process and analyse thorough, detailed and up-to-date knowledge of the domain of interest. Consider the example of a novice user having to change a tyre. Two typical questions that the product support system should be able to answer are:

- What tyre should be chosen?
- How should it be inserted?

Therefore the system should have knowledge about the products, tasks, and users supported.

- **Product.** In this case, it is useful to know the characteristics and specifications of both the vehicle and its tyres. For example, vehicle related information can be utilised to specify the required tyre size (e.g. 21" tyre) and/or material (e.g. rougher rubber compounds are needed for larger vehicles).
- **Task.** The series of actions that should be followed to insert a tyre can be realised, as long as the initial and goal states are known and knowledge about each action exists.
- **User.** The representation of the solution that the product support system delivers is based on the knowledge it has about the user. If the user is novice, detailed static images can be replaced or supported by animated multimedia (e.g. flash

animations) and examples. If the user is more experienced, a textual description of the main steps would be sufficient.

As illustrated, the product support knowledge base should contain relevant knowledge about the products, users, their tasks, and the way in which these are linked to each other in product documentation (Fig. 3.2).

Statement 2: If the knowledge available in a product support system is K_{PRSS} , product knowledge is K_p , user knowledge is K_u and task knowledge is K_t , then for the product support system to be able to deliver efficient support, the following formal requirement must be satisfied.

$$K_p \cup K_u \cup K_t \subseteq K_{PRSS} \quad (3.5)$$

Recent research has developed approaches for modelling product support systems (i.e. user-centred, task-centred, and performance-centred design). However, there are currently no uniform definitions of product, task and user knowledge within this application area. This section attempts to fill this gap.

3.3.2 Definition of knowledge

According to Webster (2006), Oxford (2006) and Cambridge (2006) dictionaries, the word 'knowledge' has the following meanings:

- Perception; clear perception of fact, truth or duty.

- **Apprehension:** awareness, experience, familiarity, awareness, or understanding gained through experience or study.
- **Learning:** a branch of learning, a course.
- **Information:** the body of facts accumulated by mankind, specific information about something, information required.

In the study of K. Srivastava et al. (2003) knowledge is defined as "information put to productive use". This highlights the fact that knowledge is created and applied within specific applications context.

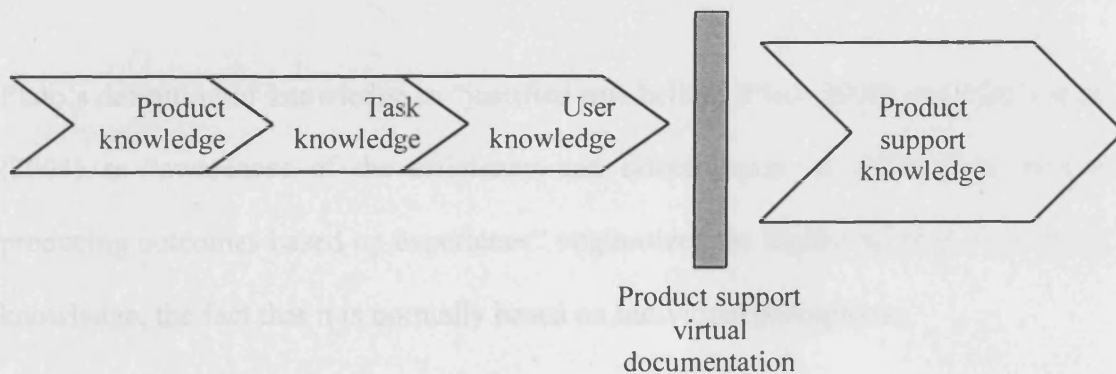


Figure 3.2. Product support knowledge

- Apprehension, awareness, experience; Familiarity, awareness, or understanding gained through experience or study.
- Learning; a branch of learning, a science.
- Information; the body of facts accumulated by mankind, specific information about something, information acquired.

In the study of Kakabadse et al. (2003) knowledge is defined as “information put to productive use”. That highlights the fact that knowledge is created and applied within specific application context.

Plato’s definition of knowledge as “justified true belief” (Plato 2006) and Yim’s et al. (2004) as “awareness of the efficiency and effectiveness of different actions in producing outcomes based on experience” emphasizes the highly subjective nature of knowledge, the fact that it is normally based on individual perceptions.

Several researchers (Kakabadse 2003, Nonaka 1994, Gunnlaugsdottir 2003, Bose 2003, Liebowitz and Megbolugbe 2003) concentrate also on the transformation of data into information, and then knowledge. Data is viewed as raw elements, which if organised in explicit way, form information. Knowledge is created when the information is structured according to certain purpose, context or perception. Accordingly, Nonaka (1991) argues that Western management sees knowledge as formal and systematic, captured in codified procedures (Belogun and Jenkins 2003). Strengthening that opinion, Stefik (1995) states that “knowledge in terms of the knowledge systems refers to the codified experience of agents”.

Following this discussion, information is viewed in this thesis as the building block of knowledge, whether it is derived from direct or indirect experience, study, or learning. However, the information acquired cannot be transformed into knowledge unless its meaning is apprehended. This understanding is tightly related to the purpose, context and beliefs within which knowledge is interpreted. Furthermore, the transformation of information in knowledge depends on the cognitive abilities of the individual users. The following working definition of knowledge is adopted in this work.

Definition 2. *Knowledge is a specific semantic interpretation of information.*

In the terminology of logic, “interpretation” is a mapping from statements to conceptualisation. In this definition, “specific interpretation” means that knowledge is context-dependent and therefore inherently different for each individual. “Semantic interpretation” denotes that the mapping to conceptualisation is carried out using semantics.

3.3.3 Product knowledge

Kaposi and Myers (2001) define a product in terms of its attributes and processes and the interrelations between them, while others (e.g. Oxford dictionary (2006)) concentrate on its property of “being produced”.

Ahn and Chang (2004) concentrate on the distinction between product and process and state that “in a knowledge intensive firm, product is the explicit output of the value-adding activities or production”, describing product as the explicit outcome of a

process. In the product support area, products are both tangible (e.g. vehicle) and intangible (e.g. software).

According to the above discussion, a product within product support is defined as follows.

Definition 3. *Product is an entity of interest created by a process (Kaposi and Myers 2001).*

In the above definition the meaning of the word “entity” is adopted by ISO 8402 (1994) and is “that which can be individually defined or considered”.

To facilitate a product knowledge definition, the meaning of product data and semantics is first considered, because product knowledge merges and extends the notions of product data and semantics as it includes and relates both of them.

In accord with ISO 10303-1 (1994), product data is “a representation of information about a product in a formal manner suitable for communication, interpretation, or processing”. In addition, Petiot and Yannou (2004) claim that product semantics is “the study of the symbolic qualities of man-made forms in the context of their use, and application of this knowledge to industrial knowledge”.

Ahn and Chang (2004) analyse product knowledge from the perspective of business performance and classify product knowledge into tacit and explicit, claiming that “tacit product knowledge is product-specific know-how that cannot be easily expressed and it resides on the human brain. Explicit product knowledge is the

knowledge accumulated in a knowledge repository...product knowledge tends to be object-oriented, focused on a specific product”.

Definition 4. *Product knowledge is a formal, temporal representation of the specific semantic interpretation of information, associated with an entity of interest created by a process.*

Or

Product knowledge is a formal, temporal representation of the knowledge related to the product.

The representation of information should be formal, as it has to be suitable for communication, interpretation, or processing, as required by ISO 10303-1 (1994). Moreover, it should be temporal, because it is valid only for a specific instance or period of time during which the information remains unchanged. Valid means that the information is within certain correct boundaries.

3.3.4 Task knowledge

A product support system should be able to advise the user on the sequence of actions or the strategy that should be followed to reach a specified goal. The definition of a task given by Wielinga et al. (1993) is adopted in this study as it reflects the above description.

Definition 5. *Task is a strategy, which is followed in order to achieve a specific goal (Wielinga et al. 1993).*

Liebowitz and Megbolugbe (2003) also describe tasks in terms of their goals and sequence of actions. So, they claim that “task knowledge describes which goal(s) an application pursues and how these goals can be realised through decomposition into tasks and inferences”. In the same manner, task knowledge for a product support system is defined as follows.

Definition 6. *Task knowledge is a formal, temporal representation of the specific semantic interpretation of information, which defines a strategy followed to achieve a specific goal.*

Or

Task knowledge is a formal, temporal representation of the knowledge related to the task.

3.3.5 User knowledge

Several definitions for the word “user” can be found in the literature. Their common characteristic is that they are all system oriented. This means that they are formed according to a reference system and its expected utilisation. As far as a product system is considered, user and user knowledge are defined as follows.

Definition 7. *User refers to any person, group or functional unit that directly interacts with a system.*

Definition 8. *User knowledge is a formal, temporal representation of the specific semantic interpretation of information, associated with a person, group or functional unit that directly interacts with a system.*

Or

User knowledge is a formal, temporal representation of the knowledge related to the user.

3.3.6 Product support virtual document

Product support knowledge is composed by product, task, and user knowledge and the understanding of the way in which these are integrated with each other in a product support system. The integration is achieved through product support electronic-based documentation.

One of the main reasons for the success of electronic-based documentation is its ability of re-purposing its components according to the requirements. For example, in many cases a paragraph or a sentence can be reused by copying and pasting in different documents. However, it is possible to reuse documentation elements in a more sophisticated way if accessing and processing them at run-time is possible. In order to do that, a flexible and dynamic but also rigid and formal underlying model of product support virtual documents is needed. As a first step towards that the notion of a product support virtual document is defined in the remaining of this section.

The Oxford dictionary (2006) describes a document as “a piece of written, printed, or electronic matter that provides information or evidence”, which means that there are two important aspects of a document.

- The means by which it is created (i.e. written, printed, or electronic).
- The purpose of its existence (i.e. it provides information or evidence).

In addition Gruber et al. (1997) has defined a Virtual Document (VD) as “a hypermedia document that is generated on demand from underlying information sources, in response to user (reader) input”. Gruber therefore, defines a VD as a specialisation of a document by elaborating on the matter utilised (i.e. hypermedia or virtual) and on the generation approach (i.e. on demand from underlying information resources). Furthermore, a Product Support Virtual Document (PSVD) is a VD that has the constraint of providing information related to a product. A PSVD is defined therefore, as follows.

Definition 9. *A product support virtual document is a piece of hypermedia that is generated on demand from underlying information sources, in response to user (reader) input, and provides information or evidence related to a product.*

The definitions given in this section are summarised in Table 3.1.

3.4 SUMMARY AND CONCLUSIONS

This chapter suggests that the first step towards developing product support that is up-to-date, accurate, coherent, and personalised is to define the knowledge contained within a product support system. The task-based analysis of product support performed in the chapter revealed the direct relationship of product support with the lack of resources for performing a task. Having that as a starting point a product support system has been defined as a medium that aims to compensate the lack of user’s knowledge. This viewpoint transforms the complex problem of creating a Product Support System (PRSS) into the more manageable goal of developing a knowledge-based platform for product support.

Table 3.1. Definitions related to product support knowledge

Term	Definition
Knowledge	Knowledge is a specific semantic interpretation of information.
Product	Product is an entity of interest created by a process (Kaposi and Myers 2001)
Product knowledge	Product knowledge is a formal, temporal representation of the specific semantic interpretation of information, associated with an entity of interest created by a process.
Task	Task is a strategy, which is followed in order to achieve a specific goal (Wielinga 1993).
Task knowledge	Task knowledge is a formal, temporal representation of the specific semantic interpretation of information, which defines a strategy followed to achieve a specific goal.
User	User refers to any person, group or functional unit that directly interacts with a system.
User knowledge	User knowledge is a formal, temporal representation of the specific semantic interpretation of information, associated with a person, group or functional unit that directly interacts with a system.
Product support virtual document	A product support virtual document is a piece of hypermedia that is generated on demand from underlying information sources, in response to user (reader) input, and provides information or evidence related to a product.

The analysis and design of a knowledge-based system follows the natural order of defining the knowledge required, modelling it with a platform-independent way, and identifying and applying appropriate reasoning techniques. As a result product support knowledge related definitions have been examined and unified by identifying their key characteristics for a product support system. Product support knowledge is identified as a synthesis of product, task, and user knowledge. It is shown that product support virtual documentation forms the link between the different product support knowledge elements and is the medium that enables provision of user-tailored product support related information to the user. The explicit description of product support knowledge elements enables the modular development of knowledge-based systems construction.

The investigation of the definitions found in the literature illustrates that knowledge is captured in terms of specific semantics. Therefore, semantically rich modelling and representation of product support knowledge is an essential part of product support system creation.

CHAPTER 4

SEMANTIC MODELLING OF PRODUCT SUPPORT KNOWLEDGE

This chapter addresses the second objective of this research. It introduces specialised notions (i.e. knowledge-specifier, and arc) for describing product support knowledge. These notions are used to construct the architectural and functional models of product, task, and user knowledge. Next, product support virtual documentation is represented by means of an ontology that integrates the aforementioned models with the documentation components.

4.1 SCOPE

Traditionally information design included two distinct views for building systems, the user and computer ones. The user view refers to the definition of data in the form of reports and screens that aid individuals to do specific tasks. The computer view is described in terms of file structures for storage and retrieval. The need for advanced flexibility in system design however led the ANSI/X3/SPARC Study Group on Database Management Systems to decide that a conceptual definition of data is needed. That advanced the development of semantic modelling techniques, which define the meaning of symbols and expressions (Stefik 1995). The semantic models developed define how the stored symbols relate to the real world.

As explained in the previous chapter, the process of developing a product support system is similar to that of implementing a knowledge-based system, where semantic models are used to guide the generation of the knowledge base.

The objective of this chapter is to provide common means of communicating the knowledge needed in developing product support systems knowledge bases. In order to do that, the following challenges have to be faced.

- Identify the notions that can be used to represent product support knowledge (e.g. concept, relation) and analyse their semantics.
- Identify and model the relations between different documentation components.
- Define the relation between documentation and product, task, and user knowledge, and the way that these can be combined in order to enable the delivery of accurate and up-to-date information to the user.

4.2 PRODUCT SUPPORT KNOWLEDGE MODEL

4.2.1 Architectural model

The *architectural model* is a formal representation of the structure of supported products, tasks, and users. It is application-independent and aims to organise data in a way that ensures homogeneity and validity of the resulting information. In this work data is considered a symbolic representation of facts with meanings. For example, the symbolic representation of a “telephone number” could be numerous variable digit numbers. In product support, raw data includes CAD models, bills of materials, drawings, assembly sequences, and user attributes. Part of the architectural model, depicted in Fig. 4.1, are the squares named “Product”, “Assembly”, “Subassembly”,

“Part”, and “Task”, as well as their associations. The squares in Fig. 4.1 represent concepts. Placing a square within a bigger one represents an “is-a” relation. The connectors between different concept hierarchies are represented either with bold straight lines or as bold curved lines. The straight lines that link the black squares (instances) are connectors linking different instances. **THING** (the most abstract concept) is the root concept. Ontologies are used due to the formality and richness of ontology-based modelling. This approach takes advantage of some widely accepted notions in KE, such as concept, instance, and relation.

Concept is a class of objects from the real world that have certain properties in common (Stuckenschmidt and Van Harmelen 2005). Take the example of having several different cars such as Ford Mondeo, BMW 330i, and Peugeot 206 ((A) in Fig. 4.1¹). All of them can be described by the general term “car” (B), which includes the characteristics that all of them share.

Instance is something that specifies a concept by illustrating a real world object that belongs to that concept, such as Peugeot 206 (A) for the notion of cars.

Relation is an attribute shared by objects in the subject domain that links and/or constrains them. There are two basic ontologically based relations, generalisation and specialisation. For example “car is-a vehicle” (C) is a generalisation type of relation.

Associating different concepts with the “is-a” relation and developing abstraction hierarchies is not enough in the case of a product support system. A relation that

¹ For simplicity, all references using capital letters in the rest of section 4.2 relate to Fig. 4.1.

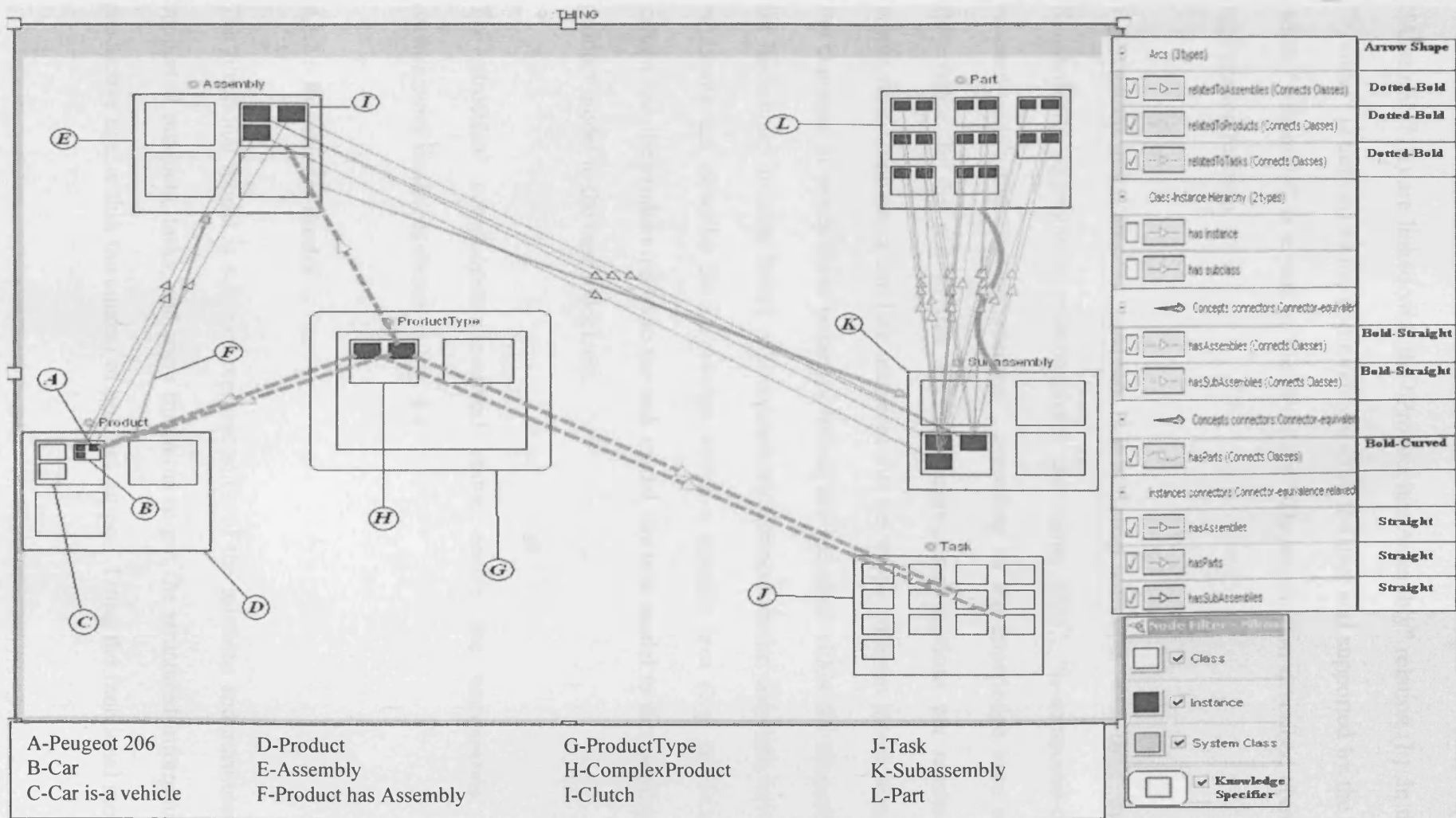


Figure 4.1. A fragment of the developed knowledge base

connects different hierarchies is also needed. For example “Product” (D) and “Assembly” (E) are linked with the “Product has Assembly” relation (F). In this case “Product” is defined as the commodity sold to the user and supported by the system, while “Assembly” is a part of the “Product”. The connection described above is an aggregation relation.

Aggregation relations are used to link different concepts and their instances. Syntactically aggregation relations have the values “has”, “is-composed-of”, “is-realised-with”, and “has-doc-element”, according to the knowledge base elements they relate. In the rest of this work all aggregation relations are referred to as *connectors*. Connectors can link instances that belong to different hierarchies, only if the concepts to which these instances belong are included within the hierarchies and the hierarchies are also linked with connectors. Connectors can only link information structures that describe the knowledge within a specific area (e.g. product). They cannot link the product model to the task model, the task model to the user model, the product model to the user model, etc.

The structural components described above enable the composition of the *architectural model*, as shown in Fig. 4.1.

4.2.2 Functional model

The functional model is a formal representation of the relations and attributes of the supported products, tasks, and users that aims to put the structured information into productive use, within the context of product support. Using the functional model, the

presented architectural model is enriched by employing two new notions, *knowledge-specifier* and *arc*, which can be further used in defining predicates, rules, and cases.

Knowledge-specifier is a property that is considered significant within the application domain (i.e. product support) and therefore is represented as a concept. For example, it may be useful to know whether a product is complex or not, since that can define the way in which the product support is adapted. Therefore, a knowledge-specifier called “ProductType” (G), which defines product’s complexity, is introduced. The level of the product support is determined using reasoning techniques that include “ProductType”, such as Predicate 1.

$((\text{ProductType} = \text{complex}) \text{ AND } (\text{User} = \text{novice}) \text{ AND } (\text{Task} = \text{complex})) \Rightarrow \text{ProductSupport} = \text{detailed}$
--

Predicate 1. Knowledge-specifier within a predicate

Predicate 1 states that if the product is complex, the user is novice and the task is complex, then the product support solution should be detailed.

Arc is a relation that links the knowledge-specifiers with other concepts. For example, “ComplexProduct” (H) can refer to concept “car” and “clutch” (I). Furthermore, knowledge-specifiers and arcs are used to relate different knowledge areas. In Fig.4.1, the task knowledge (part of which is the task (J)) is related to the product knowledge (composed by product, assembly, subassembly, and part concepts) via “ProductType”. The knowledge-specifier (“ProductType”) is shown as a rounded square and the arcs that link it to the other concepts are represented by bold dotted lines. The arcs in this case establish a constraint expressed by Predicate 2.

$$\text{Arc}(\text{ProductType}, \text{Task}) = \text{false} \Rightarrow \text{ProductSupport} = \text{null}$$

Predicate 2. Arc within a predicate

Predicate 2 states that if there is no arc between the knowledge-specifier and the task, then the product support system should not present a solution to the user. This corresponds to the case when both the product and task are too complex for the user and (s)he would not be able to perform the task even with the help of the system (e.g. assembling a car). Such a situation is considered a safety hazard and should not be allowed. More complex predicates can be developed that take in consideration the status of the user and relate it to the above arc, which decides the kind of support given (see Predicate 3).

$$\begin{aligned} & ((\text{Arc}(\text{ProductType}, \text{Task}) = \text{true}) \text{ AND } (\text{ProductType} = \text{complex}) \\ & \text{ AND } (\text{User} = \text{novice}) \text{ AND } (\text{Task} = \text{complex})) \\ & \Rightarrow \text{ProductSupport} = \text{detailed} \end{aligned}$$

Predicate 3. Arc and knowledge-specifier within a predicate

The notions introduced enable the creation of the system's functional model. The combination of the architectural and functional models results in the development of the system's *knowledge model*.

4.3 PRODUCT SUPPORT ELECTRONIC DOCUMENTATION

4.3.1 Product support electronic documentation components

Electronic documentation is one of the most important parts of a product support system. However, in order to associate the documentation with the rest of the product

support knowledge a model that delineates the documentation elements, metadata, and semantics has to be introduced. The notion of the Information Object (IO) has been widely used in product support (Pham and Setchi 2003, Pasantonopoulos 2005) and is therefore selected in this work. In addition, the IOs need to be organised into a structure that will enable direct interfacing with the rest of the product support knowledge in an organised manner. The notion of Information Object Cluster (IOC) is introduced for that purpose, as explained in section 4.3.1.2. Fig. 4.2 illustrates an abstract model that outlines virtual documentation components and their relations.

The IOs, IOCs, and VDs can be related to each other with structural and referential relations. Structural relations constitute the structure of the hypermedia. For example, a template that includes a title at the top of a hypermedia document, the main content in the middle and a conclusion at the end, defines structural relations among the title, main content, and conclusions sections. Referential relations define cross-reference linkage criteria. For example, links between different hypermedia pages are considered referential relations.

4.3.1.1 Information object

An Information Object is a finer grained element of a virtual document. It is defined as “a data structure that represents an identifiable and meaningful instance of information in a specific presentation form” (Setchi 2000). An IO can therefore be a picture that illustrates a part of a product or a textual description. In this study IOs are characterised according to their form, behaviour, type, level of detail, and theme.

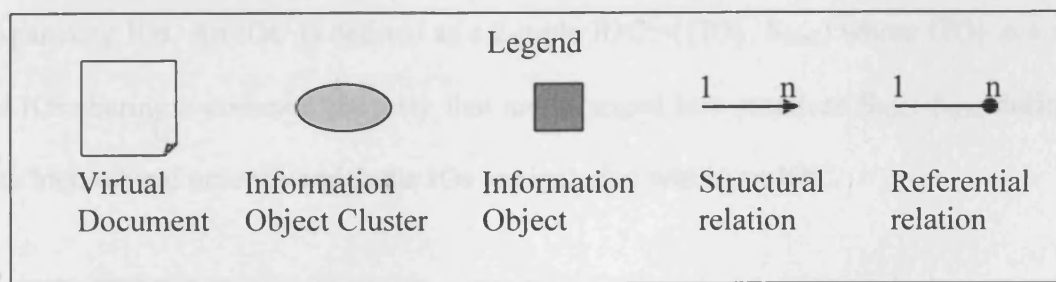
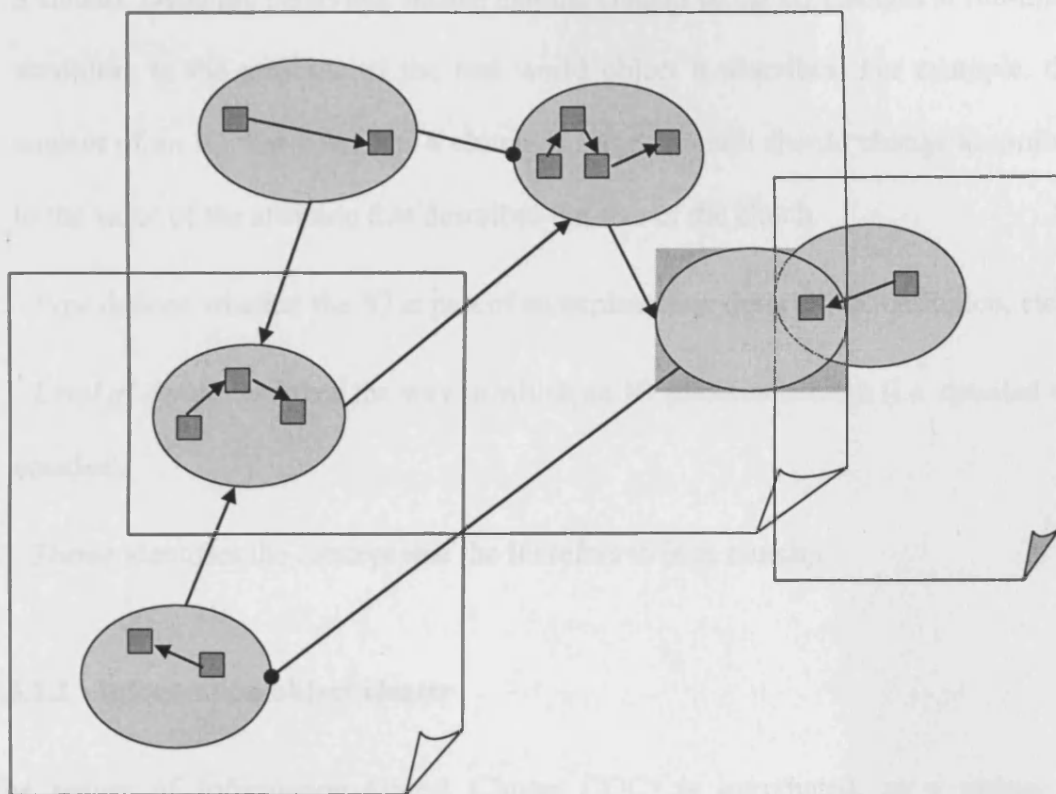


Figure 4.2. Model of virtual documentation components and their relations

1. *Form* indicates whether the IO is text, image, animation, video, audio, or a virtual reality model.
2. The IO can have two forms of *behaviour*, namely static or dynamic. Static behaviour indicates that the content of the IO does not change (e.g. the definition of a clutch). Dynamic behaviour means that the content of the IO changes at run-time, according to the attribute of the real world object it describes. For example, the content of an IO that describes a clutch as large or small should change according to the value of the attribute that describes the size of the clutch.
3. *Type* defines whether the IO is part of an explanation, description, definition, etc.
4. *Level of detail* describes the way in which an IO presents content (i.e. detailed vs. concise).
5. *Theme* identifies the concept that the IO refers to (e.g. clutch).

4.3.1.2 Information object cluster

The notion of Information Object Cluster (IOC) is introduced, as a means of organising IOs. An IOC is defined as a 2-tuple $IOC := (\{IO\}, S_{IOC})$ where $\{IO\}$ is a set of IOs sharing a common property that are arranged in a structure S_{IOC} . S_{IOC} defines the hierarchical order in which the IOs are included within an IOC.

Theme, level of detail, and type are used to identify all the IOs that belong to the same IOC. For example, if a query involves the concept of a clutch and is initiated by a user identified by the system as “novice”, the selected IOC will contain IOs that have as a theme “clutch”, level of detail value for novice users (e.g. “detailed”) and types that are included when the user is novice, such as “explanation” and “comment”.

The IOC's model (Fig. 4.3) contains three different IOC categories; *product IOC*, *task IOC*, and *reference*. Product IOC is chosen according to the theme of the query and is related to a concept (derived from the product ontology) in the knowledge base that describes a product component. Task IOC is accordingly mapped to a task that is obtained from the task ontology. Reference represents system data oriented information that can be used to instantiate elements (e.g. the unique ID of a concept within the knowledge base that can be used by the technical writer in order to identify which documentation element needs to be manually redefined).

Fig. 4.3 demonstrates that the S_{IOC} is different for each of the three different classes of IOCs. In the case of a product IOC the order of the IOs types is as follows: definition, description, explanation, comment, and example. A task IOC includes the procedure, explanation, comment, and example types. The reference involves facts, data, and numbers. S_{IOC} also conforms to presentation rules (e.g. illustrations should be presented in conjunction with their relevant text so as to be seen at the same time as the text is read) (BS4884-2 1993, BS4899-2 1992).

4.3.1.3 Product support virtual document

A Virtual Document (VD) is generated by aggregating IOCs and is defined as a 2-tuple $VD := (\{IOC\}, S_{VD})$ where $\{IOC\}$ is a set of IOCs sharing a common property and which are logically organised, with a structure (S_{VD}), for composing a document (VD). S_{VD} defines the hierarchical order in which the IOCs are included within a document. The IOCs are selected and organised according to their theme, level of detail, and type, as follows.

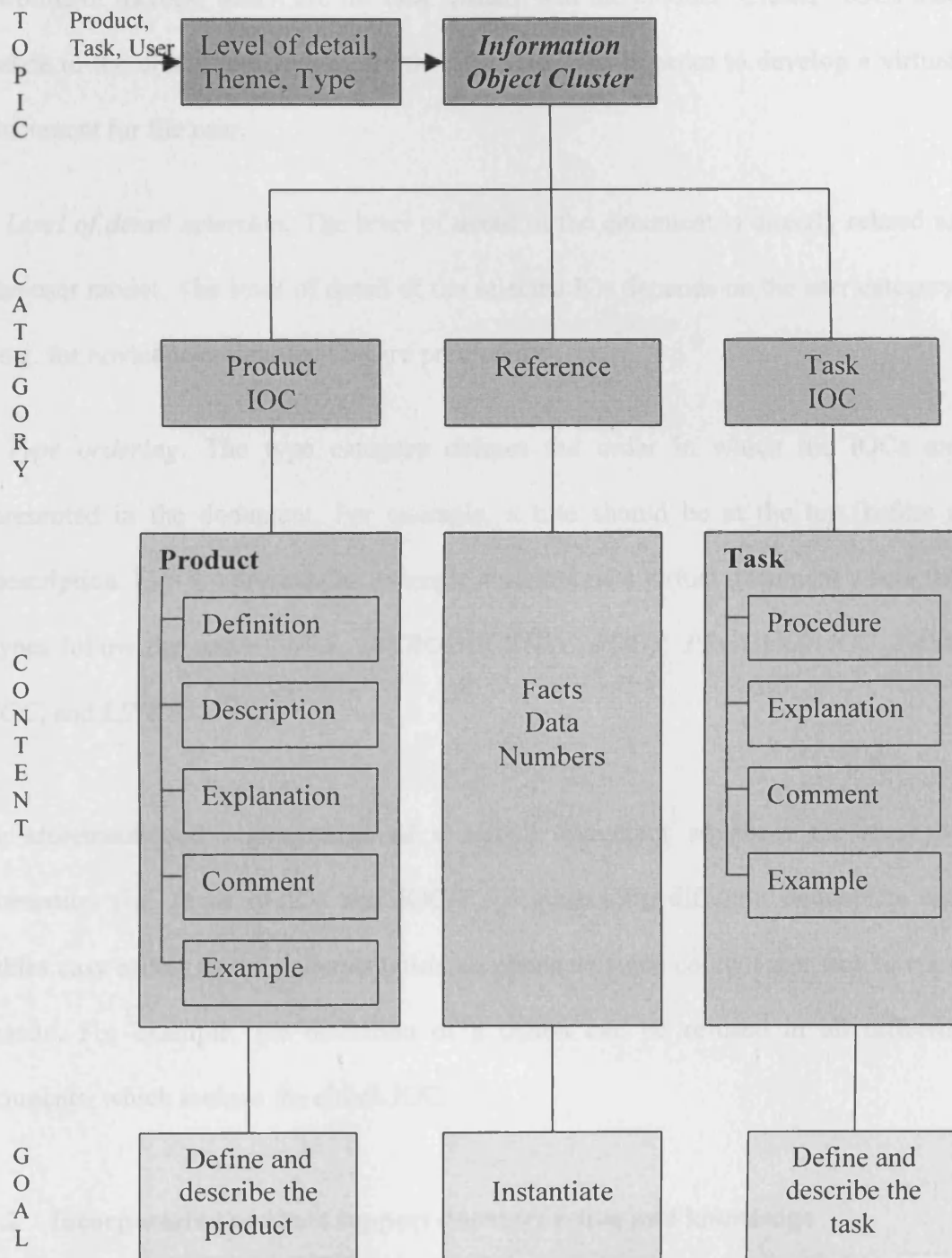


Figure 4.3. Information object cluster's model

1. *Theme configuration.* The combination of different themes that is required within the same document. For example a document about the installation of a clutch, has two major themes, which are the task 'Install' and the product 'Clutch'. IOCs that relate to the clutch installation are therefore required in order to develop a virtual document for the user.
2. *Level of detail selection.* The level of detail of the document is directly related to the user model. The level of detail of the selected IOs depends on the user category (e.g. for novice user detailed IOs are presented).
3. *Type ordering.* The type category defines the order in which the IOCs are presented in the document. For example, a title should be at the top, before a description. Fig. 4.4 presents an example structure of a virtual document where the types follow the order *TITLE, INTRODUCTION, BODY, PRODUCT IOC, TASK IOC, and LINKS.*

The aforementioned segmentation of a virtual document advances the reuse of information (i.e. reuse of IOs and IOCs) for generating different documents and enables easy access to the documentation components since content and structure are separate. For example, the definition of a clutch can be re-used in all different documents, which include the clutch IOC.

4.3.2 Incorporating product support documentation and knowledge

The knowledge base of a product support system comprises the knowledge model (section 4.2), populated with instances and their relations, and the product support

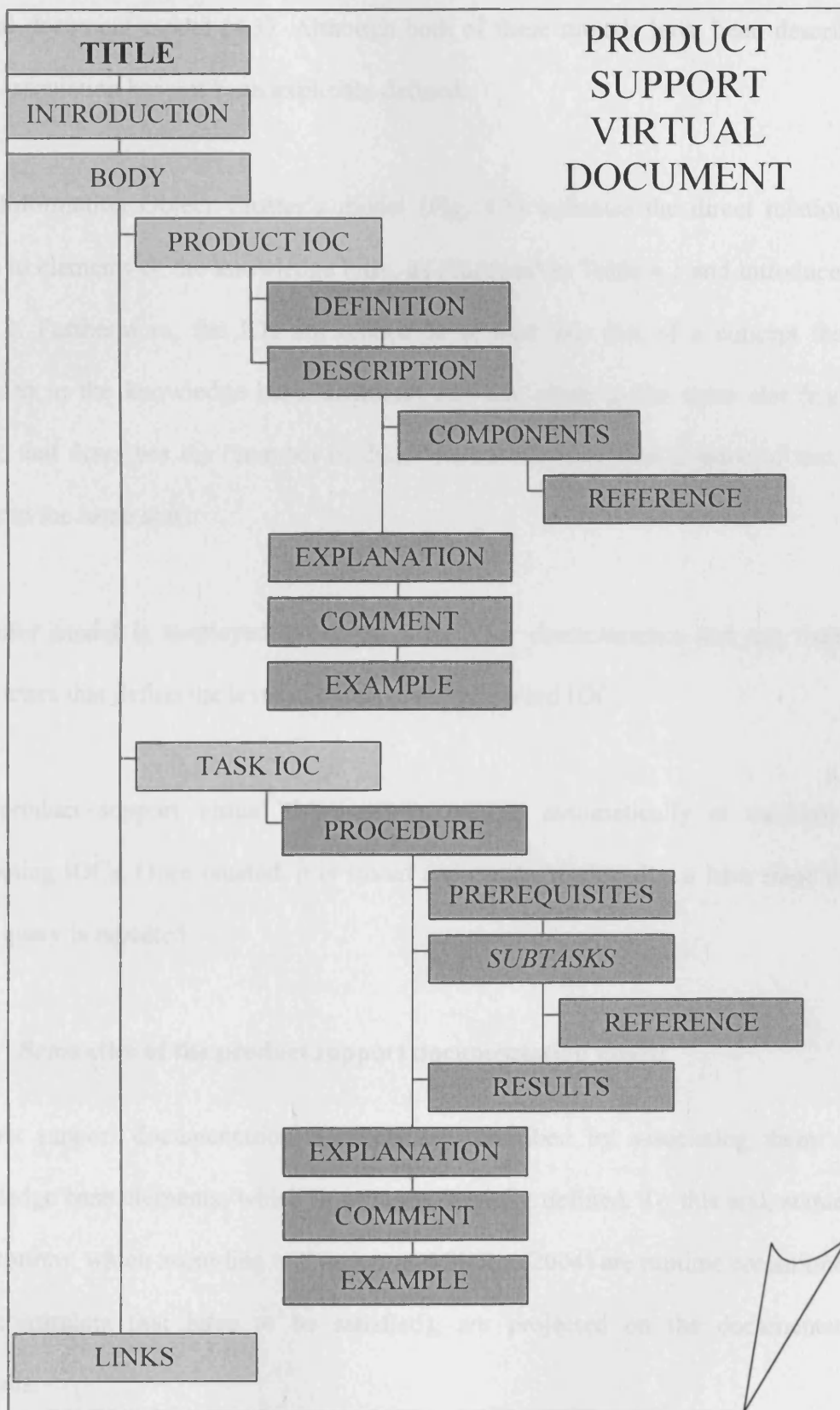


Figure 4.4. Structure of a product support virtual document

virtual document model (4.3). Although both of these models have been described, their association has not been explicitly defined.

The Information Object Cluster's model (Fig. 4.3) indicates the direct relation of IOCs to elements of the knowledge base, as illustrated in Table 4.1 and introduced in 4.3.1.2. Furthermore, the IOs are related to at least one slot of a concept that is included in the knowledge base. Different IOs can relate to the same slot (e.g. an image that describes the "number of disks" slot is different than a piece of text that refers to the same slot).

The user model is employed to represent the user characteristics and use them as parameters that define the level of detail of the presented IOC.

The product support virtual document is created automatically at run-time by combining IOCs. Once created, it is stored and can be retrieved at a later stage if the same query is repeated.

4.3.3 Semantics of the product support documentation model

Product support documentation elements are described by associating them with knowledge base elements, which in turn are formally defined. To this end, semantic implications, which according to Sanchez and Sicilia (2004) are runtime commitments (i.e. constraints that have to be satisfied), are projected on the documentation elements.

4.3.3.1 Abstraction (generalisation-definition)

Generalisation and specialisation, which are described below, are very often considered inverse relations. However, both of them can be used to derive the notion of abstraction in the design of a product user interface document by hiding knowledge details, which the system cannot provide or in which the user is not interested. This is illustrated in Table 4.1.

Table 4.1. Documentation and knowledge base elements

Documentation element	Knowledge element
Information Object Cluster	Concept
Product IOC	<ul style="list-style-type: none"> • Mapped to a concept that represents a supported product. • Related to a specific user stereotype by utilising user characteristics as input parameters for defining the level of detail and themes.
Task IOC	<ul style="list-style-type: none"> • Mapped to a concept that represents the task that the user wants to perform. • Related to a specific user stereotype by utilising user characteristics as input parameters for defining the level of detail and themes.
Reference	Related to the concept's instantiation (the concept represents either a product or a task).
Information Object	Slot
Any	Mapped to a slot that belongs to the concept described by its corresponding Information Object Cluster.

4.3.3.1 Abstraction (generalisation/specialisation)

Generalisation and specialisation, which are described below, are very often considered inverse relations. However, both of them can be used to induce the notion of *abstraction* in the design of a product support virtual document by hiding knowledge details, which the system cannot provide or in which the user is not interested. This entails that an abstracted concept includes more information than any of its specialisations but includes less particulars.

- *Generalisation*

Assuming the information represented by source concepts S_1, \dots, S_n is respectively D_{S_1}, \dots, D_{S_n} and the information represented by target concept T is D_T , then **generalisation** is a *1...n:1* relation between the source concepts and the target concept that satisfies the following:

$$S_1, \dots, S_n \mapsto T \quad (4.1)$$

$$D_{S_1} \subset D_T, \dots, D_{S_n} \subset D_T \quad (4.2)$$

$$D_{S_1} \cup \dots \cup D_{S_n} \subseteq D_T \quad (4.3)$$

- *Specialisation*

Assuming the information represented by target concepts T_1, \dots, T_n is respectively D_{T_1}, \dots, D_{T_n} and the information represented by source concept S is D_S , then **specialisation** is a *1:1...n* relation between the source concept and the target concepts, that satisfies the following:

$$S \mapsto T_1, \dots, T_n \quad (4.4)$$

$$D_{T_1} \subset D_S, \dots, D_{T_n} \subset D_S \quad (4.5)$$

$$D_{T_1} \cup \dots \cup D_{T_n} \subseteq D_S \quad (4.6)$$

Based on the aforementioned discussion abstraction can be the enabler for satisfying one of the most important requirements for creating a product support virtual document, which is availability. Availability stands for the ability of the system to provide a resource (in this case an IOC) that has been used in order to answer a user's question. If for example a question Q requires an information object cluster A , the following can occur.

1. Information object cluster A is effectively available with all needed details included in its body (all requested information objects are also available). This is the optimum case where all resources are available just-in-time and just-in-place.
2. Information object cluster A is effectively available without all needed details included in its body (not all requested information objects are available). This possibility will be discussed in more detail in the next section. It can be mentioned however, that a virtual document model can be created even without all information objects being available.
3. Information object cluster A is not effectively available. In this case, abstraction, as described in this study, can be utilised for delivering an IOC,

B, which does not contain all required details but provides (at least temporarily until a more detailed solution becomes available) a more general view of the concepts that the query involves. The abstracted IOC *B* (IOC mapped to the super-concept of the concept that is related with *A*) represents more information than the requested IOC *A* (including the information that *A* represents) but in smaller detail. Therefore, *B* should be considered only as a temporary solution, which should be eventually replaced with *A*.

Except for the implications that abstraction has on availability, it also constrains the type and the level of detail of the abstracted IOC that is delivered to the user. These should have the same value for both the abstracted and requested IOCs. For example if *A* should include the IO types “definition” and “description” with level of detail “detailed” then *B* should have the same values for these attributes. This is a consequence of the unchanged user classification, which defines and level of detail of the IOC, while the theme (related to the task or product concept) is adjusted.

4.3.3.2 Connector

The connector can take several values as explained in section 4.2.1, including *has*, which is used to describe the relation between product components and their aggregates, *is-composed-of*, which is used to characterise the relation between subtasks and their aggregates, *is-realised-with*, which is reserved for context-domain aggregation relations, and *has-doc-element*, which expresses the connections between the elements of a product support virtual document and their aggregates. Connector therefore according to its syntactic value (e.g. *has* as opposed to *is-composed-of*) can

be utilised differently by the system. However, independently of its syntactics it retains common semantics for all its different types.

In product support documentation (in terms of the model presented in this study) the connector expresses the ability of an IOC A that is mapped to a concept S_I to contain IOCs that are linked to a concept T_I , when T_I is a component of S_I (i.e. they are associated with a connector). For example, if A describes the concept “clutch”, IOC B the concept “housing”, and IOC C a “bolt”, then A could contain B , which in turn could include C . This association has immediate effect in the case when an IOC A (mapped to concept S_I) is not effectively available or its description part is missing, since it can be created as an aggregation of the IOCs that are related to the component concepts of S_I . However, in order to form a complete description of A not only components’ IOCs are needed but also knowledge about the components’ connections, in order to form S_I , is required. Hence, a description that is based only on the connector relation should be considered only a temporary solution, which should be eventually replaced by a manually elaborated version of A .

Connector imposes the same constraint on the type and level of detail of an aggregated IOC as generalisation does on an abstracted IOC, meaning that both of the aforementioned attributes should have the same values for both aggregated and requested IOC. However, when compared to an abstracted IOC a major difference of the aggregate IOC is that the theme does not change.

In addition, although a component’s IOC C can be a part of several composite IOCs (advancing reuse), if all the concepts that relate to the aggregated IOCs are deleted (e.g. product components and/or products can become obsolete) and no new concepts

are related to *C*'s mapped concept, then *C* is considered redundant and should be also deleted.

Connectors between instances determine the availability of IOs. For example, within an IOC that represents a disk clutch there are IOs that are mapped to the attribute "number_of_disks". If this attribute takes the value "1" then IOs that describe a single-disk clutch are required and in particular IOs that stand for the single disk clutch instances. Alternatively, if the value is "2" IOs for double-disk clutches (and instances) are required. However, if instances of double-disk clutches are not included in the knowledge base (either because they were never created or have been deleted), IOs that relate to them become also redundant.

Nevertheless, in the current environment that advances production of complex and customised products, instances alter continuously, which means that IOs need also to be added, removed, or changed constantly. The manual labour and effort for responding to such requirements could be excessive. Therefore an automated way is sought for creating IOs. The attribute that states whether an IO is dynamic along with connectors can be used as enablers of such an approach. The dynamic state if an IO is utilised for producing IOs that have abstracted bodies. For example, if three IOs form the following sentence, "(IO1 → Turbo-type disk clutch) (IO2 → 2 disks) (IO3 → its version number is 0234)" then instead of designing the above IOs as normal textual fragments, they are designed as follows "(IO1 → [Type] disk clutch) (IO2 → [number] disks) (IO3 → its version number is [version_number])". [Type], [number], and [version_number] are the dynamic parts of the IO, which are filled according to the values of the current instance. The range of the dynamic parts is determined either by the allowed value given to the attribute at the concept-level (constraints set on slot

value) or by the identified values of the instances retained in the knowledge base. The notion of abstracted IOs signifies that the same IO can answer several queries, as long as the changes reflect differences in the IO's dynamic parts.

4.3.3.3 Arc

The arc associates task and product concepts to knowledge-specifiers defining parameters such as “product is complex” (section 4.2.2). This means that an arc can be used to define the type and/or level of detail of the IOC/IO (if any of the defined parameters are utilised in document adaptation process). Furthermore, the IOC/IO can be either available or unavailable according to the passed parameters. For example, if an IOC should not be presented in any case to the user because of a safety critical parameter (e.g. the product is too complex for any user to maintain) then the IOC is not created.

4.3.3.4 Knowledge-specifier

Except for the relations the knowledge-specifier also imposes constraints on IOCs, since it forms a parameter that defines the presentation of an IOC as explained in the previous paragraph (see arc).

As described in this section the association between documentation and knowledge base elements has several implications on product support virtual documentation. These are summarised in Table 4.2.

4.4 SUMMARY AND CONCLUSIONS

This chapter discusses the development of knowledge-based Product Support

Table 4.2. Effects on availability, level of detail, type, and theme of IOCs and IOs from the semantic analysis of knowledge base constructs

Knowledge base element		Documentation element	
Relation	Concept	Information Object Cluster	Information Object
Generalisation/ Specialisation (Abstraction)	-	Abstracted IOC → Queries can be answered by abstracting requested but unavailable IOCs; Level of detail and Type remain unaltered; Theme changes.	-
Connector	-	Aggregate IOC → Queries can be answered by providing the components of the requested but unavailable IOCs; Level of detail, Type, and Theme remain unchanged but the themes of aggregate IOCs are also included.	Abstracted IO → Queries can be answered by utilising the dynamicity of IOs in order to design them as abstracted elements; Theme, Level of detail, and Type remain unchanged.
Arc	-	Defines if an IOC/IO should be available according to safety constraints; Level of detail and type are determined from the parameters passed through arc; Theme remains the same.	
-	Knowledge-specifier	Defines if an IOC/IO should be available according to the constraints or parameters it represents; Level of detail and type are also determined; Theme remains unchanged.	

documentation modeling by representing the structure of knowledge in a complex

4.4 SUMMARY AND CONCLUSIONS

This chapter advocates the development of knowledge-based Product Support Systems (PRSS) by semantically modelling and formally describing the knowledge that should be contained within the knowledge base of a PRSS. An architectural model is developed with the objective of organising data in a way that ensures homogeneity and validity of the resulting information, which is followed by a functional model that aims to facilitate the productive use of the structured information. Specialised notions are introduced (knowledge-specifier and arc) for formalising the acquired knowledge and applying it in the context of product support.

Product support virtual documentation is identified as an aggregation of entities called Information Objects (IOs). A new documentation concept is introduced called Information Object Cluster (IOC). IOC aggregates a number of IOs and advances meaningful reusability by being related to domain-specific concepts. This form of semantics-based assignment enables the view of a product support virtual document as a modular unit that contains several reusable IOs and IOCs, which are directly linked with the architectural and functional models.

The knowledge model introduced in this chapter is based on an ontology that can be shared and reused by product support systems. It introduces the structural components needed, the links between them, and establishes their usability. The developed ontology facilitates interoperability and seamless content exchange between product support and other knowledge intensive fields (i.e. product, task, user, and documentation modeling) by representing the elements of knowledge in a machine

processable way. Furthermore, it advances the provision of adaptive support (as illustrated with the predicates).

The semantic description of documentation elements and their direct relation with ontology concepts and slots has direct implication on IOs and IOCs. The effects include constrained values for specific qualitative attributes (i.e. type and level of detail), and facilitation of automatic transformation of content for display online.

The ontology-based representation of product support knowledge and the delineation of its underlying semantics enable the unification of the knowledge base component of a PRSS. This forms a natural step towards the creation of a framework for knowledge-based product support systems.

CHAPTER 5

A KNOWLEDGE ENGINEERING FRAMEWORK FOR PRODUCT SUPPORT SYSTEMS

This chapter addresses the third objective of this research. To do that, the synergy between product support, problem solving, and knowledge engineering is first studied and a structured problem solving approach is proposed that integrates product support systems and artificial intelligence techniques. The approach includes defining and classifying product support problems, selecting different reasoning techniques for different types of problems, and introducing a multi-modal strategy that combines case- and model-based reasoning. A framework based on that approach is also introduced..

5.1 A PROBLEM SOLVING PERSPECTIVE ON PRODUCT SUPPORT SYSTEMS

5.1.1 Problem solving and product support

Product support may be needed in different circumstances. These may range from situations that involve performing repetitive routine actions by following simple instructions, to cases requiring decision making. The demand for product support decreases when the operations that have to take place have been repeated for several times by the same user(s) within the same circumstances. In contrast, if a new

sequence of actions has to be designed or if a user has never performed a certain task before, then he/she will need more support.

The aforementioned distinction resembles having expert or novice users. Numerous studies have been conducted on changes in performance that occur as an individual is transformed progressively from a novice to an expert (Kahney 1992). Experiments that were carried out (Kahney 1992, Gunzelmann 2003), revealed that experts not only have more knowledge on particular subject areas compared to novices, but they also follow different problem solving strategies and represent problems differently.

The efficiency with which humans can solve even complex problems in relatively little time indicates that the reasoning mechanism of a product support system should resemble the operation of the human mind. Therefore, as with humans, problem representation and solution generation should be reconfigured according to the kind of problem that the PRSS is asked to solve (i.e. previously solved or new problem).

The differentiation between (i) attempting to solve new problems, and (ii) recalling previously solved problems and reusing/adapting their solutions, has also been widely addressed in the medical and human-interaction domains. For instance, MacMullin and Taylor (1984) define problems as familiar and new patterns while Gray (2001) distinguishes problems into new or unique, and previously solved ones. As Fig. 5.1 shows, the occurrence of a previously solved problem only requires reusing knowledge generated in previous cases, while attempting to solve new problems, involves both knowledge acquisition and knowledge creation. This shows that a

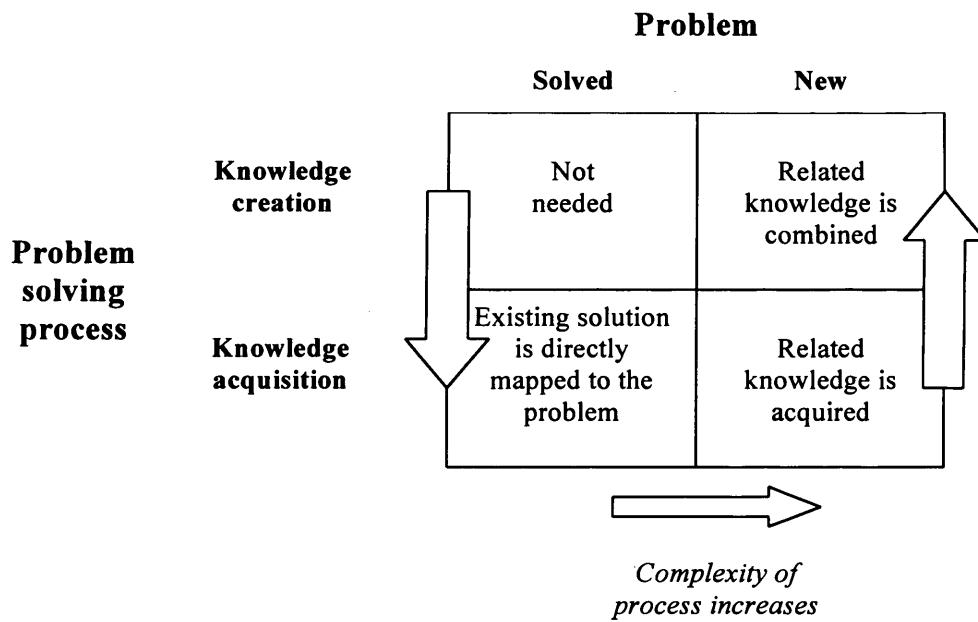


Figure 5.1. Association between problem solving and types of problems

product support system should provide solutions for both types of problems, and enable operations in support of both knowledge acquisition and knowledge creation.

However, before the problem solving process can commence, problem recognition is required. Gray (2001) describes a knowledge management framework where the detection of a problem precedes the solution process. Liao (2002) focuses on new problems and claims that the initial state of problem solving is to assess the current situation i.e. problem contextual description, and Bigus and Bigus (2001) claim, “knowing what problem you are trying to solve is one of the elementary maxims for AI”. In order to enable problem discovery product support problems are defined and consequently classified in the next section.

5.1.2 Product support problems

5.1.2.1 Product support problem definition

The knowledge that a product support system should deliver to the user is tightly linked to the query performed. There are two basic qualities that characterise a query, namely its content and context. The content is going to be relevant to the product that is supported and/or the task that the user wants to perform, while the context is determined according the user characteristics and the system’s usage. The query therefore should contain what is needed (elements of knowledge that are missing), why it is needed and under what circumstances (context).The given definition of a Product Support Problem (PSP) contains all the identified elements and is represented as follows.

Definition 10. *Product Support Problem (PSP) is a 4-tuple $PSP := (MOD, HYP, CON, OBS)$ where:*

- *MOD* is a finite set that represents the product and task models in relation to the IOCs and IOs that form the documents.
- *HYP* is a finite set of combinations of elements of MOD representing possible documentation hypotheses.
- *CON* is the context that characterises the problem and contains the User Model (UM) in combination with the usage purpose.
- *OBS* represents the observations acquired by the current query and are mapped to elements of MOD and CON.

Definition 10 identifies PSPs as a specialisation of diagnostic problems, since a product support system recognises and solves PSPs in terms of the IOs and IOCs involved. Therefore, the problem solving process includes identifying that there is a fault (e.g. PSVD asked does not exist) recognising the type of fault (e.g. difference in configuration or missing IO, IOC), and choosing a strategy to be followed (e.g. provide the missing documentation element). Portinale et al. (2004) have expressed the outcome of a diagnosis process in relation to the observations acquired from a related query. Adapting that description a Product Support Solution (PSS) is defined as follows.

Definition 11. *Given a product support problem $PSP := (MOD, HYP, CON, OBS)$ and a set of observations $OB \subseteq OBS$, then valid hypotheses are represented by $VH \subseteq HYP$ if and only if, $MOD \cup CON \cup VH \vdash OB$. Product Support Solution (PSS)*

can be every hypothesis $h(y) \in VH$. Best solutions are considered the ones for which $OB=OBS$. A PSS can take several forms, including explanations, instructions, warnings, and descriptions, which are delivered through documents.

Take the example of two typical scenarios in product support systems. In the first scenario the user asks for help from the system because a fault has been identified to the operation of the product. In that case the MOD includes behavioural aspects of the product (behavioural model) and how these are associated with the task that the user should perform in order to fix the fault, as well as relevant documentation. The context is set to the user's class (e.g. novice). OB will include the fault provided by the user (part of MOD) and the context in which this fault has been identified (part of CON). VH will be the set of hypotheses that correspond to that problem.

In the second scenario the user requests more information about the product without providing a fault. MOD in that case includes the tasks for which this information is useful and a structural model of the product, in relation to IOCs and IOs. CON is again relevant to the user type while the purpose can be determined from the usage of the system (e.g. tool used). OB contains all the information that the user has provided in the given context via the query (i.e. more information-product-user type). VH will be again the inferences best matching the given problem.

5.1.2.2 Problem classification

In line with problem solving theory, PSPs are classified as previously solved/same, similar, and new ones.

Previously solved/same PSPs include problems that have exactly the same set of OBS. In that case VH can be reused for delivering an appropriate solution.

Similar PSPs are problems for which the set of new observations is not exactly the same as earlier ones but the differences can be compensated by changing IOs and/or individual IOCs. For example, consider a case where a user has already asked for information on installing a two disk clutch and the new query concerns installing a three disk clutch, in the same context. The only thing that changes is the number of steps that have to be repeated. If an IO corresponds to the description of each step in the two-disk clutch installation process, then another IO has to be added that depicts the extra information. Therefore, a single IOC is changed by adding the extra IO and modifying IOC's structure accordingly. In such cases, previous solutions are easily adapted.

New PSPs occur when the diversity between earlier observations and new ones, requires changing IOCs as well as their associations. For instance, if a user requires a description of a transaxle and there is only a description of a clutch, based on previous experiences, then the IOC that describes the clutch cannot be efficiently adapted in order to create the IOC that corresponds to the transaxle.

5.1.3 Problem solving approach

The problem solving process includes identifying the relevant elements of the modelled knowledge (see chapter 4) and mapping them to the corresponding IOCs and IOs for producing the most appropriate document, in accordance with the OBS

set. As depicted in Fig. 5.1, depending on the problem, the process should combine knowledge acquisition and creation procedures.

Knowledge acquisition in the presence of previously solved problems

The aim in this situation is to find a way to acquire solutions without repeating the reasoning process each time a query, same to a previous one, occurs. This means that reasoning from scratch is not desirable and therefore rule-based reasoning is excluded. Moreover, in order to establish that one PSP is the same as another, specific knowledge about each problem has to be compared. Since model-based reasoning focuses on general knowledge, case-based reasoning is deemed as the most appropriate technique in this scenario. Case-based reasoning caches old situations and solutions, avoiding reasoning from scratch and reducing the time needed for delivering a solution.

The representation of cases follows a problem-solution pair structure, where the features that each case includes correspond to elements of the product, task, user, and purpose models. The solutions are pre-calculated.

Knowledge adaptation in the presence of similar problems

Case-based reasoning provides means for reasoning by analogy and therefore adapting solutions according to previous experiences. This feature is highly utilised here, by substituting IOs with new ones that reflect the changes. IOs correspond to specific features of each case. The general knowledge that is contained in the case base is extended and the adapted case forms a new problem-solution pair.

Knowledge creation in the presence of new problems

The existence of a new problem that has little or no similarity to previous experiences cannot be easily managed by CBR alone. In this case a multi-modal reasoning strategy is adopted.

General knowledge about the application domains (e.g. product domain), is utilised supporting the case-based reasoning component. Models include features of the cases while they generalise cases' specific knowledge. For example, if the new query includes transaxle as a main component while previous experiences refer only to clutch, the model-based reasoning engine can search and find, that according to the general knowledge about the domain, both of them are considered as assemblies of a car, therefore, adopting the same structure of clutch's document is a possible solution. The structure of IOCs depends on the similarities between the clutch and the transaxle (e.g. both of them are developed as a combination of different parts) and the transaxle's attributes (retrieved from the product model) can define different IOs.

The difference in the solving process of different kinds of problems is depicted in Fig. 5.2, which is based on the main steps of the case-based reasoning cycle (Aamodt and Nygard 1995), and illustrates that the reasoning approach is a hybrid of case-based (main technique) and model-based reasoning (secondary technique). The stages of the process when a previously solved problem is detected (Fig. 5.2 (A)) include retrieving relevant cases, reusing them, presenting the best match as a proposed solution, evaluating this solution, and retaining it in the knowledge-base. If the problem is similar to a previous one (Fig. 5.2 (B)), an extra stage (i.e. solution adaptation) is needed, where case-based reasoning adaptation techniques can be utilised. Fig. 5.2

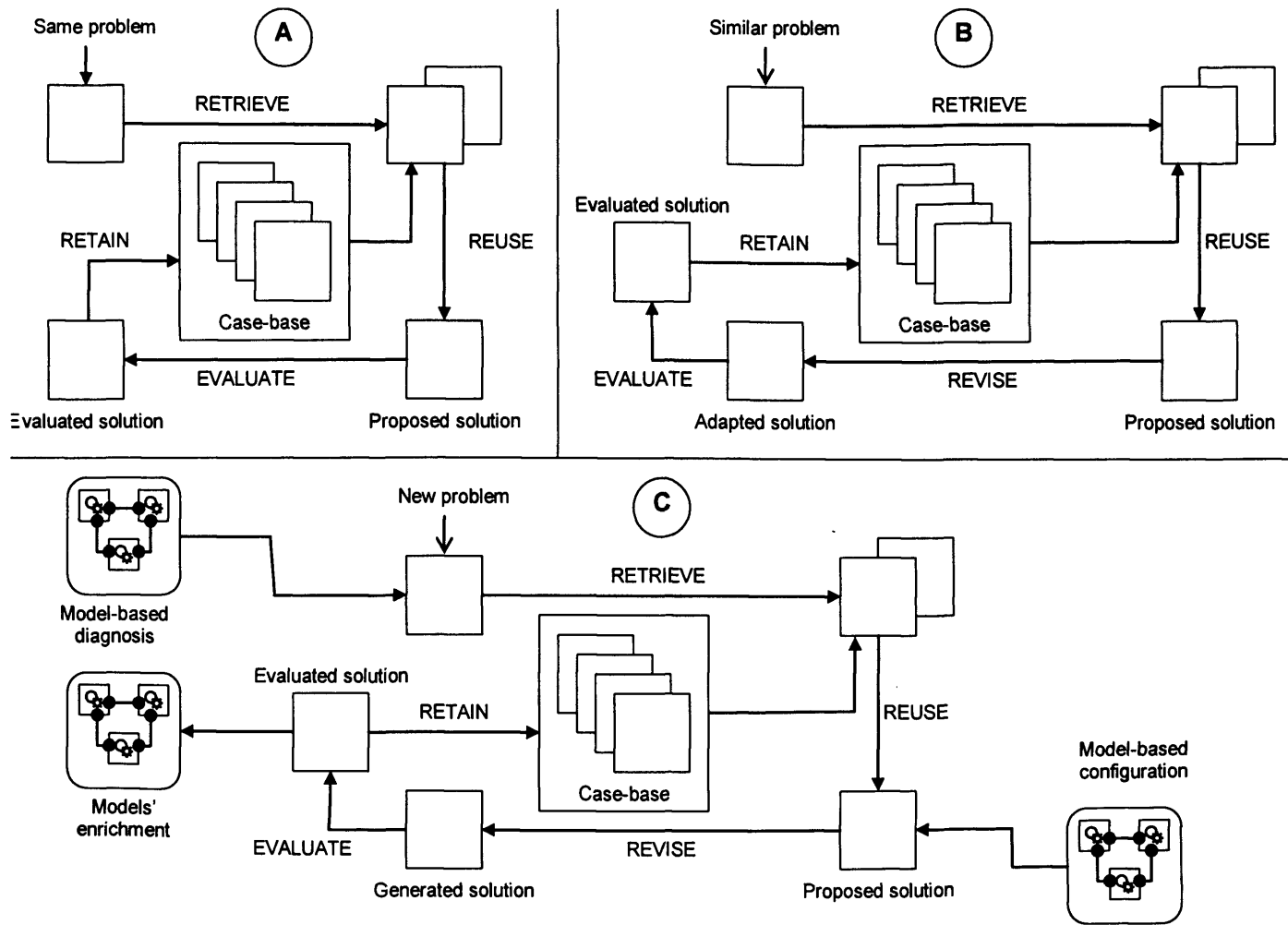


Figure 5.2. Solving approach for different kinds of product support problems

(C) shows the solving process for a new problem. Model-based reasoning is employed for identifying new problems (model-based diagnosis), and configuring the generated documents (model-based configuration) based on the product support virtual documentation model. Enrichment of the models employed is also possible.

5.2 A KNOWLEDGE-LEVEL VIEW OF A FRAMEWORK FOR PRODUCT SUPPORT SYSTEMS

5.2.1 General framework structure and operation

The framework comprises four different spaces, which are namely the data, problem, hypothesis, and solution spaces, which are further explained in subsequent sections.

The architecture of the framework is sequential, meaning that each space is involved in the process, when its preceding space has completed its operations. The sequential structure of the framework is enhanced with several feedback paths, which enable other advanced operations to take place, such as knowledge creation (Fig. 5.3).

The operation of the framework is a hybrid of loosely and tightly coupled processes. The data and problem spaces are tightly coupled, because knowledge from both spaces is used in order to generate hypotheses. Moreover, the feedback path that is initiated after a solution is derived updates them both. On the other hand, the hypothesis and solution spaces are loosely coupled, as the operations taking place within each one, are independent from operations happening in other spaces. This means that although each space serves a specific purpose, the existence of collaboration and communication between the different spaces is intensive and frequent, as the ultimate goal is to reach an optimal solution.

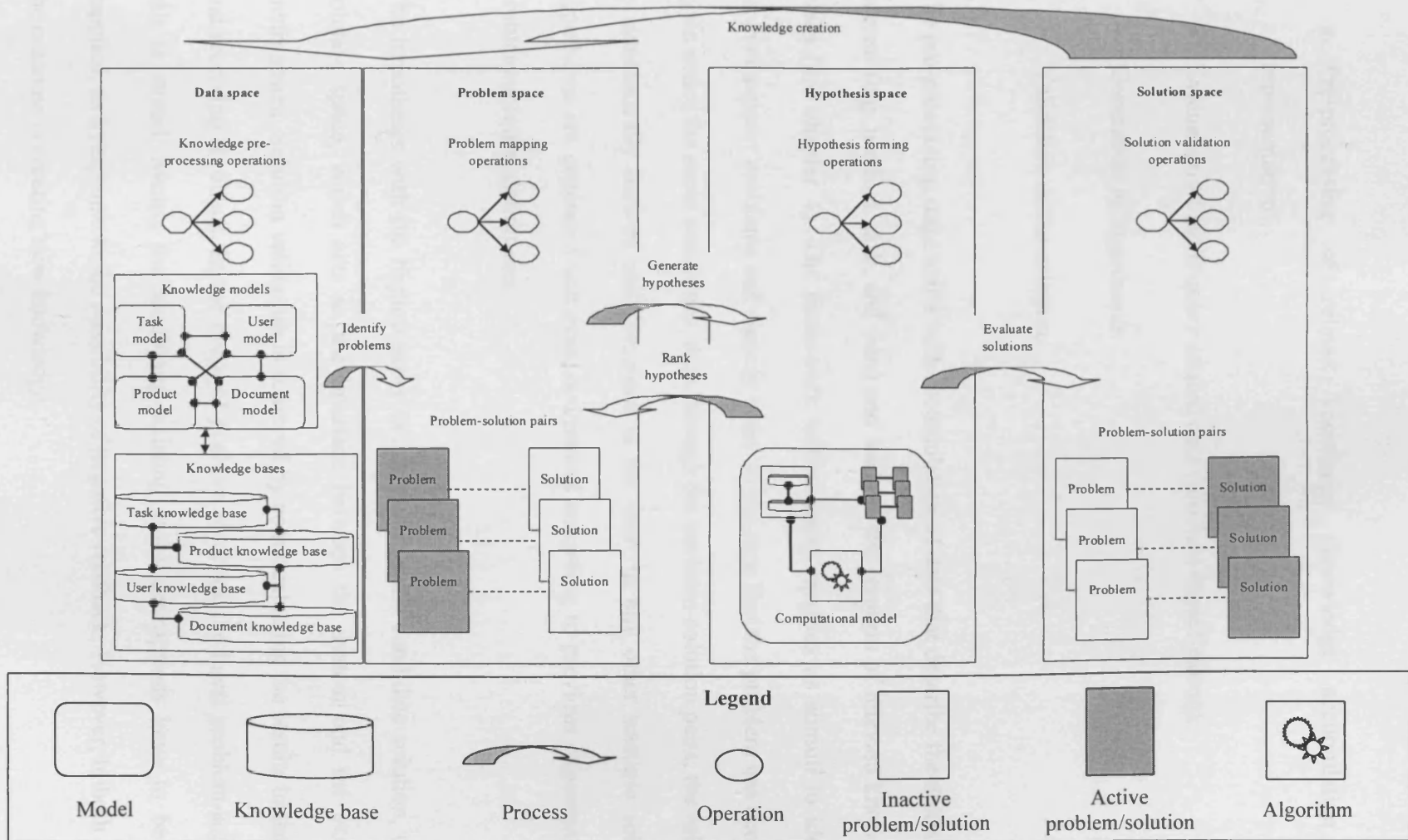


Figure 5.3. Knowledge-engineering framework for product support systems (knowledge-level view)

The operation of the framework can be segmented into the following major phases.

- Pre-processing of related knowledge (knowledge accumulation and representation).
- Evaluation of input/query related data (problem identification).
- Generation of hypotheses.
- Validation of the solution.

The pre-processing stage starts with accumulation of data that describe the domains of interest (e.g. product, task, and user) and leads to the creation of relevant knowledge bases (see chapter 4). The framework utilises users' queries as stimuli to identify product support problems and classify them. In the case that the problem has occurred again under the same conditions, then through the problem-solution pairs, the solution is automatically derived and presented to the user. In any other scenario solution hypotheses are generated and evaluated/ranked according to previous experiences of problem-solution definitions.

The hypothesis with the highest score is advanced as the candidate solution, to the solution space, which acts as the interface between the system and the external environment. Solution validation is achieved by communicating the results to the user and receiving feedback. In the event of positive feedback the latest problem-solution pair is stored. Means for modifying existing cases descriptions have to be also supplied, as a response to the possibility of negative feedback. However, in both cases the outcome is creating new knowledge.

The operation of the framework is summarised in the pseudo-algorithm shown in Fig. 5.4. The algorithm illustrates the resemblance of the framework's operation with the problem solving approach that was proposed in the previous section. Establishing this similarity means that each space of the framework should contain resources that enable the integration of a case-based with a model-based reasoning system. Accordingly, each space of the framework is further delineated in the following sections.

5.2.2 Data space

The data space includes the knowledge bases that correspond to the product, task, user, and document knowledge areas (see chapter 4). These are formed with the instantiation of the following models.

- Product model. Categorises products into product families, according to their features and component relations.
- Task model. Classifies tasks in terms of the goal pursued and the actions involved.
- User model. Differentiates users in relation to their knowledge, skills, and experiences.
- Document model. Formalises the notion of a product support virtual document and identifies the documentation constituents, as well as their relations.

IF (an identified product support problem)

THEN

IF (product support problem previously solved)

DO (retrieve the corresponding solution and deliver it to the user)

ELSE IF (product support problem similar)

DO (retrieve existing solutions and utilise them as hypotheses) AND

DO (rank similarity of hypotheses according to the problem) AND

DO (choose one of the hypotheses) AND

DO (adapt the hypothesis to form an adapted product support solution and deliver it to the user))

ELSE

DO (retrieve existing solutions) AND

DO (rank similarity of existing solutions to the current problem based on domain and behavioural models)

DO (choose one of the existing solutions)

DO (generate hypotheses based on domain and behavioural models and the aspects of the chosen solution)

DO (choose one of the hypotheses)

DO (generate the solution)

IF (solution is accepted)

THEN (final product support solution reached)

DO (store new problem-solution pair)

ELSE

DO (return reasoning process failed)

DO (hypothesis refinement) AND

DO (store modified hypothesis) AND

DO (repeat process)

Figure 5.4. Algorithm describing the operation of the framework

5.2.3 Problem space

The problem space incorporates knowledge about the product support problems that have occurred in previous problem solving iterations. These problems are literally considered as problem-solution pairs, as they are directly linked to solutions that exist in the solution space. The problem-solution pairs are represented by cases. Only the problem part of each case is regarded as active in the problem space.

5.2.3.1 Representing product support problems with cases

The problem part of a case encodes the state of the product support domain and environment as reasoning begins. Thus, the problem representation should have sufficient detail, in order to be able to judge the applicability of an existing case in a current situation. Both problem structure and content are designed towards this attainment.

Problem structure

Attribute-value pairs are a common description of case-based problems with the advantage of simplicity, preciseness, and controllability (Kolodner 1993) (important characteristics for a system used from different groups of users e.g. novice in information or web technologies). For example, the attribute “No_Disks”, which is used to describe automotive clutches, can be paired with the value “2”. The assignment indicates that the problem refers to double-disk clutches.

A product support system should enable the user to specify the level of significance that different attributes have in particular situations; therefore an importance factor (weight) is also added in the problem representation.

Another consideration is the facilitation of enhanced expressiveness, which is related to the “vagueness” quality. The natural technique of capturing non-explicit requirements is the addition of fuzziness into the system. That can be achieved by the augmentation of the problem description with fuzzy search terms (e.g. “greater than”). Fuzzy logic provides a simple way to arrive at a definite conclusion using vague, ambiguous, imprecise, noisy, or missing input information. Fuzzy inference rules may be expressed in terms such as “If the room gets hotter, spin the fan blades faster” where the temperature of the room and speed of the fan’s blades are both imprecisely (fuzzily) defined quantities, and “hotter” and “faster” are both fuzzy terms (Wang and Lin 2007).

Problem content

The content of a case-based problem contains the goals to be achieved, the situation description, and the constraints to be satisfied.

The **goals** are separated in three groups. According to the definition of a product support problem the most abstract goal is to execute diagnosis (explicitly related with the use of a product support system), which does not have to be included in the problem description. For example, identifying that specific parts of required product support virtual documentation are missing and utilising the means to fix this problem belongs in this category.

At the next group the purpose of the user is delineated, into three classes, which are information retrieval, diagnosis, and explanation (or expert advice). These can be implicitly identified, according to the usage of the system. They indicate the type of

information the user requires for the supported products and tasks. For example, the query “Loud bang or chattering is heard as vehicle vibrates” belongs to the current group since its goal is to diagnose the behaviour of the supported product, while “give more information on clutches” relates to information retrieval.

The last group differentiates between educational (i.e. knowledge enrichment) and performance (i.e. increased efficiency) objectives.

The **situation** components give descriptive information about the targeted characteristics that the solution should reflect. For example the dimension¹ “Moment_of_Inertia” with value “55.814”, depicts information about the performance of a clutch. However, although desirable, respecting the restrictions set by such descriptors is not deemed necessary for delivering a solution. Such features (i.e. *variable-level*) are therefore set to contain either highly dynamic values (e.g. the value of the moment of inertia theoretically can range from 0, in case the product doesn’t have any mass or radius, to several hundreds depending on the supported clutches) or static values (i.e. *class-level* features) that have not been included in the query (e.g. if the query is “more information on clutches” then whether the clutch contains a synchroniser or not should not disallow the presentation of a product support document for clutches). The notions of variable-level and class-level features will be further elaborated in the next section.

The **constraints** are conditions set on goals that have to be met in any acceptable solution. For example one of the goals is to diagnose the documentation constituents

¹ According to Kolodner (1993) the terms *descriptor* and *feature* refer to the attribute-value pair, while *dimension* stands for the attribute part of the pair only.

needed and deliver them (through a product support virtual document) to the user. If the query asks for more information on transaxles, then the presented product support virtual document should include such information, otherwise the system fails. Class-level and contextual features that are engaged in the query form constraints. Contextual features describe the user's category and goal. Tables 5.1 and 5.2 represent the structure and content of case-based product support problem descriptions.

5.2.3.2 Integrating case-based product support problem descriptions and ontologies

One of the major limitations of traditional attribute-value pair representations of cases is the fact that there is no relation between the different pairs. In this study the aforementioned drawback is leveraged by means of modifying the weight of each feature. However, since the CBR knowledge is most of the time stored in text format having no identified links between a large number of attribute-value pairs can influence the performance of a system in case retrieval, indexing, and adaptation.

The proposed solution is semantically disambiguating the features by assigning them to components of product support knowledge bases and ontologies. The mapping is achieved in two levels, the concept and instance ones.

- At the **concept level** the dimension of a feature is mapped to either a concept or a slot. For example, the attribute "Assembly" is assigned to the assembly concept in the product knowledge base, while the dimension "Moment_of_Inertia" represents the same information as the corresponding

Table 5.1. Structure of a case-based product support problem

Structure				
Dimension	Weight	Scale	Search Term	Value

Table 5. 2. Content of a case-based product support problem

Content	Goal	<i>Documentation</i>	Diagnosis (diagnostic task related to the product support problem definition)
		<i>Product/Task</i>	Information Retrieval
			Diagnosis (diagnostic task related to the tool chosen by the user (i.e. usage of system))
			Explanation (Expert Advice)
	<i>Context</i>	Education	
		Performance	
	Situation	<i>Variable-level features</i>	All
		<i>Class-level features</i>	Not included in query
	Constraints	<i>Class-level features</i>	Included in query
		<i>Context</i>	Goal (purpose of the user as defined in the goal part of the case)
			User

slot. The aforementioned allocation has the following repercussions.

- The descriptors of the cases are associated with each other according to the relations of the corresponding knowledge models elements. For example, the dimension “No_of_Disks” is related to the concept “Clutch” with the relation “is_slot”, which means that the value of the “Assembly” feature has to be “Clutch”, when the descriptor “No_of_Disks” belongs to the case, setting restrictions on the validity of cases. In natural language this can be expressed as “The assembly clutch has number of disks (value)”.
- The features can be classified according to the range of values they can have or the frequency with which their values are expected to change. Dimensions that denote slots are expected to demonstrate dynamism (e.g. “No_of_Disks” can change frequently within the range specified by the knowledge base), while concept-based descriptors tend to be more static and predictable (e.g. “Assembly” can have only pre-specified values that correspond to concepts in the knowledge base). The former group of features is called *variable-level* while the latter is named *class-level*. Class-level features are selected according to the represented domains and their values are always concepts. The distinction described above indicates that separate strategies are required for accommodating modifications in variable-level and class-level features.
- The sets of attribute-value pairs can be linked to different documentation components. As explained in chapter 4, concepts are described by Information Object Clusters (IOCs), while Information Objects (IOs) are mapped to slots of the knowledge base. Naturally, since variable level features are related to slots of the knowledge base they are also described by IOs and class-level



descriptors by IOCs. The identified associations between ontology-based components, case related descriptors, and product support virtual documentation elements are demonstrated in Fig. 5.5.

- Class-level features represent a more complex documentation module than variable-level ones. This means that the modification of class-level descriptors requires a lot of computational resources and knowledge-intensive techniques, in order to produce a support document. On the other hand changing variable-level descriptors is less important since they are mapped to the smallest documentation constituent (i.e. IO). This distinction is signified by the weights assigned to each group. Consequently, class-level features should have a bigger weight factor than variable-level ones, unless otherwise explicitly defined by the user.

At the **instance level** the cases contained in the case base represent combination of instances included in the knowledge base, meaning that the cases can be validated against existing instances. For example, if two features of a case are “Assembly” and “No_of_Disks” with respective values “Clutch” and “2”, then instances of the concept clutch with 2 disks should exist in the knowledge base. Variable-level descriptors are directly related with the validity of cases, since they are the ones used to instantiate concepts.

5.2.4 The hypothesis space

In the hypothesis space, the reasoning operations take place, which are enabled by a set of computational models. These include case retrieval and adaptation, as well as model-based generation algorithms.

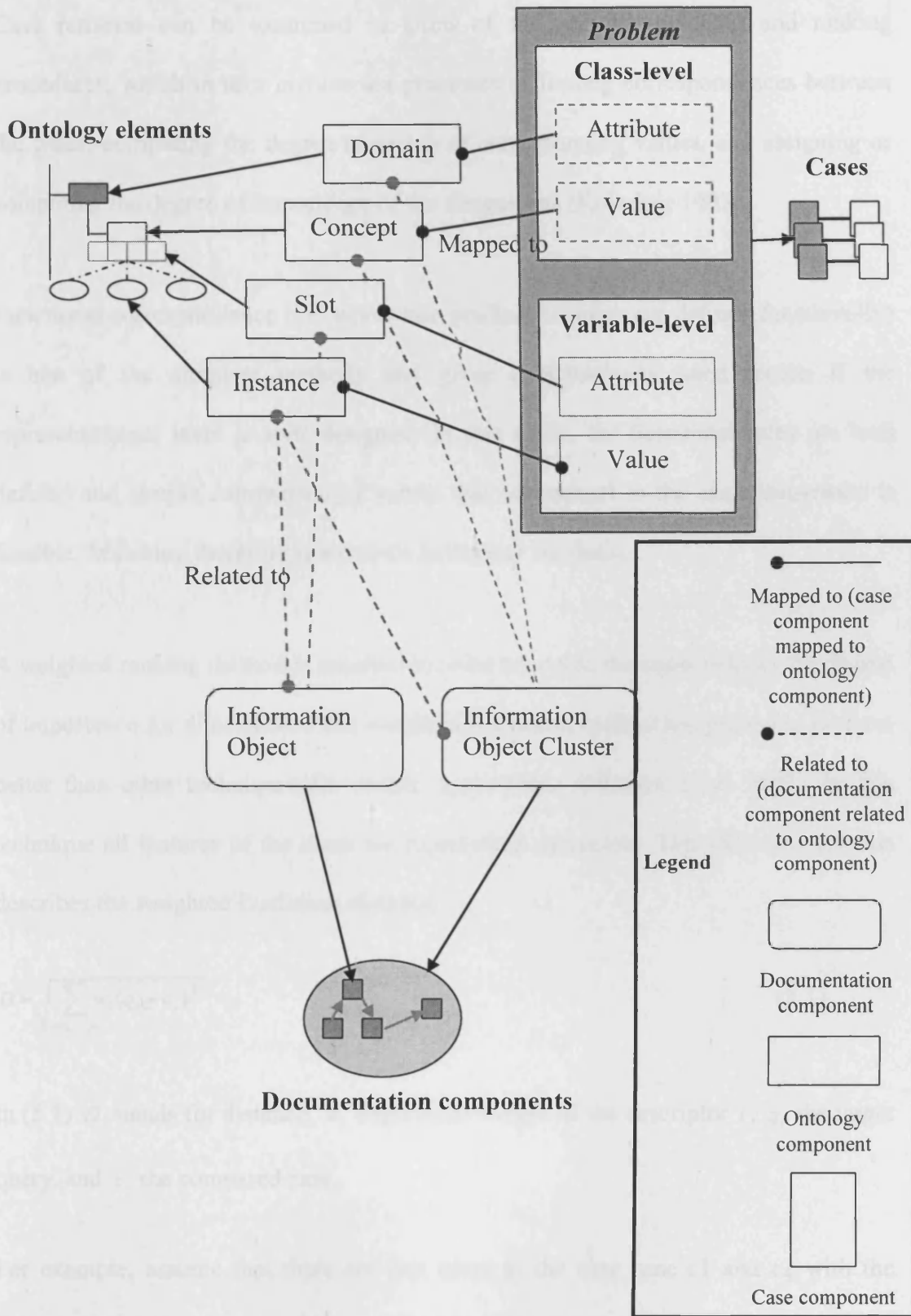


Figure 5.5. Associations between ontology elements, case-based product support problem constituents, and documentation components

5.2.4.1 Case retrieval

Case retrieval can be examined in terms of the utilised matching and ranking procedures, which in turn involve the processes of finding correspondences between the cases, computing the degree of match of corresponding values, and assigning or computing the degree of importance of the dimensions (Kolodner 1993).

Functional correspondence (i.e. when case predicate clauses are defined functionally) is one of the simplest methods and gives comparatively good results if the representational level is well designed. In this study, the functional roles are well defined and simple comparison of values that correspond to the same dimension is feasible. Matching therefore is based on functional similarity.

A weighted ranking method is required in order to enable the users to input the degree of importance for dimensions. The weighted Euclidean method has proved to perform better than other techniques for certain applications (Mendes et al. 2002). In this technique all features of the cases are represented as vectors. The following formula describes the weighted Euclidean distance.

$$D = \sqrt{\sum_{i=1, \dots, n} w_i (q_i - c_i)^2} \quad (5.1)$$

In (5.1) D stands for distance, w_i depicts the weight of the descriptor i , q_i the target query, and c_i the compared case.

For example, assume that there are two cases in the case base c_1 and c_2 with the following descriptors and values a_{c_1} : Clutch_size=10cm and b_{c_1} : Clutch_weight=3kg and a_{c_2} : Clutch_size=12cm and b_{c_2} : Clutch_weight=2kg. and the target query t

requires a clutch with a_t : Clutch_size=11cm and Clutch_weight=2.5kg and the weight descriptor is more important than the size descriptor by a factor of 2 and 1 respectively then equation 5.1 can be used as follows.

$$D_{c1} = \sqrt{w_{Clutch_size} (Clutch_size_t - Clutch_size_{c1})^2 + w_{Clutch_weight} (Clutch_weight_t - Clutch_weight_{c1})^2} = \sqrt{1(11-10)^2 + 2(2.5-3)^2} = \sqrt{1+0.5} = \sqrt{1.5} \quad (5.2)$$

$$D_{c2} = \sqrt{w_{Clutch_size} (Clutch_size_t - Clutch_size_{c2})^2 + w_{Clutch_weight} (Clutch_weight_t - Clutch_weight_{c2})^2} = \sqrt{1(11-12)^2 + 2(2.5-2.5)^2} = \sqrt{1} \quad (5.3)$$

According to (5.2) and (5.3) $D_{c2} < D_{c1}$, which means that c2 will be ranked higher than c1.

5.2.4.2 Case-based adaptation

In section 5.1.2 product support problems have been classified into previously solved, similar, and new ones. Previously solved PSPs should exhibit 100% correspondence with the current query. The case-based description of similar PSPs on the other hand can differentiate from existing solutions in two aspects.

- Variable-level descriptors can have different values. For example, “radius” is a dynamic characteristic, which requires a solution modification when changed from 2.2 cm to 3.4 cm that reflects the current situation. Most of the times such a variation doesn’t affect the structure of the presented Product Support

Virtual Document (PSVD). However, the range of the changed value should be within the boundaries set in the product support related ontologies.

- Class-level descriptors may be also altered. Case-based adaptation is employed in such a case if the Information Object Cluster that substitutes an existing solution's IOC, has the same functional role and is pre-composed or can be composed at run-time (IOs are available). For example an IOC that describes the flywheel of a clutch has the same role as an IOC depicting a countershaft (both are clutch subassemblies). In the case that both of them are available and the countershaft IOC is required to be included in the solution instead of the flywheel IOC, then case-based adaptation can be applied.

For each of the aforementioned scenarios a different substitution technique is utilised.

Parameter adjustment is utilised for enabling adaptation based on modified variable-level descriptors. This technique is one of the most popular for interpolating values in a new solution based on those from an old one, since changes in parameters of an old solution are made in response to differences between problem specifications.

A two-step process is therefore followed. First, the variable-level feature differences are identified and extracted. Then, specialised adjustment heuristics, which capture the relationships between problem features and solution parameters, are applied to the old solution to create the adapted one. For example, a solution on a PSP that asks for more information on "Clutch" with "No_Disks" 3 can be derived from a case that contains a description for "No_Disks" 2, based on the heuristics shown in Fig. 5.6 and

IF the goal is information
 retrieval AND learning
 THEN
 IF the class-level
 feature values have
 remained unchanged
 FOR
 numerical
 and non-
 numerical

Figure 5.6. Specialised adjustment heuristics for parameter adjustment

IF "No_Disks" 1,
 description d AND image
 i.
 ELSE IF "No_Disks" 2,
 description d1,d2 AND
 image i1, i2, i3.
 ELSE IF "No_Disks" x
 description x-d2 AND
 image i1, d*i2, i3.

Figure 5.7. Conditions for "No_Disks" descriptor

5.7. As illustrated, a list of parameter adjustment heuristics is maintained, and under specific circumstances a number of these heuristics is utilised, by implementing the required comparisons. This method is followed in order to avoid an exhaustive search of all possible comparisons, as the range and depth of the product support domain may be prohibitive for such an approach.

Reinstantiation is used when a new IOC needs to fill the role of an old IOC. This becomes possible by selecting an old solution (possibly the best match) and employing role bindings for creating an adapted solution. Reinstatement is feasible because of the ontology-based framework with which product support problems are related (see section 5.2.3.2) that makes the roles between different features semantically distinguishable. Thus, the problem representation except for designating the functional roles of its descriptors indicates role correspondences based on the comparison of included dimensions. This means that according to the goal that has to be achieved old problems and solutions can be abstracted. For example the two cases illustrated in Fig. 5.8 involve requesting information about a flywheel and a countershaft (according to the bindings between the ontology and the case base “flywheel” and “countershaft” are both specialisations of the subassembly concept and therefore semantically equivalent for the case reasoner) and can be used to abstract the problem of asking information about subassemblies, as shown in the right part of the figure.

5.2.4.3 Model-based generation

Model-based reasoning is utilised in the case that a class-level descriptor that is included in the query, doesn't correspond to an existing IOC. The process has three

OLD CASE: INFORMATION ON FLYWH
Goal: Information retrieval, learning
User: Inexperienced
Situation description:
 Product: Car
 Assembly: Clutch
 Subassembly: Flywheel
 Part : Bolts
 Weight: 3.14kg
 Material: Carbon
 Task: Describe

Solution:
 Object Flywheel
 find Flywheel IOC
 structure flywheel IOC:

*Figure 5.8. Example of the representation of problems
and solutions*

main stages. Initially the missing IOC is identified, then an automated computational model is utilised to locate the knowledge needed to automatically create the IOC, and finally the generated information is configured for delivering a PSVD to the user.

Diagnosis is the first step of the overall reasoning process. The final goal of this step is to isolate the fault to a single component or to a Least Replaceable Unit (LRU). Since the repair action is to construct a new IOC, the LRU denotes an IOC, although it is in turn composed of smaller constituents (i.e. IOs). Identifying the omitted IOC is a matter of exploring the model of a case description and connecting it to the ontology-based knowledge models (see section 5.2.3.2).

Once the required LRU is mapped to an ontology component, the **configuration** process starts by automatically creating an IOC based on the relations defined in the knowledge base. More specifically, the generalisation relation (“is-a”) is utilised in providing information that covers the queried domains, while aggregation and reference relations are employed to compose required IOCs.

For example, if an IOC that corresponds to a transaxle is needed but no such description is available, then aggregation relations are exploited to find the assemblies and parts with which a transaxle is developed. Each assembly, part, and interface is individually used to portray the description of a transaxle. In the case that transaxle is not recognised as an internal part of the model, its dimension (i.e. assembly) is utilised as the transaxle’s abstracted concept, which can deliver more information based on the general qualities that characterise the domain (e.g. transaxle is-a assembly and therefore the definition of assemblies is true for transaxles as well).

A component-based model is therefore adopted where the structural description of the products and tasks is supplied by the knowledge base, while the behaviour is simulated through the relations of the ontology-based concepts.

The search in the knowledge base is exhaustive (i.e. all aggregation and abstraction relations are used) and several different hypotheses are formed. Each of them is created as a combination of the information retrieved from different concepts. A candidate solution covers the query (e.g. transaxle description) and is consistent with all requirements (e.g. other assemblies are not included within the IOC). Candidate solutions are ranked according to a specialisation of the parsimony rule (i.e. if a composite solution is a data subset of another one, then the smallest set is selected). For example, a transaxle has a number of subassemblies, which in turn have a number of parts. At least two hypotheses are formed in such a case. The first one has data about the subassemblies only, while the second one about both subassemblies and parts. If the IOCs that describe the subassemblies have been manually pre-composed then the former hypothesis is selected otherwise the latter one is chosen.

Configuration models are employed to merge the IOCs and generate the product support virtual document. From the documentation ontology, the modules of a PSVD are recognised as being IOCs and IOs. The IOCs created with Model-Based Reasoning (MBR) are based on less elaborate resources² than the ones in case-based reasoning, since MBR is used when no manually pre-composed or automatically adapted constituents can be used to create the PSVD.

² The assumption is that the most elaborate or “best” solutions (PSVDs) are the ones that are manually developed by a technical writer or in which the automated processes are minimal (i.e. value substitution (part of parameter adjustment) is only required).

Therefore, the general structure of previous PSVDs is used as the arrangement model (see section 4.2). For example, a PSVD that describes a transaxle includes a title, an introduction and a body. The body should include the IOC describing the transaxle, which can be substituted by the compositional solution developed and selected in the previous stages.

Fig. 5.9 illustrates the three different kinds of product support virtual documents that can be automatically generated. Case-based reasoning is used when the document is adapted and there are enough resources (i.e. IOCs) to create the new document automatically. That document is considered the best solution of the three because enough information is available to construct the PSVD.

The aggregation-based approach is utilised when there is information in the ontology-based model that can be used to compose the required IOC, according to the relations between different concepts. That approach is preferred to the abstraction-based one because enough information can be retrieved by the model to compose a solution at run-time. However, it should be manually edited and validated at a later stage.

The abstraction-based approach identifies information related to the required IOCs by analysing relations to more abstract concepts than the ones needed. The information that is provided to the user with this approach refers to a relevant concept and not the one required. It is therefore the least desirable of the three approaches introduced in this study and is used only if information cannot be retrieved either from the case-based or the ontology-based model (aggregation-based approach).

5.2.5 The solution space

The solution space contains information about the product support virtual documents that have been derived throughout previous problem solving iterations and includes the following.

- Unique Identifiers (UIs) for every PSVD. Each UI is comprised of the concepts and slots that were involved in the problem specification and solution generation.

**Goal- User-
Product-ProductInstantiation-
Assembly-AssemblyInstantiation-
Subassembly-SubassemblyInstantiation-
Part-PartInstantiation-
Task-Subtask-Action**

All instantiations denote the existence of a number of variable-level features (or ontology slots). The parts of the UI that do not have any value are filled with “null” or “0”.

- Status Identifiers (SIs) for every PSVD. Each SI can take the values “validated” or “not validated”, indicating whether a solution has been manually validated. The ones that have not been validated are removed after a period of time.

A tool is also provided for modifying or validating the case descriptions used as hypotheses except for the solutions themselves. The tool is only accessible to authorised users.

5.3 SUMMARY AND CONCLUSIONS

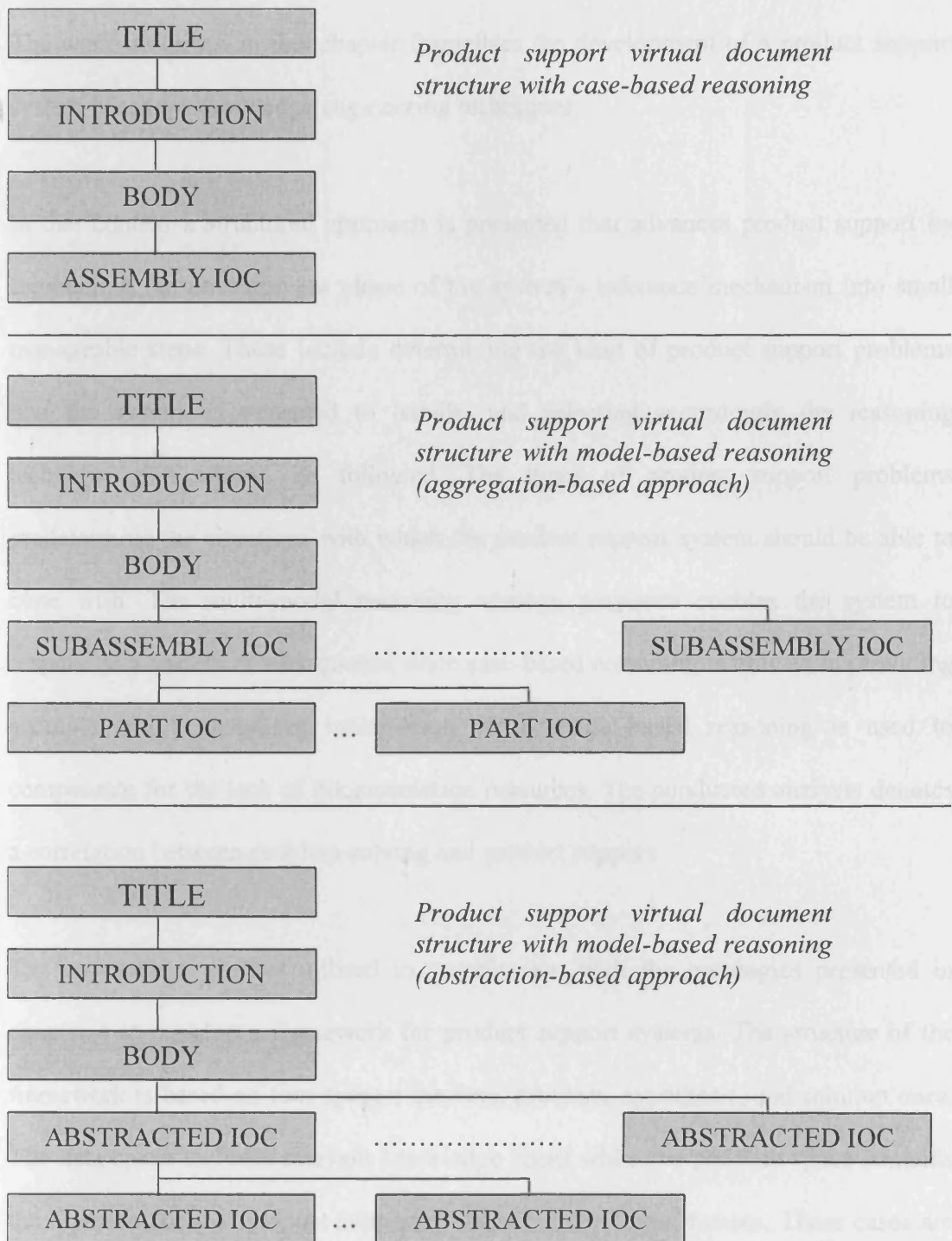


Figure 5.9. Difference in structure of a product support virtual document using case-based reasoning and model-based reasoning

5.3 SUMMARY AND CONCLUSIONS

The work described in this chapter formalises the development of a product support system based on knowledge engineering techniques.

In that context a structured approach is presented that advances product support by segmenting the development phase of the system's inference mechanism into small manageable steps. These include determining the kind of product support problems that the system is expected to handle, and selecting accordingly the reasoning technique that should be followed. The types of product support problems predetermine the situations with which the product support system should be able to cope with. The multi-modal reasoning strategy proposed enables the system to respond to a variety of user queries since case-based reasoning is utilised in providing accurate and personalised information while model-based reasoning is used to compensate for the lack of documentation resources. The conducted analysis denotes a correlation between problem solving and product support.

The approach is further utilised in combination with the ontologies presented in chapter 4 to develop a framework for product support systems. The structure of the framework is based on four spaces; the data, problem, hypothesis, and solution ones. The data space includes relevant knowledge bases while the problem space contains the representation of product support problems in the form of cases. These cases are semantically analysed by mapping their dimensions to ontology-based components. The proposed integration of case-based reasoning with ontologies enables case-based adaptation (reinstantiation) and model-based generation to be applied in the hypothesis space.

CHAPTER 6

CONTEXT-AWARE PRODUCT SUPPORT SYSTEMS

This chapter advocates the view that product support could be substantially enhanced by developing context-aware Product Support Systems (PRSSs) where the context of use is modelled through a set of context-related attributes and interactions. Towards this attainment, the chapter presents a formal, ontology-based model of context-aware PRSSs that integrates knowledge about users, their activities, environment, and physical constraints. Next, a context-related adaptation method is introduced that utilises defined documentation attributes and task structures. An ontology-based approach is also proposed to semi-automatically integrating design data into product support according to internal stimuli. The objective is to transform a product support system in a responsive entity able to adapt to different situations.

6.1 CONCEPTUAL ANALYSIS OF A CONTEXT-AWARE PRODUCT SUPPORT SYSTEM

6.1.1 Scope

A Product Support System (PRSS) aims to alleviate the lack of knowledge of the user in a particular subject or situation related to a supported product by providing user-tailored information just-in-time and just-in-place. Researchers in this field have utilised the Internet and digital technologies as enablers of achieving the “just-in-time” and “just-in-place” requirements. User-tailored delivery on the other hand has been one of the main themes for several years in fields such user interfaces design,

human-computer interaction and web-based systems. However, personalisation (i.e. the ability to deliver information that most closely corresponds to an individual's profile) has been largely based on reasoning techniques that utilise solely user characteristics as a resource. The adaptation process has been therefore considered as independent of the time, place, and intention of the user.

Recently *context* has received attention as a way of representing different situations in a universe of discourse. It has been mainly used in mobile applications where location is a very important aspect towards achieving personalisation. Attempts to use other context information as well have increased over the last few years (Baldauf et al. 2004). The increasingly widespread use of context in different applications indicates the trend towards adaptive information delivery. However, to date the integration between domain and context knowledge in the field of support systems has been limited to the representation of user characteristics.

The main assumptions underlying this work are two. First, in addition to user specific data, context should include other data used to characterise a situation such as time. Second, domain and context knowledge can be integrated within a PRSS and utilised to generate *context-tailored* information. The objective is to transform product support systems into a medium that not only offers just-in-time support but also enables users to improve their skills by acquiring context-specific information.

6.1.2 Definition of a context-aware product support system

Several definitions describing context have been proposed in the literature. In this work context is regarded as “any information that can be used to characterise the

situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves” (Dey 2001).

Context-aware systems are designed as the means of allowing operational autonomy by accurately recognising the context and determining the appropriate action to be taken (Erickson 2002). As a result automatic contextual reconfiguration and context-triggered actions are core categories in context-aware applications (Schilit et al. 1994). The following definition summarises the goal of Context-Aware PRSSs.

Definition 12: *Context-Aware PRSSs are knowledge-based PRSSs that can monitor internal or external stimuli, identify contextual changes, and adapt their operation in order to generate context-specific information.*

6.1.3 Conceptual model of a context-aware PRSS

As illustrated in Fig. 6.1., context-aware PRSSs are viewed as an extension of knowledge-based ones (discussed in previous chapters), with an extra subsystem called *Context Manager* and an added *Sensors Layer*. The main components of the Context Manager include the *Context Processing Module*, the Context Knowledge Base (*Context KB*), and the *Adaptation Module*.

The **Sensors Layer** contains several sensors. The word “sensor” in this study refers mainly to software used to monitor internal and external stimuli. External stimuli are factors that characterise the system’s environment, the user, the user’s environment, and the interaction between the user and the system. Internal stimuli are all the

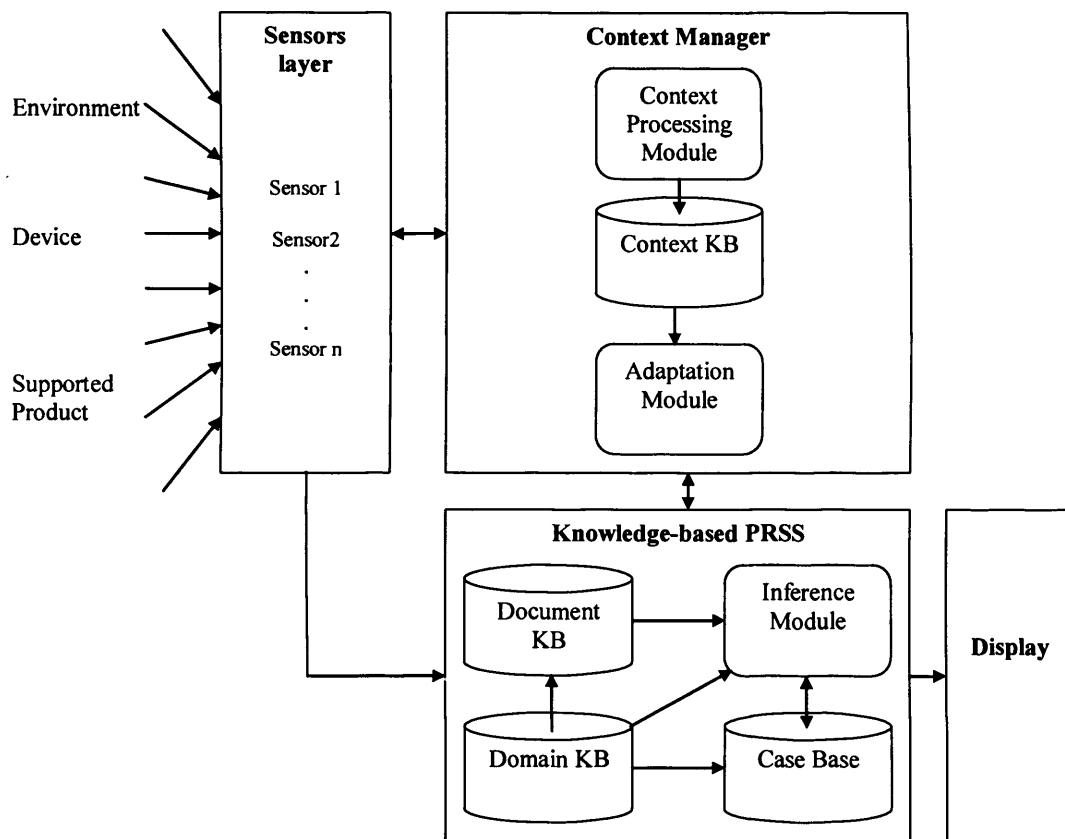


Figure 6. 1. The concept of context-aware PRSSs

changes happening internally to the system, such as variations in domain information. Sensing information is captured explicitly (e.g. questionnaires) and implicitly (e.g. use of the system) while sensors operation is either automatic or semi-automatic (e.g. sensing is initiated by the user with the use of a system's function). For example, a programme that monitors the key strokes of the user and transmits them to the system is a type of a software sensor.

The sensors layer is used to gather all contextual data. Some of this data is directly stored in the Context KB (e.g. administrative data such as the name of the user). However, most of the acquired contextual data is too primitive to be used directly.

The **Context Processing Module** converts such data into meaningful contextual information. Take the example of having data about user's years of experience and job function. Individually, this data has no effect on the system's operation. Nevertheless, their combination can identify whether a user is experienced in performing a specific job function and therefore be classified as an expert in related operations. That can have an immediate effect on the way information is delivered to him (e.g. remove introductory material).

The **Context KB** retains all contextual data according to a formal model. This context model needs to define and store relevant data in a machine processable form enabling the exchange of contextual information within a PRSS. Ontology-based representations are therefore used. Except for building a consensus between researchers ontologies are considered a very promising instrument for modelling contextual information due to their high and formal expressiveness and the possibilities for applying ontology reasoning techniques (Baldauf et al. 2004).The

Adaptation Module retrieves the contextual characteristics retained in the Context KB and adapts information delivery accordingly. More information on this operation will be given in subsequent sections.

6.2 MODELLING CONTEXT

6.2.1 General ontology of context for product support systems

The aim of Context-Aware PRSSs is to deliver context-specific information to the user to support him/her in all tasks related to a product. The development of a PRSS is therefore highly interdisciplinary requiring the integration of several application areas including computer science, knowledge engineering, product design, and technical writing. Detailed context information represented in a formal way should be provided to facilitate the aforementioned integration, as well as application adaptation and context-specific delivery. The following four models are utilised in order to achieve this objective.

- *Activity Model* (or Purpose Model) (AM), which is a finite set $\{a_1, a_2, \dots, a_n\}$ where each a_i stands for a specific activity i.e. purpose of using the system. In this study two main abstract activities are included, which are ‘perform’ and ‘learn’.
- *User Model* (UM), which is a finite set $\{u_1, u_2, \dots, u_k\}$ where each u_i represents a user stereotype. Different user stereotypes have diverse characteristics, indicating that different types of information should be provided to each user.

- *Physical Model* (PM), which is a finite set $\{p_1, p_2, \dots, p_m\}$ where each p_i stands for any hardware or software related property, such as the operating system or the graphics card memory. The Physical model designates the medium used to deliver information and can be used to support automatic code conversions between different formats.
- *Environment Model* (EM), which is a finite set $\{e_1, e_2, \dots, e_n\}$ where each element represents other environmental conditions (e.g. light conditions, location). EM specifies characteristics that are important in user interface optimisation. For example, differences in light conditions may trigger the brightness in the monitor to change accordingly.

A context instantiation C_i , is denoted by the aggregation of the aforementioned models' instantiations and can be represented as $\langle a_i, u_i, p_i, e_i \rangle$. In this study UM and AM are utilised to illustrate the notion of context-aware product support systems while PM and EM are not extensively analysed since distributed, service-based, and mobile PRSSs (where physical and environmental conditions frequently change) are not within the scope of this work. However, in order to create an extensible representation all models are included in the context ontology, as shown in Fig. 6.2.

The main concepts that originate the creation of the context ontology are depicted in Fig. 6.2 in the form of shadowed boxes. The *Physical* configuration of a device is regarded as a compositional concept, which can be largely defined according to its *Software* and *Hardware* aggregates (normal boxes in Fig. 6.2). Each of these concepts can in turn be described using the specialisation relation, as in the case of *Hardware*, which can be further classified into *CoreComponent*, *StorageDevice*, and *Peripheral*

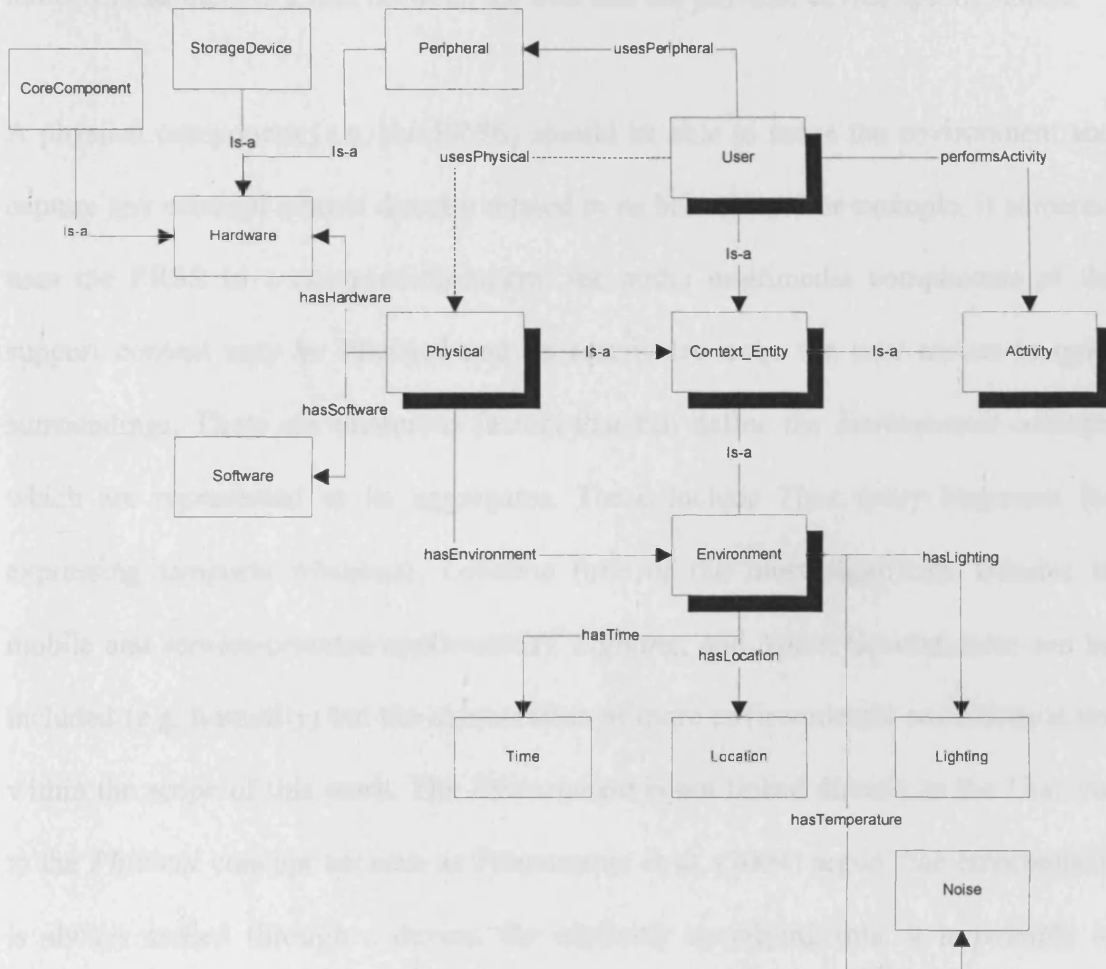


Figure 6. 2. Part of the context ontology

concepts. Peripherals are utilised by the user in order to access the physical device. Consequently, *Peripheral* is linked with the *User* concept with the relation *usesPeripheral*, which enables the system to adapt its display accordingly (e.g. if a small monitor is only supported as in mobiles, the resolution of words can change accordingly). The *usesPhysical* relation between the *User* and the *Peripheral* concepts indicates that there is a link between the user and the physical device specifications.

A physical component (e.g. the PRSS) should be able to sense the environment and capture any external stimuli directly related to its behaviour. For example, if someone uses the PRSS in a noisy environment, the audio multimedia components of the support content may be disabled and be reactivated once the user moves to quiet surroundings. There are numerous factors that can define the *Environment* concept, which are represented as its aggregates. These include *Time* (very important for expressing temporal relations), *Location* (one of the most significant features in mobile and service-oriented applications), *Lighting*, and *Noise*. Several more can be included (e.g. humidity) but the enumeration of more environmental conditions is not within the scope of this work. The *Environment* is not linked directly to the *User* but to the *Physical* concept because as Preuveneers et al. (2004) argue “the environment is always sensed through a device. By explicitly specifying this, it is possible to reason about several properties of the sensed environment that require knowledge of the measuring device, e.g. accuracy.” In the following sections the creation of the activity and user models is discussed.

6.2.2 Activity context

The *Activity* concept is utilised to describe the user’s intention. Utilising *Activity*

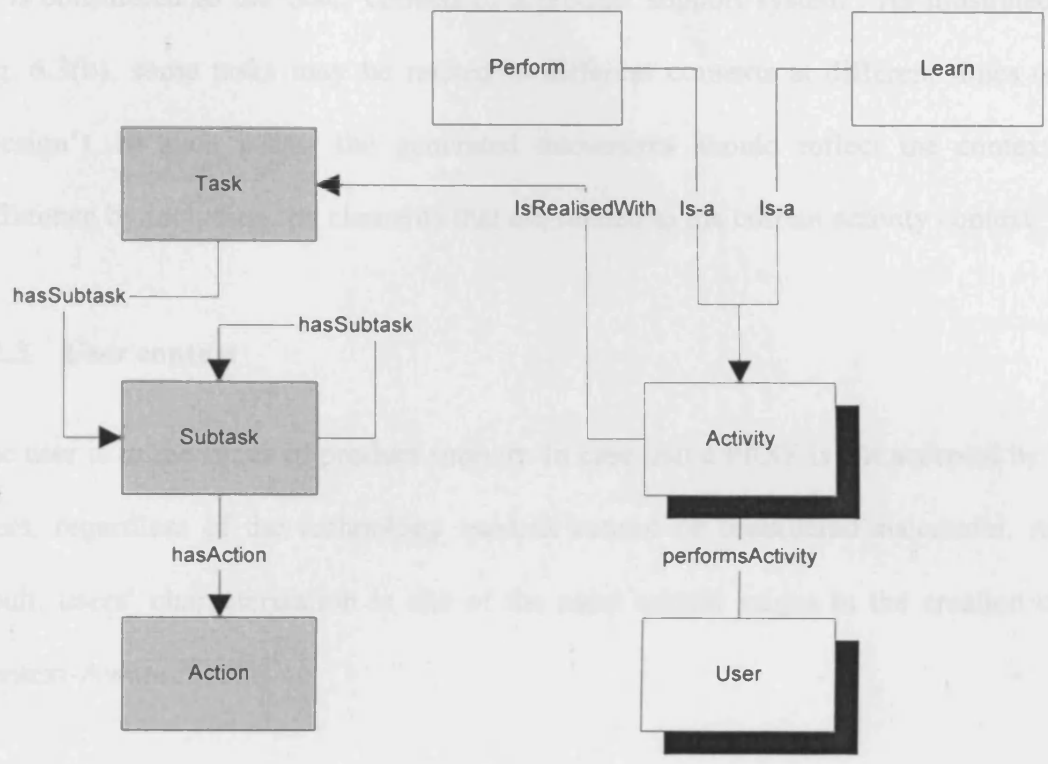
performance support and e-Learning techniques can be integrated within the context of a single product support system to achieve performance and learning goals. In this study *Learn* and *Perform* are used to signify the difference between those goals. As shown in Fig. 6.3, the *User* is directly linked to the *Activity* with the relation *performsActivity*, which suggests that context information is only relevant if it enables adaptation according to user goals.

Activity is also connected to *Task* through the spatial relation *isRealisedWith*. *Task* (represented in a grey box because it is part of the domain knowledge) includes a number of subtasks and actions (i.e. is the most elementary step of a task) that a user may want to be supported for (e.g. installation, and design). It is therefore viewed as an aggregation of the *Subtask* and *Action* concepts.

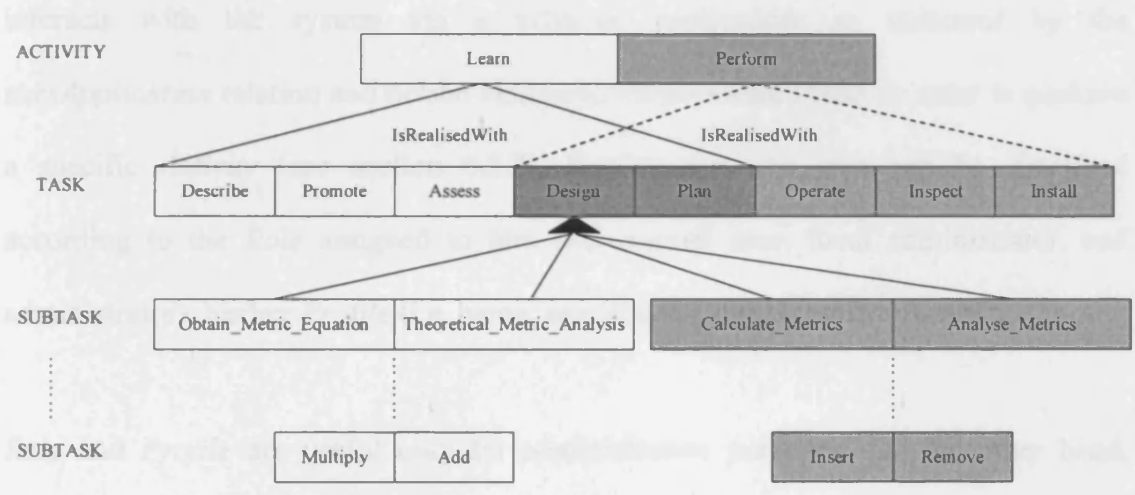
The activity model is used to develop particular configurations of the task model according to the variations in the activity selection. For example, as illustrated in Fig. 6.3 (b), if the ‘Learn’ activity is selected, the task model includes the ‘Describe’, ‘Promote’, ‘Assess’, ‘Design’, and ‘Plan’ tasks and their corresponding subtasks. On the other hand, if the ‘Perform’ activity is selected, the task model is (re)configured and contains the ‘Design’, ‘Plan’, ‘Operate’, ‘Inspect’, and ‘Install’ tasks, as well as their corresponding subtasks. Each configuration of the task model is defined through the *IsRealisedWith* relation. For instance, using McCarthy’s (1993) formalization, the relation ‘IsRealisedWith’ between the ‘Learn’ activity and the ‘Describe’ task, is equivalent to the following.

$$c0: \quad \text{ist}(c_{\text{activity}}(\text{learn}), \text{“describe is a task”}) \quad (6.1)$$

(1) asserts that it is true in the context of the activity "Learn" that "Describe" is a task. It is considered as the user's knowledge about the system. An illustration of this is shown in Fig. 6.3(a), where tasks may be realized by activities.



(a)



(b)

Figure 6.3. Relation between the context (activity model) and domain knowledge (task model)

(1) asserts that it is true in the context of the activity “Learn” that “Describe” is a task. c_0 is considered as the outer context of a product support system¹. As illustrated in Fig. 6.3(b), some tasks may be related to different contexts at different times (e.g. ‘Design’). In such cases, the generated documents should reflect the contextual difference by including the elements that are related to the current activity context.

6.2.3 User context

The user is in the focus of product support. In case that a PRSS is not accepted by the users, regardless of the technology used, it cannot be considered successful. As a result, users’ characterisation is one of the most critical stages in the creation of a Context-Aware PRSS.

Fig. 6.4 shows the place of the *User* concept within the context ontology. *User* always interacts with the system via a software application, as indicated by the *usesApplication* relation and he/she visits a Context-Aware PRSS in order to perform a specific *Activity* (see section 6.2.2). Furthermore, the user can be described according to the *Role* assigned to him (i.e. normal user, local administrator, and administrator), his/her *Profile* (i.e. name, age, and sex) and *Characteristics*.

Role and *Profile* are useful only for administrative purposes. On the other hand, *Characteristics* includes data used as parameters throughout the computation of the user’s information needs. For example, an “expert” characterisation will indicate the ability of the user to understand technical information included in a document.

¹ McCarthy (1993) argues that every context has an outer context.

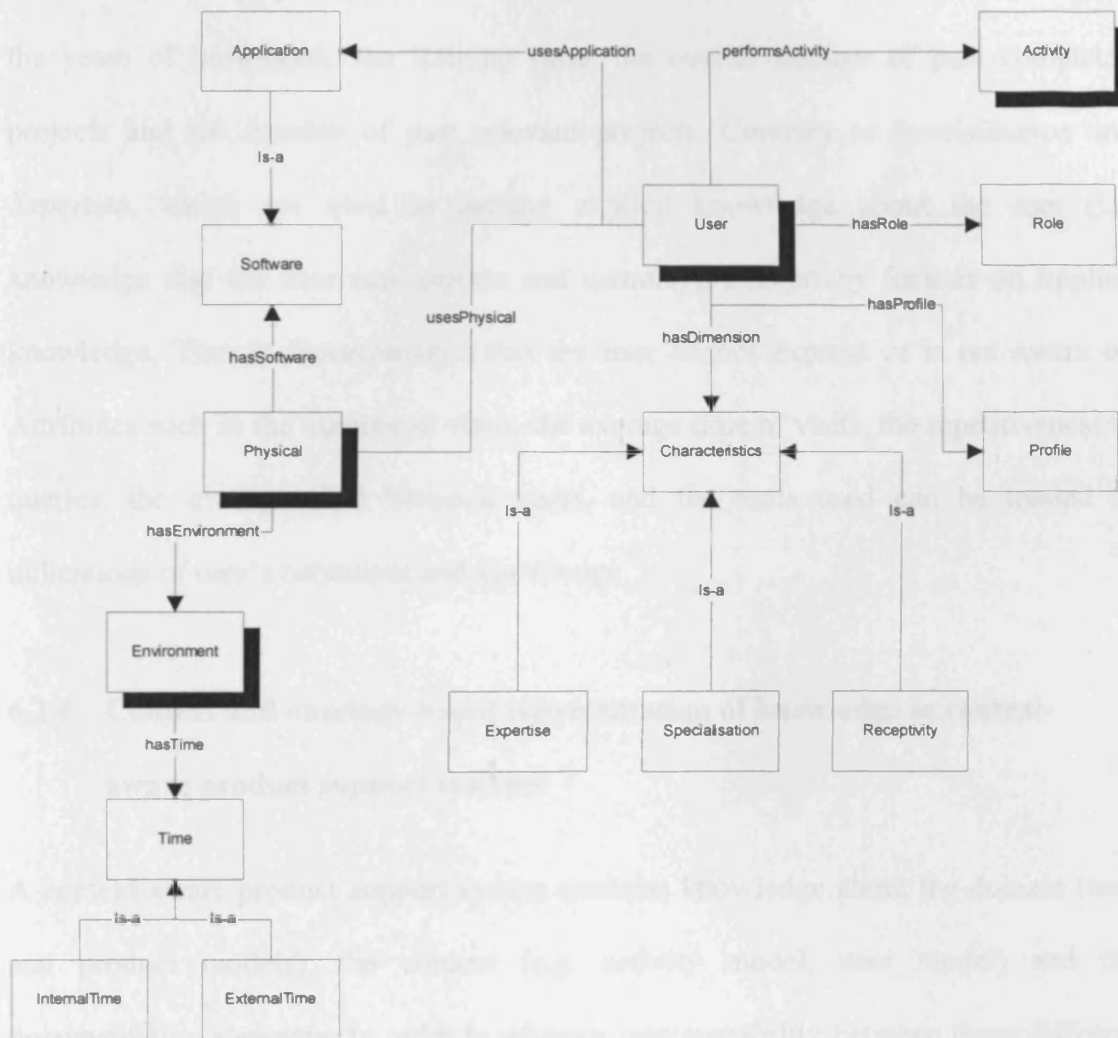


Figure 6. 4. User context concept

Expertise, *Specialisation* and *Receptivity* are identified as the main user features directly related to a PRSS.

The attributes defining *Specialisation* include the education level, the training type, the job title, the project type, and the project role. *Expertise* is determined based on the years of profession, the training time, the overall number of past completed projects and the number of past relevant projects. Contrary to *Specialisation* and *Expertise*, which are used to capture explicit knowledge about the user (i.e. knowledge that the user can express and quantify), *Receptivity* focuses on implicit knowledge. That is characteristics that the user cannot express or is not aware of. Attributes such as the number of visits, the average time of visits, the repetitiveness of queries, the average time between visits, and the tools used can be treated as indications of user's behaviour and knowledge.

6.2.4 Context and ontology-based representation of knowledge in context-aware product support systems

A context-aware product support system contains knowledge about the domain (task and product models), the context (e.g. activity model, user model) and the documentation elements. In order to advance interoperability between these different areas and product support, an ontology that formalizes the aforementioned knowledge has been developed. A part of the ontology is illustrated in Fig. 6.5.

The ontology can be described according to the product, task, context, and document models it includes.

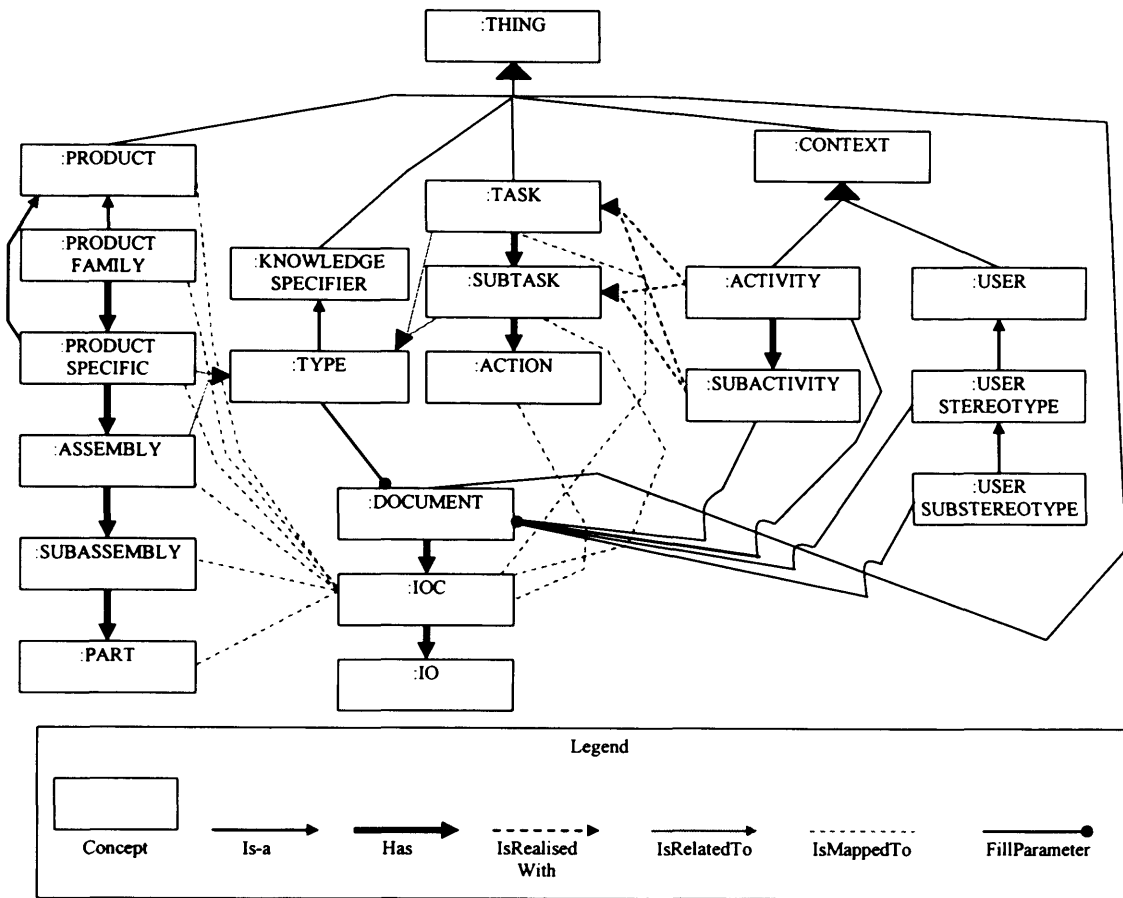


Figure 6.5. Part of the ontology for context-aware product support systems

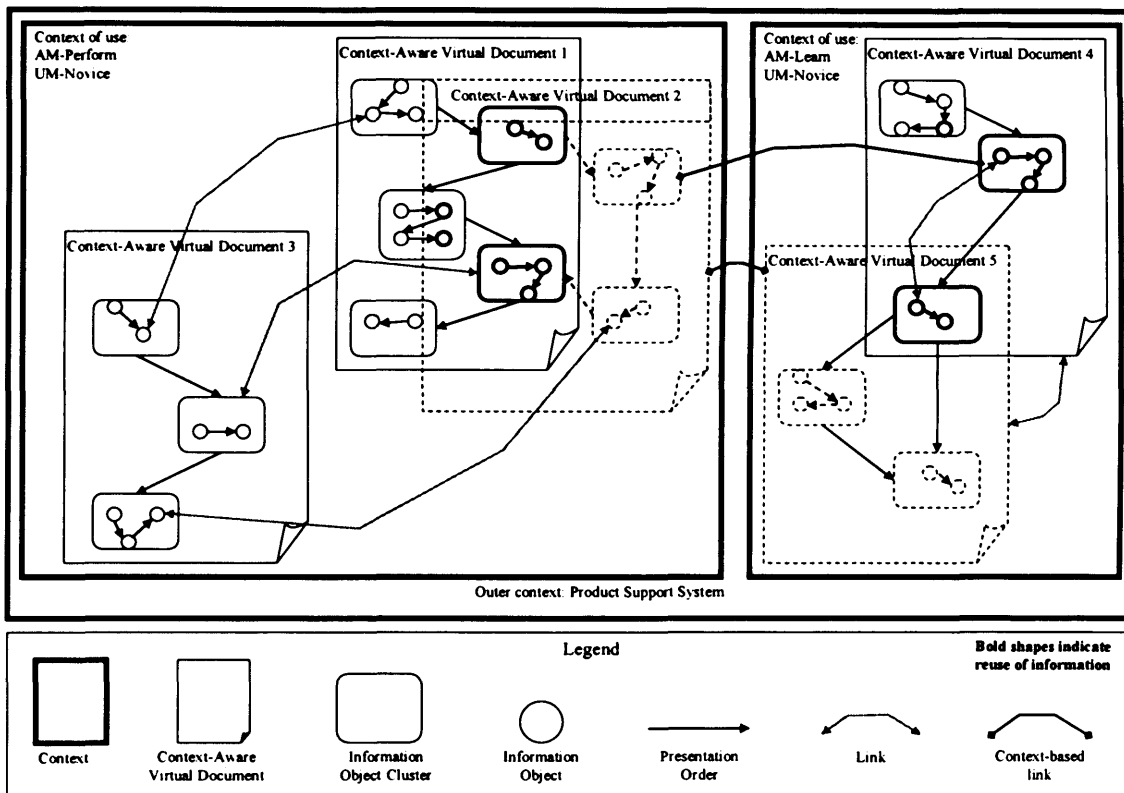


Figure 6.6. Model of context-aware virtual documents and their relations

- The product model represents the structure of the product. All its concepts are mapped to Information Object Clusters (IOCs) as explained in the rest of the section. IOC has been defined (see chapter 4), as a documentation element that is semantically distinguishable and related to the concepts describing supported products and tasks. It contains smaller elements called Information Objects (IOs) as shown in Fig. 6.5. Concepts “:PRODUCT SPECIFIC” and “:ASSEMBLY” are linked to the concept “:TYPE”. This is a specialisation of “:KNOWLEDGE SPECIFIER”, which abstracts all concepts that represent domain significant properties. For example, the type of an assembly, i.e. whether it is considered as complex or not, affects the generation of a document (see chapter 4).
- The task model contains the tasks, subtasks, and actions that are supported, where action is the most elementary step of a task. All are mapped respectively to Information Object Clusters and are related to “:TYPE”. Furthermore, the task model is configured according to its relation to the activity model (see section 6.2.2).
- The context model includes the activity and user models (more models are included in context as explained in previous sections). Both models are related to the “:DOCUMENT” concept with the relation “FillParameter”, which denotes that the characteristics of the user and the activity are passed as adaptation parameters to the document. Furthermore, the activity model also defines the variations of the task model.
- The document model embodies the structure of a context-aware virtual document. As illustrated in Fig. 6.6, each virtual document is composed of a number of IOs and IOCs. These are connected to each other with different relations including

their presentation order, hypertext, and context-related links. Virtual documents generated in different contexts may have differences in both content and presentation. For example, in Fig. 6.6 the difference in context is signified by having different values for the Activity Model (AM). However, all documents are contained within the outer context of a PRSS. Bolded shapes in the figure indicate that the same documentation elements are reused to create different documents. The “IsMappedTo” relation included in Fig. 6.5 defines the content by mapping each IOC to a different concept. Moreover, each slot (or each attribute) of a concept, is mapped to at least one IO. The document that is generated includes all the IOCs, and therefore concepts, that the query requires. The context of use is passed to the document as a set of parameters, as explained in the next section.

6.3 ADAPTATION APPROACH

6.3.1 Correlation between context, tasks, and product support documentation

In her work on adaptive product manuals, Pham and Setchi (2003) has identified and summarised the strong correlation that exists between users, tasks and support documentation (Fig. 6.7). That is extended for context-aware product support systems, as illustrated in Fig. 6.7. In adaptive manuals the main contextual information included the user profile while in this study involves the models discussed in section 6.2. That results in the following differences.

- The learning and task performing activities are not integrated, since the user can achieve both (i.e. learning and performing goals) by selecting the activity context he/she prefers. The content is then adapted accordingly by utilising different

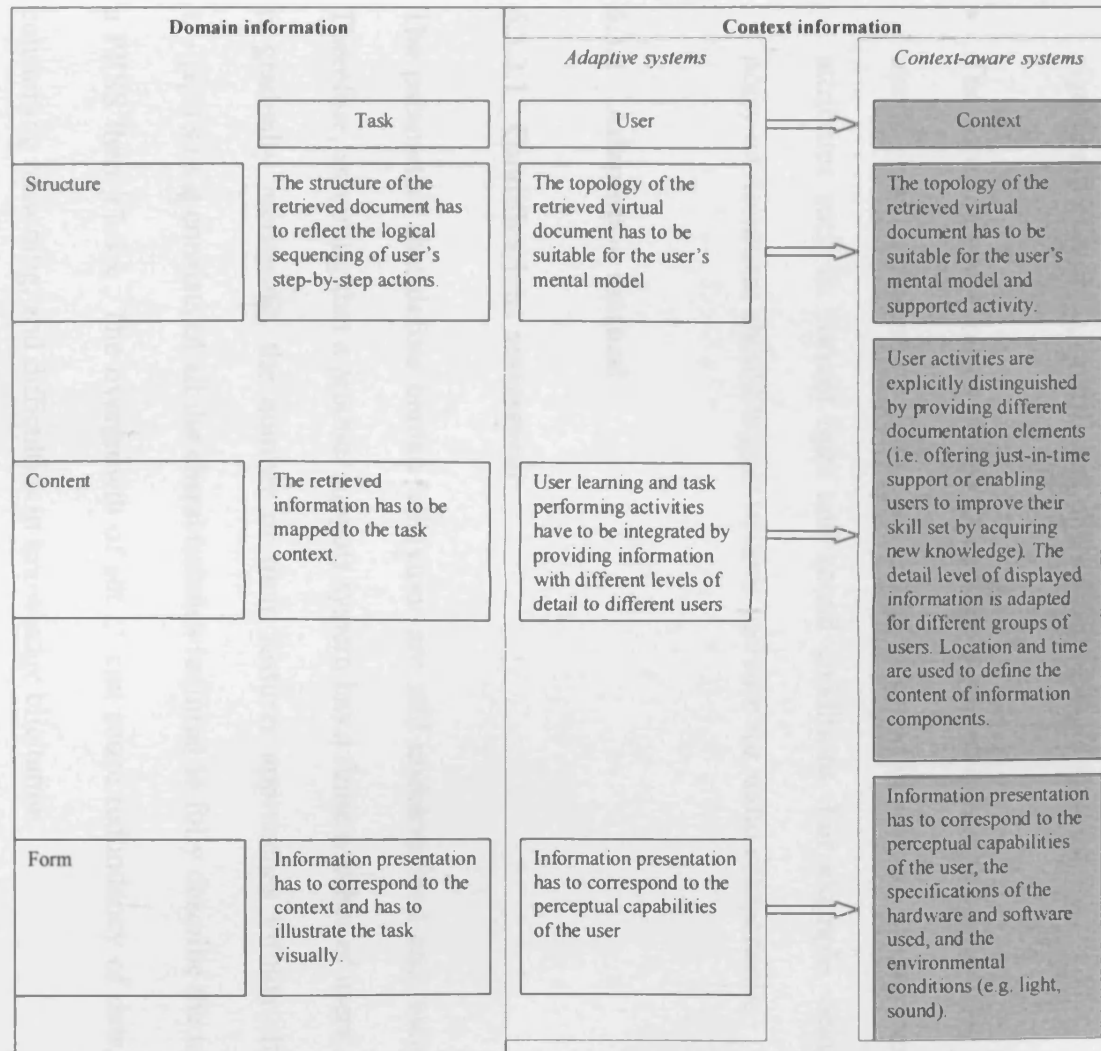


Figure 6. 7. Correlation between task, context and product support documentation and comparison with adaptive systems

documentation elements. The adaptation process is still however also related to the user.

- The location and time can also be used to define to content of the product support documents. For example, the service centres that a system should propose to a driver should be tailored to his/her current location. Location and time are mostly significant in mobile, distributed, and services related systems.
- The form or presentation of the document is not only adapted according to the user's characteristics and perceptual model but also in relation to environmental attributes such as current light and sound conditions. For example, being in a noisy environment should trigger volume increase for audio components.

6.3.2 Adaptation method

6.3.2.1 Qualifications assessment

The parameters that define human behaviour are still undetermined and uncounted. Therefore, assuming that a product support system has a finite number of users, which is gradually increasing, the number of their features approaches infinity. If a set $C = \{x_i : 1 \leq i \leq c\}$ consists of all the characteristics required to fully describe the users of a PRSS then $|C| \rightarrow \infty$. The overgrowth of set C can cause redundancy of data, time-consuming reasoning, and difficulties in knowledge elicitation.

A solution to the above problem can be based on the analysis given by Morakis et al. (2003) on the mathematical nature of hierarchy trees in information security, which is adapted for the current requirements as follows. There is a relation on set C with symbol R and $c_1, c_2, c_3 \in C$ for which the following is satisfied.

$$c_1 R c_1 \quad (6.2)$$

$$c_1 R c_2 \Rightarrow c_2 R c_1, \forall c_1, c_2 \in C \quad (6.3)$$

$$(c_1 R c_2 \text{ and } c_2 R c_3) \Rightarrow c_1 R c_3, \forall c_1, c_2, c_3 \in C \quad (6.4)$$

Then R is referred to as an equivalence relation and is denoted by ‘ \sim ’. Equivalence relations provide a method of classing together all the elements that are related to each other. The equivalence class of an element $c_i \in C$ is defined as the following subset.

$$[c_i] = \{x \in C : x \sim c_i\} \quad (6.5)$$

(6.5) indicates that it is possible to enumerate a small representative number of user characteristics relevant to the application domain and relate them to other user attributes through equivalence relations (e.g. generalisation is an equivalence relation). For example, in this study the qualifications of the users are assessed according to the user context ontology described in section 6.2.3. As a result, user attributes can be grouped according to the specialisation, expertise, and receptivity characteristics. User groups can accordingly be formed based on the aforementioned characteristics. For example, a user can belong to a group that is characterised by values “specialised” and “inexperienced”. The user qualifications model is created as the following non empty set.

$$E = \{[c_i] : 1 \leq i \leq n\} \quad (6.6)$$

(6.6) means that E represents the vertices of a directed graph consisted of all defined equivalence classes. The edges of this graph are defined by the following set.

$$D = \{([c_1], [c_2]) \in V \times V : [c_1] R [c_2]\} \quad (6.7)$$

(6.7) stands for different user groups that are linked to each other with equivalence relations. For example, the user group “specialised” and “inexperienced” is related to the group “specialised” with a generalisation relation. The relations between different groups create an acyclic directed graph which has n vertices (groups) and $n-1$ edges (equivalence relations between groups). Stereotypical classification of users has proved to be useful in several fields and superior to direct personalisation in information filtering (Kuflik et al. 2003).

6.3.2.2 Content adaptation

In order to create a context-tailored document the following parameters are employed.

- Context parameters including user’s stereotype group defined through qualifications assessment, selected activity, environmental and physical conditions.
- Documentation elements attributes including their form (i.e. text, image, etc.), type (i.e. explanation, description, etc.), and level of detail (i.e. detailed and concise).

The content is adapted by mapping different documentation elements attributes to different context instantiations. Table 6.1 shows the values of three different documentation attributes and Table 6.2 illustrates their usage for learn and perform activities and different user contexts. For example, in a scenario where an expert-specialised user wants to perform a task the attributes of the elements contained in the

Table 6. 1. Documentation element attributes

Documentation Element Attributes		
Type	Form	Level of Detail
Definition (DEF)	Text (TX)	Detailed (D)
Fact (FAC)	Image (IM)	Concise (C)
Information (INF)	Animation (AN)	
Rule_of_thumb (ROT)	Video (VD)	
Explanation (EXP)	Audio (AD)	
Description (DES)		
Example (EXA)		
Recommendation (REC)		
Comment (COM)		
Warning (WAR)		

Table 6. 2. Documentation element attributes for different contexts

		Qualification level		
		Expert-Specialised	Intermediate-Intermediate	Novice-Unspecialised
Activity	Learn	DEF-FAC-WAR-TX-IM-C	DEF-DES-FAC-WAR-ROT-TX-IM-VD-AD-D	DEF-DES-WAR-ROT-INF-TX-AN-VD-AD-D
	Perform	DEF-FAC-WAR-TX-IM-C	DEF-FAC-WAR-ROT-REC-TX-IM-VD-AU-D	DEF-INF-WAR-ROT-EXA-EXP-REC-TX-IM-AN-VD-AD-D

generated document should include definitions and facts, be limited to text and still images with concise level of detail.

As exemplified, the more specialised, expert, and receptive the user is considered the less clarification items such as explanations, rules-of-thumb, and comments are included in the generated and adapted documents. Furthermore multimedia and in particular animations and video are targeted towards inexperienced, unspecialised, and unreceptive audiences.

Learn and perform activities also require different content. For example, for novice-unspecialised users rules-of-thumb with examples are preferred in performing (considered more practically oriented approach) than only rules-of-thumb (considered more theoretically related approach).

6.4 RESPONDING TO INTERNAL STIMULI – INTEGRATING PRODUCT DESIGN AND SUPPORT DOCUMENTATION DATA

6.4.1 Product data

Domain knowledge is primarily synthesised from data related to the supported products and tasks. Although the structure of the tasks does not change very frequently, the life-cycle of the products introduced in the market is shortened continuously by either updating or modifying parts of them. In order to be able to respond to internal stimuli, these changes have to be captured as early as possible at

the design stage, where the data is produced from CAD/CAM tools, and be propagated to the product support one.

Most of the approaches towards database storage of CAD models (a solid or surface one) include representing the model in neutral format (such as IGES or ISO 10303, the Standard for the Exchange of Product model data, commonly referred to as STEP), and create images or volume/voxel representations. Design databases also typically contain images (drawings), and unstructured text (documentation) (Szykman 2001). Product knowledge is however continuously evolving, which results in excessive growth of the design databases contents and in problematic storage and retrieval of design data.

In the approach presented in this study, the knowledge base of the product support system contains a formal and generic description of product knowledge, the product ontology. Such a representation allows a product to exist in the absence of any specified geometrical characteristics at multiple levels of abstraction simultaneously, since it allows not only models of physical entities but also their abstracted concepts to be depicted. For example, a clutch can be modelled as a single entity at one level and as collection of components at another.

The support of conceptual abstractions at different levels allows the product ontology to be mapped to the design data irrespectively of the product's complexity and structure (i.e. it can be as simple as a single part).

In addition, the existence of an ontology lessens the ambiguities that can occur when multiple terms are used to mean the same things or when the same term is used with

multiple meanings, as in the case of distributed design data. The existence of a formal and shared dictionary (ontology) enables the association of the product support system's knowledge base with the design data, as explained in the following sections.

6.4.2 Required resources

The approach introduced in this section for integrating the knowledge base of a product support system and the design data, is based on the following resources.

- The product model (and ontology) is based on the structure of a product and its fragmentation into assemblies, subassemblies, and parts. Therefore, function and low-level technical characteristics (e.g. features) are not essential in this study. The main source for deriving the product's architecture is the Bill Of Materials (BOM).
- The slots in the model describe the attributes of the product's components (e.g. volume, size, etc.). Consequently, data derived from the models that describe these attributes are required. These can be produced very easily with a CAD/CAM system's analysis tools.
- The STEP files can be used to produce drawings, figures, and other multimedia except for textual descriptions. Furthermore STEP is a standard that contains information for both artefacts and documents (identification, versioning, structures, approvals, authorization, project, work order, requests, effectiveness, classification and properties) (Gao et al. 2003). The majority of the current CAD/CAM systems are able of creating STEP representations of the product design models.

6.4.3 Integration approach

6.4.3.1 Overall approach

The overall approach (Fig. 6.8) has three stages. First, data derived from the document base (where the design files are stored) are combined with information from the knowledge base in order to produce an integrated model based on the represented domain knowledge.

The second stage utilises information acquired from cases that describe different situations (based on previous experiences) and therefore may contain different product instantiations. Case-based knowledge is integrated with data derived from design-related data files (i.e. files created with CAD/CAM analysis tools describing attributes such as centre of gravity, mass, volume, material, etc.), as explained in section 6.4.3.3.

In the third stage the case that has been created is compared against the ontology that represents the domain. For example, if the new case expresses a situation where a space shuttle is depicted as the main product, while a product within the ontology must belong to one of the following categories: car, bus, lorry (meaning that the domain is restricted to automotive industry applications), the case is not considered valid for the specific universe of discourse. The inconsistency in this case is the value of the feature that stands for the product concept and has to be replaced with valid data. Inconsistencies can be characterised as syntactic and semantic ones. Syntactic problems include data types irregularities (e.g. the mass of a product should be a number with several decimal points and not a character) and values out of boundaries (e.g. restrictions on volume range), while examples of semantic conflicts include

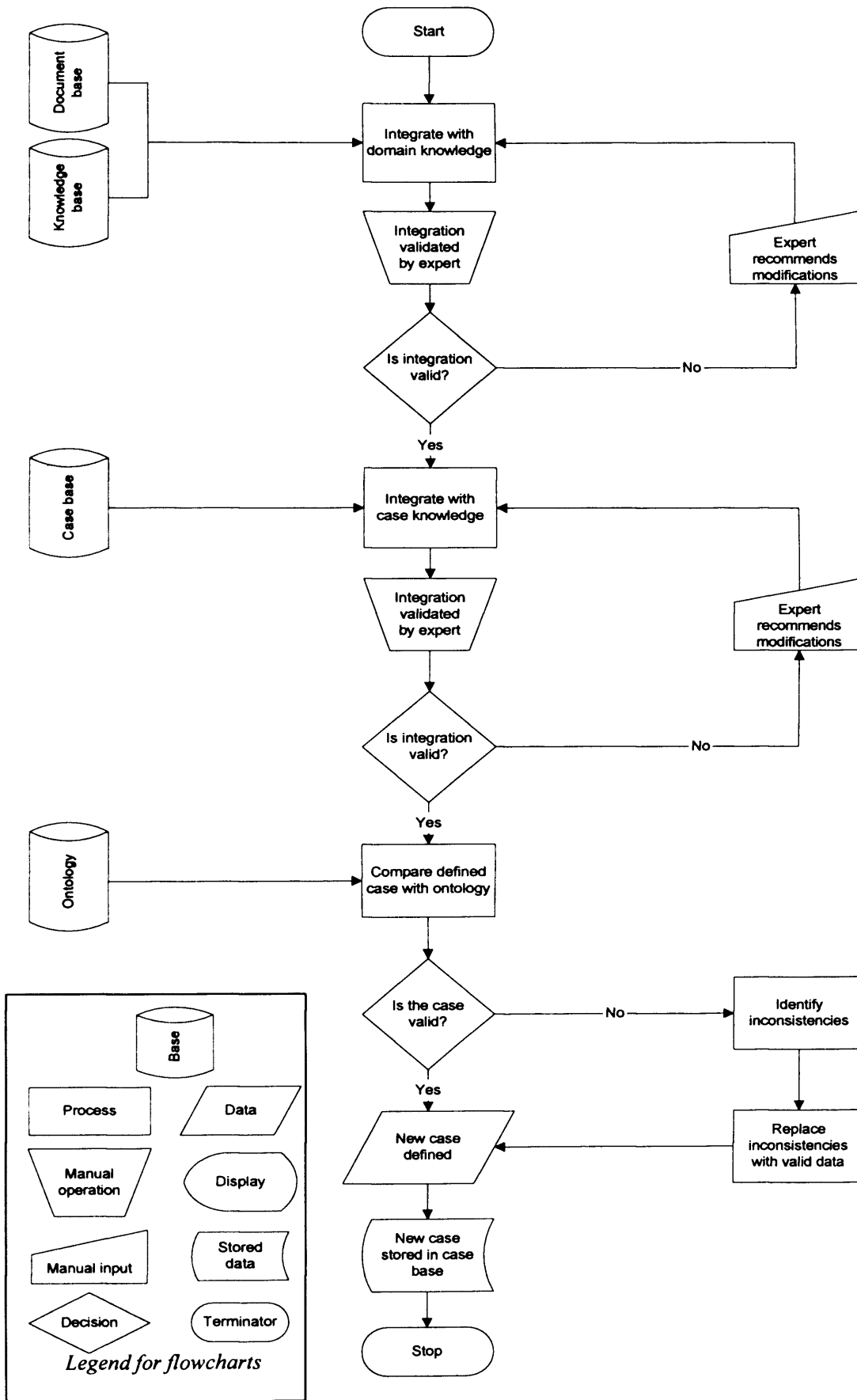


Figure 6. 8. Overall approach

contradictions with concept definitions (e.g. according to the ontology a concept has to be related with an aggregation relation “Has” with at least one “part” in order to be defined as “product”) and relations (e.g. a car can be a “product” within the domain of “automotive applications” only if it is related with the “subassembly” wheels with an aggregation relation, which denotes that all cars should have wheels).

The integration stages incorporate feedback paths that lead to the user (only users classified as experts are allowed to recommend changes, in order to avoid pitfalls), disseminating control over the integration process between the user and the system itself.

6.4.3.2 Integration of product design data with the knowledge base

Fig. 6.9 illustrates the approach followed for integrating product design data with the knowledge base. Initially the STEP files are used in order to create the BOM file(s) (this can be done with any CAD/CAM tool). For example, the abstracted form of a BOM file created by Pro/Engineer is shown in Figure 6.10(b), where <<NAME_...>> substitutes the name of the assembly, subassembly, or part defined by the designer. The BOM is constructed from a STEP file, as illustrated in Fig. 6.10 (Fig. 6.10(a) contains a very small part of a STEP file). In the example, the <<NAME_SUBASS_2>> and <<NAME_SUBASS_3>> are included in both files, while in BOM they are related to each other with the aggregation relation (denoted by “<<NAME_SUBASS_2>> contains <<NAME_SUBASS_3>>”). A Product Data Management (PDM) system or a simple directory structure can be used to store and manage those files. The user then chooses the BOM file on which the process will be based.

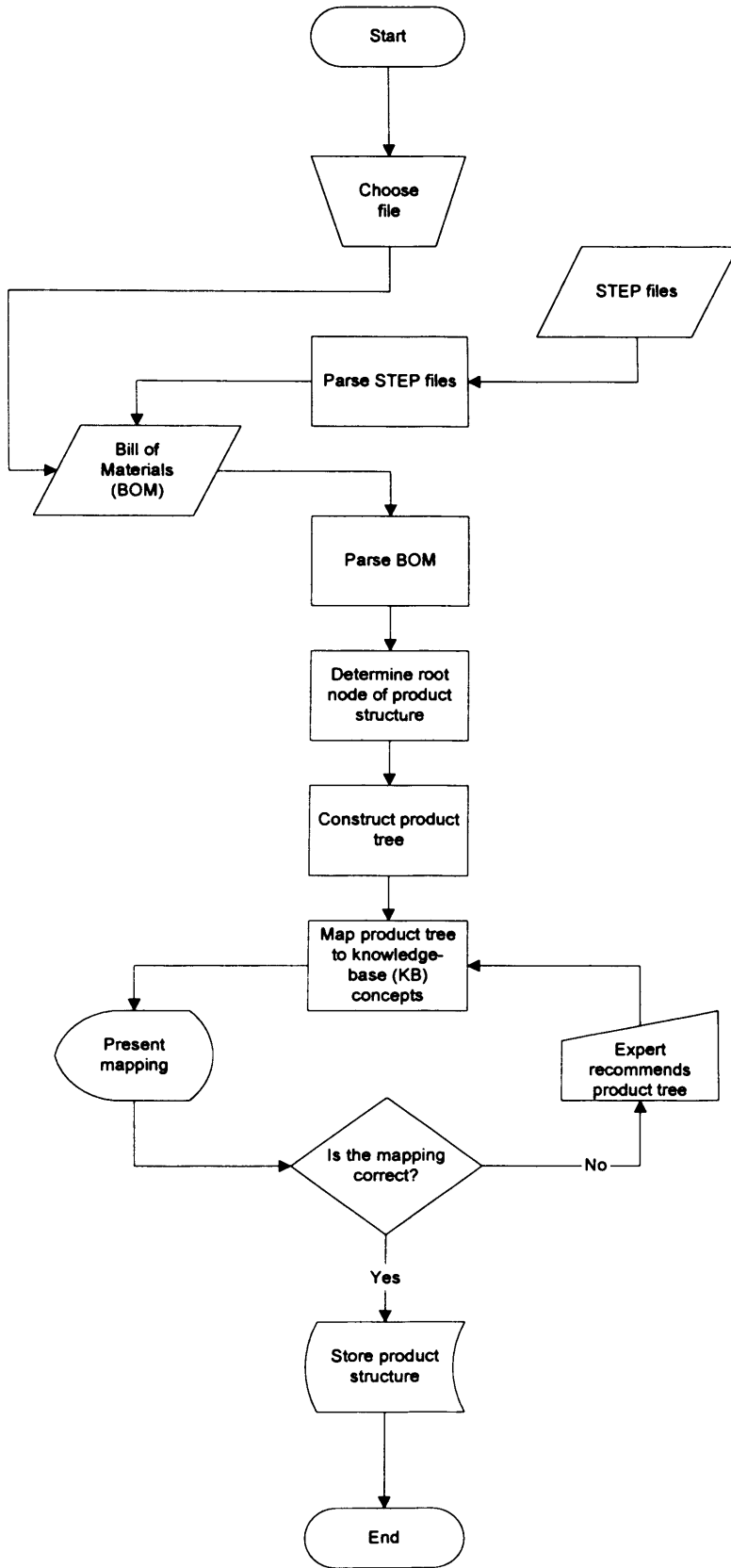


Figure 6. 9. Integration with domain knowledge

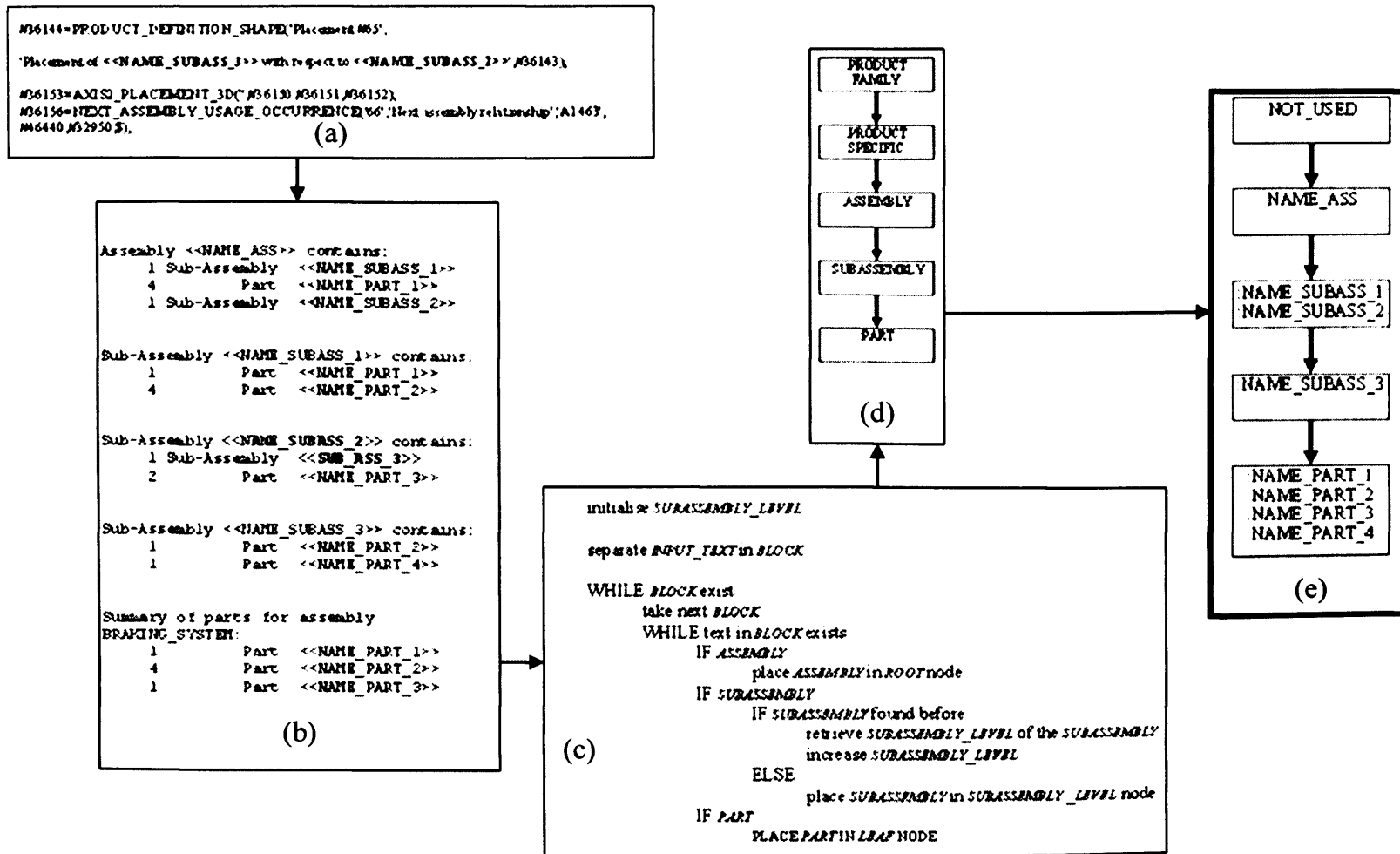


Figure 6. 10. Transforming a STEP file into an ontology-based product structure

The selected file is parsed in order to identify the structure of the product that is described by the BOM (the relations of between the different product components are aggregation relations in accordance with the Knowledge Base (KB)) and map KB concepts to the nodes of the structure. As illustrated by the pseudocode of the parsing algorithm (Fig.6.10(c)) the text of the selected file is segmented into several blocks of text. Each of these blocks represents a tree that includes a root node and leafs. The value of the artefact that contains all others is assigned to the root node. By processing all blocks of text and recording the relative position of one root node to another, a generic tree of the product components is created. Each node of the produced tree is assigned to a concept of the ontology that represents the product structure. For example in Fig. 6.10, <<NAME_SUBASS_3>> is recognised as a direct component of <<NAME_SUBASS_2>>, and therefore, in accordance with the ontology model, it is allocated as a subassembly of the assembly <<NAME_SUBASS_2>>. The concept “:PRODUCT_FAMILY” has the default value “:NOT_USED” because it represents a conceptual abstraction of real world products (e.g. a conceptual abstraction of a car is the family of four wheel vehicles), which BOM should not contain.

The allocation of the concepts is presented to the expert user. In case the result is deemed incorrect, the user can manually define the product structure or change any other parameters (i.e. different STEP file, different BOM file) and repeat the process again. Otherwise, the product structure is stored in a new case.

6.4.3.3 Integration of product design data with the case base

The second stage of the integration approach (Fig. 6.11) requires the selection of an existing case, on which the creation of a new case (and therefore new knowledge) is

based. The STEP files are utilised again in order to produce Data Files (DFs). An example template of a DF, as generated with Pro/Engineer's analysis tools, is shown in Fig. 6.12(a). The illustrated part contains information about the volume (in bold), surface area, average density, and mass product attributes among others. In Fig. 6.12(b) a case is displayed as the aggregation of features, their datatypes (e.g. String, double, etc.), and their values. In this example, one of the features represents the attribute "volume", which is also contained in the DF.

Once both the DF and case are selected, they are processed by the system. Based on the features included in the case the DF file is parsed (process "parse DF" in Fig. 6.11), in order to find relevant information. The parsing algorithm (Fig. 6.12(c)) retrieves the names of the case features and compares them with the text included in the DF. All the values of the matching words are retrieved (as a value is taken the number that is assigned to each word) and stored. In the next step, the feature values of the new case are filled with the data from the DF. For example, in Fig. 6.12(d) the value of the feature <<VOLUME>> is replaced with <<VALUE_1>>, which is acquired from the DF. All the other features retain the values of the initially selected case (e.g. <<FEATURE_1>> has <<VALUE_F_1>>, <<FEATURE_2>> has <<VALUE_F_2>>, etc.).

The new case is presented to the expert user, who verifies its validity and is able to manually replace any of the values (the inserted values are checked against the ontology, as explained in section 6.4.3.1). The case, if validated, is stored in the case base.

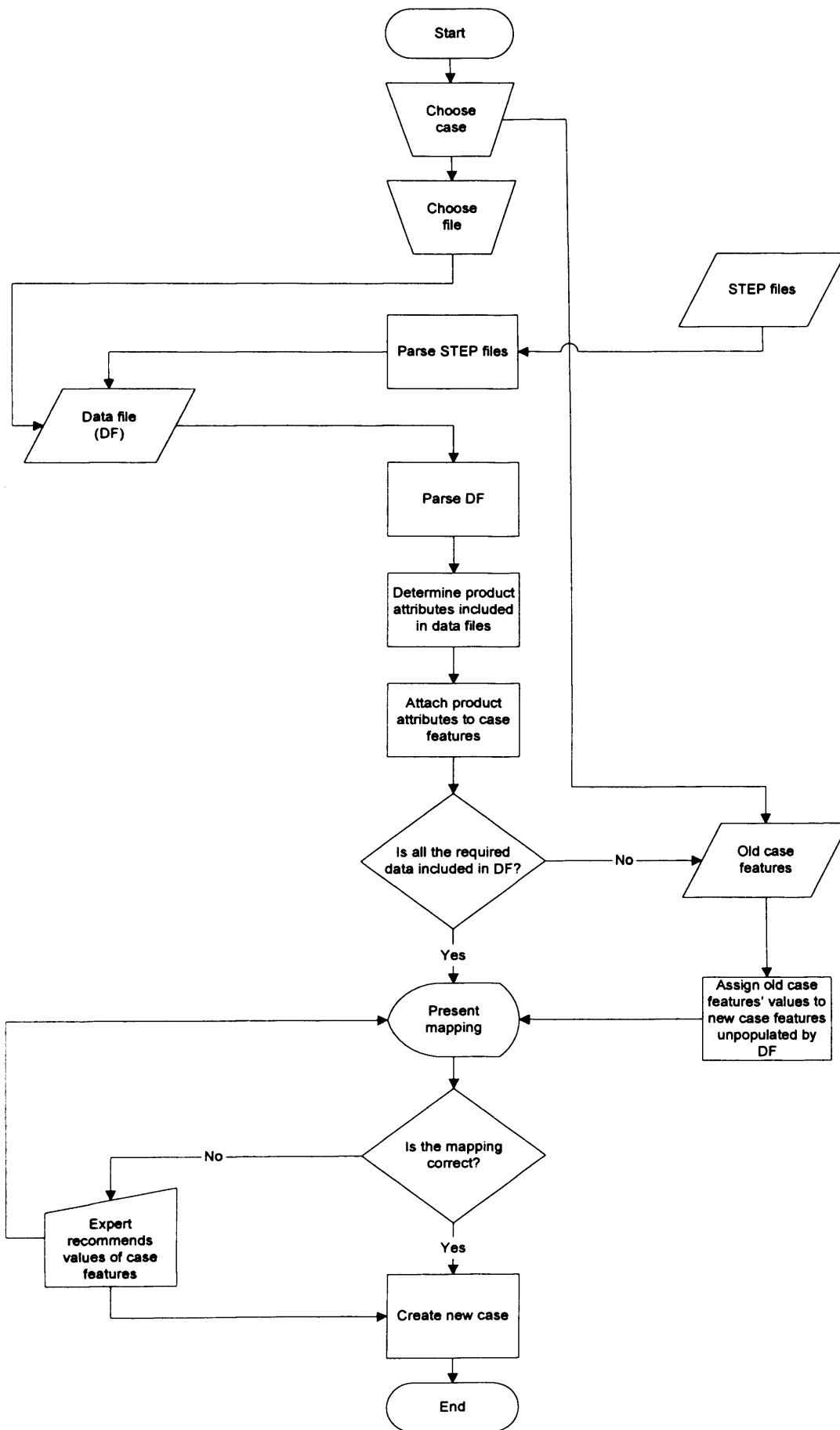


Figure 6. 11. Integration with case knowledge

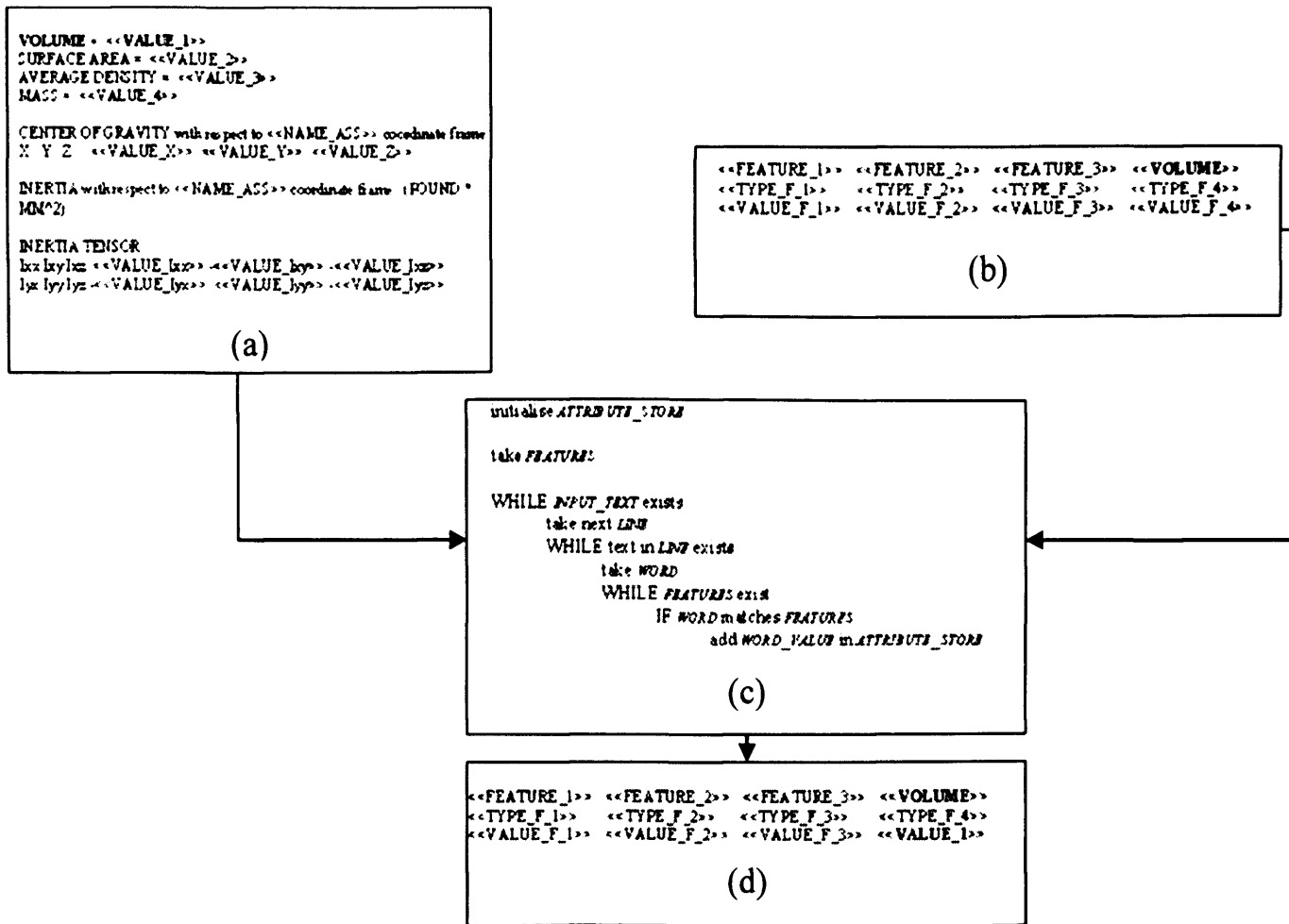


Figure 6. 12. Utilising a data file to create a new case

6.4.3.4 Supporting facilities

The integration approach introduced in the previous sections involves manual input and user feedback in several validation steps. To be able to recommend modifications on the automatically generated results, in-depth experience and up-to-date knowledge need to be readily available. Consequently, only users classified as experts are able to access this knowledge generation tool.

However, expertise alone is not enough as products and trends are continuously evolving. The addition of facilities that enable collaboration and access to information resources is necessary. A context-aware product support system supports these operations throughout the whole integration process by incorporating a range of visualisation environments and applications that allow access to design data (including STEP and BOM files) and the knowledge base at run-time.

6.5 SUMMARY AND CONCLUSIONS

Till today, adaptation in product support systems has been solely based on user models, delivering insufficiently adapted information. This chapter introduces the concept of context-aware product support systems, which are needed to meet the requirements of the user.

To facilitate the development of such systems, a context ontology is proposed that includes four basic models the activity, user, environment, and physical ones. The environment and physical models are particularly important in mobile and distributed applications. Utilising the activity model, performance support and e-Learning techniques can be integrated within the context of a single product support system to

achieve both performance and learning goals for the first time. Through the activity model, the task model (domain knowledge) can be reconfigured according to user needs. As a result, the user model is not considered independent of the other context models, creating a holistic view of different situations to which the system needs to react.

Context-specific content adaptation is achieved by adjusting the type, form, and level-of-detail documentation element attributes in different context instantiations. Furthermore, the delivered information is modified to reflect the configuration of the task model and the information describing supported products. It is therefore shown that knowledge about the context, the universe of discourse (including products and tasks), and documentation can be integrated to transform a product support system in a responsive entity able to adapt to different situations described with context-specific attributes.

The idea that both the external and internal stimuli can influence information delivery has been also introduced in this chapter. To illustrate this concept, an approach to automatically use design based data in product support is proposed. The approach utilises the concepts of ontology, case-based reasoning and their inter-relations in order to create new product support knowledge. This step presents an important contribution towards the semi-automatic integration of the design and product support stages of the product lifecycle.

CHAPTER 7

VERIFICATION AND VALIDATION

This chapter presents case studies that are used to validate the theories introduced in previous chapters and provide verification results based on different application scenarios.

7.1 SEMANTIC PRODUCT SUPPORT ENVIRONMENT: PROSON

7.1.1 PROSON environment

The technology solution developed in this work is called PROSON (PROduct Support ONtology). PROSON is based on the conceptual model of product support knowledge organisation and the methodology for PRSSs knowledge-base development introduced in chapter 4.

7.1.1.1 PROSON technical specification

As part of the methodology validation, clutch systems for automotive applications have been modelled. For reasons of copyright, the values of concepts' slots have been slightly modified but are deemed representative of general configurations. To keep the analysis manageable, a limited account of clutch characteristics (slots) and subassemblies (concepts connected to clutch with connector "has") is considered here as illustrated in Table 7.1.

Table 7.1. Clutch subassemblies and attributes considered in the case study

Abstract concept (Assembly)	
Concrete concept linked to “assembly” with the “is-a” relation (Clutch)	
Concepts connected to clutch with a connector (Subassemblies)	Cover_Plate
	Pressure_Plate
	Housing
	Flywheel
	Countershaft
	Synchroniser
Slots of clutch (Attributes)	Assembly_ID (Integer: 5342-8998)
	Assembly_Material (String: aluminium, carbon, magnesium)
	Assembly_Model (String: e.g. 4.5 Carbon Drive Ti)
	Assembly_Moment Of Inertia (Double: 52.255-95.904)
	Assembly_Radius (Double: 3.55-4.42)
	Assembly_Type (String: e.g. Cone, Conical_Disk, Disk, Hydraulic)
	Assembly_Weight (Double: 5.6-15.0)
	hasSubAssemblies (Concept: Subassemblies)

PROSON has been developed on the Windows platform. Four types of files are created after project initiation as follows.

- *.pins* has examples of the code for the knowledge base instances and constraints (Appendix A).
- *.pont* contains small part the code for the classes and their slots (ontology) (Appendix B).
- *.pprj* includes some of the code for the user interface of the project (Appendix C).
- *.jsp* (or Java Server Pages) contain the code that defines Information Objects, Information Object Clusters and Virtual Documents (small annotated parts are included in Appendix D).

The knowledge base, documentation components, and the general software technical specification are summarised in Table 7.2.

7.1.1.2 PROSON enabling technologies and tools

The following enabling technologies and tools have been used in this case study.

- Java 1.5.0_05 – a programming language to create the IOs, IOCs, and integrate them with the ontology.
- JCreator LE 2.5 (Stcherbatchenko 2005 et al.) – a software programming tool used to develop the *.jsp* files for the case study.

Table 7.2. Technical specification of the developed system components

(**System** indicates knowledge base components and software created by the tool itself and documentation components related to the ontology, while **direct** stands for knowledge base components and software coded by the author of this thesis).

TECHNICAL SPECIFICATION				
Knowledge base element		<i>System</i>	<i>Direct</i>	<i>Total</i>
	<i>Classes</i>	15	147	162
	<i>Slots</i>	34	110	144
	<i>Facets</i>	10	0	10
	<i>Instances</i>	0	369	369
	<i>Frames</i>	59	626	685
Documentation element	<i>Information Objects</i>	116	232	348
	<i>Information Object Clusters</i>	32	52	84
	<i>Total</i>	148	284	432
Software file lns (lns-lines of code)	<i>.pins lns (instances)</i>	1234	2145	3379
	<i>.pont lns (ontology)</i>	0	1772	1772
	<i>.pprj lns (user interface)</i>	1184	0	1184
	<i>.jsp lns</i>	0	782	782
	<i>Total lns</i>	2418	4699	7117
	<i>.gif, .jpeg, .bmp, .flash</i>	0	53	53

- COOL – a programming language (CLIPS object-oriented version) that is used as the frame-based ontology’s representation language.
- JBoss 2.0 – an application server to support the publishing process.
- Protégé 3.0 (Crubezy et al. 2005) – a free, open source ontology editor and knowledge-base framework that is based on Java and provides a foundation for customised knowledge-based applications. It supports frames, eXtensible Markup Language (XML) Schema (Fallside and Walmsley 2004), Resource Description Framework Schema (RDFS) (Brickley and Guha 2004) and the Web Ontology Language (OWL) (McGuinness and Van Harmelen 2004), which currently serves as the ontology standard proposed by the World Wide Web Consortium (W3C) community. Protégé 3.0 therefore provides a plug-and-play environment that makes it a flexible base for application development. The following Protégé 3.0 plug-ins have been employed in this study:
 - The OntoViz Tab (Sintek 2005a) allows the visualisation of Protégé ontologies.
 - The Protégé-OWL plug-in (Knublauch et al. 2004), which is an extension supporting the creation and management of OWL-based ontologies.
 - The Protégé Axiom Language (PAL) Constraints Tab (Yeh et al. 2003), which is a front end for the constraints system.
 - The RDF backend (Sintek 2005b), which facilitates the creation, import, and management of RDF(S) files.

- The Jambalaya Tab Widget (Storey et al. 2002) is a plug-in created for Protégé, which uses SHriMP to visualize regular Protégé and OWL knowledge-bases. SHriMP (Simple Hierarchical Multi-Perspective) (Rayside et al. 2003) is a domain-independent visualisation technique designed to enhance how people browse and explore complex information spaces.
- Apache Tomcat 4.1.31 (Bakore et al. 2004) – a servlet container that is used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies.
- Web browsers for PCs including Mozilla Firefox 1 and Internet Explorer 6 for displaying the developed product support virtual documentation components.

7.1.2 PROSON development

In accordance with the models presented in this chapter, PROSON is based on conventional frame ontology, thus making it accessible to a range of legacy applications. This means that PROSON can be exchanged as a static and complete information structure to third parties.

During the development of PROSON, several textbooks and formal resources were used as the general knowledge skeleton, which when organised and refined formed the knowledge base (Appendix E).

7.1.2.1 Architectural model development

As illustrated in Fig. 7.1, the clutch concept is connected to super and sub-concepts with the “is-a” relation.

In the figure “CLUTCH”¹ includes the code “*is-a Assembly*”, which denotes that clutch’s immediate super-concept is “ASSEMBLY”. The highest level of abstraction is represented by concept “THING”.

Connectors that express the “has” relation between “CAR”, “CLUTCH”, and “COUNTERSHAFT” have also been developed (Fig. 7.1). The range of the connector between the first two concepts is set to include all sub-concepts of “ASSEMBLY” including itself.

This means that all assemblies that are included in the knowledge base are car components. In the case of the latter two concepts, the connector is much more focused indicating several less abstract (compared to “ASSEMBLY”) concepts such as “COUNTERSHAFT”, which signifies that the subassemblies included in the knowledge base describe other assemblies of the car except for the clutch (e.g. transmission).

The role of the concepts is either “Concrete” or “Abstract”. For example “CLUTCH” is concrete as illustrated from the CLIPS line “*role concrete*”. This means that “CLUTCH” contains direct instances as opposed to “ASSEMBLY”.

¹ For this section, text in the form “CAPITALS” denotes a concept while “*Italics*” refers to a code fragment.

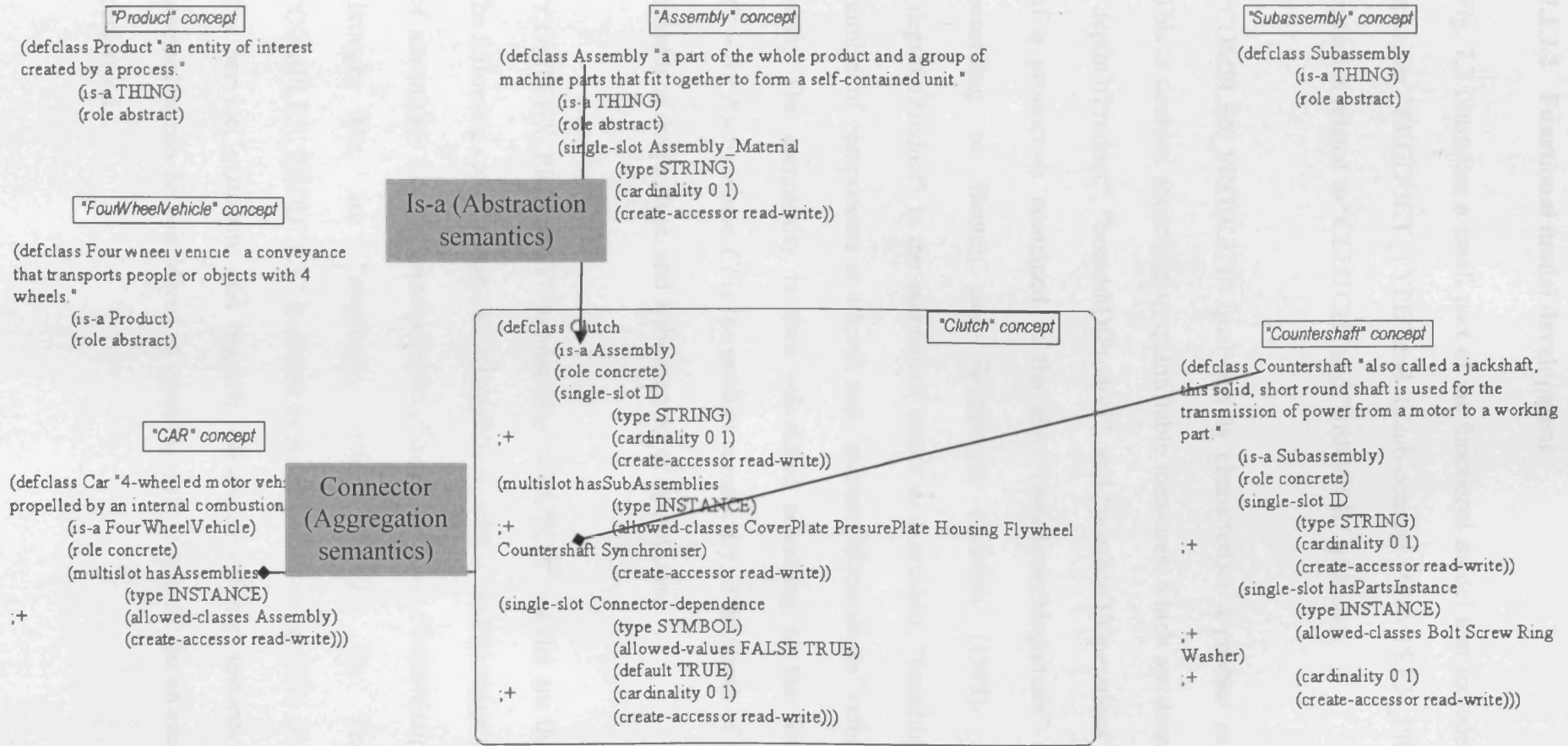


Figure 7.1. CLIPS code that describes small part of the architectural model for "Clutch" assembly

7.1.2.2 Functional model development

Fig. 7.2 illustrates a small part of the functional model that includes the knowledge-specifier “PRODUCT_TYPE” and its sub-concept “COMPLEX_PRODUCT”, which in turn is related to “CLUTCH” and “SERVICE” with arcs.

“COMPLEX_PRODUCT” qualitatively characterises a product as complex or not. This is decided according to quantifiable measures, which are described by the slots “depthOfProduct”, “breadthOfProduct”, and “numberOfOperations”. The complexity of a product is contained in the slot “complexityMagnitude” and is calculated according to Benton and Svistrava’s equation (1993). Based on that, “depthOfProduct” is the number of levels in a product, “breadthOfProduct” is the number of components at a level, and “numberOfOperations” refers to the stages of work. The complexity is then calculated according to the following equation, $Cl = \delta \times \beta \times \eta$, where Cl is the product complexity, δ the depth of the product, β the breadth of the product, and η the number of operations.

“COMPLEX_PRODUCT” is related to “CLUTCH” via the arc that is defined with the following code “*single-slot relatedToAssemblies*” and its range includes a number of assemblies i.e. “*allowed-parents Clutch Transaxle Transmission*”. Furthermore, through the arc “*single-slot relatedToTasks*” the knowledge-specifier “COMPLEX_PRODUCT” is related to combinations of specific assemblies and tasks (i.e. service, maintain, and inspect), which means that instances of the allowed assemblies can be considered as complex only if one of the aforementioned tasks is performed.

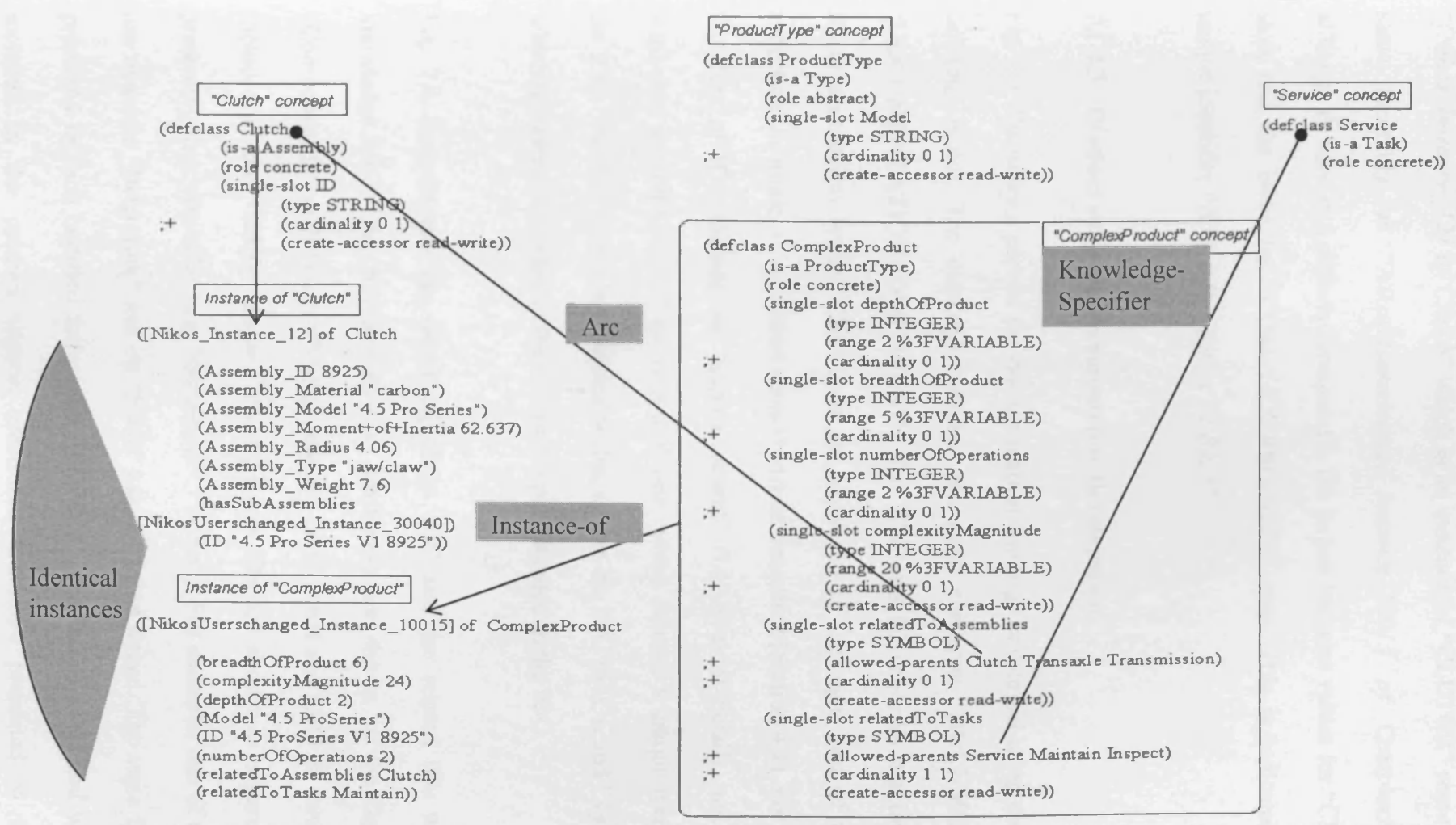


Figure 7.2. CLIPS code that describes small part of the functional model for "Clutch" assembly

“*[Nikos_Instance_12] of Clutch*” which is an instance of “CLUTCH” represents the same assembly as “*NikosUserschanged_Instance_10015 of ComplexProduct*”, although defined in a different manner i.e. the former includes values for “CLUTCH” slots and the latter for “COMPLEX_PRODUCT” slots. This is indicated by the unique identifier “*ID 4.5 Pro Series VI 8925*”.

7.1.2.3 Product support documentation development

Fig. 7.3 illustrates a part of the documentation ontology developed using the Protégé ontology editor. The ontology contains two main concepts: “DOCUMENT” and “DOCUMENTATIONCOMPONENT”. The IOC and IO concepts are included at lower abstraction levels. Their slots are displayed in the right part of the page, reflecting the model of a product support virtual document (section 4.2). For example, the slots of IOC include the “expressiveness” (i.e. level of detail), and “theme” attributes, as well as the “ComponentID” one, which denotes a unique identifier for the IOC, “hasIO”, which is a connector that relates the IOC to IOs, and “TypeOrder” which represents the order of the different types included in the IOC.

Fig. 7.4 demonstrates the modelling of an IOC and the related IOs within the knowledge base. The IOC’s slots (i.e. attributes) are shown in the figure (e.g. “ComponentID” and “Theme”) while the set of IOs that are parts of this IOC are included in the “hasIO” frame. Two of these IOs are shown as separate boxes presenting their attributes (e.g. “Behaviour”, “Form”). For example, one of these IOs has dynamic “Behaviour” and its “Form” has the value image. The same IO is also presented in each codified form (only HTML is used in this case) and when it is included in the product support electronic document presented to the user.

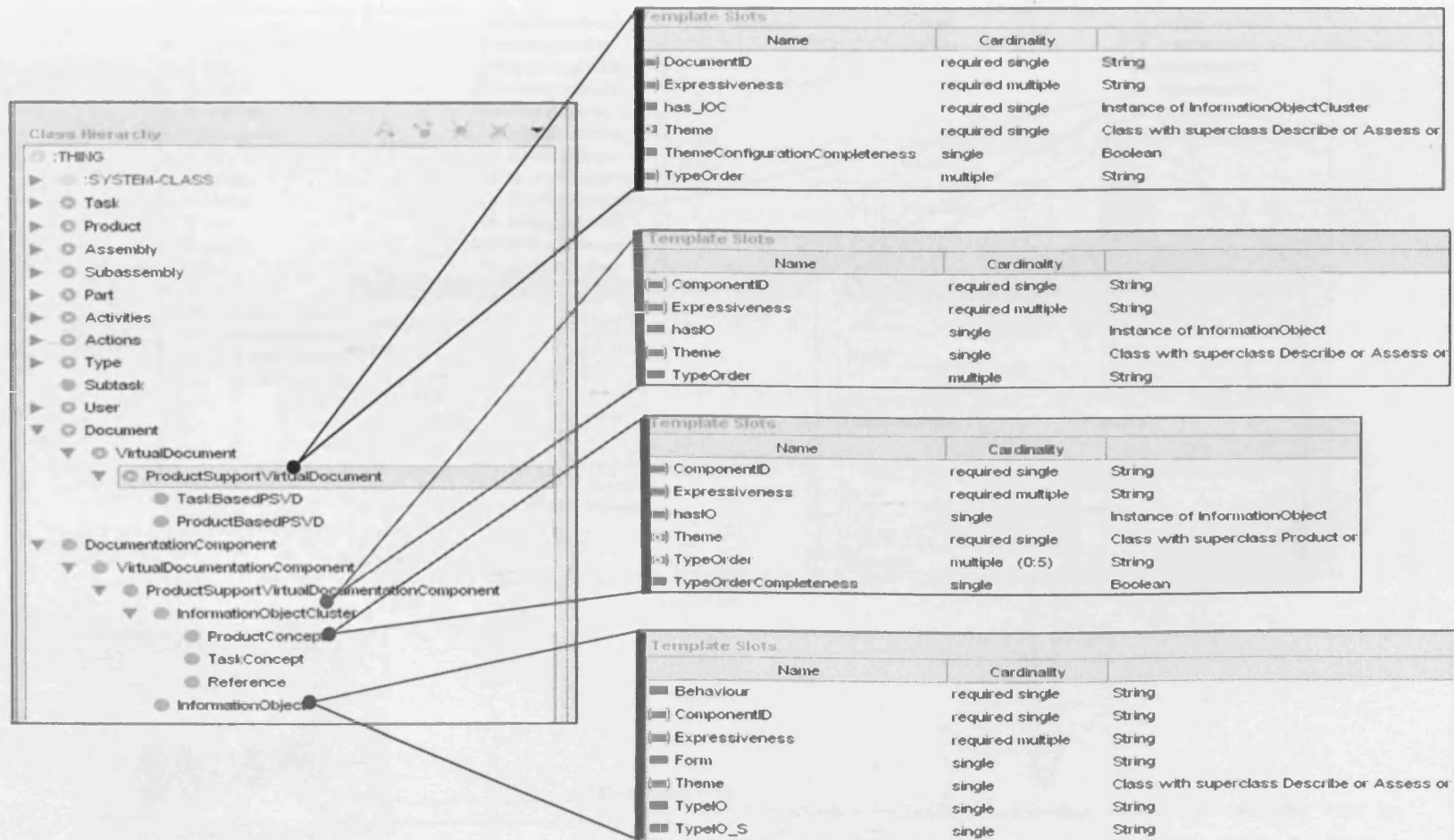


Figure 7.3. Product support documentation model

The screenshot displays the PROSON software interface, which is used for creating product support virtual documentation. The interface is divided into several panels:

- Class Hierarchy:** A tree view on the left showing a hierarchy of classes. The selected path is: `THING` > `SYSTEM-CLASS` > `Task` > `Product` > `Assembly` > `Subassembly` > `Part` > `Activities` > `Actions` > `Type` > `Subtask` > `User` > `Document` > `DocumentationComponent` > `VirtualDocumentation` > `ProductSupportV` > `InformationOI` > `ProductC` > `TaskCon` > `Referenc` > `InformationOI`.
- ComponentID:** A list of component IDs with their corresponding IDs. The selected component is `54765267638628862`.
- Expressiveness:** A panel showing the selected component's expressiveness settings, including `Concise` and `Complete`.
- Theme:** A panel showing the selected component's theme settings, including `Clutch`.
- Form:** A panel showing the selected component's form settings, including `Image` and `Description`.
- Rendered HTML:** A snippet of HTML code is shown, which is used to generate the virtual documentation page. The code is:


```
<IMG height=393
src="ExperiencedDescribeClutch_files/double disk clutch exploded view.jpg"
width=642></P>
<P align=center><B><FONT size=-1>Double-disk exploded</FONT></B></P>
```

Arrows indicate the flow of information from the software interface to the rendered HTML code, which is then used to generate the virtual documentation page.

Figure 7.4. Product support virtual documentation creation using PROSON

7.2 KNOWLEDGE-BASED PRODUCT SUPPORT SYSTEM: PROGNOSIS

7.2.1 PROGNOSIS environment

The technology solution developed is called PROGNOSIS. PROGNOSIS is based on the conceptual problem solving model of product support systems and the knowledge engineering framework for PRSSs presented in chapter 5.

7.2.1.1 PROGNOSIS technical specification

The ontologies and knowledge bases developed in section 7.1 (PROSON) are utilised as the backbone of PROGNOSIS. Additional components such as the case base and the reasoning engine are developed on Windows platform and are represented by the following types of files.

- *.java* includes the code for the reasoning engine, login authentication, and the links between framework modules.
- *.jsp* contains the implementation for the system, query, and case-based reasoning interfaces.
- *.txt* files represent the case bases developed within the system.

A summary of PROGNOSIS technical specification is listed in Table 7.3. The illustrated numbers are only indicative, since PSVDs are automatically created throughout the operation of the system.

Table 7.3. Technical specification of PROGNOSIS system components

(System keyword indicates software created by the tool itself while the direct keyword stands for software created by the developer manually).

TECHNICAL SPECIFICATION				
Software file lns		System	Direct	Total
	<i>.java lns</i>	5399	4464	9863
(lns-lines of code)	<i>.jsp lns</i>	1527	3044	4571
	<i>Total lns</i>	6926	7508	14434

7.2.1.2 PROGNOSIS enabling technologies and tools

Besides the tools and technologies utilised in the previous section the following tool was also employed in this work.

- FreeCBR – a case-based reasoning tool used to match and rank existing cases (Johanson 2005). Several characteristics and tools were added to the basic FreeCBR application, including adaptation mechanisms, validation mechanisms, and links to the product, task, and user knowledge bases.

7.2.2 PROGNOSIS system architecture

The PROGNOSIS system comprises a number of separate components. It has been designed using scalable Web-based language standards (e.g. XHTML, JSP, CLIPS, RDF(S)). Its modular architecture (Fig. 7.5) includes three independent layers.

- *The Knowledge Base Layer*, which contains the product, documentation, task, and user knowledge bases, as well as the case and file system bases. These can be accessed by the administrators or knowledge engineers/managers through specialised interfaces. Manipulation tools are provided for editing, maintaining, troubleshooting, and querying all knowledge bases, including Protégé editor and FreeCBR.
- *The Logic Layer*, which contains the algorithms for reusing, adapting, and generating new product support virtual documents. Three main groups of internal processes are also managed by the logic layer. User administration, authentication, and registering procedures are regulated by a program called LoginHandler (Appendix F). The ApplicationController (Appendix G) realises

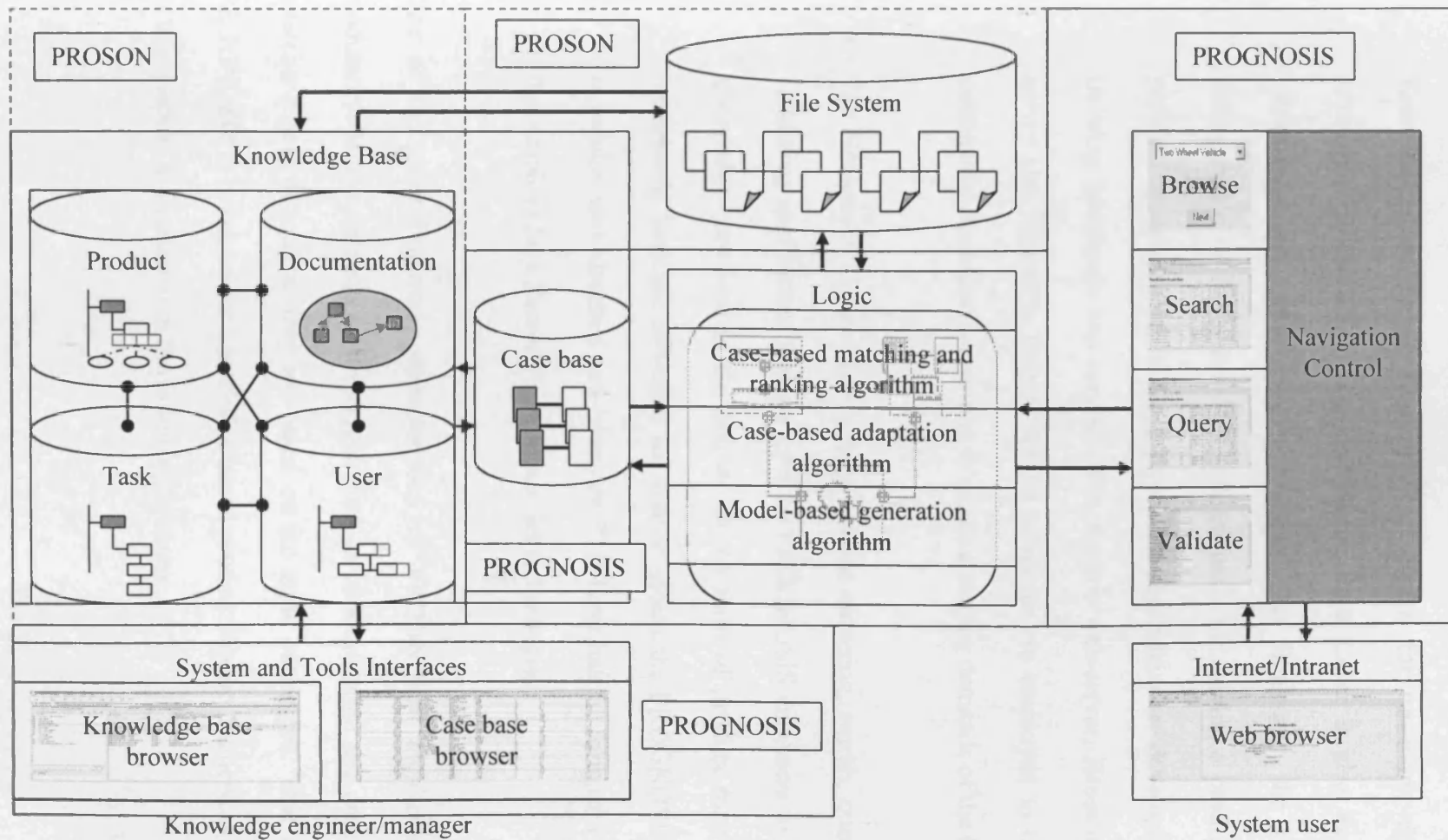


Figure 7.5. PROGNOSIS system architecture

connections between the different layers, the case base-knowledge base-file system modules, and the diverse tools offered from the user interface. The KnowledgeBaseWrapper (Appendix H) except for traversing through the knowledge contained in the Knowledge Base Layer it also explores the information that needs to be extracted. As illustrated the operational configuration of the system's functional units for solving a product support problem is selected in the logic layer. The algorithms are developed in Java, utilising JavaBeans and servlets. The Apache web-server, JBoss application server and Apache's Tomcat servlet container are employed to control the continuous fluctuations resulting from the changing demands of the users.

- *The Navigation Control Layer* encompasses browsing, search, querying, and validating applications that are used by PROGNOSIS end-users to access the knowledge contained in the system, in the form of product support virtual documents. Multiple end-users are able to access the PROGNOSIS interface in parallel with standard web-browsers. This layer has the form of a thin client that employs Java Server Page and servlet technologies.

One of the most important characteristics of PROGNOSIS architecture is its modularity. Major elements of the system such as its knowledge and case bases are interchangeable as long as they are based on the same ontologies. That is because PROGNOSIS is based on the knowledge based product support framework presented in this chapter, as illustrated in the following sections.

7.2.2.1 PROGNOSIS data space

The data space corresponds mainly to PROGNOSIS Knowledge Base Layer. Details of the constructs and structure of the knowledge included in this space (i.e. PROSON) can be found in section 7.1. Appendix I illustrates an integrated view of a part of PROSON in combination with the case-base introduced in the current chapter.

7.2.2.2 PROGNOSIS problem space

Fig. 7.6 illustrates the process of implicitly identifying the user's goal in PROGNOSIS. As explained in section 5.2.3, once the user logs in PROGNOSIS (Fig. 7.6(A)), the diagnosis goal is set. The first page (Fig. 7.6(B)) enables the user to either browse through existing pre-composed documents (Fig. 7.6(C)) or use one of the query tools located at the right frame of the page (Fig. 7.6(D)). These tools serve the information retrieval, diagnosis, and explanation functions. Each of them includes several cases contained in different case bases in order to retain the modular nature of the system (example cases can be found in Appendix J). However, the structure of the cases is homogeneous following the order shown in Fig. 7.6(E). In the leftmost cells the dimensions are described (i.e. "Activity" and "Action"). The next columns allow the selection of weights (i.e. "5"), search algorithms (i.e. "Fuzzy linear", "Inverted"), search terms (i.e. "="), and appropriate feature values (e.g. "Learn").

Fig. 7.7 depicts the connection between the PROSON knowledge-base and the case-based tools. In this scenario, the user requires more information about the countershaft. In PROSON the structure of the product is partly described through the *subassembly* concept, which in turn includes the *countershaft* concept. Therefore, "countershaft" is a value of the descriptor "SUBASSEMBLY" (part of the case base)

Pro Gnosis
Knowledge Based Product Support System

Please enter your Login Name, Password, and PIN to log in

Account:

Password:

PIN:

(A)

Diagnosis

WELCOME

LEGENDS INDEX

- Search
- Troubleshooter
- Expert Advisor

(C)

(B)

Product Name	Version	Release Date	Support Status
Product A	1.0	2000-01-01	Active
Product B	2.0	2000-02-01	Active
Product C	3.0	2000-03-01	Active
Product D	4.0	2000-04-01	Active
Product E	5.0	2000-05-01	Active
Product F	6.0	2000-06-01	Active
Product G	7.0	2000-07-01	Active
Product H	8.0	2000-08-01	Active
Product I	9.0	2000-09-01	Active
Product J	10.0	2000-10-01	Active

Information Retrieval

Search cases in case base

Case ID	Product	Version	Issue	Status
1001	Product A	1.0	Issue 1	Resolved
1002	Product B	2.0	Issue 2	Open
1003	Product C	3.0	Issue 3	Resolved
1004	Product D	4.0	Issue 4	Open
1005	Product E	5.0	Issue 5	Resolved
1006	Product F	6.0	Issue 6	Open
1007	Product G	7.0	Issue 7	Resolved
1008	Product H	8.0	Issue 8	Open
1009	Product I	9.0	Issue 9	Resolved
1010	Product J	10.0	Issue 10	Open

Diagnosis

Search Tool

Troubleshooter

Expert Advisor

(D)

Search cases in case base

Case ID	Product	Version	Issue	Status
1001	Product A	1.0	Issue 1	Resolved
1002	Product B	2.0	Issue 2	Open
1003	Product C	3.0	Issue 3	Resolved
1004	Product D	4.0	Issue 4	Open
1005	Product E	5.0	Issue 5	Resolved
1006	Product F	6.0	Issue 6	Open
1007	Product G	7.0	Issue 7	Resolved
1008	Product H	8.0	Issue 8	Open
1009	Product I	9.0	Issue 9	Resolved
1010	Product J	10.0	Issue 10	Open

Explanation

Performance or Educational

(E)

ACTIVITY [String]	5	Fuzzy linear	<input type="checkbox"/> Inverted	=	? [String]	
ACTION [String]	5	Fuzzy linear	<input type="checkbox"/> Inverted	=	? [String]	Learn Perform

Figure 7.6 Identifying goals by system usage

The screenshot displays the PROGNOSIS software interface, illustrating the connection between concepts in the PROSON knowledge base and concept-level features. The interface is divided into several main sections:

- Left Panel (Navigation Tree):** A hierarchical tree structure showing concepts. The 'Assembly' node is expanded, showing sub-nodes like 'Clutch', 'Transaxle', 'Transmission', 'Body', 'Subassembly', 'Countershaft', 'Housing', 'Synchroniser', 'PlateAssembly', and 'Flywheel'. Other nodes include 'Part', 'Activities', 'Actions', 'Type', 'Subtask', 'User', 'Document', and 'DocumentationComponent'. A 'Superclasses' section at the bottom shows 'Assembly' as a superclass.
- Top Panel (Feature Table):** A table with five rows, each representing a feature. The first column lists the feature name (e.g., PRODUCTSPECIFIC [String], ASSEMBLY [String], SUBASSEMBLY [String], PART [String], Assembly_Material [String]). The second column shows a value of '5'. The third column is 'Fuzzy linear'. The fourth column has an 'Inverted' checkbox. The fifth column is '='. The sixth column shows a dropdown menu with values like 'Car', 'Clutch', 'countershaft', '?', and 'aluminium'.
- Central Panel (Template Slots Table):** A table with columns 'Name', 'Cardinality', 'Type', and 'Other Facets'. It lists various slots such as 'Assembly_ID' (single, Integer), 'Assembly_Material' (single, String), 'Assembly_Model' (single, String), 'Assembly_Moment of Inertia' (single, Float), 'Assembly_Radius' (single, Float), 'Assembly_Type' (single, String), 'Assembly_Weight' (single, Float), 'Connector-dependence' (single, Boolean, default=true), 'hasSubAssemblies' (multiple, Instance of CoverPlate or Prest...), and 'hasSubAssemblies (instance of :STANDARD-SLOT)'.
- Right Panel (Slot Detail View):** A detailed view of the 'hasSubAssemblies' slot. It includes fields for 'Name' (hasSubAssemblies), 'Value Type' (Instance), 'Allowed Classes' (Flywheel, Countershaft, Synchroniser), 'Cardinality' (multiple checked), and 'Domain' (Assembly).

Dashed lines indicate the mapping between the 'Clutch' and 'countershaft' options in the top panel, the 'Clutch slots' label in the central panel, and the 'Countershaft' class in the right panel's 'Allowed Classes' list.

Figure 7.7. Connection between concepts in PROSON knowledge base and concept-level features in PROGNOSIS

that is used to express the same information as the *subassembly* concept in the ontology. The *clutch* concept (linked to the assembly ontology concept with the generalisation relation is-a) is related to *countershaft* via the aggregation relation *hasSubAssemblies*, which means that the countershaft is part of the clutch in automotive systems (indicated by the feature with dimension “PRODUCTSPECIFIC” and value “Car”). In the scenario conveyed by the illustrated case, the “Assembly_Material” descriptor (related to the *Assembly_Material* slot) has the value “aluminium”, denoting that the clutch is made out of aluminium.

7.2.2.3 PROGNOSIS hypothesis space

Matching and ranking the relevance of different cases according to the query is the first process taking place in the hypothesis space. That is based on the engine included in FreeCBR, which utilises the Euclidean distance algorithm discussed in section 5.2.4.1.

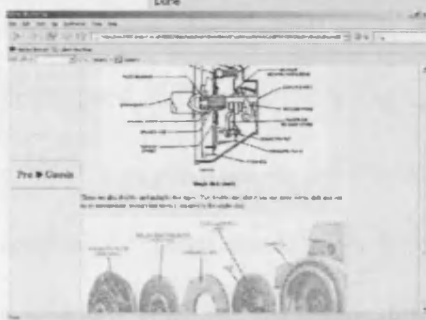
The selection of simple retrieval, case-based adaptation or model-based generation depends on the cases retrieved. Fig. 7.8(A) illustrates an example of a pre-composed document that describes the clutch assembly. The first picture included in Fig. 7.8(A) corresponds to single-disk clutches while the second picture portrays double-disk clutches. Fig. 7.8(B) shows a scenario where the user asks for a description of single-disk clutch. That is expressed by the highlighted row in Fig. 7.8(B), which is one of the retrieved cases that match the user’s query. The resultant virtual document includes only the documentation components related to single-disk clutches such as the related pictures and facts (denoted by the arrows), produced with parameter adjustment heuristics.

Firefox browser window showing a search interface with a table of results and a technical diagram of a single-disk clutch.

Hit number	Hit percentage	PRODUCTSPECIFIC	ASSEMBLY	SUBASSEMBLY	PART	Subassembly_Material	of disks
Searched values							
		= Car	= Clutch	= null	= null ?		
Results							
1	100.0	Car	Clutch	null	null	carbon	2
2	96.6	Car	Clutch	null	null	steel	1

The clutch mostly encountered is the single-disk one

Single-disk clutch



(B)

(A)

(C)

Firefox browser window showing a search interface with a table of results and a 3D model of a car.

Hit number	Hit percentage	PRODUCTSPECIFIC	ASSEMBLY	SUBASSEMBLY	PART
Searched values					
		= Car	= Body	?	?
Results					
1	100.0	Car	Body	null	null

DESCRIPTION OF A CAR

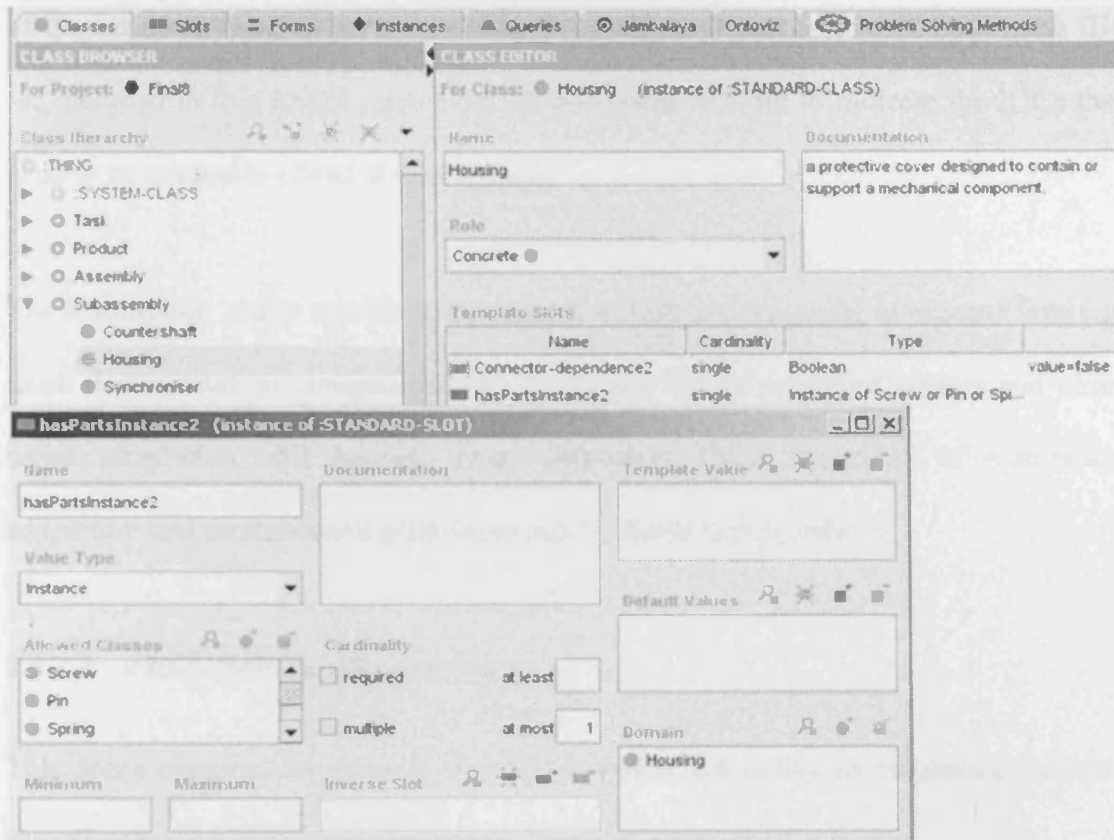
A car is a four wheel vehicle. This model is called BMW-330i. Its volume is 6.0969332e 00. Therefore it is considered as a large one.

Its CAD-generated image is illustrated below. More information is included in the CAD viewer.

Figure 7.8. Case-based adaptation

The next scenario involves a user who requests a description for the body of a car. The query can be satisfied by replacing the *Clutch* concept from the previous case with the *Body* concept (represented by the values of the “ASSEMBLY” dimension in the cases) since both of them are considered specialisations of the *Assembly* concept in the product ontology (Fig. 7.7). Reinstantiation is therefore utilised and the IOC related to the *Clutch* is replaced by the IOC describing the *Body* of a car, in order to respond to the changes between the previous and current selected cases. The resultant document is shown in Fig. 7.8(C).

A rather similar query is examined in the following setting where the user presumably asks for the description of a transaxle. However, in this scenario there is no pre-composed IOC corresponding to the *Transaxle* concept. This means that the IOC and PSVD content has to be created according to information generated based on the ontology-related models. A simple solution is to utilise the aggregation relations, as discussed earlier, between the subassemblies and parts that compose a transaxle additionally to other information found in the slots of each concept and the information extracted from the generalisation relations. For example, in Fig. 7.9 the text describing the *Housing* concept is a combination of its relation to the *Subassembly* concept (i.e. “A housing is a **Cls(Subassembly, FrameID(1:10133))**”), and to the parts *Screw, Pin, Spring, etc.* (“**Has Screw, Pin, Spring, Washer, Bolt parts**”). Furthermore, the definition of *Housing* is included in the *Documentation* slot and used in the generated document. As illustrated, the produced document follows the basic structure of a PSVD having title (“DESCRIPTION OF A TRANSAXLE”) and introduction (“In four wheel vehicles and specifically in cars”) related to the features of the case and the themes of the PSVD (i.e. Describe, Transaxle). The body



DESCRIPTION OF A TRANSAXLE

In four wheel vehicles and specifically in cars

A transaxle is a `Cls(Assembly, FrameID(1:10079))`, combines the functionality of the transmission, the differential and the drive axle into a single unit. Transaxles are near universal in all automobile configurations that have the engine placed at the same end of the car as the driven wheels: the front-wheel drive, rear wheel drive and mid-engined arrangements. Has Housing, Countershaft subassemblies [to_image](#)

A housing is a `Cls(Subassembly, FrameID(1:10133))`, a protective cover designed to contain or support a mechanical component Has Screw, Pin, Spring, Washer, Bolt parts [to_image](#)

A countershaft is a `Cls(Subassembly, FrameID(1:10133))`, also called a jackshaft, this solid, short round shaft is used for the transmission of power from a motor to a working part. Has Bolt, Screw, Ring, Washer parts [to_image](#)

A pin is a `Cls(Part, FrameID(1:10160))`, cylindrical tumblers consisting of two parts that are held in place by springs, when they are aligned with a key the bolt can be thrown [to_image](#)

A ring is a `Cls(Part, FrameID(1:10160))`, a rigid circular band of metal or wood or other material used for holding or fastening or hanging or pulling [to_image](#)

A spring is a `Cls(Part, FrameID(1:10160))`, a device on the suspension system to cushion and absorb shocks and bumps and to keep the vehicle level on turns. After the stress or pressure exerted by the flexing of the spring has been removed, the spring returns to its original state. The spring does this by first absorbing and then releasing a certain amount of energy. The form of spring may be leaf springs, coil springs, torsion bars, or a combination of these [to_image](#)

Figure 7.9. Model-based generation

of the document is a composition of subassembly and part related IOCs. Frame IDs are included in this initial version of the document in order to indicate the IOCs that need to be manually edited at a later stage.

The assumption is that model-based generated solutions are going to become fewer as more documentation components are developed by the technical writers and case-based adaptation will become more important. Other scenarios of case-based adaptation and model-based generation can be found in Appendix K.

7.2.2.4 PROGNOSIS solution space

This space contains the created documents and a tool aiding in validation and new case creation. More information can be found in Appendix L.

7.3 CONTEXT-AWARE PROGNOSIS

7.3.1 Environment and enabling technologies

Context-awareness and the approach proposed in this study are illustrated through a case study that involves PROSON ontology (see section 7.1) and PROGNOSIS (see section 7.2). In addition to presented enabling technologies, Pro-Engineer (PTC 1998) has been used to create, process, and manipulate design files (i.e. STEP) and NSViewer (STEPSTONE 2006) to reconstruct and view STEP-based representations. Furthermore, GIMP and QuickScreenCapture have been utilised in capturing and processing the images created from the design files, in order to make them understandable for non-specialists.

7.3.2 Context-based adaptation in PROGNOSIS

7.3.2.1 Ontology-based context model

Fig. 7.10 illustrates a small part of the code (in CLIPS) used to create the context ontology. The root concept for all context related concepts is “CONTEXT_ENTITY”². “USER”³ and “PHYSICAL” are the specialisations of “CONTEXT_ENTITY” as signified by the relation “*is-a Context_Entity*”. “USER” has slots that describe two other relation types, which are the aggregation and dependency ones.

The “*hasCharacteristic*” aggregation relation links “USER” with the “CHARACTERISTIC” concept and its instances. The cardinality of “*hasCharacteristic*” is denoted as “*multislot*”, which means that a user can have several characteristics. These are taken from instances of the concepts “EXPERTISE”, “SPECIALISATION”, and “RECEPTIVITY” (designated by the code “*allowed-classes Expertise Specialisation Receptivity*”). These instances are created by populating the slots of each of the concepts with appropriate values (Appendix M). For example, “EXPERTISE” has slots to describe the training time (“*single-slot TrainingTime*”), the past projects that the individual has participated (“*single-slot PastProjects*”), the ones that are relevant to the current project (“*single-slot PastRelatedProjects*”), and the number of years that the user is working on such tasks (“*single-slot ProfessionYears*”).

² For the rest of this section, text in the form “CAPITALS” denotes a concept while “*Italics*” refers to a code fragment.

³ “USER” in Fig. 7.10 denotes an abstract entity defined by the system in order to store the knowledge base. It has nothing to do with “User”, which is the concept that represents the users of the system.

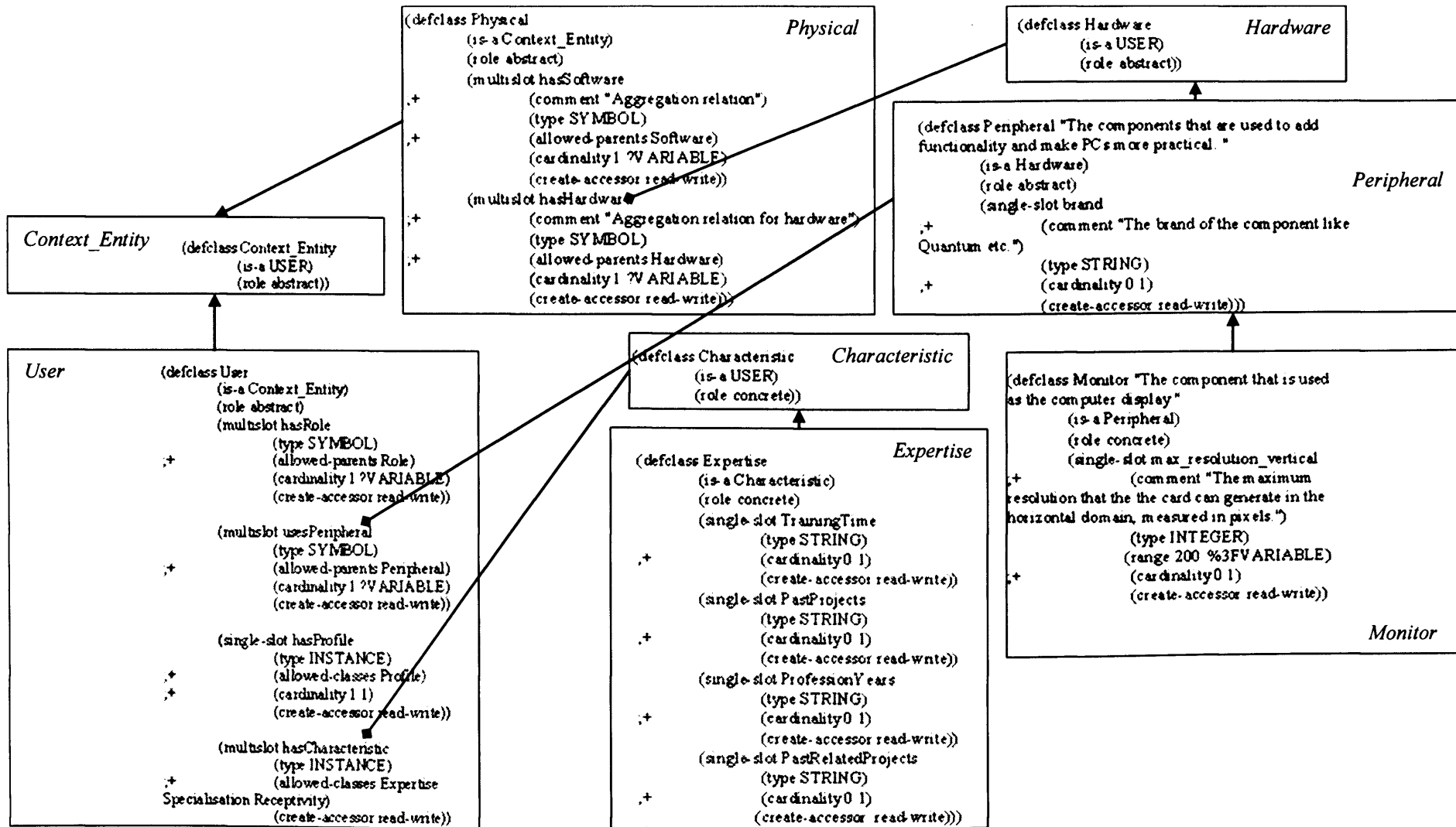


Figure 7. 10. Part of the context ontology in PROGNOSIS


The “usesPeripheral” dependency relation connects “USER” with the “PERIPHERAL” concept, which in turn is related to “PERIPHERAL” through generalisation (“*is-a Hardware*”) and aggregation relations (“*hasHardware*” in “*defclass Physical*”). One of the peripherals that is continuously used from people is the computer monitor (i.e. “MONITOR”). More code fragments describing the context ontology can be found in Appendix N.

7.3.2.2 Adaptive information delivery

To facilitate comparison between different contexts, it is assumed that two types of users, experienced and inexperienced, are interested in either designing a clutch (perform activity) or acquiring more information about clutch design (learn activity). The differences identified between traditional learning and performing applications are summarised in Appendix O. PROGNOSIS, the prototype system presented in section 7.2 and extended with context models to provide context-specific information, is used to enable users to pose queries and retrieve solutions adapted to their needs (shown in Fig. 7.11-7.14). Several documentation types are utilised in this study as listed in Table 7.4, to illustrate the differences between different use cases (more scenarios can be found in Appendix P).

CASE ONE (Fig. 7.11), involves an inexperienced user who is performing the task of designing a clutch. The presented information includes information, rules-of-thumb, and warnings. The goal, constraints metrics, metric attributes, and calculations are specific to the supported product, which in this case is the “clutch”. Example metrics are the torque and moment of inertia of the clutch. Furthermore, tools that can be used

Table 7. 4. Symbols used in case study

		Symbol
Documentation type	Information	i
	Rule-of-thumb	
	Warning	!
	Definition	No Symbol
	Description	D
	Assignment	A
	Case study	C
	Fact	No Symbol

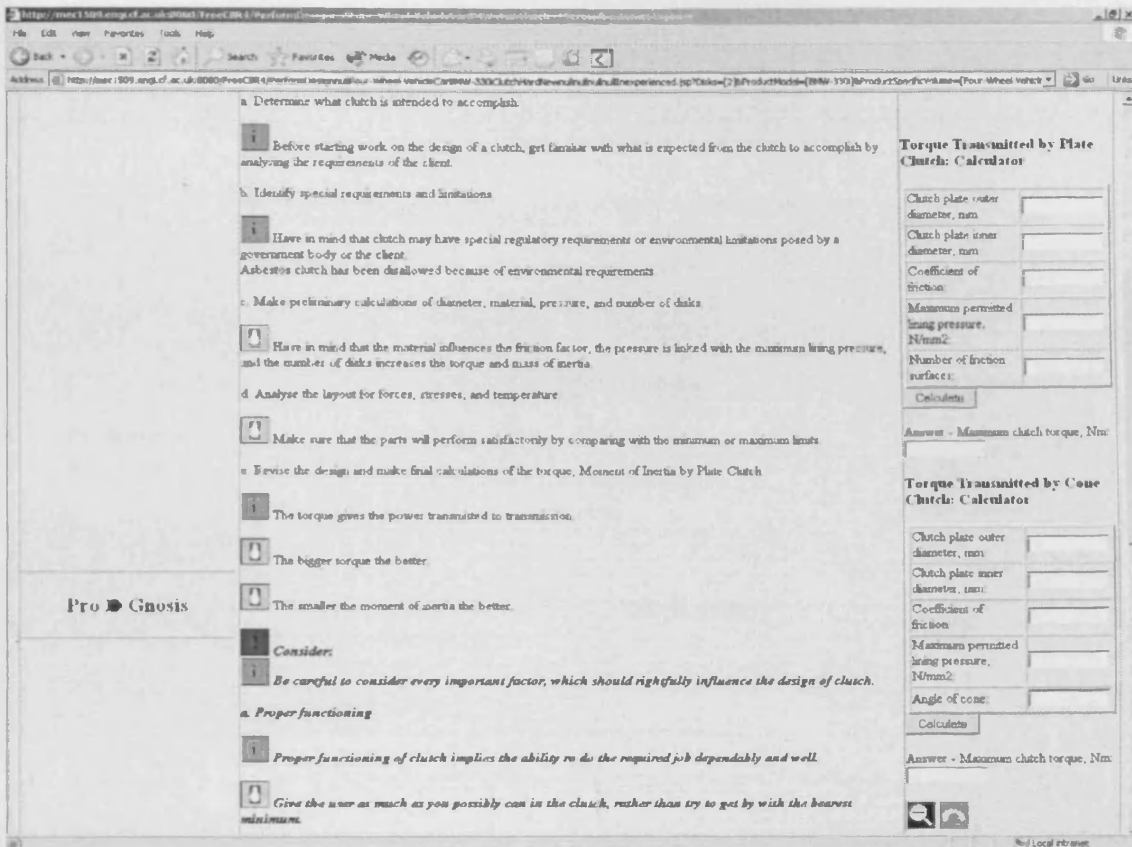


Figure 7.11. Case 1: An inexperienced user designs a clutch (perform activity)

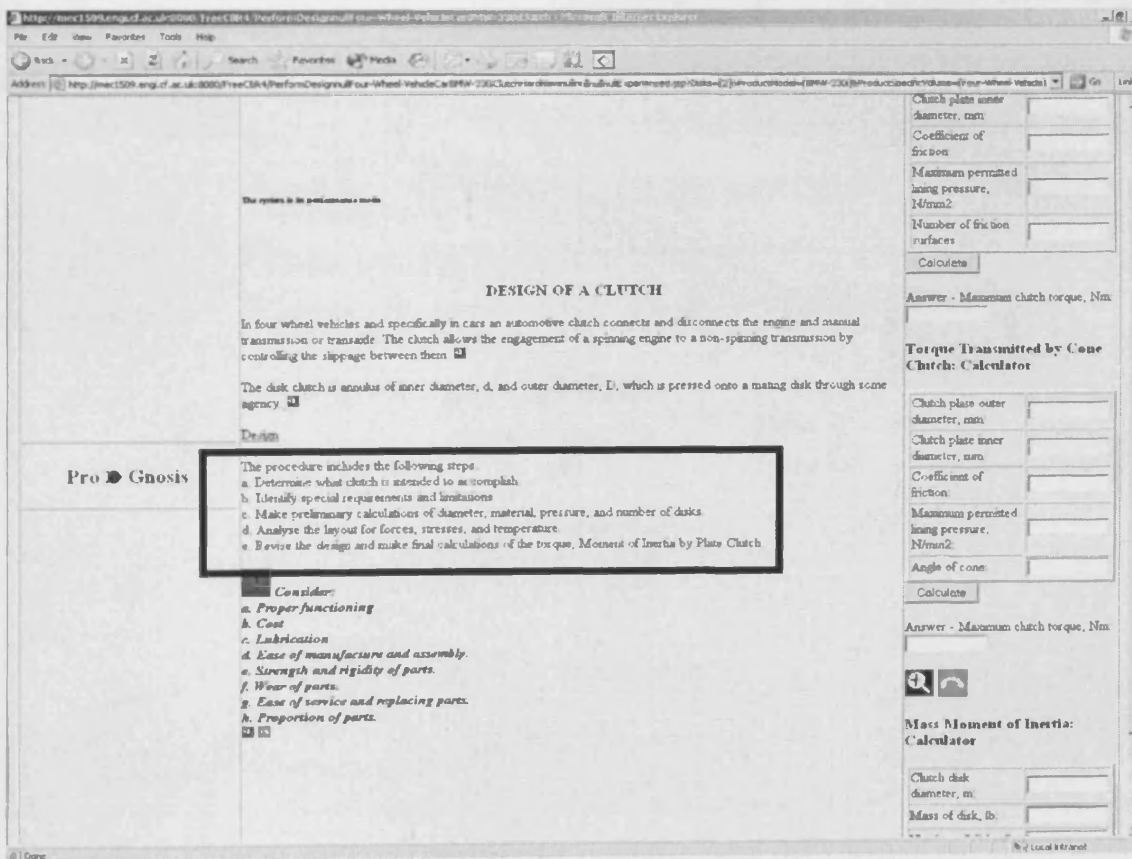


Figure 7.12. Case 2: An experienced user designs a clutch (perform activity)

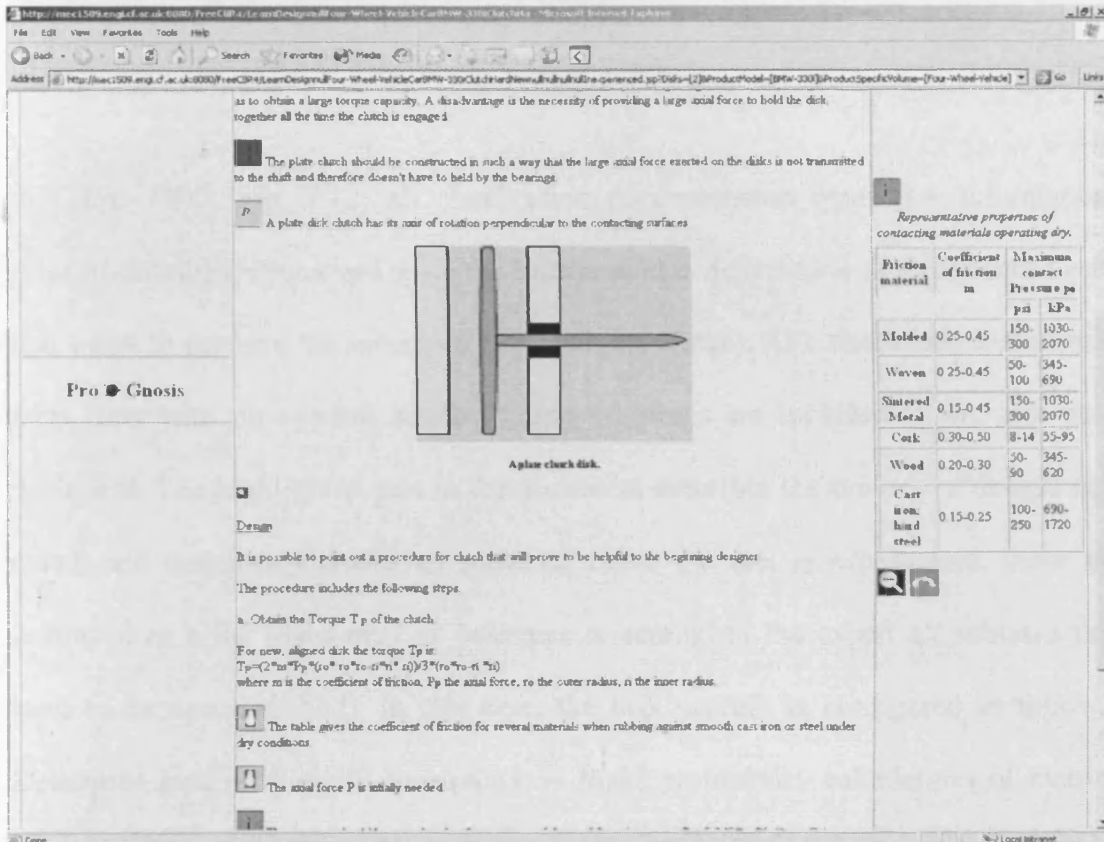


Figure 7.13. Case 3: An inexperienced user requests more information on clutch design (learn activity)

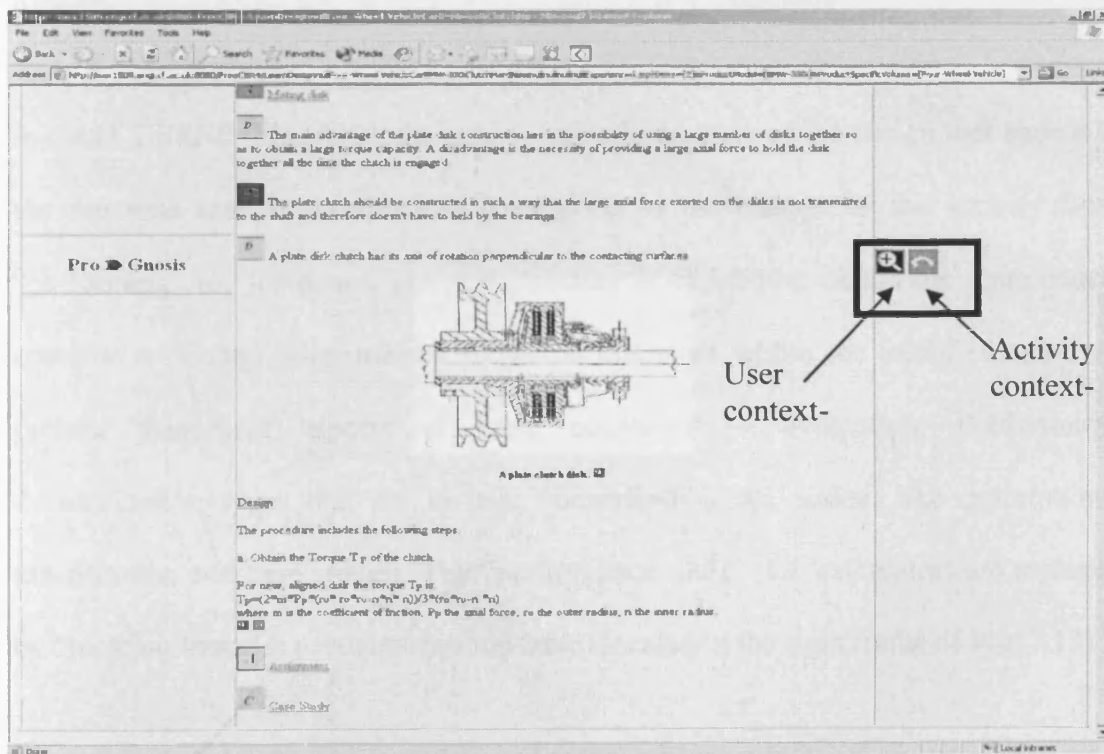


Figure 7.14. Case 4: An experienced user requests more information on clutch design (learn activity)

to ease the design process are provided, such as the torque calculator contained in the right frame of the product support virtual document.

In *CASE TWO* (Fig. 7.12) all clarification documentation types (i.e. information, rules-of-thumb) are removed since the information is delivered to an experienced user that wants to perform the same task (i.e. design a clutch). As a result only definitions, facts (text with no symbol attached), and warnings are included in the presented document. The highlighted part in the document describes the process of designing a clutch and therefore includes all subtasks. Since the user is experienced, these are portrayed as a list (used only as reference to remind to the expert all subtasks that have to be accomplished). In this case, the task process is configured as follows: Determine goal → Identify constraints → Make preliminary calculations of metrics → Analyze layout of metric attributes → Revise and make final calculations of metrics.

In *CASE THREE* (Fig. 7.13), the task → subtask structure of the design task shown in the previous case is modified to correspond to the change of the activity from “performing” to “learning”. The new structure is as follows; obtain the main metric equation → obtain other metrics equations that exist within the initial equation → present theoretical aspects of these equations → evaluation. Furthermore, documentation types that aid in user understanding are added, like descriptions, assignments, and case studies. The “performance tools” (i.e. calculator) are replaced by “learning tools” (i.e. summarization table) located in the right frame of Fig. 7.13).

CASE FOUR (Fig. 7.14), corresponds to an experienced user seeking for more information on clutch design (i.e. user “experienced”, activity “learning”).

Information documentation types are therefore removed while other types that are contained within both cases have different levels of detail (e.g. the pictorial description of a clutch is more concise for an experienced user). The highlighted part of Fig. 7.14 shows the “context-change” buttons, which enable the user to change context at run-time and access documents belonging to another context instantiation, formulating a highly interactive experience.

7.3.3 Responding to internal stimuli in PROGNOSIS

Fig. 7.15 illustrates a scenario in which a case is created based on the data retrieved from CAD related files (other scenarios can be found in Appendix Q). In this example, the user decides to structure the case based on the STEP file produced during the design of a driving axle (parts of this file can be found in Appendix R). Part of the product’s BOM is shown in Fig. 7.15 (a) (other tested BOM files can be found in Appendix S). Fig. 7.15 (b) shows the screen that enables the user to choose different BOM files (by clicking the “VERIFY CHOSEN FILE” button) and/or use any of the supporting facilities (see Appendix T).

Parsing the BOM file selected by the user as input (see Appendix U for a part of the parsing code), the system constructs a tree of the product and categorises its components according to the PROGNOSIS ontology. This means that the classification distinguishes between products, assemblies, subassemblies, and parts. Fig. 7.15 (c) demonstrates the recommendation of the system for the current setting.

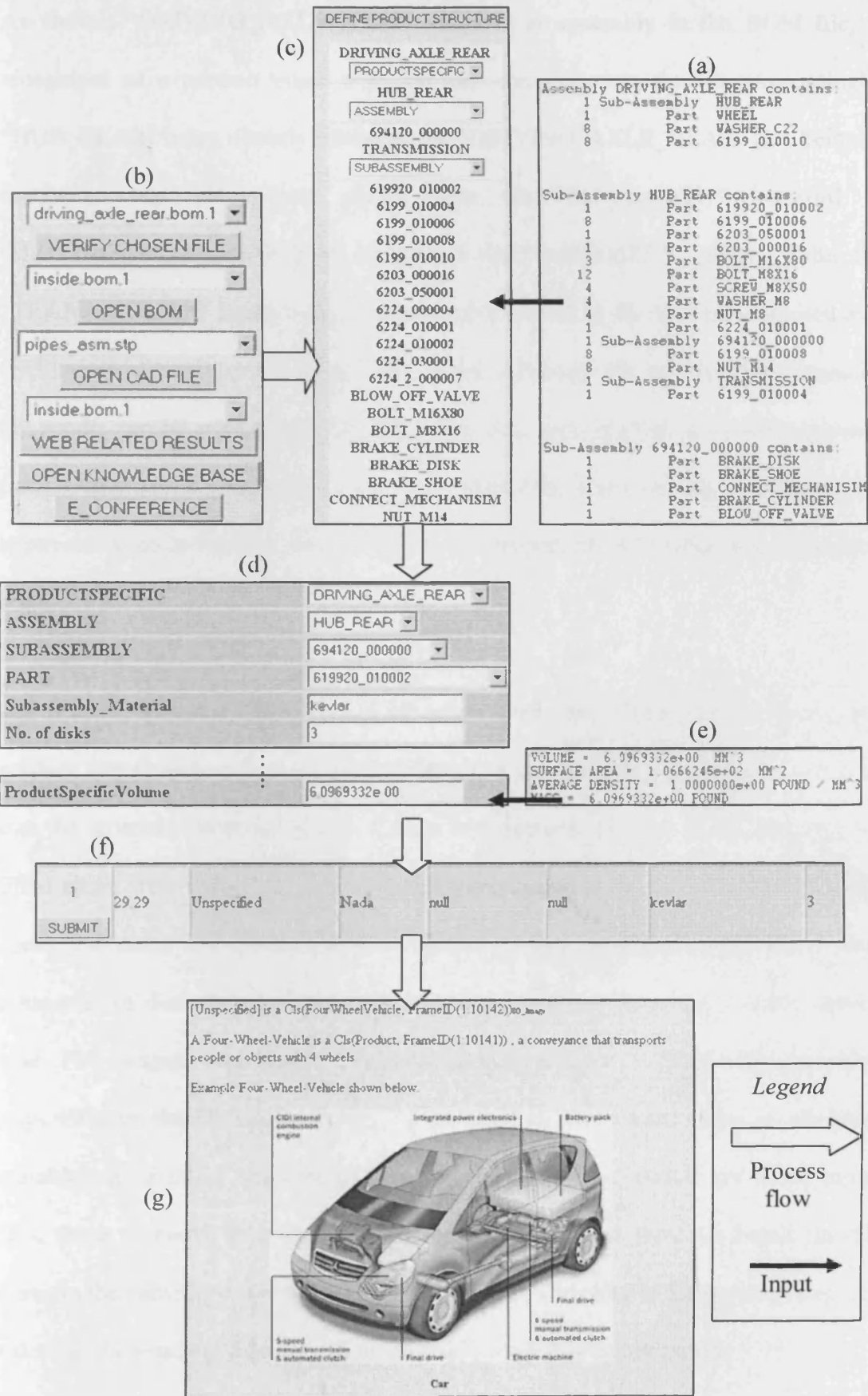


Figure 7.15. Case creation in PROGNOSIS using CAD related files

As shown, “DRIVING_AXLE_REAR” which is an assembly in the BOM file, is recognised as a product since it is the root component in the file. Accordingly, “HUB_REAR” being directly related to the “DRIVING_AXLE_REAR” (i.e. being in the same text block and related with the “Sub-Assembly” keyword to “DRIVING_AXLE_REAR”) is located in the “ASSEMBLY” level, while the “TRANSMISSION” being a direct subassembly of “HUB_REAR” is recognised as a “SUBASSEMBLY” by the system, and so on. Although this step is fully automatic, the results can be manually modified by the user, meaning that the aforementioned components can be assigned at any level (however some validity constraints are applicable such as one that does not allow all components to be allocated at the same level).

Fig. 7.15 (d) includes a fragment of the constructed case. The features related to the product components are defined in the previous step (in this scenario it is assumed that the structure proposed by the system is accepted). The rest of the features are filled either with values that are retrieved from data files (e.g. files containing data about the mass and other attributes of the product produced using STEP data (examples of data files can be found in Appendix V)) or from the initially chosen case. For example the feature “ProductSpecificVolume” is filled with the value acquired from the DF shown in Fig. 7.15 (e). On the other hand, since no attributes matching the features “Subassembly_material” and “No. of disks” are found in the DFs, these “inherit” their values from the case on which they are based (in this scenario the values are correspondingly kevlar and 3). Similar to the previous step, the values of the resultant features can be manually modified by the expert.

As illustrated in Fig. 7.15 (f), the case stored in the case base is different than the one submitted by the user. That is because the submitted case is checked against the knowledge base and ontology that describes products. In this example, the values of the features “PRODUCTSPECIFIC” and “ASSEMBLY” are considered invalid and therefore are replaced by system keywords (e.g. “Unspecified”), which indicate that there is an error with these features. However, the case is retained in the case base although another expert has to change and validate it before being removed. The product support virtual document created at the end of the process is shown in Fig. 7.15 (g).

7.4 CONCLUSIONS

Two product support environments have been developed, PROSON and PROGNOSIS, to illustrate the applicability of the proposed approaches.

The semantic models presented in chapter 4, have been integrated into PROSON. The development of such an environment illustrates that documentation objects can be reused in a meaningful way for producing new documents by mapping them to ontology components and representing them using ontology-based language..

The framework introduced in chapter 5, is used as the skeleton of PROGNOSIS. Its inference mechanism corresponds to the reasoning approach introduced and its modules relate to the spaces of the framework. The scenarios presented illustrate that changes in the domain knowledge are instantly reflected in the solutions that are delivered to the user, by reusing existing documentation components or automatically creating them. This framework ensures that the frequent changes expected when

dealing with much more complex and highly customised products are instantly propagated into product support documentation.

PROGNOSIS has been further extended according to the ideas described in chapter 6. The context ontology has been used to acquire context attributes including user profile and usage purpose and to adapt automatically generated documents to specific requirements. Internal stimuli have been used to update documentation objects from design data with minimal intervention from an expert. This work also illustrates that STEP contains data that can be automatically transformed in a form suitable for developing product support virtual documents.

CHAPTER 8

CONTRIBUTIONS, CONCLUSIONS, LIMITATIONS AND FUTURE WORK

This chapter summarises the contributions made and conclusions reached and suggests possible directions for further research.

8.1 CONTRIBUTIONS

The main contribution of this research is the development of a context-aware knowledge engineering framework for product support. This work is a step towards the new paradigm of context-specific “just-in-time” knowledge delivery that would enable corporate knowledge to be captured and delivered to users and developers of online training systems when, where and in the form they need it. The specific contributions are summarised below.

1. Definition of product support knowledge.

Product support knowledge is defined as a synthesis of product, task, and user knowledge. Product support virtual documentation forms the link between the different product support knowledge elements and is the medium that enables provision of user-tailored product support related information to the user.

2. Knowledge model of Product Support Systems (PRSSs).

The knowledge model is developed as a formal representation of the structure, relations, and attributes of the product support domain. It enables the development of

PRSSs knowledge bases and is used with the objective of organising data in a way that ensures homogeneity and validity of the resulting information when used for product support.

3. Conceptual model of a product support virtual document.

Product support virtual documentation is defined as an aggregation of entities called Information Objects (IOs). A new documentation concept is introduced called Information Object Cluster (IOC). IOC aggregates a number of IOs and is linked to concepts of the domain knowledge. The runtime commitments/constraints related to a product support virtual document are analysed in terms of the semantic relations of its components (i.e. IOs and IOCs).

4. Problem solving approach to developing PRSSs.

The problem solving approach includes determining the kind of product support problems that the system is expected to handle, and selecting accordingly the reasoning technique that should be followed. This approach is based on a multi-modal reasoning strategy that facilitates the generation of responses to a variety of user queries since case-based reasoning is utilised in providing personalised information while model-based reasoning is used to compensate for the lack of documentation resources.

5. Knowledge engineering framework for PRSSs.

The structure of the developed framework is based on four spaces: the data, problem, hypothesis, and solution spaces. The data space includes the knowledge bases that contain product, task, user, and documentation related knowledge. The problem space incorporates knowledge about the product support problems that have occurred in

previous problem solving iterations in terms of problem-solution pairs. The problem-solution pairs are represented by cases. The hypothesis space includes the case-based adaptation and model-based generation algorithms and the solution space contains information about the product support virtual documents that have been derived throughout previous problem solving iterations.

6. Conceptual model of context-aware PRSSs.

The conceptual model of context-aware PRSSs that is introduced extends the knowledge engineering framework (contribution 5). It includes a component that manages context-related information and a layer that captures internal and external stimuli.

7. Ontology of context for PRSSs.

The developed ontology of context includes four basic models: activity, user, environment, and physical models. The activity model is used to identify the purpose of visiting the PRSS (i.e. learn or perform). The user model represents users' characteristics and attributes. The environment and physical models include environmental features and the specifications of the system's hardware and software. In this ontology, the user model is integrated with the other context models, creating a holistic view of different situations to which the system needs to react.

8. Approach for integrating product design and support documentation data.

The integration approach proposed in this research is based on the idea that both the external and internal stimuli influence information delivery. The approach utilises the concepts of ontology, case-based reasoning and their inter-relations to create new product support knowledge from design data.

8.2 CONCLUSIONS

1. Traditional product support applications have a limited ability of adapting to the user's needs and responding to unforeseen situations.
2. Knowledge-based techniques have been used recently in developing training and support related applications but most of these attempts have been application-oriented.
3. The complex problem of creating a PRSS can be addressed by solving a more manageable goal of developing a knowledge-based platform for product support.
4. The development of an ontology-based model for representing the knowledge base of a PRSS facilitates interoperability and seamless content exchange between product support and other knowledge intensive fields (i.e. product, task, user, and documentation modeling) by representing knowledge in a machine-processable way.
5. The specification of product support problems and the active response to them by reusing, adapting or generating new information is among the most essential aspects of responsive product support.
6. A multi-modal reasoning strategy that utilises case and model-based reasoning can be employed to automatically develop product support information for every product support situation.
7. A knowledge-engineering framework can be used to uniformly represent product support systems and their components.

8. Effective knowledge delivery depends not only on user's characteristics but also on user's goals, environmental and device related features. These form the context of a PRSS.
9. Knowledge about the product support domain and context can be integrated and incorporated in the design of informative, effective, and expressive representations of context-specific support information.
10. Changes in product design data can be automatically translated into new product support documentation data by utilising the semantic-based representation of product support cases and product documentation components.

8.3 LIMITATIONS

The method used to rank the cases is based on the approach adopted by the FreeCBR tool, which uses the Euclidean distance algorithm. However, that algorithm cannot support adequately fuzzy searches in natural language based queries.

The solutions provided to the user are automatically generated, which poses some risk of delivering an incorrect result. In this work no metrics are proposed for calculating that risk.

The operation of the framework includes the selection of a hypothesis and its delivery to the user. The selection process is based on the cases ranking provided by CBR and on manual selection in the current prototype. Hypothesis selection could be improved by using learning algorithms that can utilise previous iterations and user preferences.

The documents generated by the system are not evaluated in this prototype. The reason is that no metrics have been devised for calculating the quality of technical documents in quantifiable terms. Usability studies could be used to acquire the opinions of expert and novice users, however, professional resources (i.e. graphics, videos, interactive maps, etc.) should be used in such a case.

8.4 FUTURE WORK

Currently the input for the querying tool is based on controlled text interfaces. Natural language processing could be utilised to offer to the users a more natural way of expressing their needs.

In this work adaptive knowledge delivery has been demonstrated in terms of simple techniques that illustrated the applicability of the proposed approaches. More complex methods can be used to perform content adaptation.

The product support system used to demonstrate the research conducted in this project is developed on a PC system. However, software embedded on the supported device or on a mobile platform (e.g. PDA) is another promising area where the framework could be tested.

In this research, product design data is retrieved by STEP data once this has been transformed into ASCII files. One of the future steps will be to integrate the document model and ontology within the STEP standard and make it an integral part of product data.

The proposed enhancements are based on some technical aspects of the current project. However, the work presented in this thesis stimulates research in other scientific areas, as well.

A possible research direction is using the same documentation models and testing their applicability in situations that are not related to product support but to other areas such as e-government and e-education.

Adaptive knowledge delivery could be further improved by developing mental models of human cognition and perception, as well as increasing the levels of adaptation. This would enable adaptation to different learning and task performance styles.

The automatic generation of electronic documentation raises the need for automatic evaluation of the delivered information. A step towards that direction would be the development of quantifiable measures and methodologies that indicate the appropriateness of electronic documents in different situations.

APPENDIX A

Examples of instances of the PROSON-based KB

1. Clutch instance

([Nikos_Instance_12] of Clutch

(Assembly_ID 8925)
(Assembly_Material "carbon")
(Assembly_Model "4.5 Pro Series")
(Assembly_Moment+of+Inertia 62.637)
(Assembly_Radius 4.06)
(Assembly_Type "jaw/claw")
(Assembly_Weight 7.6)
(hasSubAssemblies [NikosUserschanged_Instance_30040]
(ID "4.5 Pro Series V1 8925")
(No_Disks 0))

2. IO instance

([NikosUserschanged2_Instance_10] of InformationObject

(Behaviour "Static")
(ComponentID "47425475574772")
(Expressiveness "Detailed")
(Form "Text")
(Theme Clutch)
(TypeIO "Explanation")
(TypeIO_S [No_Disks]))

3. Reference instance

([NikosUserschanged2_Instance_10000] of Reference

(ComponentID "1432458724573474256")
(Expressiveness "Technical")
(hasIO [NikosUserschanged2_Instance_10246])
(Theme %3ASTANDARD-CLASS)
(TypeOrder "Data"))

4. Task-based product support virtual document instance

([NikosUserschanged2_Instance_10001] of TaskBasedPSVD

(DocumentID "6533264586548635628665365")
(Expressiveness "Detailed")
(has_IOC [NikosUserschanged2_Instance_10345]))

(ProductReference [NikosUserschanged_Instance_30039])
(Theme Design)
(ThemeConfigurationCompleteness TRUE)
(TypeOrder
 "Title"
 "Introduction"
 "Body"
 "TaskConcept"
 "Procedure"
 "Prerequisites"
 "Subtasks"
 "Reference"
 "Results"
 "Explanation"
 "Comment"
 "Example"))

5. Product-based product support virtual document instance

([NikosUserschanged2_Instance_10002] of ProductBasedPSVD

(DocumentID "45734768244864867476")
(Expressiveness
 "Detailed"
 "Partial"
 "Technical")
(has_IOC [NikosUserschanged2_Instance_10308])
(TaskReference Describe)
(ThemeConfigurationCompleteness TRUE)
(TypeOrder
 "Title"
 "Introduction"
 "Body"
 "ProductConcept"
 "Definition"
 "Description"))

6. IOC instance

([NikosUserschanged2_Instance_10328] of ProductConcept

(ComponentID "56743436584864234")
(Expressiveness
 "Detailed"
 "Partial")
(hasIO
 [NikosUserschanged2_Instance_20008]
 [NikosUserschanged2_Instance_20009]
 [NikosUserschanged2_Instance_20006]
 [NikosUserschanged2_Instance_20007])

```
[NikosUserschanged2_Instance_10215])
(Theme Clutch)
(TypeOrder
  "Definition"
  "Description"
  "Explanation"
  "Comment"
  "Example")
(TypeOrderCompleteness TRUE))
```

7. Pressure plate instance

([NikosUserschanged_Instance_40018] of PressurePlate

```
(hasParts
  [NikosUserschanged_Instance_40019]
  [NikosUserschanged_Instance_40020])
(ID "Pressure_Ultraweight 428")
(Subassembly_ID 428)
(Subassembly_Material "carbon")
(Subassembly_Model "Pressure_Ultraweight")
(Subassembly_Moment+of+Inertia 22.224)
(Subassembly_Radius 3.67)
(Subassembly_Type "diaphragm")
(Subassembly_Weight 3.3))
```

APPENDIX B

Examples of classes and slots of the PROSON-based KB

Examples of slots related to product support virtual documentation

1. Description of the theme slot

```

(single-slot Theme
  (type SYMBOL)
;+      (allowed-parents Describe Assess Assemble Inspect Schedule
        Promote Operate Install Launch Maintain Support Plan Service
        Three+Wheel+Vehicle Two+Wheel+Vehicle FourWheelVehicle Car
        Bus Lorry Truck Transmission Transaxle Body Clutch
        AutomotiveTransmission Countershaft PlateAssembly Flywheel
        Synchroniser Housing PressurePlate CoverPlate Screw Bearing Button
        Bolt Disk Pin Lever Cone Spring Capscrew Ring Washer Plate Nut
        Retaining_bolt Diaph_spring Pressure_spring Adjusting_screw
        Lock_Nut Pivot_Ring Retaining_capscrew Internal_cone Pilot_bearing
        Driving_plate Driven_plate Friction_plate Backplate Subtask Actions
        Attach Insert Screw1 Put Remove Turn Assembly Design)
;+      (cardinality 0 1)
        (create-accessor read-write))

```

2. Description of the has IOC connector

```

(multislot has_IOC
;+      (comment "Defines the IOCs included in the Product Support Virtual
Document")
        (type INSTANCE)
;+      (allowed-classes InformationObjectCluster)
        (cardinality 1 ?VARIABLE)
        (create-accessor read-write))

```

Examples of slots related to the user knowledge

3. Description of the Training time slot

```

(single-slot TrainingTime
;+      (type STRING)
        (cardinality 0 1)
        (create-accessor read-write))

```

4. Description of the username slot

```

(single-slot Username
  (type STRING)

```



```
;+          (cardinality 0 1)
           (create-accessor read-write))
```

Examples of slots related to the product knowledge

5. Description of the product weight slot

```
(single-slot Product_Weight
 (type FLOAT)
;+          (cardinality 0 1)
           (create-accessor read-write))
```

6. Description of the complexity magnitude slot in the Knowledge specifier

```
(single-slot complexityMagnitude
 (type INTEGER)
 (range 20 %3FVARIABLE)
;+          (cardinality 0 1)
           (create-accessor read-write))
```

7. Description of the has-parts connector

```
(multislot hasParts
 (type INSTANCE)
;+          (allowed-classes Part)
           (create-accessor read-write))
```

Examples of concepts

8. Description of the task concept

```
(defclass Task
 (is-a USER)
 (role abstract)
 (single-slot IsComposedOfSubtask
 (type INSTANCE)
;+          (allowed-classes Subtask)
;+          (cardinality 0 1)
           (create-accessor read-write)))
```

9. Description of the assembly concept

```
(defclass Assembly "a part of the whole product and a group of machine parts that fit
together to form a self-contained unit."
 (is-a USER)
 (role abstract)
 (multislot hasSubAssemblies
 (type INSTANCE)
;+          (allowed-classes CoverPlate PressurePlate Housing Flywheel
Countershaft Synchroniser)
```

```

        (create-accessor read-write))
(single-slot Assembly_Model
  (type STRING)
  (cardinality 0 1)
;+
  (create-accessor read-write))
(single-slot Assembly_Type
  (type STRING)
  (cardinality 0 1)
;+
  (create-accessor read-write))
(single-slot Assembly_ID
  (type INTEGER)
  (cardinality 0 1)
;+
  (create-accessor read-write))
(single-slot Assembly_Material
  (type STRING)
  (cardinality 0 1)
;+
  (create-accessor read-write))
(single-slot hasSubAssembly
  (type INSTANCE)
  (allowed-classes Subassembly)
;+
  (cardinality 0 1)
;+
  (create-accessor read-write)))

```

APPENDIX C

Example of the CLIPS code describing protégé interface

([BROWSER_SLOT_NAMES] of Property_List

```
(properties
  [NikosUserschanged2_ProjectKB_Instance_125]
  [NikosUserschanged2_ProjectKB_Instance_126]
  [NikosUserschanged2_ProjectKB_Instance_127]
  [NikosUserschanged2_ProjectKB_Instance_128]
  [NikosUserschanged2_ProjectKB_Instance_129]
  [NikosUserschanged2_ProjectKB_Instance_130]
  [NikosUserschanged2_ProjectKB_Instance_131]
  [NikosUserschanged2_ProjectKB_Instance_132]
  [NikosUserschanged2_ProjectKB_Instance_133]
  [NikosUserschanged2_ProjectKB_Instance_134]
  [NikosUserschanged2_ProjectKB_Instance_135])
```

([CLSES_TAB] of Widget

```
(is_hidden FALSE)
(label "Classes")
(property_list [Nikos_ProjectKB_Instance_10097])
(widget_class_name "edu.stanford.smi.protege.widget.ClsesTab"))
```

([FORMS_TAB] of Widget

```
(is_hidden FALSE)
(label "Forms")
(property_list [Nikos_ProjectKB_Instance_10133])
(widget_class_name "edu.stanford.smi.protege.widget.FormsTab"))
```

([INSTANCE_ANNOTATION_FORM_WIDGET] of Widget

```
(height 476)
(is_hidden FALSE)
(name ":INSTANCE-ANNOTATION")
(property_list [KB_083170_Instance_33])
(widget_class_name "edu.stanford.smi.protege.widget.FormWidget")
(width 603)
(x 0)
(y 0))
```

APPENDIX D

Examples of IOCs, IOs, and PSVD

Information Objects

- **Java Server pages format**

1. *Textual description of a disk clutch with 2 disks for experienced users*

<P align=left>The double-disk clutch has one extra driven disk and one more intermediate driving plate when compared to the single-disk.</P>

2. *Multimedia description of a disk clutch with 1 disk for experienced users*

</P>
<P align=center>Single-disk clutch</P>

3. *Textual description of a disk clutch with 1 disk for experienced users*

<P>The clutch mostly encountered is the single-disk one.</P>

4. *Textual description of a disk clutch with multiple disks for experienced users*

<P align=left> The <%=disks%>-disk clutch has <%=(disksNumber-1)%> extra driven disks and <%=(disksNumber-1)%> more intermediate driving plates when compared to the sigle-disk.</P>

5. *Textual definition of a disk clutch for experienced users*

B>A clutch is a coupling that connects or disconnect driving and driven parts of a driving mechanism.</P>

6. *Textual definition of a disk clutch for inexperienced users*

B>A clutch is a coupling that connects or disconnect driving and driven parts of a driving mechanism.</P>
<P>An automotive clutch connects and disconnects the engine and hand-shifted transmission or transaxle.

7. *Comment of a disk clutch for inexperienced users*

The clutch in cars, is located between the back of the engine and the front of the transmission.</P>
<P>

8. *Textual explanation of a disk clutch for inexperienced users*

<P>Clutches are useful in devices with two rotating shafts. The clutch locks the two shafts together, so that they can spin at the same speed or

decouples them, in order to spin at different speed.</P>

9. *Multimedia (Flash) explanation of a disk clutch for inexperienced users*

<OBJECT

```
codeBase=http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#
version=4,0,2,0
```

```
height=301 width=354 classid=clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000><PARAM NAME="movie"
VALUE="http://static.howstuffworks.com/flash/clutch-fig2.swf"><PARAM
NAME="quality" VALUE="high">
```

<embed

```
src="http://static.howstuffworks.com/flash/clutch-fig2.swf" quality=high
```

```
pluginspage="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod
_Version=ShockwaveFlash"
```

```
type="application/x-shockwave-flash" width="354" height="301">
```

</embed> </OBJECT>

10. *Multimedia (Flash) example of a disk clutch for inexperienced users*

<OBJECT

```
codeBase=http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#
version=4,0,2,0
```

```
height=400 width=450 classid=clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000><PARAM NAME="movie"
```

```
VALUE="http://static.howstuffworks.com/flash/clutch-fig3.swf"><PARAM
NAME="quality" VALUE="high">
```

```
<embed src="http://static.howstuffworks.com/flash/clutch-fig3.swf"
quality=high
```

```
pluginspage="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod
_Version=ShockwaveFlash"
```

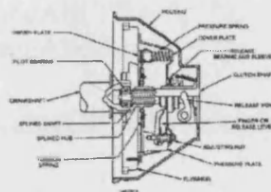
```
type="application/x-shockwave-flash" width="450" height="400"></embed>
```

</OBJECT>
Exploded view of car

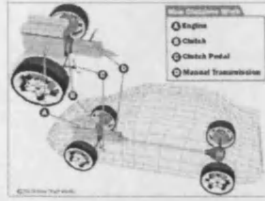
clutch

• **Other formats**

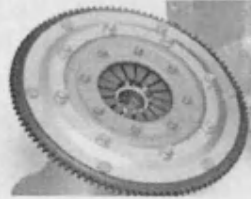
1. *Multimedia description of a disk clutch with 1 disk for experienced users*



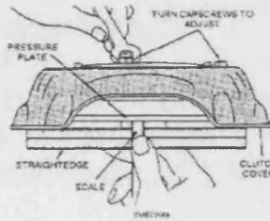
2. *Multimedia comment of a disk clutch for inexperienced users*



3. Multimedia example of a disk clutch for inexperienced users



4. Multimedia explanation of a pressure plate adjustment for inexperienced users



Information Object Cluster

- **Java Server pages format**

1. Disk clutch related IOC

```
<OBJECT
<%//Taking the number of disks as a parameter%>
<%
    String disks = request.getParameter("No_Disks");
    disks.toString();

    char leftBracket = '[';
    char rightBracket = ']';
    char empty = ' ';

    String disksNew = disks.replace(rightBracket, empty);
    disksNew = disksNew.replace(leftBracket, empty);
    disksNew = disksNew.replaceAll("^\\s+", "");
    disksNew = disksNew.replaceAll("\\s+$", "");

    int disksNumber = Integer.valueOf(disksNew).intValue();
%>
<P>The disks in this case are :
<%=disks
```

```

%>

<%//Using the Information Objects%>
<%if (disks.equals("[1]")){
%>

<%StringBuffer get = new StringBuffer();
if (USER.equals ("Inexperienced"))
get.add("Single-disk clutch textual definition.jsp");
get.add("Single-disk clutch textual.jsp");
get.add("Single-disk clutch describe multimedia.jsp");%>

<%}
if (disks.equals("[2]")){
%>

<%StringBuffer get2 = new StringBuffer();
if (USER.equals ("Inexperienced"))
get2.add("double-disk clutch textual definition.jsp");
get2.add("double-disk clutch describe textual.jsp");
get2.add("double-disk describe multimedia.jsp");%>

<%
    }
    if (disksNumber > 2){
%>

<%StringBuffer get2 = new StringBuffer();
if (USER.equals ("Inexperienced"))
get2.add("multiple-disk clutch textual definition.jsp");
get2.add("multiple-disk clutch describe textual.jsp");
get2.add("multiple-disk clutch describe multimedia.jsp");%>

```

Product support virtual document

- **Java Server pages format**

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE></TITLE>
<META http-equiv=Content-Type content="text/html; charset=iso-8859-7">
<META content="MSHTML 6.00.2800.1491" name=GENERATOR></HEAD>
<BODY bgColor=#faebd7>
<FORM action=http://MEC1503.engi.cf.ac.uk:8080/Nikos/servlet/LoginHandler
method=post>
<TABLE height="100%" width="100%" border=1>
  <TBODY>
    <TR width="100%">

```

<TD width="20%" bgColor=#e6dfcf>

<CENTER><HR><H2>Pro <IMG height=15
src="spin_files/spin.gif" width=20>
Gnosis</H2><HR>

</TD>

<TD width="60%"><!-- prosthetic edw -->

<P align=left>The system is in learning
mode</P>

<H1 align=center>DESCRIPTION OF A
CLUTCH</H1>

<P align=left>Clutches can be found in many things that are probably used
everyday, such as chainsaw, cordless drills and cars.</P>

<P><IMG height=32 src="InexperiencedDescribeClutch_files/definition.bmp"
width=32 name=Definition> A clutch is a coupling that connects or
disconnect driving and driven parts of a driving mechanism.</P>

<P>An automotive clutch connects and disconnects the engine and
hand-shifted transmission or transaxle. The clutch in cars, is located
between the back of the engine and the front of the transmission.</P>

<P>

<CENTER><IMG height=300
src="InexperiencedDescribeClutch_files/diagram of car showing clutch
location.jpg"
width=400></CENTER>

<P></P>

<DIV align=center>Location of clutch
</DIV>

<P>Clutches are useful in devices with two rotating shafts. The clutch
locks the two shafts together, so that they can spin at the same speed or
decouples them, in order to spin at different speed.</P>

<P align=center>

<OBJECT

codeBase=http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#
version=4,0,2,0

height=301 width=354 classid=clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000><PARAM NAME="movie"

VALUE="http://static.howstuffworks.com/flash/clutch-fig2.swf"><PARAM
NAME="quality" VALUE="high">

<embed

src="http://static.howstuffworks.com/flash/clutch-fig2.swf" quality=high

pluginspage="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod
_Version=ShockwaveFlash"

type="application/x-shockwave-flash" width="354" height="301">

</embed> </OBJECT>
Basic
clutch</P>


```
<P><IMG height=32 src="InexperiencedDescribeClutch_files/useful_info.bmp"
width=32> <I>In a car, a clutch is needed in order to connect the engine
shaft and the wheel shaft. This connection enables the car to move or stop
independently of whether the engine is stopped or not.</I></P>
<P><IMG height=32 src="InexperiencedDescribeClutch_files/subparts.bmp"
width=32> The main components of a clutch include the pressure plate, the
studs, the diaphragm spring (or coil-spring), the throw-out bearing, the
clutch housing, the release fork, and the bell housing.</P>
<P>&nbsp;
<CENTER>
<OBJECT
```

```
codeBase=http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#
version=4,0,2,0
height=400 width=450 classid=clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000><PARAM NAME="movie"
VALUE="http://static.howstuffworks.com/flash/clutch-fig3.swf"><PARAM
NAME="quality" VALUE="high">
<embed src="http://static.howstuffworks.com/flash/clutch-fig3.swf"
quality=high
```

```
pluginspage="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod
_Version=ShockwaveFlash"
type="application/x-shockwave-flash" width="450" height="400"></embed>
</OBJECT><BR><FONT size=-1><B>Exploded view of car
clutch</B></FONT>
</CENTER>
<P></P>
<P></P></TD>
<TD width="20%"
bgColor=#e6dfcf></TD></TR></TBODY></TABLE></FORM></BODY></HTML
>
```

APPENDIX E

The knowledge base

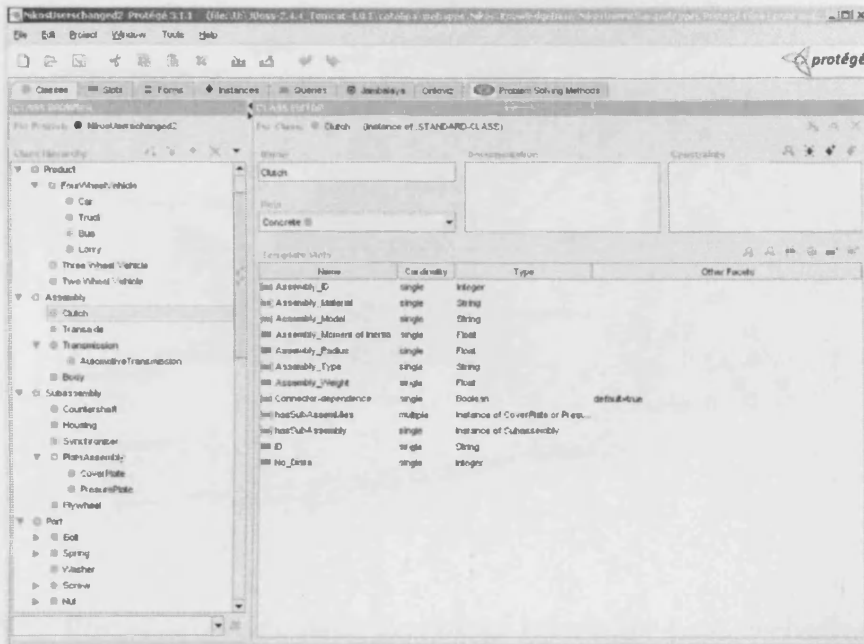


Figure E. 1. The product ontology with the clutch concept selected

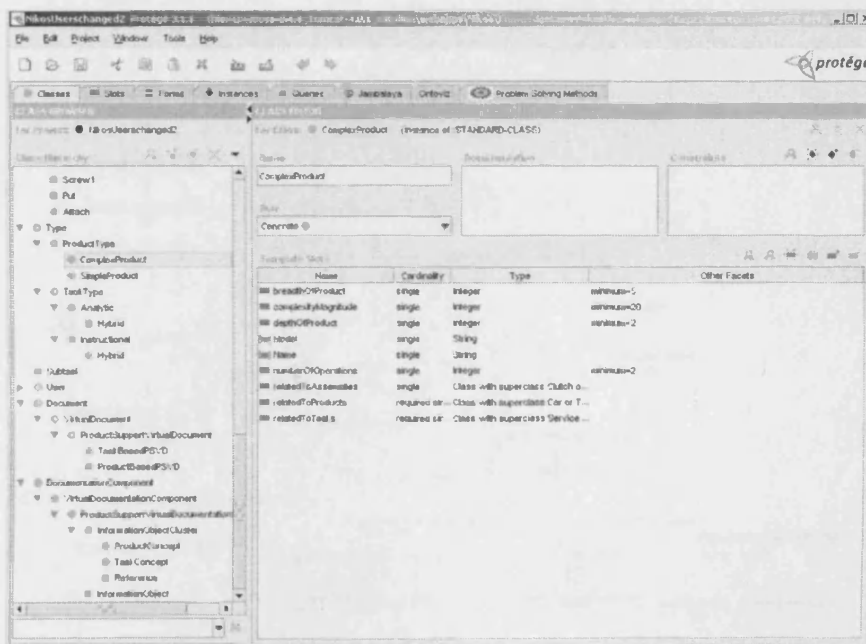


Figure E. 2. Part of the task ontology (actions), knowledge-specifiers (with ComplexProduct concept highlighted) and documentation ontology

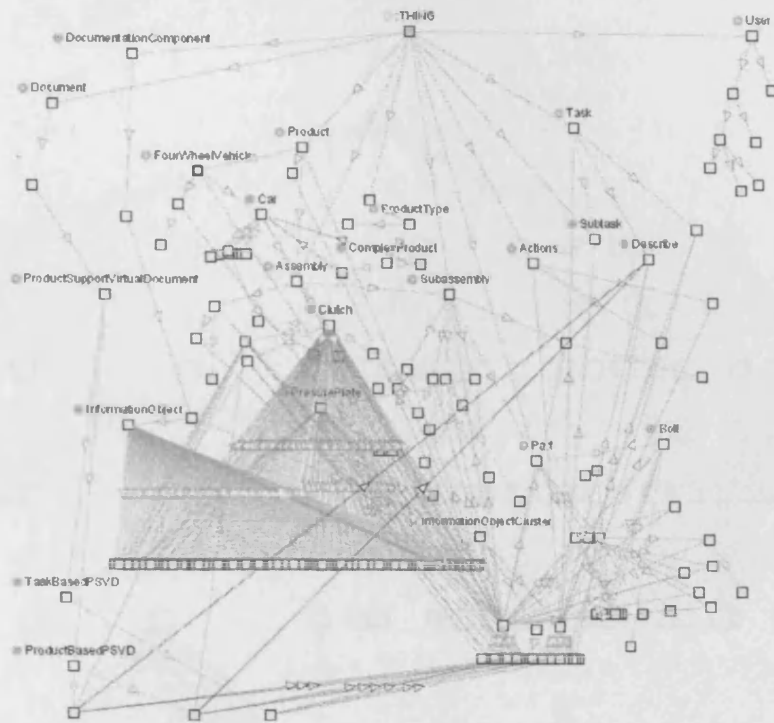


Figure E. 3. Visualisation of a small part of the knowledge base (including relations and instances)

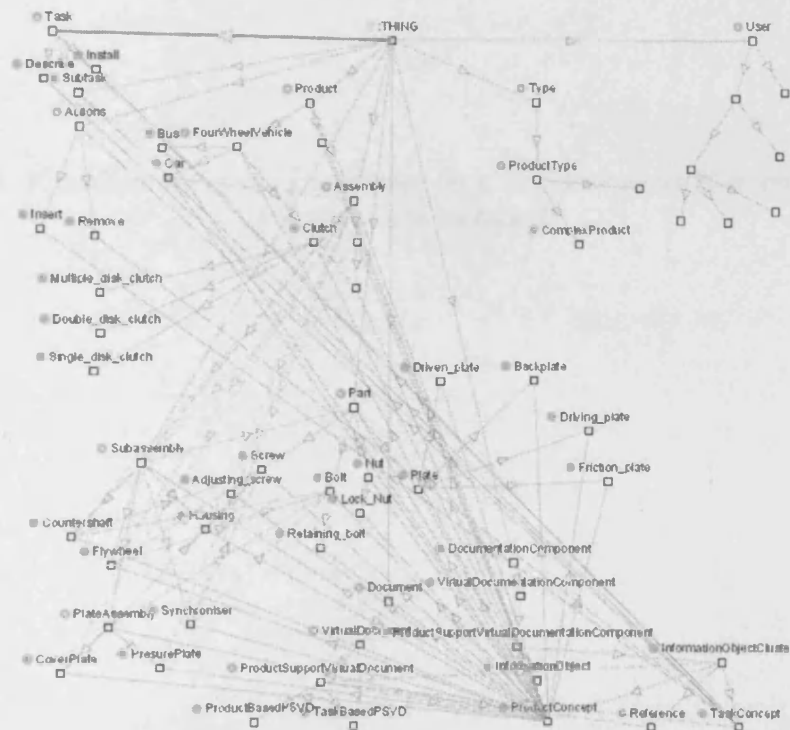


Figure E. 4. Visualisation of a part of the knowledge base subclasses (including relations)

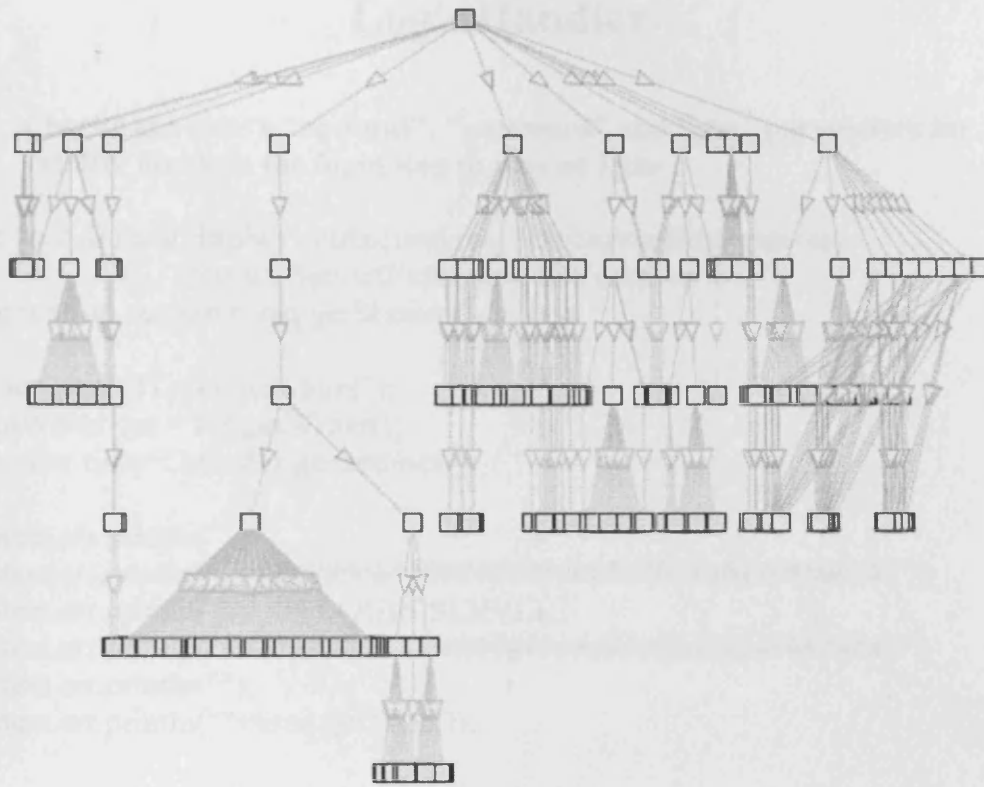


Figure E. 5. Visualisation of the knowledge base as a tree (only is-a and instance-of relations are included)

APPENDIX F

LoginHandler

1. Checks the user's "account", "password" and "pin" parameters for validity and sets the login flag to true or false

```

public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    HttpSession session = req.getSession();

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    Calendar time=Calendar.getInstance();

    System.err.println("");
    System.err.println("#####");
    System.err.println("## IN LOGIN SERVLET      ");
    System.err.println("#####");
    System.err.println("");
    System.err.println(""+time.getTime());

    // Get the user's account number, password, and pin
    String account = req.getParameter("account");
    String password = req.getParameter("password");
    String pin = req.getParameter("pin");

    // Check the name and password for validity
    if (!allowUser(account, password, pin, session)) {

res.sendRedirect("http://MEC1503.engi.cf.ac.uk:8080/Nikos/servlet/nick.LoginHandl
er");

        // if login fails invalidate session
        session.invalidate();
    }
    else {
        // Valid login. Make a note in the session object.
        session.putValue("logon.isDone", account);           // initialise username
        session.putValue("ApplicationStateFlag", "LOGIN" ); // initialise
applications state
        session.putValue("UMStateFlag", "IS_SET" ); // initialise user-model state
        System.err.println("value set .....");
        String Uaccount = (String) session.getValue("logon.isDone");
        System.err.println("in Login Uname:"+Uaccount);
        System.err.println("Login redirecting @:"+time.getTime());
    }
}

```

- 2. Retrieves the login, password, and pin parameters from the session, which were set according to the values found in the knowledge base.**

```
protected boolean allowUser(String LoginName, String password, String pin,  
HttpSession sess) {
```

```
    KnowledgeBaseWrapper kbw= new KnowledgeBaseWrapper();  
    result=kbw.process(LoginName,password,pin);  
    System.out.println("result:"+result);  
    if(result.equals("Experienced")||result.equals("Inexperienced")) {  
        sess.putValue("USER",result);  
        return true;  
    }  
    else  
        return false;
```

```
}
```

```
public void init() throws ServletException {
```

```
}
```

APPENDIX G

ApplicationController

1. **Code to take session parameters and specifically the value of the parameter “SUBMIT”, which holds the browsing trail of the user (from the browsing tool)**

```
public void doPost (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {

    HttpSession session = req.getSession();
    Calendar time=Calendar.getInstance();
    String parameter=req.getParameter("State");
    System.out.println("Request:"+req);
    System.out.println("State:"+parameter);
        // INITIALISE SERVLET
    res.setContentType("text/html");
    ServletOutputStream out = res.getOutputStream();
    Enumeration e =req.getParameterNames();
    while(e.hasMoreElements() ){
        System.out.println(e.nextElement());
    }
    boolean BooleanVolume = false;

    String showParameter = req.getParameter("SUBMIT");

    if(parameter.equals("Task")) {
        String parameter1=req.getParameter("ActivityChoise");
        System.out.println("ActivityChoise:"+parameter1);
        session.setAttribute("ACTIVITY",parameter1);

    res.sendRedirect("http://MEC1503.engi.cf.ac.uk:8080/Nikos/Task.jsp");
    }
```

2. **Component that takes the “CBR” parameters, checks if a precomposed solution exists and records the required time**

```
if(parameter.equals("CBR")) {
    String parameter13=req.getParameter("Present");
    System.out.println("Present:"+parameter14);

    KnowledgeBaseWrapper kbwclses = new
    KnowledgeBaseWrapper();

    if(showParameter.equals("SUBMIT")){
```

```

long startTime = System.currentTimeMillis();

if (checkIfPageExists(parameter13)==true){
res.sendRedirect("http://MEC1503.engi.cf.ac.uk:8080/F
reeCBR4/"+parameter14+".jsp?Disks="+parameter20+"
&ProductModel="+parameter29+"&ProductSpecificVo
lume="+parameter44);

long stopTime = System.currentTimeMillis();
long runTime = stopTime - startTime;
System.out.println("Run time: " + runTime);

}

```

3. **If a pre-composed solution doesn't exist start creating a document. The retrieval starting IOC is portrayed in the code below.**

```

else {

    try {

        char leftBracket = '[';
        char rightBracket = ']';
        char empty = ' ';

        char comma = ',';
        char bulletPoints = ':';

        StringBuffer call = new StringBuffer();

        ////////////////////////////////////////////////////////////////////Start IOC//////////////////////////////////////////////////////////////////
        StringBuffer callStart = new StringBuffer ("");

        KnowledgeBaseWrapper referenceURISStart = new KnowledgeBase Wrapper();

        LinkedList referenceURISStartComp = new LinkedList ();

        referenceURISStartComp.add((referenceURISStart.getReferenceIDStart("Refere
nce")).toString());

        String kbwReferenceURISStartComp= new String
(referenceURISStartComp.toString());

        kbwReferenceURISStartComp =
kbwReferenceURISStartComp.replace(rightBracket, empty);
        kbwReferenceURISStartComp =
kbwReferenceURISStartComp.replace(leftBracket, empty);
        kbwReferenceURISStartComp =
kbwReferenceURISStartComp.replaceAll("\s+", "");

```



```

kbwReferenceURIStartComp =
kbwReferenceURIStartComp.replaceAll("\\s+$", "");

//System.out.println("The reference is :"+kbwReferenceURIStartComp);

File URIaddressStart = new File(kbwReferenceURIStartComp);

String URIallStart=getContentsOfFile(URIaddressStart).toString();

```

```

.....

/*****Getting IDs of start reference and relevant URIs IOs*****/
public LinkedList getReferenceIDStartIO(String match) { // Add As
Argument The Previous Choose From The Specific Product Menu
//String match = new String("Car");

LinkedList list = new LinkedList(kb.getCls(match).getDirectInstances());

//System.out.println("");
System.out.println("ID of reference is :");

LinkedList output = new LinkedList();

for(int k=0; k<list.size(); k++) {
LinkedList slotList= new LinkedList(((Instance)list.get(k)).getOwnSlots());
    for(int j=0; j<slotList.size(); j++) {
        if(((Slot)slotList.get(j)).getName().equals("ComponentID")){
            if(((Instance)list.get(k)).getOwnSlotValue((Slot)slotList
                .get(j)).equals("1245252353243")){
                for (int n=0; n<slotList.size(); n++){
                    if(((Slot)slotList.get(n)).getName().equals("URI")){
                        output.add(((Instance)list.get(k)).getOwnSlotValue((Slot)slotList.get(n)));
                    }
                }
            }
        }
    }
}

return output;
}

```

4. Passing parameters to different classes and algorithms according to the tool used

```

else if (showParameter.equals("Add case")){
res.sendRedirect("http://MEC1503.engi.cf.ac.uk:8080/FreeCBR2/page3.jsp?ProductSpecificNew="+parameter53+"&AssemblyNew="+parameter54+"&SubassemblyNew="+parameter55+"&FirstIndication="+parameter56+"&SecondIndication="

```

```
+parameter57+"&RecognisedProblem="+parameter58+"&RelatedReason="+parameter59+"&ProposedSolution="+parameter60+"&IndexSearchTerms="+parameter61);  
}
```

APPENDIX H

KnowledgeBaseWrapper

1. Get concepts related by slot type (according to the concept selected by the user)

```
/*
 * Get Related By Slot Type of Extraction
 */

public LinkedList getActionMenu(String match) { // Add As Argument The
Previous Choise From The Activity Menu

    Cls cls1 = kb.getCls(match);

    LinkedList list = new LinkedList(cls1.getDirectTemplateSlots());

    int x = list.size();

    LinkedList listOfClasses = new LinkedList();

    if(x>0)
listOfClasses.addAll(((Slot)list.get(0)).getAllowedClses());

    System.out.println("");
    System.out.println("Action Menu :");

    LinkedList output = new LinkedList();

    for(int k=0; k<listOfClasses.size(); k++)
        output.add(((Cls)listOfClasses.get(k)).getName());

    return output;
}
```

2. Get slots related to specific instance

```
/*
 * Get Class Instances Related to Specific InstanceType of Extraction
 */
// Add As Argument The Previous Choice From The Product Model menu and
the choice from the assembly menu

public LinkedList getPartModelMenu(String PartType, String match) {
    //= new String("Bolt");
    //= new String("Ford Hewland 164501");
}
```

```

System.out.println(" ");
System.out.println("Part Model Menu:");
LinkedList list = new
LinkedList(kb.getMatchingReferences(match,100));
//System.out.println(list);

LinkedList slotList= new
LinkedList(((Instance)((Reference)list.get(0)).getFrame()).getOwnSlot
s());
//System.out.println(slotList);
LinkedList parts = new LinkedList();
for(int j=0; j<slotList.size(); j++) {
    if(((Slot)slotList.get(j)).getName().equals("hasParts")) {

//System.out.println(((Instance)((Reference)list.get(0)).getFrame()));
//System.out.println((Slot)slotList.get(j));
parts.addAll(((Instance)((Reference)list.get(0)).getFrame()).getOwnSlotValue
s((Slot)slotList.get(j)) );
    }
}

LinkedList selectedPartsOfType = new LinkedList();
for(int m=0; m<parts.size(); m++) {

if(((Instance)parts.get(m)).getDirectType().getName().equals(PartType))
    selectedPartsOfType.add(parts.get(m));
}

LinkedList output = new LinkedList();

for(int j=0; j<selectedPartsOfType.size(); j++) {
    LinkedList slotList2= new
LinkedList(((Instance)selectedPartsOfType.get(j)).getOwnSlots());
//System.out.println(slotList2);
for(int a=0; a<slotList2.size(); a++) {
    if(((Slot)slotList2.get(a)).getName().equals("ID"))

output.add(((Instance)selectedPartsOfType.get(j)).getOwnSlotValue((Slot)slot
List2.get(a)) );
    }
}
return output;
}

```

3. Get instances related to specific instance type

```

/*
 * Get Class Instances Related to Specific InstanceType of Extraction
 */
// Add As Argument The Previous Choice From The Product Model menu and
the choice from the assembly menu

```

```

public LinkedList getPartMenu(String match) {
    // String match = new String("Ford Hewland 164501");

    System.out.println(" ");
    System.out.println("Part Menu:");
    LinkedList list = new
LinkedList(kb.getMatchingReferences(match,100));
    //System.out.println(list);
    LinkedList slotList= new
LinkedList(((Instance)((Reference)list.get(0)).getFrame()).getOwnSlots());
    //System.out.println(slotList);
    LinkedList parts = new LinkedList();
    for(int j=0; j<slotList.size(); j++) {
        if(((Slot)slotList.get(j)).getName().equals("hasParts")) {

//System.out.println(((Instance)((Reference)list.get(0)).getFrame());
//System.out.println((Slot)slotList.get(j));

        parts.addAll(((Instance)((Reference)list.get(0)).getFrame()).getOwnSlotValue
s((Slot)slotList.get(j)) );
        }
    }

    LinkedList output = new LinkedList();

    for(int k=0; k<parts.size(); k++) {

output.add(((Instance)parts.get(k)).getDirectType().getName());
    }
    return output;
}

```

APPENDIX I

Integrated view of PROSON and case base

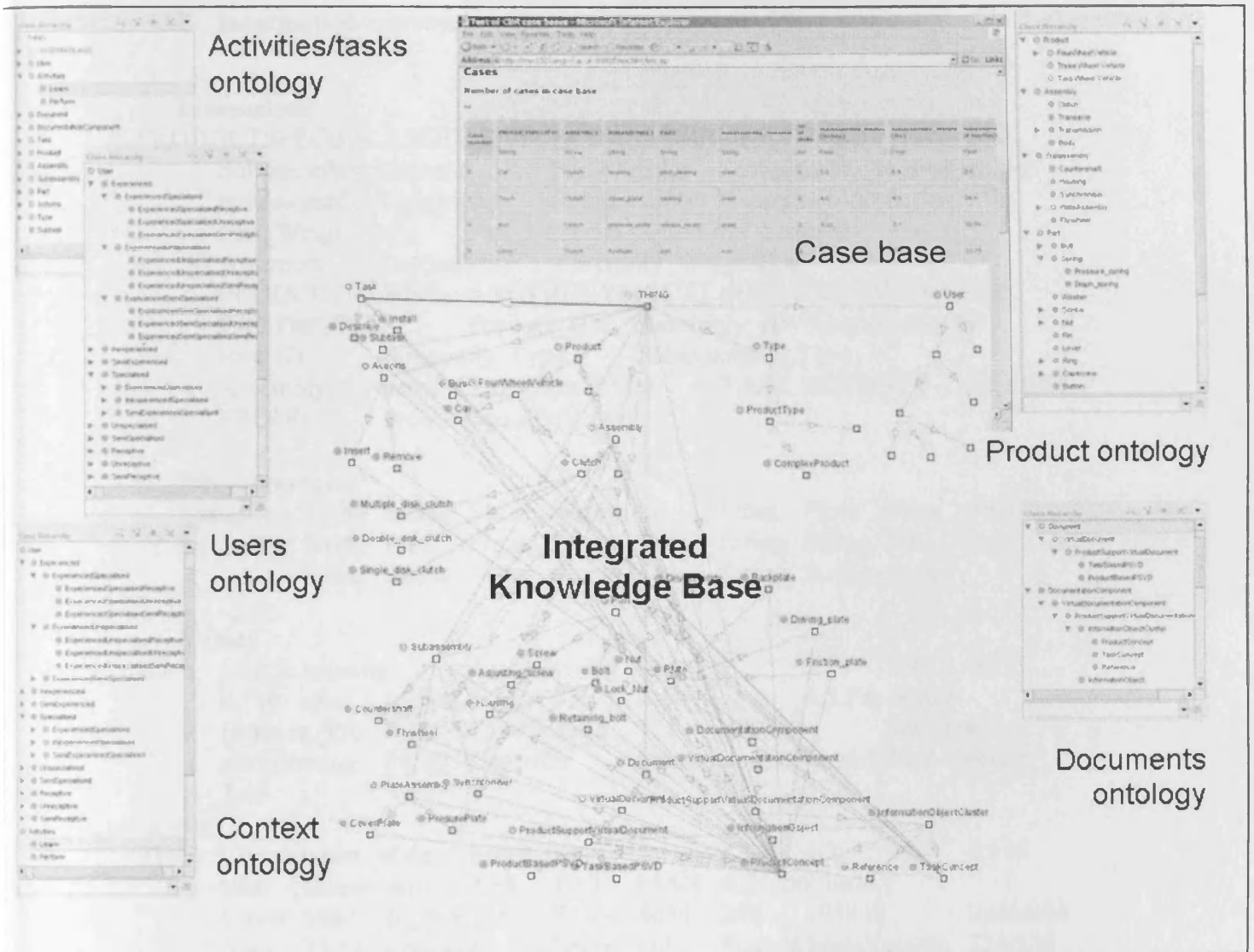


Figure I. 1. Integrated knowledge base of PROGNOSIS

APPENDIX J

Example cases

1. Information retrieval

Dimensions

```

PRODUCTSPECIFICASSEMBLY SUBASSEMBLY PART
Subassembly_Material No. of disks Subassembly_Radius(inches)
Subassembly_Weight(lbs) Subassembly_Moment of Inertia(in-lb*lb)
Part_Weight(lbs) Part_Material Assembly_Material
Assembly_Radius(inches) Assembly_Weight(lbs)
PRODUCTMODEL ASSEMBLYMODEL SUBASSEMBLYMODEL
PARTMODEL Product_ID Assembly_ID Subassembly_ID
Part_ID Assembly_Type Subassembly_Type
Assembly_Moment of Inertia(in-lb*lb) TASK ACTIVITY ACTION
PRODUCT ProductSpecificVolume
  
```

Data structures

```

String String String String String Int Float Float Float Float
String String Float Float String String String String Int Int Int
Int String String Float String String String String String
  
```

Values

```

Car Clutch housing pilot_bearing steel 1 5.2 5.4 0.0
0.756 steel carbon 4.16 6.8 Ford_Focus 4.5 Pro Series
Housing_Pro Pilot_S15463 8345 134 198342 jaw/claw
semifloating 58.84 Describe Learn null Four-Wheel-Vehicle
3.45

Truck Clutch cover_plate bearingsteel 2 4.0 4.3 34.4 0.645
steel aluminium 3.88 10.3 MAN 4.5 Optimum-V
Cover_Steel B_Steel 75634 8634 256 293842 hydraulic
front 77.53 Promote Learn null Four-Wheel-Vehicle 324.234

Bus Clutch pressure_plate release_levers steel 2 4.22 3.7 32.94
0.032 aluminium aluminium 3.8 11.9 Mercedes_S 4.5 V-
Drive Pressure_Steel RL_Lightweight 34521 8967 444 938472
hydraulic coil_spring 85.918 Support Learn null Four-
Wheel-Vehicle 322.2

Lorry Clutch flywheel bolt iron 3 4.35 2.3 21.76 0.0042
aluminium magnesium 3.87 10.5 MAN_New 5.5 Pro Series
Flywheel_Lightweight Bolt_Pro 73456 5342 563 847329
hydraulic Type_IV 78.628 Assess Perform null Four-
Wheel-Vehicle 3.46
  
```

2. Diagnosis

Dimensions

Product	Assembly	Subassembly	First_Indication	Second_Indication
Recognised_Problem	Related_Reason		Proposed_Solution	
Index_Search_Terms				

Data structures

String	String	String	String	String	String	String	String	String
--------	--------	--------	--------	--------	--------	--------	--------	--------

Values

car clutch	clutch_linkage	Engine speed rises rapidly on acceleration, while the vehicle gradually increases in speed	N/A	clutch_slippage	wear of clutch lining	replace clutch lining	PerformInstallnullFour-Wheel-Vehiclecarnullclutchnullclutch_linkagenullnullnull
------------	----------------	--	-----	-----------------	-----------------------	-----------------------	---

car clutch	clutch_linkage	Engine runs momentarily even with the clutch pedal fully released	N/A	clutch_slippage	improper clutch adjustment	adjust clutch linkage	PerformServicenullFour-Wheel-Vehiclecarnullclutchnullclutch_linkagenullnullnull
------------	----------------	---	-----	-----------------	----------------------------	-----------------------	---

car clutch	clutch_release_mechanism	Engine speed rises rapidly on acceleration, while the vehicle gradually increases in speed	Engine runs momentarily even with the clutch pedal fully released	clutch_slippage	release mechanism has rusted, bent, sticking or damaged components	replace release mechanism	PerformInstallnullFour-Wheel-Vehiclecarnullclutchnullbearing_mountnullnullnull
------------	--------------------------	--	---	-----------------	--	---------------------------	--

3. Expert advice

Dimensions

Purpose	Product	Assembly	Material	No. of Disks
Radius(inches)	Weight(lb)	Moment_of_Inertia (in-lb*lb)		

Data structures

String	String	String	String	Int	Float	Float	Float
--------	--------	--------	--------	-----	-------	-------	-------

Values

Maximise_Performance	car	clutch	carbon	2	4.16	6.8	58.84
Average (Recommended)	car	clutch	aluminium	2	3.84	10.8	79.62
Standard	car	clutch	aluminium	3	3.75	12.4	87.2
Maximise_Reliability	car	clutch	magnesium	3	3.76	12.7	89.77

APPENDIX K

Other scenarios

1. Case based adaptation

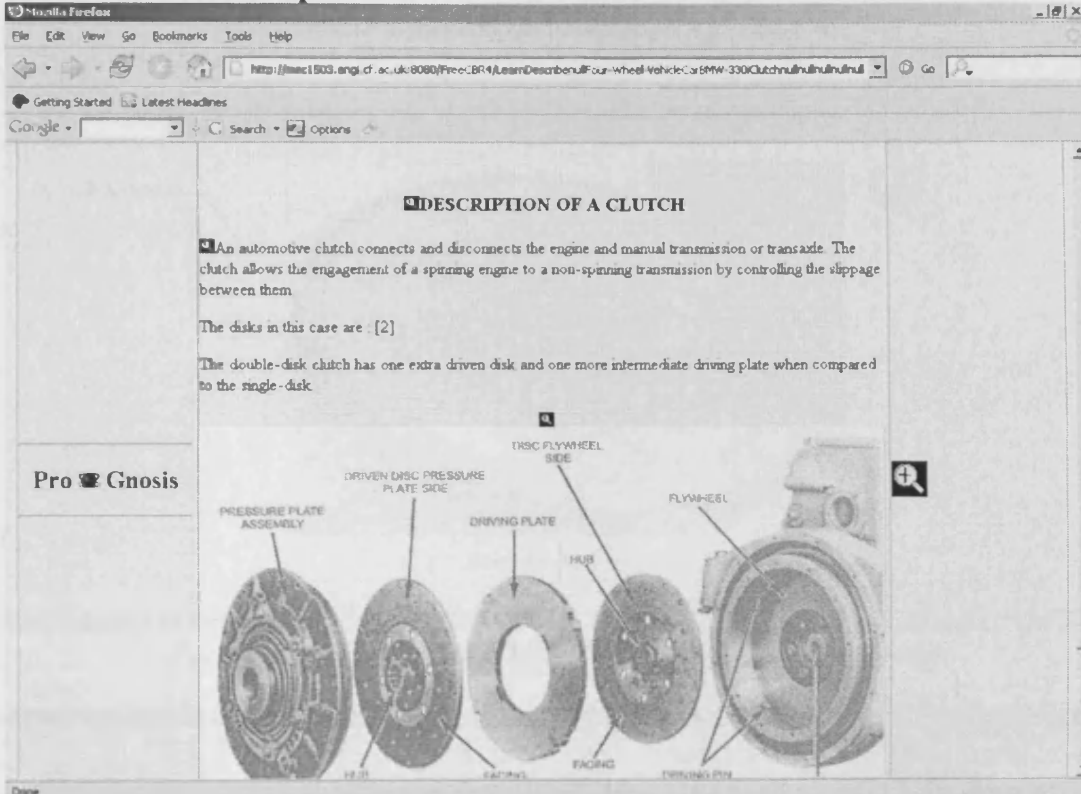


Figure K. 1. Double disk clutch

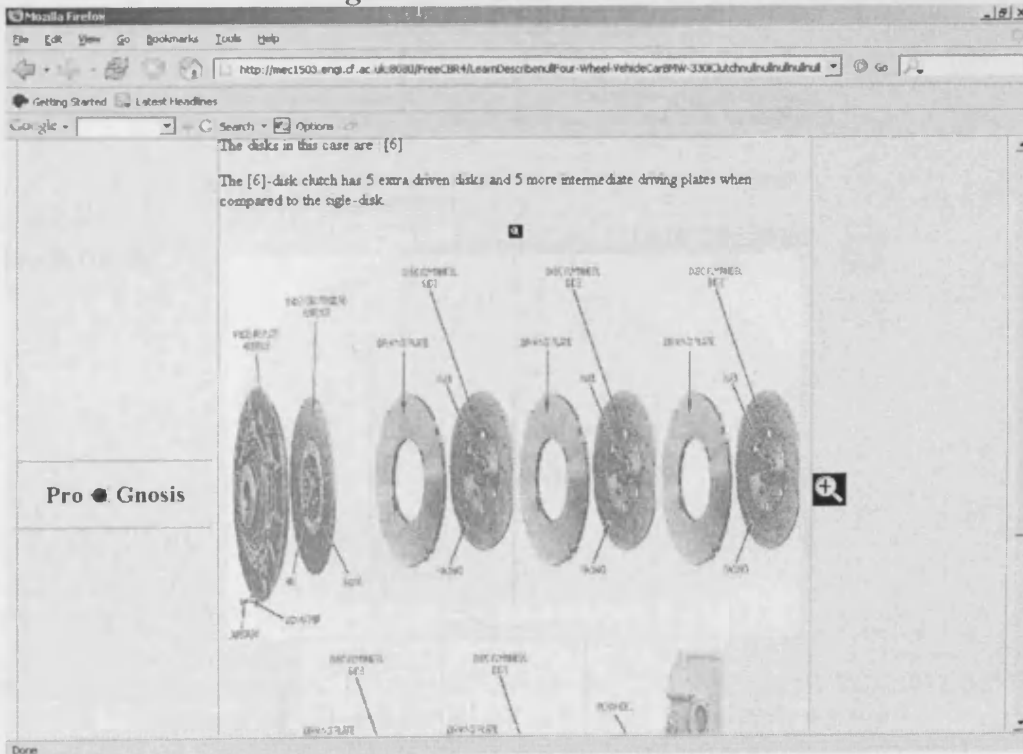


Figure K. 2. Six-disk clutch

Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://mec1503.engi.cf.ac.uk:8080/Nikos/web/LearnDescription/Four-wheel-VehicleCarBMW_Z4Clutch5.5-V-Drivepress...

Getting Started Latest Headlines

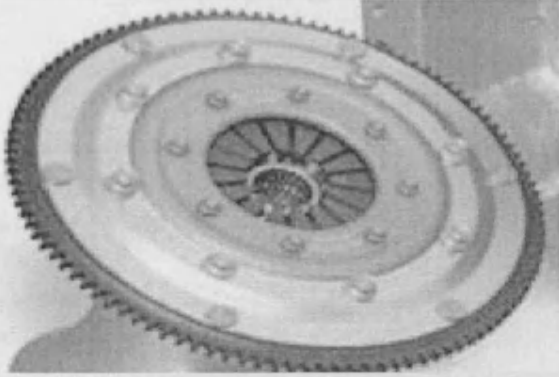
Google Search Options

DESCRIPTION OF A CLUTCH

In four wheel vehicles and specifically in cars an automotive clutch connects and disconnects the engine and manual transmission or transaxle. The clutch allows the engagement of a spinning engine to a non-spinning transmission by controlling the slippage between them. The clutch mostly encountered is the single-disk one.

For the requested car, which is the BMW Z4 the corresponding clutch is the Normal-Racing version of the 5.5 Pro, as shown below.

Pro Gnosis



5.5 Normal-Racing version

http://mec1503.engi.cf.ac.uk:8080/FreesCBR4/MoreInfo1.jsp

Figure K. 3. Clutch for BMW_Z4 (Normal-racing version)

Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://mec1503.engi.cf.ac.uk:8080/Nikos/web/LearnDescription/Four-wheel-VehicleCarBMW_AlpineClutch5.5-Lightweig...

Getting Started Latest Headlines

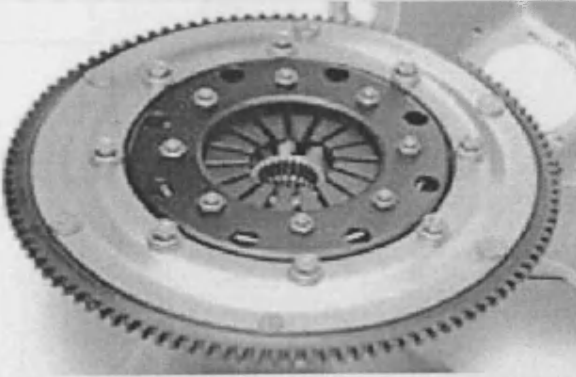
Google Search Options

DESCRIPTION OF A CLUTCH

In four wheel vehicles and specifically in cars an automotive clutch connects and disconnects the engine and manual transmission or transaxle. The clutch allows the engagement of a spinning engine to a non-spinning transmission by controlling the slippage between them. The clutch mostly encountered is the single-disk one.

For the requested car, which is the BMW Alpine the corresponding clutch is the Lightweight version of the 5.5 Pro, as shown below.

Pro Gnosis



5.5 Pro-Lightweight version

Done

Figure K. 4. Clutch for BMW_Alpine (Lightweight version)

2. Model-based generation

INSPECTING A TRANSAXLE

In four wheel vehicles and specifically in cars

STEP 1:
Inspect transaxle by inspecting each of Housing, Countershaft subassemblies.

STEP 2:
Inspect Housing by inspecting each of Screw, Pin, Spring, Washer, Bolt parts.

STEP 3:
Inspect Screw by checking for alteration in helical grooves.

INTERMEDIATE STEP:
Inspect Pin by checking each cylindrical tumbler.

INTERMEDIATE STEP:
Inspect Spring by checking for alteration on surfaces or rust.

INTERMEDIATE STEP:
Inspect Washer by checking for alteration in the surfaces.

STEP 7:
Inspect Bolt by checking for alteration in wrench or socket.

Pro Gnosis

<http://mec1503.eng.cf.ac.uk:8080/Nikos/web/PerformInspectnuffour-Wheel-VehicleCarBMW-330Transaxle.html#ins>

Figure K. 5. Inspecting a transaxle (based on behaviour model)

The system is in learning mode

Pro Gnosis

In four wheel vehicles and specifically in cars

[Nada] is a `Clc(Assembly, FrameID(1.10062))to_issap`

This assembly family is a part of the whole product and a group of machine parts that fit together to form a self-contained unit `to_issap`

Done

Figure K. 6. Concept that does not exist in the knowledge base but in a case (assembly that does not exist but the IOC for assemblies is precomposed)

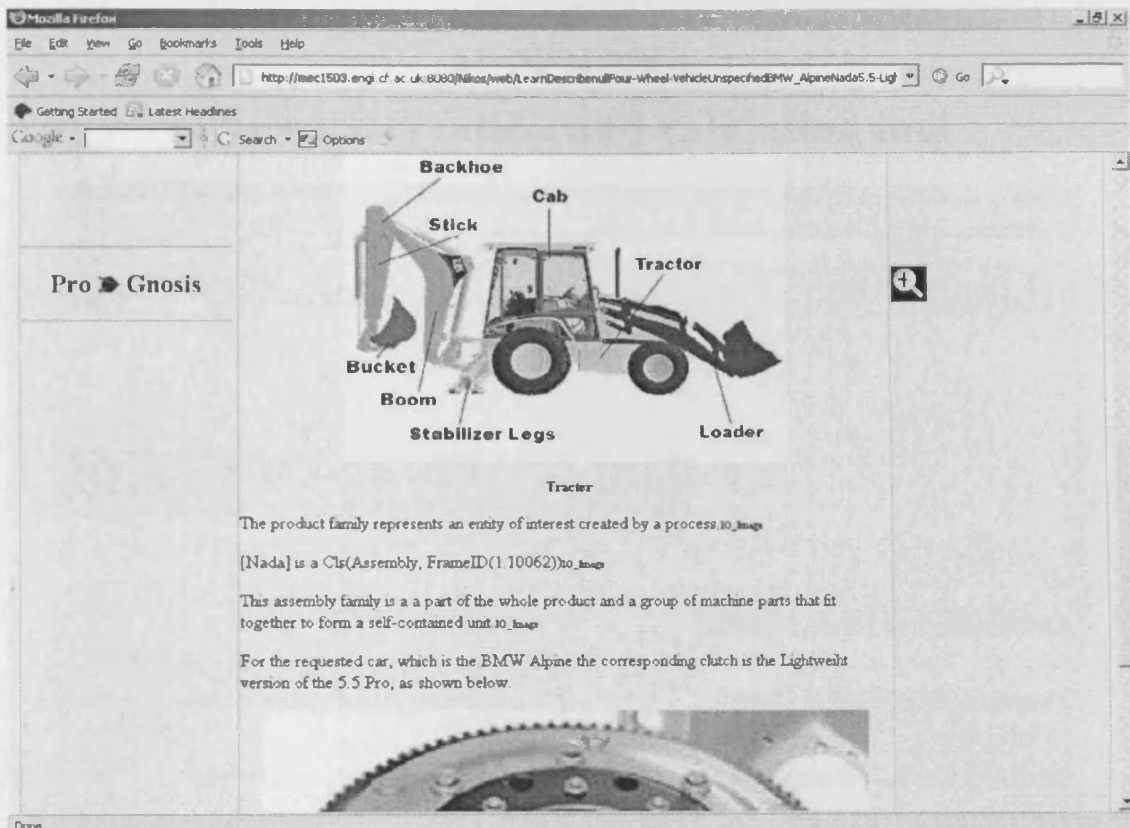


Figure K. 7. Concept that does not exist in the knowledge base but in a case (assembly that does not exist but the IOC for assemblies is NOT precomposed)

APPENDIX L

The case creation and validation tool

Assembly_Moment of Inertia(in-lb*lb)	TASK	ACTIVITY	ACTION	PRODUCT	ProductSpecific Volume			
95.904	Describe	Learn	null	Four-Wheel-Vehicle	3453.6	Add case	Adapt case	Create case using CAD data
59.375	Plan	Learn	null	Four-Wheel-Vehicle	9.64	Add case	Adapt case	Create case using CAD data
55.054	Assemble	Perform	remove	Four-Wheel-Vehicle	8.4	Add case	Adapt case	Create case using CAD data
62.0	Describe	Learn	null	Four-Wheel-Vehicle	?	Add case	Adapt case	Create case using CAD data

Figure L. 1. Selecting the case creation and validation tool by clicking on “Add case” or “Adapt case”. “Add case” permits the modification of static and dynamic features, while the “Adapt case” only dynamic features.

Adding new case in CBR search

PRODUCTSPECIFIC	{Car
ASSEMBLY	{Clutch
SUBASSEMBLY	{pressure_plate
PART	{lock_nut
Subassembly_Material	aluminium
No. of disks	3
Subassembly_Radius(inches)	4.14
Subassembly_Weight(lbs)	3.75
Subassembly_Moment of Inertia(in-lb*lb)	32.136
Part_Weight(lbs)	0.0012
Part_Material	aluminium
Assembly_Material	magnesium
Assembly_Radius(inches)	3.6
Assembly_Weight(lbs)	1.48
PRODUCTMODEL	{Seat_bike
ASSEMBLYMODEL	{5.5 Pro Series
SUBASSEMBLYMODEL	{Pressure_Lightweight
PARTMODEL	{LNut_Pro
Product_ID	{68354
Assembly_ID	{5342

Figure L. 2. The values of the new case are based on previous cases

APPENDIX M

User attributes' values and weights

Table M. 1. Characteristics, their attributes, and values (weights)

Characteristic	Attribute	Attribute weight	Value	Value Weight
Specialisation	<i>Training Type</i>	4	Classroom Instruction	2
			Hands-on Practice	3
			Both of the above	4
			None of the above	1
	<i>Education</i>	2	High School	1
			Technical School	2
			Post-secondary	3
			College	4
			Higher	5
			Postgraduate Taught	6
			Postgraduate Research	7
	<i>Job Title</i>	8	Mechanic	4
			Service Technician	3
			Manager	2
			User	1
	<i>Project Type</i>	3	Repair	2
			Assemble	3
			Research	4
			None of the above	1
	<i>Project Place</i>	4	Worker	2
Supervisor			3	
Managing supervisor			4	
None of the above			1	
Experience	<i>Years of Profession</i>	4	Graduate (<1)	1
			Trainee (<2)	2
			Professional (2-12)	8
			Experienced professional (>12)	12
	<i>Training Time</i>	2	Low (months<6)	2
			Average(6-24)	3
			High(>24)	5
			None of the above	1
	<i>Past Projects</i>	3	Low (<4)	2
			Average (4-24)	4
			High (>24)	8
			None of the above	1
	<i>Past Related Projects</i>	5	Low (<1)	2
Average (1-4)			6	
High(>4)			10	

			None of the above	1
Receptivity	<i>Number of visits</i>	2	None (<2)	6
			Few (2-10)	4
			Average (10-30)	2
			A lot (>30)	1
	<i>Average visit time</i>	2	Low (<10 minutes)	6
			Average (10-20)	4
			High (>20)	2
	<i>Time between visits</i>	2	Low (<60 minutes)	6
			Average	4
			High	2
	<i>Repetitiveness of query</i>	8	Low (<4)	6
			Average (4-8)	4
			High (>8)	2
	<i>Tool used</i>	2	Browsing	1
Searching			2	

APPENDIX N

Context ontology (CLIPS)

Activities and tasks

1. *Activities concept*

```
(defclass Activities
  (is-a Context_Entity)
  (role abstract)
  (single-slot IsRealisedWith
    (type INSTANCE)
    ;+ (allowed-classes Task)
    ;+ (cardinality 0 1)
    (create-accessor read-write)))
```

2. *Learn activity concept*

```
(defclass Learn
  (is-a Activities)
  (role concrete)
  (multislot hasLearningTasks
    (type INSTANCE)
    ;+ (allowed-classes Describe Plan Support Assess Promote Launch
Schedule Design)
    (create-accessor read-write)))
```

3. *Perform activity concept*

```
(defclass Perform
  (is-a Activities)
  (role concrete)
  (multislot hasPerformTasks
    (type INSTANCE)
    ;+ (allowed-classes Assemble Operate Maintain Inspect Service Install
Design)
    (create-accessor read-write)))
```

4. *Task concept*

```
(defclass Task
  (is-a USER)
  (role abstract)
  (single-slot IsComposedOfSubtask
    (type INSTANCE)
    ;+ (allowed-classes Subtask)
    ;+ (cardinality 0 1)
    (create-accessor read-write))
  (single-slot hasSubtask
    (type INSTANCE)
    ;+ (allowed-classes Subtask))
```



```
;+          (cardinality 0 1)
           (create-accessor read-write)))
```

5. *Design task concept*

```
(defclass Design
  (is-a Task)
  (role concrete))
```

6. *Determine goal task concept*

```
(defclass DetermineGoal
  (is-a Subtask)
  (role concrete)
  (single-slot hasSubtask
    (type INSTANCE)
    (allowed-classes Subtask)
    (cardinality 0 1)
    (create-accessor read-write)))
```

7. *Install task concept*

```
(defclass Install
  (is-a Task)
  (role concrete)
  (single-slot hasActionsInstall2
    (type INSTANCE)
    (allowed-classes)
    (cardinality 0 1)
    (create-accessor read-write)))
```

Physical context

8. *Input Devices concept*

```
(defclass Input+Device "Devices that feed data into the computer that have not been
included to the system in other classes."
```

```
  (is-a Peripheral)
  (role concrete)
  (single-slot type
    (comment "Whether the device is a hub, a router, a switch etc.")
    (type STRING)
    (cardinality 0 1)
    (create-accessor read-write))
  (single-slot interface_type
    (comment "The way the component is connected to the computer.")
    (type STRING)
    (cardinality 0 1)
    (create-accessor read-write)))
```

9. *Scanner concept*

```
(defclass Scanner "A device that can read text or illustrations printed on paper and
translate the information into a form the computer can use."
  (is-a Peripheral))
```

```

(role concrete)
(single-slot automatic_load
;+ (comment "Automatic load is supported.")
    (type SYMBOL)
    (allowed-values FALSE TRUE)
;+ (cardinality 0 1)
    (create-accessor read-write))
(single-slot max_resolution_horizontal
;+ (comment "The maximum resolution that the card can generate in the
vertical domain, measured in pixels.")
    (type INTEGER)
    (range 400 %3FVARIABLE)
;+ (cardinality 0 1)
    (create-accessor read-write))
(single-slot colour_resolution
;+ (comment "The supported colour resolution in bits.")
    (type INTEGER)
    (range 16 %3FVARIABLE)
;+ (cardinality 0 1)
    (create-accessor read-write))
(single-slot type
;+ (comment "Whether the device is a hub, a router, a switch etc.")
    (type STRING)
;+ (cardinality 0 1)
    (create-accessor read-write))

```

10. CoreComponent concept

```

(defclass CoreComponent "The components that are inside the cover of the computer
and absolutely necessary for data processing."
  (is-a Hardware)
  (role abstract)
  (single-slot model
;+ (comment "Product-line, model or combination of these two, as
provided by the manufacturer, wherever available.")
    (type STRING)
;+ (cardinality 0 1)
    (create-accessor read-write))
  (single-slot brand
;+ (comment "The brand of the component like Quantum etc.")
    (type STRING)
;+ (cardinality 0 1)
    (create-accessor read-write)))

```

11. NetworkingHardware concept

```

(defclass NetworkingHardware "The component that is used to connect the computer
with other computers."
  (is-a CoreComponent)
  (role abstract))

```

12. Network Interface Card concept

```

(defclass NIC "NIC (Network Interface Card) is a computer circuit (board or card)
that connects the PC to other computers."
  (is-a NetworkingHardware)
  (role concrete)
  (single-slot data_transfer_rate
;+      (comment "The rate in which the data is tranferred through the
component.")
        (type FLOAT)
        (range 0.0 %3FVARIABLE)
;+      (cardinality 0 1)
        (create-accessor read-write))
  (single-slot interface_type
;+      (comment "The way the component is connected to the computer.")
        (type STRING)
;+      (cardinality 0 1)
        (create-accessor read-write))
  (single-slot maximum_indoor_range
;+      (comment "The distance that the component can be used away from
the PC, measured in metres.")
        (type STRING)
;+      (cardinality 0 1)
        (create-accessor read-write))
  (single-slot OS_compatibility
;+      (comment "With which operating systems the card is compatible.")
        (type STRING)
;+      (cardinality 1 1)
        (create-accessor read-write))
  (single-slot networking_connection_type
;+      (comment "The protocol(s) that is (are) used for network
connections.")
        (type STRING)
;+      (cardinality 1 1)
        (create-accessor read-write)))

```

APPENDIX O

Learning and performing activities in traditional systems and in context-aware product support

Table O. 1. Differences between performance support, e-Learning, and context-aware product support

<i>Features</i>	<i>Performance support</i>	<i>E-learning</i>	<i>Context-aware product support (Learning activity)</i>
Goal	Enhance performance by providing a work context aligned to user's task and skills	Enable the acquisition of 'richer' knowledge states and/or improved skill set	Enhance task performance and improve skill set
Content	Material for current query	All material covered	All material related to current query
Content structure	Query-related conceptual model	Subject-related conceptual model	Product/Task related conceptual model
Content association	Question-based methodology	Lessons with associated tests	Question-based methodology
Content identification	User-based	Instructor-based	User-based
Flow of content	Random-parallel flow	Sequential flow	Random-parallel flow with sequential structure
Temporal association	Just-in-time	Scheduled	Just-in-time/Scheduled
Practical examples	Real case-studies with real issues	Artificial/Incomplete case-studies	Real case-studies with real issues
Assessment strategy	On-the-job performance (apply)	Tests (remember)	Tests and projects
Supporting mechanisms	Integrated with system	Outside resources	Integrated with system
Collaboration	Contact with peers and experts	Contact with peers and instructor	Contact with peers and experts
Knowledge level	User is able to apply the acquired knowledge to solve particular problem	Learner acquires knowledge and understands its fundamentals and the underlying reasoning process	User/Learner is able to successfully understand related complex problems and apply acquired knowledge.

APPENDIX P

Content adaptation scenarios

1. Learn activity

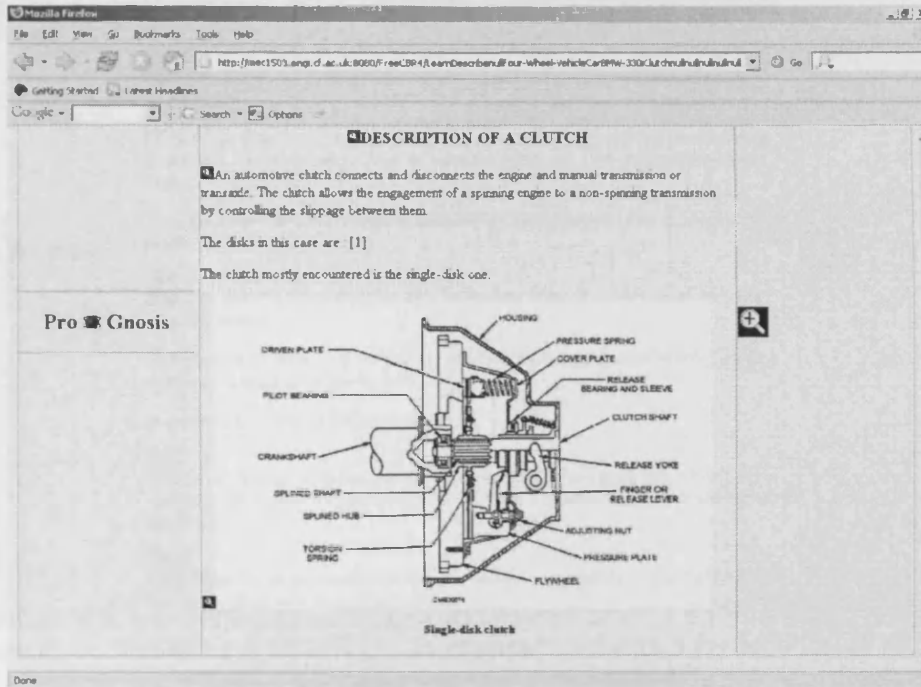


Figure P. 1. Describing a clutch for an experienced user (concise descriptions, still images, and facts compose the document in this scenario)

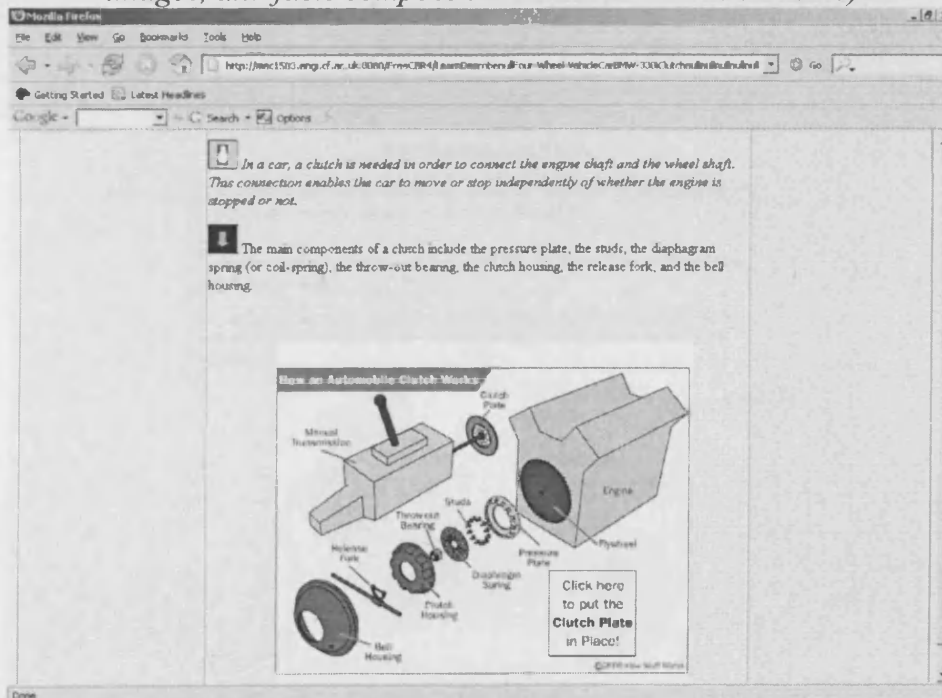


Figure P. 2. Describing a clutch for an inexperienced user (detailed descriptions, animations, and clarifications such as rules-of-thumb form the document)

2. Perform activity

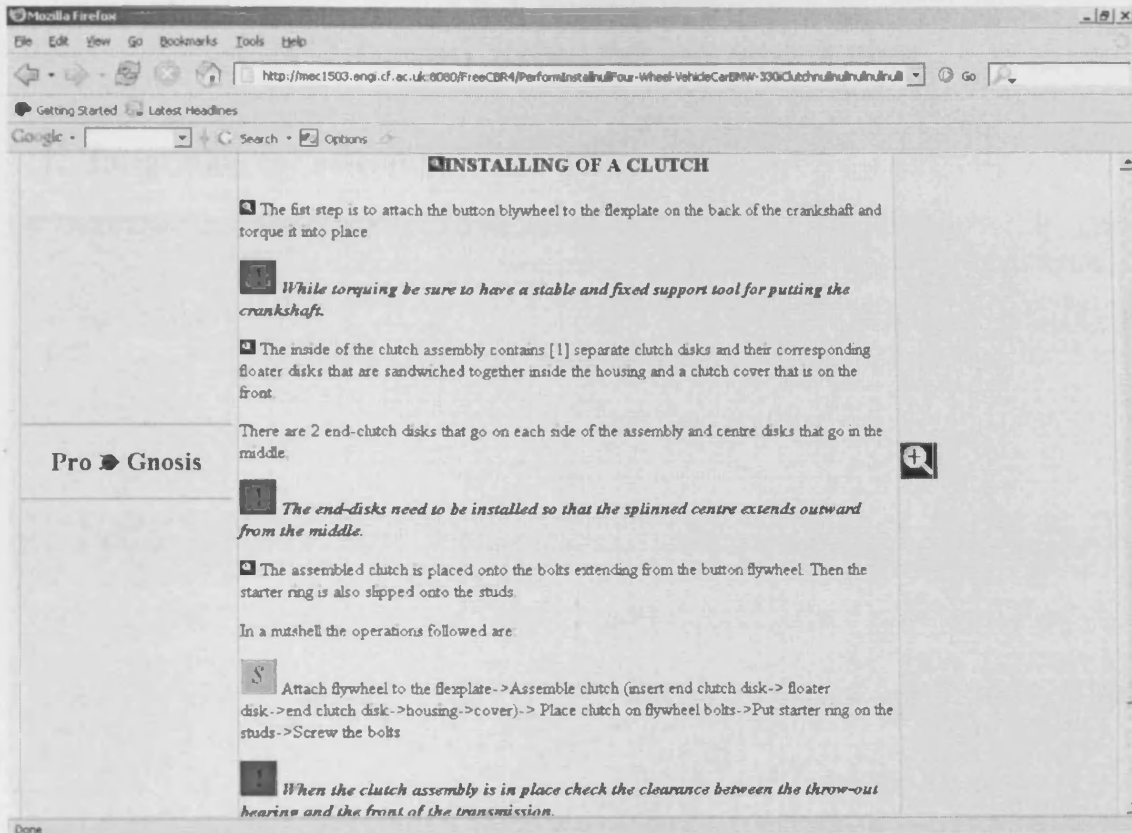


Figure P. 3. Installing a clutch for an experienced-receptive user (a lot of text that includes warnings and facts)

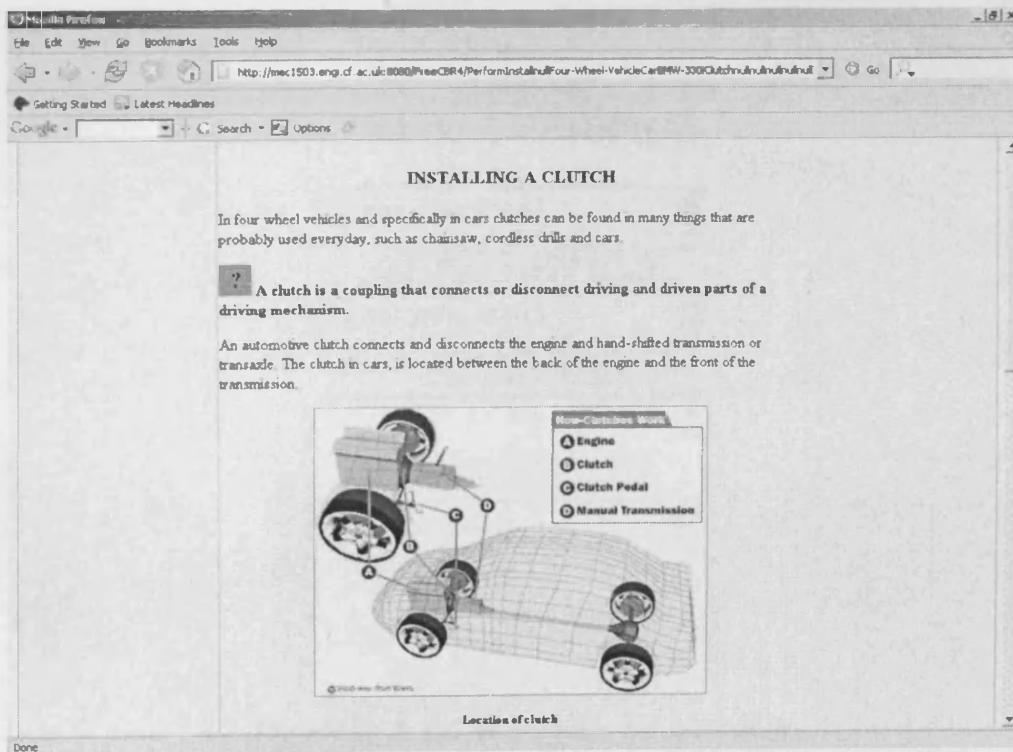


Figure P. 4. Installing a clutch for an inexperienced-unreceptive user (big clear pictures including clarification types)

APPENDIX Q

CAD based integration scenarios

1. Integrating car assembly

Result of CBR search - Mozilla Firefox
http://mec1503.engr.cf.ac.uk:8080/FreeCBR4/page2.jsp

oment)	Part_Weight(lbs)	Part_Material	Assembly_Material	Assembly_Radius(inches)	Assembly_Weight(lbs)	PRODUCTMODEL	ASSEMBLYMODEL
?	?	?	?	?	?	?	?
	8.4E-4	null	asbestos	3.6	9.8	BMW-330i	HardNew
	8.4E-4	null	asbestos	3.6	9.8	BMW-330i	HardNew
	8.4E-4	null	steel	3.6	9.8	BMW-330i	HardNew
	8.4E-4	steel	steel	3.6	9.8	BMW-330i	null

Figure Q. 1. Select the case

car_asm.bom.1
VERIFY CHOSEN FILE
car_asm.bom.1
OPEN BOM
car_asm_full.stp
OPEN CAD FILE
car_asm.bom.1
WEB RELATED RESULTS
OPEN KNOWLEDGE BASE
E_CONFERENCE

Figure Q. 2. Select the file and operation

DEFINE PRODUCT STRUCTURE

CAR_ASM

PRODUCTSPECIFIC ▾

CAR_ASM_2

ASSEMBLY ▾

null

PART ▾

BODY1_3

CHASSIS1_2

WHEEL_9

SUBASSEMBLY ▾

Figure Q. 3. Validate the product structure

Result of sCBR search - Mozilla Firefox

http://mec1503.eng.cf.ac.uk:8080/FreeCBR4/page7.jsp?ProductSpecific=CAR_ASM&FirstAssembly=CAR_ASM_2%206

Getting Started Latest Headlines

Google Search Options

PRODUCTSPECIFIC	CAR_ASM ▾
ASSEMBLY	CAR_ASM_2 ▾
SUBASSEMBLY	BODY1_3 ▾
PART	null ▾
Subassembly_Material	steel
No. of disks	2
Subassembly_Radius(inches)	4.0
Subassembly_Weight(lbs)	3.5
Subassembly_Moment of Inertia(in-lb*lb)	23.6
Part_Weight(lbs)	8.4E-4
Part_Material	null
Assembly_Material	esbestos
Assembly_Radius(inches)	3.6
Assembly_Weight(lbs)	9.8
PRODUCTMODEL	BMW-330i
ASSEMBLYMODEL	HardNew
SUBASSEMBLYMODEL	null
PARTMODEL	null
Product_ID	44444
Assembly_ID	2222
Subassembly_ID	0
Part_ID	0
Assembly Type	Irish

Done

Figure Q. 4. Validate the values of the new case's features

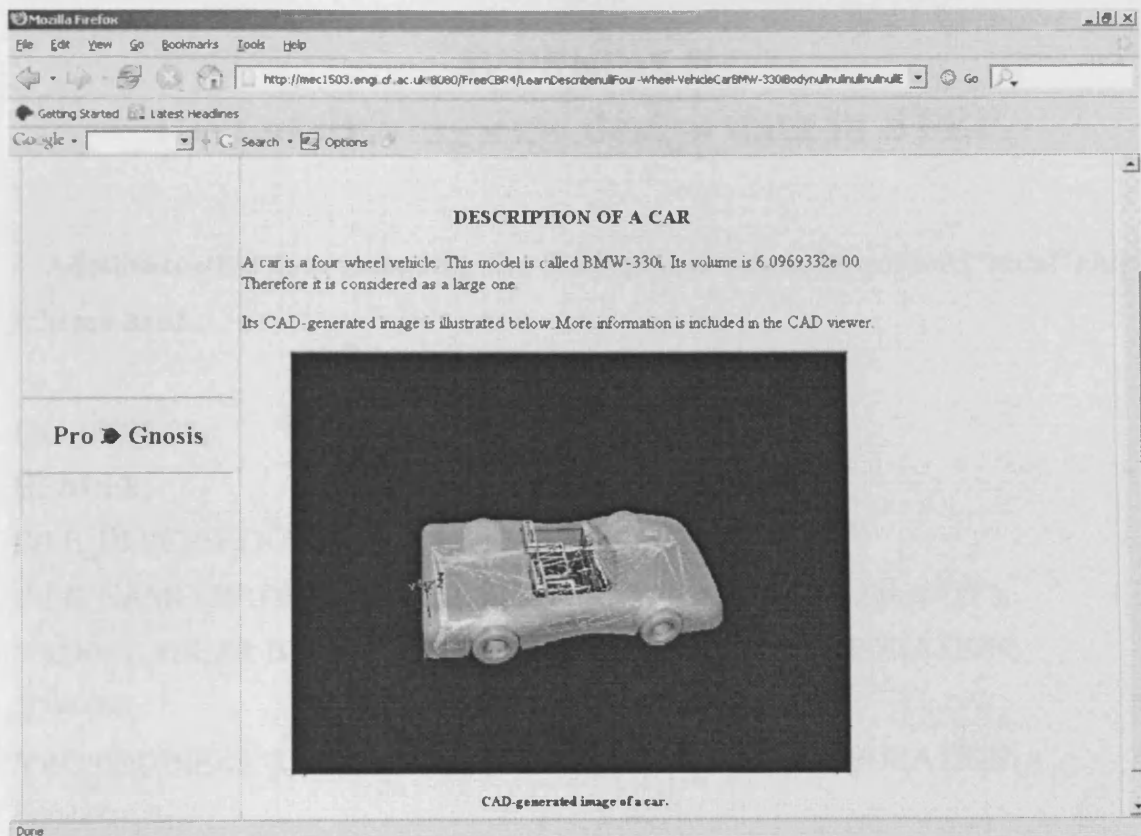


Figure Q. 5. Generated product support document based on CAD data

APPENDIX R

Part of driving axle design data in STEP

1. Administrative data including the file type and name, its author (“scenl”) and schema used.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(("','2;1');
FILE_NAME('DRIVING_AXLE_REAR_ASM','2006-03-16T','scenl'),('',
'PRO/ENGINEER BY PARAMETRIC TECHNOLOGY CORPORATION,
2004460',
'PRO/ENGINEER BY PARAMETRIC TECHNOLOGY CORPORATION,
2004460','');
FILE_SCHEMA(('CONFIG_CONTROL_DESIGN'));
ENDSEC;
DATA;
```

2. Geometrical data.

```
#2=CARTESIAN_POINT(",(0.E0,0.E0,0.E0));
#3=DIRECTION(",(0.E0,0.E0,-1.E0));
#4=DIRECTION(",(-1.E0,0.E0,0.E0));
#5=AXIS2_PLACEMENT_3D("#2,#3,#4);
#7=DIRECTION(",(2.258904820183E-1,0.E0,-9.741527037039E-1));
#8=VECTOR(",#7,7.083078426786E1);
#882=EDGE_LOOP(",(#876,#877,#879,#881));
#883=FACE_OUTER_BOUND(",#882,.F.);
#884=ADVANCED_FACE(",(#883),#874,.T.);
```

3. Defining the units of attributes such as length.

```
#1699=(LENGTH_UNIT()NAMED_UNIT(*)SI_UNIT(.MILLI.,.METRE.));
#1700=LENGTH_MEASURE_WITH_UNIT(LENGTH_MEASURE(2.54E1),#1699)
;
#1701=(CONVERSION_BASED_UNIT('INCH',#1700)LENGTH_UNIT()NAMED_
UNIT(#1698));
#1702=DIMENSIONAL_EXPONENTS(0.E0,0.E0,0.E0,0.E0,0.E0,0.E0,0.E0);
#1703=(NAMED_UNIT(*)PLANE_ANGLE_UNIT()SI_UNIT($,.RADIAN.));
#1704=PLANE_ANGLE_MEASURE_WITH_UNIT(PLANE_ANGLE_MEASURE(
1.745329251994E-2),
#1703);
#1705=(CONVERSION_BASED_UNIT('DEGREE',#1704)NAMED_UNIT(#1702)P
LANE_ANGLE_UNIT(
));
#1706=(NAMED_UNIT(*)SI_UNIT($,.STERADIAN.)SOLID_ANGLE_UNIT());
```

4. Defining that “619920_010002” is a subassembly of “HUB_REAR” and their relative placement.

```
#1726=NEXT_ASSEMBLY_USAGE_OCCURRENCE('0','Next assembly
relationship',
'619920_010002',#45159,#1720,$);
#1727=PRODUCT_DEFINITION_SHAPE('Placement #0',
'Placement of 619920_010002 with respect to HUB_REAR_ASM',#1726);
#1736=AXIS2_PLACEMENT_3D("#1733,#1734,#1735);
```

APPENDIX S

Three BOM tested files

The test data is chosen with the objective of having a representative sample for different scenarios. These include having: one intermediate level between the assembly (root) and parts nodes (scenario 1), no intermediate levels between assembly and parts nodes (scenario 2), all intermediate levels between assembly and part nodes (scenario 3 and the one presented in chapter 6).

1. “CAR_ASM” has no subassemblies.

Assembly CAR_ASM contains:

- 1 Sub-Assembly CAR_ASM_2

Sub-Assembly CAR_ASM_2 contains:

- 1 Part CHASSIS1_2
- 4 Part WHEEL_9
- 1 Part BODY1_3

Summary of parts for assembly CAR_ASM:

- 1 Part CHASSIS1_2
- 4 Part WHEEL_9
- 1 Part BODY1_3

2. “INSIDE” has only parts.

Assembly INSIDE contains:

- 1 Part 02060101_1
- 4 Part 020005_1
- 1 Part 020500_1
- 2 Part 020601_1
- 4 Part WASHER_M8
- 5 Part BOLT_M8X35
- 1 Part NUT_M8
- 4 Part BOLT_M10X25
- 4 Part WASHER_M10
- 4 Part NUT_M10
- 1 Part 020004_1

Summary of parts for assembly INSIDE:

- 1 Part 02060101_1
- 4 Part 020005_1
- 1 Part 020500_1
- 2 Part 020601_1
- 4 Part WASHER_M8
- 5 Part BOLT_M8X35
- 1 Part NUT_M8
- 4 Part BOLT_M10X25
- 4 Part WASHER_M10
- 4 Part NUT_M10
- 1 Part 020004_1

3. "BRAKING_SYSTEM" has assemblies, subassemblies, and parts.

Assembly BRAKING_SYSTEM contains:

- 1 Sub-Assembly INSIDE
- 1 Sub-Assembly OUTSIDE
- 4 Part WASHER_M10
- 4 Part BOLT_M10X25
- 1 Part AXIS_10X28
- 1 Sub-Assembly PIPES

Sub-Assembly INSIDE contains:

- 1 Part 02060101_1
- 4 Part 020005_1
- 1 Part 020500_1
- 2 Part 020601_1
- 4 Part WASHER_M8
- 5 Part BOLT_M8X35
- 1 Part NUT_M8
- 4 Part BOLT_M10X25
- 4 Part WASHER_M10
- 4 Part NUT_M10
- 1 Part 020004_1

Sub-Assembly OUTSIDE contains:

- 1 Part 020700_1
- 1 Part AMPLIFIER
- 1 Part DUST_BOOT
- 2 Part NUT_M10
- 1 Part A1130
- 3 Part WASHER_M10
- 3 Part BOLT_M10X25
- 1 Part ADAPTER
- 1 Part 020006_1

Sub-Assembly PIPES contains:

- 1 Sub-Assembly 020900
- 2 Part A1531
- 1 Sub-Assembly 021000
- 1 Part 020002
- 7 Part A1463
- 1 Part 020001
- 1 Part HOSE
- 1 Sub-Assembly 020400
- 1 Part 020003
- 1 Sub-Assembly 020100
- 1 Sub-Assembly 020100_1
- 1 Sub-Assembly 020800_1
- 1 Sub-Assembly 020800

Sub-Assembly 020900 contains:

- 1 Part 020901
- 1 Part 020102
- 1 Part A1463

Sub-Assembly 021000 contains:

- 1 Part 021001
- 1 Part 020102

Sub-Assembly 020400 contains:

- 1 Part 020401
- 1 Part 020102

Sub-Assembly 020100 contains:

- 1 Part 020101
- 2 Part 020102

Sub-Assembly 020100_1 contains:

- 1 Part 020101_1
- 2 Part 020102

Sub-Assembly 020800_1 contains:

- 1 Part 020801_1
- 2 Part 020102

Sub-Assembly 020800 contains:

- 1 Part 020801
- 1 Part 020802

Summary of parts for assembly BRAKING_SYSTEM:

- 1 Part 02060101_1
- 4 Part 020005_1
- 1 Part 020500_1
- 2 Part 020601_1
- 4 Part WASHER_M8
- 5 Part BOLT_M8X35
- 1 Part NUT_M8
- 11 Part BOLT_M10X25
- 11 Part WASHER_M10
- 6 Part NUT_M10
- 1 Part 020004_1
- 1 Part 020700_1
- 1 Part AMPLIFIER
- 1 Part DUST_BOOT
- 1 Part A1130
- 1 Part ADAPTER
- 1 Part 020006_1
- 1 Part AXIS_10X28
- 1 Part 020901
- 9 Part 020102
- 8 Part A1463
- 2 Part A1531
- 1 Part 021001
- 1 Part 020002
- 1 Part 020001
- 1 Part HOSE
- 1 Part 020401
- 1 Part 020003
- 1 Part 020101
- 1 Part 020101_1
- 1 Part 020801_1
- 1 Part 020801
- 1 Part 020802

APPENDIX T

Supporting facilities for the integration approach

The supporting facilities incorporated in PROGNOSIS include the following (see also Fig. T.1).

- “OPEN BOM” – opens a BOM file with an appropriate application
- “OPEN CAD FILE” – calls the visualisation tool linked with the management of STEP files
- “WEB RELATED RESULTS” – initiates a search on the Web and acquires information relating to the represented product
- “OPEN KNOWLEDGE BASE” – displays the knowledge base of PROGNOSIS related to product analysis with an appropriate application
- “E-CONFERENCE” – commences e-conference tools (such as NetMeeting) to enable collaboration between different experts

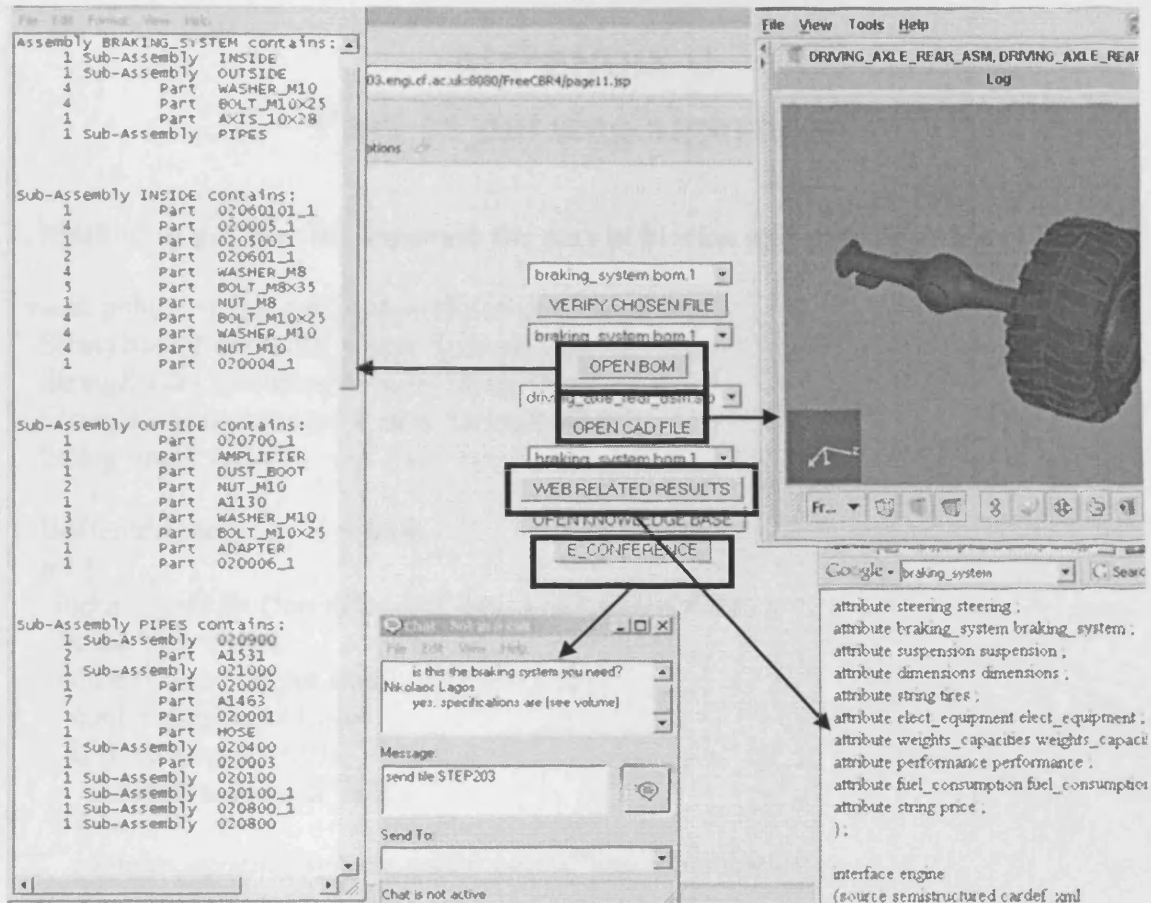


Figure T. 1. Supporting facilities for the integration process

APPENDIX U

Part of parsing algorithm

1. Method to get the file, separate the text in blocks, and read each block

```
static public String getContentsFileSub (File aFile) {
    StringBuffer contents = new StringBuffer();
    StringBuffer contents3 = new StringBuffer();
    StringBuffer contents4 = new StringBuffer();
    String line3=null;

    BufferedReader input = null;
    try {
        input = new BufferedReader( new FileReader(aFile) );
        String line = null;
        while (( line = input.readLine()) != null){
            contents.append(line);
            if (line.length()==0){
                contents.append("$");
            }
            contents.append(System.getProperty("line.separator"));

                }
            //System.out.println ("The contents are "+ contents);
            String[] st = (contents.toString()).split("\\s");

            String[] temp = (contents3.toString()).split("\\s");

            //String[] temp2 = new String [2000];

            int symbol = 0;
            for (int x=0; x<st.length; x++){

                if (st[x].equals("$")){
                    symbol++;
                }

                if ((symbol<4)&&(st[x].equals(InBlockText))){
                    contents4.append(st[x+2]);
                    contents4.append(" ");
                    //temp2[x]=st[x+2];
                }

                else if ((symbol>=4)&&(st[x].equals(OutofBlockText))){
                    if (st[x+2].equals(keyword)){
                        System.out.println ("Not correct");
                    }
                }
            }
        }
    }
}
```

```

        }
        else {
            contents3.append(st[x+2]);
            contents3.append(" ");
        }

    }
    else{
        line3 = contents.substring(121,131);
    }
}

} catch (FileNotFoundException ex) {
    ex.printStackTrace();
}
catch (IOException ex){
    ex.printStackTrace();
}
finally {
    try {
        if (input!= null) {
            //flush and close both "input" and its underlying FileReader
            input.close();
        }
    }
    catch (IOException ex) {
        ex.printStackTrace();
    }
}

return contents4.toString();
}

```

2. Class to compare stored values and classify them as existing or not, storing them in an index for fast retrieval.

```
package nick;

import java.io.*;
import java.util.*;

public class searcher
{
    private static long [] indexes;

    private static class temp_data
    {
        public final String text;
        public final long starts_at;

        public temp_data(String t, long l)
        {
            text = t;
            starts_at = l;
        }
    };

    private static class temp_cmp implements Comparator
    {
        public int compare(Object o1, Object o2)
        {
            return ((temp_data)o1).text.compareTo(
                ((temp_data)o2).text);
        }
    };

    /** creates index table. This method has high peak memory usage but it is
    easy to optimize it.*/
    public static void buildIndex(RandomAccessFile file) throws Exception
    {
        List temp = new LinkedList();
        String st;
        long p = file.getFilePointer();
        while((st = file.readLine())!=null)
        {
            temp.add(
                new temp_data(st,p)
            );
            p = file.getFilePointer();
        }
        Collections.sort(temp,new temp_cmp());
        indexes = new long[temp.size()];
        int i=0;
        for(Iterator I=temp.iterator();I.hasNext();i++)
```

```

    {
        temp_data tt = ((temp_data)I.next());
        System.out.println("indexing :"+tt.text+" ["+tt.starts_at+""]);
        indexes[i]=tt.starts_at;
    };
};
/** returns position at which text starts or -1 if not found */
public static long find(String text,RandomAccessFile file)throws Exception
{
    int ncp = indexes.length/2;
    int n = 2;
    int cp;
    do{
        cp = ncp;
        file.seek(indexes[cp]);
        String tt = file.readLine();
        System.out.println("comparing with "+tt);
        int cmpr = text.compareTo(tt);
        if (cmpr==0)
            return indexes[cp];
        else
            if (cmpr>0)
                ncp = cp+(indexes.length / (1<<n));
            else
                ncp = cp-(indexes.length / (1>>n));
        n++;
    }while(ncp!=cp);
    return -1;
};

public static void Primary (String args [] )throws Exception
{
    RandomAccessFile f = new RandomAccessFile(args[0],"r");
    buildIndex(f);
    for(int i=1;i<args.length;i++)
    {
        System.out.println("searching for \""+args[i]+"\"");
        System.out.println("found at:"+find(args[i],f));
    }
    f.close();
};
};
};

```

APPENDIX V

Data files examples

1. Braking system

VOLUME = 2.6639375e+06 INCH³
SURFACE AREA = 8.5076725e+05 INCH²
AVERAGE DENSITY = 8.4867902e+00 POUND / INCH³
MASS = 2.2608278e+07 POUND

CENTER OF GRAVITY with respect to _BRAKING_SYSTEM coordinate frame:
X Y Z -1.1508418e+02 -4.6106796e+01 -2.5495884e+00 INCH

INERTIA with respect to _BRAKING_SYSTEM coordinate frame: (POUND * INCH²)

INERTIA TENSOR:

Ixx Ixy Ixz 4.6586006e+11 2.8334606e+11 2.5657460e+10
Iyx Iyy Iyz 2.8334606e+11 2.1799930e+12 -8.8027032e+08
Izx Izy Izz 2.5657460e+10 -8.8027032e+08 2.3536163e+12

INERTIA at CENTER OF GRAVITY with respect to _BRAKING_SYSTEM coordinate frame: (POUND * INCH²)

INERTIA TENSOR:

Ixx Ixy Ixz 4.1765159e+11 4.0330927e+11 3.2291120e+10
Iyx Iyy Iyz 4.0330927e+11 1.8804137e+12 1.7774086e+09
Izx Izy Izz 3.2291120e+10 1.7774086e+09 2.0061224e+12

PRINCIPAL MOMENTS OF INERTIA: (POUND * INCH²)
I1 I2 I3 3.1326069e+11 1.9805855e+12 2.0103415e+12

ROTATION MATRIX from _BRAKING_SYSTEM orientation to PRINCIPAL AXES:

0.96829	0.22721	0.10389
-0.24917	0.90857	0.33529
-0.01821	-0.35054	0.93637

ROTATION ANGLES from _BRAKING_SYSTEM orientation to PRINCIPAL AXES (degrees):

angles about x y z -19.701 5.963 -13.206

RADII OF GYRATION with respect to PRINCIPAL AXES:

R1 R2 R3 1.1771158e+02 2.9598047e+02 2.9819556e+02 INCH

MASS PROPERTIES OF COMPONENTS OF THE ASSEMBLY
(in assembly units and the _BRAKING_SYSTEM coordinate frame)

DENSITY	MASS	C.G.: X	Y	Z
	INSIDE	MATERIAL:		UNKNOWN
8.61284e+00	1.14192e+07	-7.35899e+01	-8.37503e+00	-3.08152e+00
	OUTSIDE	MATERIAL:		UNKNOWN
9.08783e+00	9.74055e+06	-2.51668e+02	-4.03928e+01	-2.37410e+00
	WASHER_M10	MATERIAL:		UNKNOWN
2.00000e+00	6.95245e+02	-1.54723e+02	-1.48810e+01	5.46317e+01
	WASHER_M10	MATERIAL:		UNKNOWN
2.00000e+00	6.95245e+02	-1.75244e+02	-7.12626e+01	5.46317e+01
	WASHER_M10	MATERIAL:		UNKNOWN
2.00000e+00	6.95245e+02	-1.54723e+02	-1.48810e+01	-5.53683e+01
	WASHER_M10	MATERIAL:		UNKNOWN
2.00000e+00	6.95245e+02	-1.75244e+02	-7.12626e+01	-5.53683e+01
	BOLT_M10X25	MATERIAL:		UNKNOWN
9.00000e+00	3.35776e+04	-1.58196e+02	-1.36169e+01	-5.50000e+01
	BOLT_M10X25	MATERIAL:		UNKNOWN
9.00000e+00	3.35776e+04	-1.78717e+02	-6.99985e+01	-5.50000e+01
	BOLT_M10X25	MATERIAL:		UNKNOWN
9.00000e+00	3.35776e+04	-1.78717e+02	-6.99985e+01	5.50000e+01
	BOLT_M10X25	MATERIAL:		UNKNOWN
9.00000e+00	3.35776e+04	-1.58196e+02	-1.36169e+01	5.50000e+01
	AXIS_10X28	MATERIAL:		UNKNOWN
5.00000e+00	1.36369e+04	-4.88141e+01	-4.03554e+01	2.12353e+00
	PIPES	MATERIAL:		UNKNOWN
5.24912e+00	1.29776e+06	5.49887e+02	-4.21515e+02	4.96217e-01

2. Truck data

VOLUME = 6.8572298e+09 MM³
 SURFACE AREA = 6.9530871e+07 MM²
 AVERAGE DENSITY = 4.6355666e+02 POUND / MM³
 MASS = 3.1787146e+12 POUND

CENTER OF GRAVITY with respect to _TRUCK2 coordinate frame:
 X Y Z 1.6737096e+03 9.5000968e+02 7.6710179e+02 MM

INERTIA with respect to _TRUCK2 coordinate frame: (POUND * MM²)

INERTIA TENSOR:
 Ixx Ixy Ixz 6.5886787e+18 -5.0543125e+18 -3.7172951e+18
 Iyx Iyy Iyz -5.0543125e+18 1.5364885e+19 -2.3165015e+18
 Izx Izy Izz -3.7172951e+18 -2.3165015e+18 1.6022707e+19

INERTIA at CENTER OF GRAVITY with respect to _TRUCK2 coordinate frame:
 (POUND * MM²)

INERTIA TENSOR:

Ixx Ixy Ixz 1.8493312e+18 -2.8231730e+13 3.6387437e+17
Iyx Iyy Iyz -2.8231730e+13 4.5898408e+18 0.0000000e+00
Izx Izy Izz 3.6387437e+17 0.0000000e+00 4.2493133e+18

PRINCIPAL MOMENTS OF INERTIA: (POUND * MM^2)

I1 I2 I3 1.7953752e+18 4.3032693e+18 4.5898408e+18

ROTATION MATRIX from _TRUCK2 orientation to PRINCIPAL AXES:

0.98918 0.14668 0.00001
0.00001 0.00001 -1.00000
-0.14668 0.98918 0.00001

ROTATION ANGLES from _TRUCK2 orientation to PRINCIPAL AXES (degrees):

angles about x y z 89.999 0.000 -8.434

RADII OF GYRATION with respect to PRINCIPAL AXES:

R1 R2 R3 7.5153956e+02 1.1635190e+03 1.2016363e+03 MM

MASS PROPERTIES OF COMPONENTS OF THE ASSEMBLY

(in assembly units and the _TRUCK2 coordinate frame)

DENSITY	MASS	C.G.: X	Y	Z
FRONT-CHASSIS-29-4 MATERIAL:				
UNKNOWN				
5.00000e+02	3.12668e+11	6.00409e+02	9.50000e+02	3.40529e+02
CAB-20-3 MATERIAL: UNKNOWN				
6.00000e+02	1.37513e+12	1.26413e+03	9.50003e+02	1.27891e+03
REAR-25-4 MATERIAL: UNKNOWN				
3.49092e+02	9.74559e+11	2.83130e+03	9.50027e+02	4.79500e+02
WHEEL-8-4 MATERIAL: UNKNOWN				
4.39000e+02	1.10231e+11	2.00000e+02	1.56571e+02	5.50000e+01
WHEEL-8-4 MATERIAL: UNKNOWN				
4.39000e+02	1.10231e+11	2.00000e+02	1.74343e+03	5.50000e+01
ARMS-6-5 MATERIAL: UNKNOWN				
6.54780e+02	7.19945e+10	-1.96446e+02	9.50000e+02	1.12991e+03
VERRIN-1 MATERIAL: UNKNOWN				
3.41982e+02	9.51385e+08	2.51178e+02	5.21059e+02	5.65116e+02
VERRIN-1 MATERIAL: UNKNOWN				
3.41982e+02	9.51385e+08	2.51178e+02	1.37894e+03	5.65116e+02
FORK-6-5 MATERIAL: UNKNOWN				
5.40000e+01	1.53427e+09	-4.53346e+02	9.50000e+02	2.49902e+02
WHEEL-8-4 MATERIAL: UNKNOWN				
4.39000e+02	1.10231e+11	2.74500e+03	8.65707e+01	5.00000e+01
WHEEL-8-4 MATERIAL: UNKNOWN				
4.39000e+02	1.10231e+11	2.74500e+03	1.81343e+03	5.00000e+01

References

1. Aamodt, A. and Nygard, M. 1995. Different roles and mutual dependencies of data, information, and knowledge- An AI perspective on their integration. *Data and Knowledge Engineering*, 16 (3), pp. 191-222.
2. ADL (Advanced Distributed Learning). 2004. SCORM-Sharable content object reference model, 2nd edition-Overview. URL: <http://www.adlnet.gov/scorm/history/2004/index.cfm> (Last accessed: 15/12/2005).
3. Ahn, J.H. and Chang, J.G. 2004. Assessing the contribution of knowledge to business performance: the KP³ methodology. *Decision Support Systems*, 36 (4), pp. 403-416.
4. Auriol, E., Crowder, R. M., McKendrick, R., Rowe, R., and Knudsen. T. 1999. Integrating case-based reasoning and hypermedia documentation: an application for the diagnosis of a welding robot at Odense steel shipyard. *Engineering Applications of AI*, 12 (6), pp. 691-703.
5. Bakore, A., Galbraith, B., Wiggers, C., Eaves, J., Li, S., and Chopra, V. 2004. *Professional Apache Tomcat 5*, Wiley Publishing Inc., Indianapolis.
6. Baldauf, M., Dustdar, S., and Rosenberg, F. 2004. A survey on context aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, Forthcoming. URL: <http://citeseer.ist.psu.edu/baldauf04survey.html> (Last accessed: 28/09/2006).

7. Bastiaens, T. J. 1999. Assessing an electronic performance support system for the analysis of jobs and tasks. *International Journal of Training and Development*, 3 (1), pp. 54-61.
8. Belogun, J. and Jenkins, M. 2003. Re-conceiving change management: A knowledge-based perspective. *European Management Journal*, 21 (2), pp.247-257.
9. Benko, S. and Webster, S. 1997. Preparing for EPSS projects. *Communications of the ACM*, 40 (7), pp. 60-63.
10. Benton, W.C. and Srivastava, R. 1993 Product structure complexity and inventory storage capacity on the performance of a multi-level manufacturing system. *International Journal of Production Research*, 11 (31), pp. 2531-2545.
11. Bezanson, W. R. 1995. Performance support: Online, integrated documentation and training. *ACM Proceedings*, Savannah, Georgia, USA, pp. 1-10.
12. Bigus, J.P. and Bigus, J. 2001. *Constructing Intelligent Agents Using Java*. 2nd edition. John Wiley & Sons, New York.
13. Bose, R. 2003. Knowledge management-enabled health care management systems: capabilities, infrastructure, and decision support. *Expert Systems with Applications*, 24 (1), pp.59-71.
14. Boose, M. L., Shema, D. B., and Baum, L. S. 2003. A scalable solution for integrating illustrated parts drawings into a class IV interactive electronic

technical manual. Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR '03). IEEE Computer Society.

15. Brickley, D. and Guha, R. V. 2004. RDF Vocabulary Description Language 1.0: RDF Schema. (Eds). McBride, B., W3C Recommendation. URL: <http://www.w3.org/TR/rdf-schema/> (Last accessed: 10/12/2005).
16. Bronson, G. J. and Rosenthal, D. 2006. Object-oriented program development using Java: a class-centred approach. Thomson Course Technology.
17. Brusilovsky, P. and Cooper, D. W. 2002. Domain, task, and user models for an adaptive hypermedia performance support system. Proceedings of the Intelligent User Interfaces (IUI '02), pp. 23-30.
18. BS 4884-2. 1993. Technical Manuals, Part 2: Guide to Content (London: British Standards Institution).
19. BS 4899-2. 1992. Guide to User's Requirements for Technical Manuals (Based on the Principles of BS 4884). Part 2: Presentation (London: British Standards Institution).
20. Burkett, W. C. 2001. Product data markup language: a new paradigm for product data exchange and integration. *Computer-Aided Design*, 33 (7), pp. 489-500.
21. Cambridge English Dictionary, 2006. URL: <http://dictionary.cambridge.org/> (Last accessed: 09/01/2006).

22. Cantando, M. 1996. Vision 2000: Multimedia electronic performance support. Special Interest Group on design of Communication Conference (SIGDOC '96), pp. 111-114.
23. CISCO Systems. 1999. Reusable information object strategy-Definition, creation overview, and guidelines, version 3.0. URL: http://www.cisco.com/warp/public/779/ibs/solutions/learning/whitepapers/el_cisco_rio.pdf#search=%22CISCO%20Reusable%20Information%20Object%22 (Least accessed: 28/09/2006).
24. Cliff, S. 1999. Information is power? Envisioning the Minnesota public internet - public service and community information and interaction in the public interest. Information for Change conference, St. Paul, Minnesota.
25. Coffey, J. W., Canas, A. J., Hill, G., Carff, R., Reichherzer, T., and Suri, N. 2003. Knowledge modelling and the creation of EI-Tech: a performance support and training system for electronic technicians. *Expert Systems with Applications*, 25 (4), pp. 483-492.
26. COM2001 (Commission of the European Communities). 2001. The eLearning action plan: Designing tomorrow's education. Communication for the Commission to the Council and the European Parliament, Brussels, pp.1-19.
27. Cornet, R. and Abu Hanna, A. 2005. Description logic-based methods for auditing frame-based medical terminological systems. *Artificial Intelligence in Medicine*, 34 (3), pp. 201-217.

28. Crowder, R., Sim, Y. W., Wills, G., and Greenough, R. 2001. A review of the benefits of using hypermedia manuals. Hypertext and Hypermedia conference (HT '01), pp. 245-246.
29. Crubezy, M., O'Connor, M. J., Buckeridge, D. L., Pincus, S. S., and Musen, M. A. 2005. Ontology-centered syndromic surveillance for bioterrorism. IEEE Intelligent Systems, 20 (5), pp. 26-35.
30. Dale, N. and Weems, C. 2005. Programming and Problem Solving with C++ (4th edition). Jones and Bartlett, Massachusetts.
31. Day, D., Priestley, M., and Schell, D. 2005. Introduction to the Darwin Information Typing Architecture. URL: <http://www-128.ibm.com/developerworks/xml/library/x-dita1/> (Last accessed: 28/09/2006).
32. Davies, J., Fensel, D., and Van Harmelen, F. 2003. Towards the Semantic Web- Ontology-driven knowledge management. John Wiley & Sons.
33. Desmarais, M. C., Leclair, R., Fiset, J. Y., and Talbi, H. 1997. Cost-justifying electronic performance support systems. Communications of the ACM, 40 (7), pp. 39-48.
34. Dey, A.K. 2001. Understanding and using context. Personal and Ubiquitous Computing, 5 (1), pp. 4-7.

35. Dublin Core. 2005a. Using Dublin Core. URL: <http://dublincore.org/documents/usageguide/> (Last accessed: 28/09/2006).
36. Dublin Core. 2005b. DCMI Abstract Model. URL: <http://dublincore.org/documents/abstract-model/> (Last accessed: 28/09/2006).
37. Duval, E. and Hodgins, W. 2003. A LOM research agenda. Proceedings of the twelfth international conference on World Wide Web, pp. 1-9.
38. Elsbernd, G. 2001. Performance-centered portals. Performance Improvement, 40 (10), pp. 24-29.
39. Erickson, T. 2002. Some Problems with the notion of context-aware computing. Communications of the ACM, 45 (2), pp. 102-104.
40. Fallside, D. C. and Walmsley, P. 2004. XML Schema Part 0: Primer Second Edition. W3C Recommendation. URL: <http://www.w3.org/TR/xmlschema-0/> (Last accessed: 10/12/2005).
41. Foo, S., Hui, S. C., and Leong, P. C. 2002. Web-based intelligent helpdesk-support environment. International Journal of Systems Science, 33 (6), pp. 389-402.
42. Forzese, A. C. 1997. How do performance support and minimalist systems designers consider audience?: Some design tools for user-focused systems. IPCC '97 Proceedings. IEEE Crossroads in Communication, pp. 167-180.

43. Frost, A. 1998. Interactive electronic technical manual research study. Proceedings of the IEEE Aerospace and Electronics Conference (NAECON '98), pp. 80-87.
44. Gao, J.X., Aziz, H., Maropoulos, P.G., and Cheng, W.M. 2003. Application of product data management technologies for enterprise integration. International Journal of Computer Integrated Manufacturing, 16 (7-8), pp. 491-500.
45. Gery, G. 1995. Attributes and behaviors of performance-centered systems. Performance Improvement Quarterly, 8 (1), pp. 47-93.
46. Gharbi, A., Kenne, J. P., and Beit, M. 2007. Optimal safety stocks and preventive maintenance periods in unreliable manufacturing systems. International Journal of Production Economics, 107 (2), pp. 422-434.
47. Goffin, K. 1998. Evaluating customer support during new product development- an exploratory study. Journal of Production Innovation Management, 15 (1), pp. 42-56.
48. Gonzalez, A. J. and Dankel, D. D. 1993. The engineering of knowledge-based systems: Theory and Practice. Prentice Hall. USA.
49. Gray, P.H. 2001. A problem-solving perspective on knowledge management practices. Decision Support Systems, 31 (1), pp. 87-102.

50. Gruber, T. R., Vemuri, S., and Rice, J. 1997. Model-based virtual document generation. *International Journal of Human Computer Studies*, 46 (6), pp. 687-706.
51. Gunnlaugsdottir, J. 2003. Seek and you will find, share and you will benefit: organising knowledge using groupware systems. *International Journal of Information Management*, 23 (5), pp.363-380.
52. Gunzelmann, G., Anderson, J.R. 2003. Problem-solving: Increased planning with practice. *Cognitive Systems Research*, 4 (1), pp. 57-76.
53. Hennum, E. 2005. Specialising domains in DITA. URL: <http://www-128.ibm.com/developerworks/xml/library/x-dita5/> (Last accessed: 28/09/2006).
54. Horn, R. 1999. Two approaches to modularity: Comparing the STOP approach with structured writing. *Journal of computer documentation*, 23 (3), pp. 87-95.
55. Huneiti, A. M. 2004. Hypermedia-based performance support systems for the Web. Ph.D Thesis, Cardiff University.
56. IEEE LOM. 2002. Draft standard for learning object metadata. URL: http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf#search=%22IEEE%20LOM%22 (Last accessed: 28/09/2006).
57. ISO 8402: 1994. 1994. Quality Management and Quality Assurance. Vocabulary of the ISO 9000 Quality Standard.

58. ISO 10303-1: 1994. 1994. Industrial Automation Systems and Integration-Product Data Representation and Exchange. Part 1: Overview and Fundamental Principles, TC184/ SC 4, ISO.
59. Johanson L. 2003. FreeCBR documentation. URL:
http://freecbr.sourceforge.net/api_1.1.2/ (Last accessed: 10/12/2005).
60. Joyce, D. A. 2002. Electronic performance support for royal navy technicians-a new paradigm for engineering training? Engineering Education 2002: Professional Engineering Scenarios (Ref. No. 2002/056), IEE, 2 (3-4), pp. 36/1-36/6.
61. Kabel, M. A. and Kiger, R. 1997. Convergence of knowledge engineering and electronic performance support systems. Professional Communication Conference (IPCC '97). IEEE Crossroads in Communication, pp. 45-51.
62. Kahney, H. 1992. Problem solving-current issues, 2nd edition. Open Guides to Psychology, Open University Press, McGraw Hill, London.
63. Kakabadse, N. K., Kakabadse, A., and Kouzmin, A. 2003. Reviewing the knowledge management literature: Towards a taxonomy. Journal of Knowledge Management, 7 (4), pp.75-91.
64. Kaposi, A. and Myers, M. 2001. Systems for All. Imperial College Press, London.

65. Khalifa, M. and Liu, V. 2006. Semantic network representation of computer-mediated discussions: Conceptual facilitation form and knowledge acquisition. Omega, In Press.
66. Knublauch, H., Fergerson, R. W., Noy, N. F., Musen, M. A. 2004. The Protégé OWL plugin: An open development environment for semantic web applications. Third International Semantic Web Conference - ISWC 2004, Hiroshima, Japan.
67. Kolodner, J. Case-Based Reasoning. Morgan Kaufmann, San Mateo, 1993.
68. Kraidli, R., Ammar, H., Reynolds, D., and Copen, G. 2003. Adaptive electronic technical manuals. Computer Systems and Applications. Book of Abstracts. International Conference on ACS/IEEE.
69. Kuflik, T., Shapira, B., and Shoval, P. 2003. Stereotype-based versus personal-based filtering rules in information filtering systems. Journal of the American Society for Information Science and Technology, 54 (3), pp. 243-250.
70. Lagos, N., Setchi, R. M., and Dimov, S. S. 2005. Towards the integration of performance support and e-Learning: Context-aware product support systems, (Eds). Meersman, R., Tari, Z., and Herrero, P., Lecture Notes in Computer Science (LNCS), 3762, pp. 1149 - 1158.
71. Liao, S.H. 2002. Problem solving and knowledge inertia. Expert Systems with Applications. 22 (1), pp. 21-31

72. Liebowitz, J. and Megbolugbe, I. 2003. A set of frameworks to aid the project manager in conceptualising and implementing knowledge management initiatives. *International Journal of Project Management*, 21 (3), pp.189-198.
73. Mackenzie, C. 2002. The need for a Design Lexicon: Examining minimalist, performance-centered and user-centered design. *Technical Communication*, 49 (4), pp. 405-410.
74. MacMullin, S. E. and Taylor, R. S. 1984. Problem dimensions and information traits. *Information Society*, 3 (1), pp. 91-111.
75. Marion, C. 2002. Attributes of Performance-Centered Systems: What can we learn from five years of EPSS/PCD competition award winners?. *Technical Communication*, 49 (4), pp. 428-443.
76. Matheiu, V. 2001. Product Services: From a service supporting the product to a service supporting the client. *Journal of Business and Industrial Marketing*, 16 (1), pp. 39-58.
77. McCarthy, J. 1993. Notes on Formalizing Context. Proceedings of the 23rd Artificial Intelligence International Joint Conference (AI IJCAI'93), Chambéry, France, pp. 555-560.
78. McGuinness, D. L. and Van Harmelen, F. 2004. OWL Web Ontology Language Overview. W3C Recommendation. URL: <http://www.w3.org/TR/owl-features/> (Last accessed: 10/12/2005).

79. McMahon, C., Lowe, A., Culley, S., Corderoy, M., Crossland, R., Shah, T., and Stewart, D. 2004. Waypoint: An integrated search and retrieval system for engineering documents. *Journal of Computing and Information Science in Engineering*, 4 (4), pp. 329-338.
80. Mendes, E., Mosley, N., and Watson, I. 2002. A comparison of case-based reasoning approaches. *Proceedings of 11th International Conference on World Wide Web*, pp. 272-280.
81. Morakis, E., Vidalis, S., and Blyth, A. 2003. Measuring vulnerabilities and their exploitation cycle. *Information Security Technical Report*, 8 (4), pp. 45-55.
82. Namahn. 2001. Darwin Information Typing Architecture (DITA), a research note by Namahn. URL: <http://www.namahn.com/resources/documents/note-DITA.pdf> (Last accessed: 28/09/2006).
83. Nonaka, I. 1991. The knowledge creating company. *Harvard Business Review*, pp.96-104.
84. Nonaka, I. 1994. A dynamic theory of organisational knowledge creation. *Organisation Science*, 5 (1), pp.14-37.
85. OASIS DITA. 2005. OASIS Darwin Information Typing Architecture (DITA) architectural specification v1.0. URL: <http://xml.coverpages.org/DITAv10-OS-ArchSpec20050509.pdf> (Last accessed: 28/09/2006).

86. Ockerman, J. J., Najjar, L. J., and Thomson, J. C. 1999. FAST: future technology for today's industry. *Computers in Industry*, 38 (1), pp. 53-64.
87. Oxford English Dictionary 2nd edition, 2006. URL: <http://dictionary.oed.com/>
(Last accessed: 09/01/2006).
88. Pasantonopoulos, C. 2005. Automatic construction of virtual technical documentation. Thesis submitted to Manufacturing Engineering Centre (MEC), Cardiff University.
89. Paul, C., Zeiler, G., and Nolan, M. 2003. Integrated support system for the self protection system. *IEEE Systems Readiness Technology Conference Proceedings (AUTOTESCON 2003)*, pp. 155-160.
90. Petiot, J.F. and Yannou, B. 2004. Measuring consumer perceptions for a better comprehension, specification, and assessment of product semantics. *International Journal of Industrial Ergonomics*, 33 (6), pp. 507-525.
91. Pham, D. T., Dimov, S. S., and Huneiti, A. M. 2003. Semantic data model for product support systems. *IEEE International Conference on Industrial Informatics (INDIN 2003)*, pp. 279-285.
92. Pham, D. T., Dimov, S. S., and Peat, B. J. 2000. Intelligent product manuals. *Proceedings of the Institution of Mechanical Engineers IMechE*, 214 (B), pp. 411-419.

93. Pham, D. T., Dimov, S. S., and Setchi, R. M. 1999a. Intelligent product manuals. Proceedings of the Institution of Mechanical Engineers IMechE, 213 (I), pp. 65-76.
94. Pham, D. T., Dimov, S. S., and Setchi, R. M. 1999b. Concurrent Engineering: a tool for collaborative working. Human Systems Management, 18 (3-4), pp. 213-224.
95. Pham, D. T. and Setchi, R. M. 2000. Adaptive Product Manuals. Proceedings of the Institution of Mechanical Engineers IMechE, 214 (C), pp. 1013-1018.
96. Pham, D. T. and Setchi, R. M. 2003. Case-based generation of adaptive product manuals. Proceedings of the Institution of Mechanical Engineers, 217 (B), pp. 313-322.
97. Pham, D. T., Setchi, R. M., and Dimov, S. S. 2002. Enhanced product support through intelligent product manuals. International Journal of Systems Science, 33 (6), pp. 433-449.
98. Plato on Knowledge in the Theaetetus, Stanford Encyclopedia of Philosophy. 2006. URL: <http://plato.stanford.edu/entries/plato-theaetetus/> (Last accessed: 09/01/2006).
99. Portinale, L., Magro, D., and Torasso, P. 2004. Multi-modal diagnosis combining case-based and model-based reasoning: a formal and experimental analysis. Artificial Intelligence. 158 (2), pp. 109-153.

100. Prekop, P. and Burnett, M. 2003. Activities, context and ubiquitous computing. *Computer Communications*, 26 (11), pp. 1168-1176.
101. Preuveneers, D., Van den Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., and De Bosschere, K. 2004. Towards an extensible context ontology for ambient intelligence. In: Markopoulos, P., Eggen, B., Aarts, E., Crowley, J.L. (Eds.). *Ambient Intelligence. Lecture Notes in Computer Science*, 3295, pp. 148-159.
102. PTC. 1998. *Pro/Engineer Online Books*, Parametric Technology Corporation, Waltham, MA, USA.
103. Quesenbery, W. 2002. Who is in control? The logic underlying the intelligent technologies used in performance support. *Technical Communication*, 49 (4), pp. 449-457.
104. Ranwez, S. and Crampes, M. 1999. Conceptual documents and hypertext documents are two different forms of virtual document. *Workshop on Virtual Documents, Hypertext Functionality and the Web*, In: 8th International World Wide Web Conference, Toronto, Canada.
105. Raybould, B. 1995. Performance support engineering: an emerging development methodology for enabling organisational learning. *Performance Improvement Quarterly*, 8 (1), pp. 7-22.

106. Raybould, B. 2000. Building performance-centered web-based systems, information systems and knowledge management systems, in the 21st century. *Performance Improvement*, 39 (6), pp. 32-39.
107. Rayside, D., Litoiu, M., Storey, M. A., Best, C., and Lintern, R. 2003. Visualizing flow diagrams in Websphere studio using SHriMP views (visualizing flow diagrams). *Information Systems Frontiers*, 5 (2), pp. 161-174.
108. Ruland, D. and Spindler, T. 1995. Integration of product and design data using a metadata- and a rule-based approach. *Computer Integrated Manufacturing Systems*, 8 (3), pp. 211-221.
109. Russell, S. and Norvig, P. 1995. *Artificial Intelligence-A modern approach*. In: *Artificial Intelligence Series*, Prentice Hall.
110. Sanchez, S. and Sicilia, M. A. 2004. On the semantics of aggregation and generalisation in Learning Object contracts. *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT'04)*. IEEE Computer Society, pp. 425 – 429.
111. Savsar, M. 2006. Effects of maintenance policies on the productivity of flexible manufacturing cells. *Omega-The International Journal of Management Science*, 34 (3), pp. 274-282.

112. Schilit, B. N., Adams, N. I., and Want, R. 1994. Context-aware computing applications. Proceedings of the Workshop on Mobile Computing Systems and Applications, pp. 85-90.
113. SEMTA-Science, Engineering, Manufacturing Technologies Alliance. 2004. Aerospace Sector Skills Agreement: Stage 1. Consultation Draft: Aerospace SSA.
114. Setchi, R. M. 2000. Enhanced product support through intelligent product manuals. Thesis submitted to Cardiff School of Engineering, University of Wales Cardiff.
115. Setchi, R. M., Huneiti, A. M., and Pasantonopoulos, C. 2006. The evolution of intelligent product support. Proceedings of the 5th CIRP International Seminar on intelligent computation in manufacturing engineering (CIRP ICME '06), 5, pp. 639-644, Ischia, Italy.
116. Setchi, R. M. and Lagos, N. 2005. A problem solving approach to the development of product support systems. Proceedings of the 1st I*PROMS Virtual International Conference on Intelligent Production Machines and Systems (IPROMS 2005), 4-15 July 2005, Cardiff, UK, pp. 79-84.
117. Setchi, R. M. and Lagos, N. 2006a. Semantic-based authoring of technical documentation. World Automation Congress 2006 (WAC2006) - 10th International Symposium on Manufacturing and Applications (ISOMA 2006), Budapest, Hungary.

118. Setchi, R. M. and Lagos, N. 2006b. Context modelling for product support systems. *Acta Mechanica Slovaca*, 2 (A), pp. 425-434.
119. Setchi, R. M., Lagos, N., and Dimov, S. S. 2005. Semantic modelling of product support knowledge. *Proceedings of the 1st I*PROMS Virtual International Conference on Intelligent Production Machines and Systems (IPROMS 2005)*, 4-15 July 2005, Cardiff, UK, pp. 275-281.
120. Sintek, M. 2005a. *OntoViz documentation*.
URL: <http://protege.stanford.edu/plugins/ontoviz/ontoviz.html> (Last accessed: 10/12/2005).
121. Sintek, M. 2005b. *RDF Backend*.
URL: <http://protege.stanford.edu/plugins/rdf> (Last accessed: 10/12/2005).
122. Stry, C. and Stoiber, S. 2003. Model-based electronic performance support. *Lecture Notes in Computer Science*. (Eds.) J. A. Jorge, N. J. Nunes, J. Falcao e Cunha. 2844, pp. 258-272.
123. Stcherbatchenko, A., Prantl, F., and Levin, S. 2005. *JCreator 2.5 Help*. Xinox software.
124. Stefik, M. 1995 *Introduction to Knowledge Systems*. Morgan Kaufmann Publishers, London.

125. STEPSTONE. 2006. NSViewer Help. URL:
<http://www.microcad.co.jp/NSViewerSTEP-E/products/> (Last accessed:
02/01/2006).
126. Storey, M. A., Noy, N. F., Musen, M., Best, C., and Ferguson, R. 2002. Jambalaya: an interactive environment for exploring ontologies. International Conference on Intelligent User Interfaces, pp. 239.
127. Stuckenschmidt, H. and Van Harmelen, F. 2005. Information sharing on the semantic web. In: Advanced Information and knowledge processing series. Springer.
128. Su, L. P., Bosco, C. D., and England, W. 1997. Application of new information technology to DOD legacy paper technical manuals. IEEE Proceedings of the Autotestcon conference (AUTOTESTCON '97), pp. 18-22.
129. Szykman, S., Sriram, R.D., and Regli, W.C. 2001. The role of knowledge in next-generation product development systems. Journal of Computing and Information Science in Engineering, 1 (1), pp. 3-11.
130. Tucker, H. and Harvey, B. 1997. SGML documentation objects within the STEP environment. SGML Europe '97, Barcelona, Spain. URL:
<http://www.eccnet.com/papers/step.html> (Last accessed: 27/09/2006).

131. Van Amstel, P., Van der Eijk, P., Haasdijk, E., and Kuilman, D. 2000. An interchange format for cross-media personalised publishing. *Computer Networks*, 33 (1-6), pp. 179-195.
132. Verbert, K. and Duval, E. 2004. Towards a global architecture for learning objects: a comparative analysis of learning object content models. *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-MEDIA 2004)*, pp. 202-209.
133. Vercoustre, A. M., Dell'Oro, J., and Hills, B., 1997, Reuse of information through virtual documents. *Second Australian Document Computing Symposium, Melbourne, Australia*, pp. 55-64.
134. Wagner, E. D. 2002. Steps to creating a content strategy for your organisation. *The e-Learning Developers Journal*. URL: <http://www.elearningguild.com/pdf/2/102902MGT-H.pdf#search=%22Steps%20to%20creating%20a%20content%20strategy%20for%20your%20organisation%22> (Last accessed: 28/09/2006).
135. Waldeck, E. N. and Lefakis, Z. N. 2005 HR perceptions and the provision of workforce training in an AMT environment: An empirical study. *Omega*, 35 (2), pp. 161-172.
136. Wang, K. J. and Lin, Y. S. 2007. Resource allocation by genetic algorithm with fuzzy inference. *Expert Systems with Applications*, 33 (4), pp. 1025-1035.

137. Webster Online Dictionary, 2006. URL: <http://www.websters-online-dictionary.org/> (Last accessed: 09/01/2006).
138. Wielinga, B., Schreiber, G., and Breuker, J. 1993. Modelling Expertise. In: KADS: A principled approach to knowledge-based system development. XI, pp. 21-47, G. Schreiber, B. Wielinga, J. Breuker, Eds. New York: Academic.
139. Wills, G., Sim, Y. W., Crowder, R., and Hall, W. 2002. Open hypermedia for product support. *International Journal of Systems Science*, 33 (6), pp. 421-432.
140. Yao, H. and Eitzkorn, L. 2006. Automated conversion between different knowledge representation formats. *Knowledge Based Systems*, 19 (6), pp. 404-412.
141. Yeh, I., Karp, P. D., Noy, N. F., and Altman, R. B. 2003. Knowledge acquisition, consistency checking and concurrency control for Gene Ontology (GO). *Bioinformatics*, 19 (2), pp. 241-248.
142. Yim, N. H., Kim, S. H., Kim, H. W., and Kwahk, K. Y. 2004. Knowledge based decision making on higher level strategic concerns: system dynamics approach. *Expert Systems with Applications*, 27 (1), pp. 143-158.

