

# Feature Grammar Systems

Incremental Maintenance  
of Indexes to  
Digital Media Warehouses

Menzo Windhouwer

... *van M voor M* :-)

# Feature Grammar Systems

Incremental Maintenance  
of Indexes to  
Digital Media Warehouses

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor  
aan de Universiteit van Amsterdam,  
op gezag van de Rector Magnificus  
prof. mr. P. F. van der Heijden  
ten overstaan van een door het  
college voor promoties ingestelde commissie  
in het openbaar te verdedigen  
in de Aula der Universiteit  
op donderdag 6 november 2003, te 14.00 uur

door Menzo Aart Windhouwer  
geboren te Apeldoorn

## Promotiecommissie

Promotor: prof. dr M. L. Kersten

Overige commissieleden: prof. dr P. M. G. Apers  
prof. dr P. M. E. de Bra  
prof. dr P. van Emde Boas  
prof. dr P. Klint

## Faculteit

Faculteit der Natuurwetenschappen, Wiskunde en Informatica  
Universiteit van Amsterdam



The research reported in this thesis was carried out at CWI, the Dutch national research laboratory for mathematics and computer science, within the theme *Data Mining and Knowledge Discovery*, a subdivision of the research cluster *Information Systems*.



SIKS Dissertation Series No-2003-16.

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

The research and implementation effort reported in this thesis was carried out within the *Advanced Multimedia Indexing and Searching*, *Digital Media Warehouses* and *Waterland* projects.

ISBN 90 6196 522 5

Cover design and photography by Dick Windhouwer ([www.windhouwer.nl](http://www.windhouwer.nl)).

# Contents

<b>1</b>	<b>Digital Media Warehouses</b>	<b>1</b>
1.1	Multimedia Information Retrieval	2
1.2	Annotations	4
1.2.1	The Semantic Gap	4
1.2.2	Annotation Extraction Algorithms	5
1.2.3	Annotation Extraction Dependencies	6
1.2.4	Annotation Maintenance	7
1.3	The Acoi System	8
1.3.1	A Grammar-based Approach	8
1.3.2	System Architecture	9
1.3.3	Case Studies	10
1.3.3.1	The WWW Multimedia Search Engine	10
1.3.3.2	The Australian Open Search Engine	11
1.3.3.3	Rijksmuseum Presentation Generation	11
1.4	Discussion	12
<b>2</b>	<b>Feature Grammar Systems</b>	<b>13</b>
2.1	A Grammar Primer	15
2.1.1	A Formal Specification of Languages and Grammars	15
2.1.1.1	The Chomsky Hierarchy	16
2.1.1.2	Mild Context-sensitivity	17
2.1.2	The Derivation Process	18
2.1.2.1	Bidirectional Grammars	19
2.1.2.2	Parse Trees	19
2.1.2.3	The Delta Operation	19
2.1.2.4	Parse Forests	20
2.1.2.5	Disambiguation	21
2.1.3	Regulated Rewriting	21
2.1.3.1	Conditional Grammars	22
2.1.3.2	Tree-controlled Grammars	22
2.1.4	Grammar Systems	24

2.1.4.1	CD Grammar Systems . . . . .	24
2.1.4.2	Internal Control . . . . .	26
2.2	Feature Grammar Systems . . . . .	27
2.2.1	Detectors . . . . .	28
2.2.2	Atoms . . . . .	32
2.2.3	Dependencies . . . . .	33
2.2.3.1	Detector Input . . . . .	33
2.2.3.2	Detector Output . . . . .	37
2.2.4	Ambiguous Feature Grammar Systems . . . . .	39
2.2.5	Mildly Context-sensitive Feature Grammar Systems . . . . .	41
2.3	Discussion . . . . .	42
<b>3</b>	<b>Feature Grammar Language</b>	<b>45</b>
3.1	The Basic Feature Grammar Language . . . . .	45
3.1.1	Production Rules . . . . .	46
3.1.2	Atoms . . . . .	48
3.1.3	Detectors . . . . .	49
3.1.3.1	The XPath Language . . . . .	50
3.1.3.2	Detector Confidence . . . . .	52
3.1.4	The Start Symbol . . . . .	52
3.2	The Extended Feature Grammar Language . . . . .	53
3.2.1	Production Rules . . . . .	53
3.2.1.1	Additional Sequence Types . . . . .	53
3.2.1.2	Constants . . . . .	54
3.2.2	Detectors . . . . .	54
3.2.2.1	Whitebox Detectors . . . . .	54
3.2.2.2	Plugins . . . . .	56
3.2.2.3	Classifiers . . . . .	57
3.2.3	The Start Symbol . . . . .	58
3.2.3.1	References . . . . .	58
3.2.4	Feature Grammar Modules . . . . .	61
3.2.5	Change Detection . . . . .	62
3.3	Discussion . . . . .	63
<b>4</b>	<b>Feature Detector Engine</b>	<b>65</b>
4.1	A Parser Primer . . . . .	66
4.1.1	More Parsing Algorithms for Context-free Grammars . . . . .	69
4.2	Parsing Feature Grammar Systems . . . . .	73
4.2.1	Exhaustive Backtracking for Feature Grammar Systems . . . . .	75
4.2.1.1	Left-recursion . . . . .	78
4.2.1.2	Lookahead . . . . .	80
4.2.1.3	Memoization . . . . .	81
4.3	The Feature Detector Engine . . . . .	82

4.3.1	The Symbol Table	84
4.3.1.1	Rewriting	84
4.3.1.2	Semantic Checks	85
4.3.2	The Parser	86
4.3.3	The Parse Forest	86
4.3.3.1	XML and DOM	86
4.3.3.2	Labeling Parse Trees	88
4.3.3.3	Memoized Parse Trees	90
4.3.4	The Sentences	92
4.3.5	Detectors	93
4.3.5.1	Detector Input	93
4.3.5.2	Blackbox Detectors	94
4.3.5.3	Plugins	94
4.3.5.4	Classifiers	95
4.3.5.5	Start Symbols and References	95
4.3.5.6	Deadlock Resolution	96
4.4	Discussion	97
<b>5</b>	<b>Feature Databases</b>	<b>99</b>
5.1	The Monet Database Kernel	99
5.1.1	Monet and XML	100
5.1.1.1	Semistructured Data	100
5.1.1.2	Monet XML and XMark	101
5.1.1.3	XQuery	102
5.2	A Feature Database	102
5.2.1	A Database Schema	103
5.2.2	A parse forest XML document	105
5.2.3	Inserting a Parse Forest	107
5.2.4	Replacing a (Partial) Parse Forest	107
5.2.5	Query Facilities	108
5.2.6	Adding Database Management to a Database Kernel	109
5.3	Discussion	110
<b>6</b>	<b>Feature Detector Scheduler</b>	<b>111</b>
6.1	The Dependency Graph	111
6.2	Identifying Change	114
6.2.1	External Changes	114
6.2.2	Internal Changes	115
6.2.2.1	The Start Condition	115
6.2.2.2	The Detector Function	115
6.2.2.3	The Stop Condition	116
6.3	Managing Change	116
6.3.1	The (Re)validation Process	117

6.3.2	Lookup Table Management . . . . .	117
6.4	Discussion . . . . .	118
<b>7</b>	<b>Case Studies</b>	<b>119</b>
7.1	The Acoi Implementation . . . . .	119
7.1.1	Acoi Prehistory . . . . .	119
7.1.2	The Acoi Project . . . . .	120
7.1.3	Acoi 1998 . . . . .	120
7.1.4	Acoi 2000 . . . . .	120
7.1.5	Acoi 2002 . . . . .	121
7.1.6	Acoi Future . . . . .	122
7.2	The WWW Multimedia Search Engine . . . . .	123
7.2.1	The Feature Grammars . . . . .	123
7.2.2	The System Architecture . . . . .	124
7.2.3	Lessons Learned . . . . .	125
7.3	The Australian Open Search Engine . . . . .	126
7.3.1	The Webspace Method . . . . .	127
7.3.2	COBRA . . . . .	128
7.3.3	The Australian Open DMW Demonstrator . . . . .	128
7.3.4	Lessons Learned . . . . .	132
7.4	Rijksmuseum Presentation Generation . . . . .	133
7.4.1	The Style Repository . . . . .	133
7.4.2	The Data Repository . . . . .	135
7.4.3	The Presentation Environment . . . . .	136
7.4.4	A Style Feature Grammar . . . . .	137
7.4.5	Generating the Presentation . . . . .	139
7.4.6	Lessons Learned . . . . .	141
7.5	Discussion . . . . .	142
<b>8</b>	<b>Conclusion and Future Work</b>	<b>143</b>
8.1	Conclusion . . . . .	143
8.2	Future Work . . . . .	144
8.2.1	Feature Grammar Systems . . . . .	144
8.2.2	Feature Grammar Language . . . . .	144
8.2.3	Feature Detector Engine . . . . .	145
8.2.4	Feature Database . . . . .	145
8.2.5	Feature Detector Scheduler . . . . .	145
8.2.6	Digital Media Warehouses . . . . .	146
8.3	Discussion . . . . .	146
<b>A</b>	<b>The Feature Grammar Language</b>	<b>147</b>



---

<b>B</b>	<b>Feature Grammars</b>	<b>151</b>
B.1	The WWW Feature Grammar . . . . .	151
B.2	The Text Feature Grammar . . . . .	151
B.3	The HTML Feature Grammar . . . . .	152
B.4	The Image Feature Grammar . . . . .	153
B.5	The Audio Feature Grammar . . . . .	153
B.6	The MIDI Feature Grammar . . . . .	154
B.7	The MP3 Feature Grammar . . . . .	154
B.8	The Video Feature Grammar . . . . .	154
B.9	The MPEG Feature Grammar . . . . .	155
B.10	The Acoi Feature Grammar . . . . .	155
B.11	The Tennis Feature Grammar . . . . .	155
B.12	The Australian Open Feature Grammar . . . . .	156
B.13	The Rijksmuseum Feature Grammar . . . . .	156
<b>C</b>	<b>XML documents</b>	<b>159</b>
C.1	A schema document . . . . .	159
C.2	A parse forest document . . . . .	161
	<b>Abbreviations</b>	<b>163</b>
	<b>Bibliography</b>	<b>165</b>
	<b>Index</b>	<b>179</b>
	<b>Samenvatting</b>	<b>183</b>
	<b>Curriculum Vitae</b>	<b>185</b>
	<b>Acknowledgments</b>	<b>189</b>
	<b>SIKS Dissertation Series</b>	<b>191</b>



# Chapter 1

## Digital Media Warehouses

*Yama said, “Surely knowledge should be free to everyone, since all knowledge is the gift of the Preservers.”*

*“Ah, but if it was freed,” Kun Nurbo said, “who would look after it? Knowledge is a delicate thing, easily destroyed or lost, and each part of the knowledge we look after is potentially dependent upon every other part. I could open the library to all tomorrow, if I was so minded, but I will not. You could wander the stacks for a dozen years, Yama, and never find what you are looking for. I can lay my hand on the place where the answer may lie in a few hours, but only because I have spent much of my life studying the way in which the books and files and records are catalogued. The organization of knowledge is just as important as knowledge itself, and we are responsible for the preservation of that organization.”*

Paul J. McAuley – *Ancients of Days*

Encouraged by the low price of digitizing methods (*e. g.* digital cameras, scanners) and storage capacity (*e. g.* DVDs) collections of media objects are quickly becoming popular. Public services like libraries and museums digitize their collections and make parts of it available to the public. Likewise, the public digitizes private information, *e. g.* holiday pictures, and shares it on the World Wide Web (WWW). Vast collections of digital media are thus constructed in a relatively easy manner.

The management of these media objects encompasses many more aspects than just populating the *digital media warehouse* (DMW). First, there is the retention issue of the digital content; will the digital image be accessible in 25 years from now? Dedicated database [Sub97] and file systems [Bos99] have been developed to handle the input, storage and output of media streams. Second, security issues play a role: who is allowed to retrieve the data and should the data be encrypted? Third, the mass of information in a DMW stresses our capability to find relevant information: how to retrieve all images related to, for example, jazz music? The next section will delve

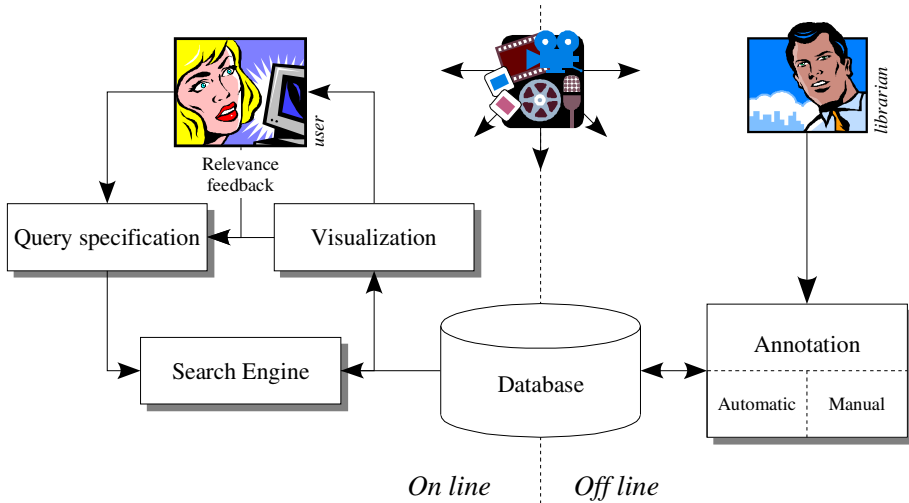


Figure 1.1: Multimedia information retrieval system

deeper into this last issue, because the contribution of this thesis lies within this grand challenge.

## 1.1 Multimedia Information Retrieval

A known and limited subset of a warehouse can be accessed by browsing: the common practice of every day life on the WWW. However, if this subset is unknown, identifying relevant media objects in the vast collection poses a major problem.

Identifying relevant media objects is studied in the area of *multimedia information retrieval*. This research community is multidisciplinary and thus attracts scientist from various disciplines, *e. g.* computer vision, artificial intelligence, natural language processing and database technology. These disciplines play their specific role in the subsystems of the generic multimedia information retrieval system sketched in Figure 1.1 (based on [Del99]). A small walk through this generic system will clarify the information flow between and the individual roles of the various subsystems.

The user, *i. e.* the person in the left part of the figure, starts a session with the system to resolve a query intention, for example: find a portrait of the jazz trumpeter Chet Baker. Query specification tools offer assistance in translating the query intentions into query clues understood by the system. For example: *query-by-sketch* (QbS) [DP97, JFS95] or *query-by-text* (QbT) [dJGHN00, KKK<sup>+</sup>91, OS95] are well-known paradigms being used. In QbS a global impression of the image has to be drawn. Keywords or phrases, like “Chet Baker”, “jazz” or “trumpet”, form the clues used by the

QbT paradigm.

These query clues are subsequently translated by the search engine into database queries. The type of the information stored in the database, and thus these translations, is as diverse as the query specification paradigms. For example: the QbS paradigm maps the clues on numerical feature vectors containing information about color, texture and shapes. The keywords and phrases from the QbT paradigm may map on entries in an ontology [SDW01], controlled vocabulary or textual annotations. This mapping from query clues to the information stored forms the basis to find matching media objects. When the mapping is also used for the ranking of matching objects the search engine needs a notion of similarity: how similar are two objects in the space induced by the mapping? Using this distance metric the objects can be ranked from the best to the worst match [Fal96].

The database executes the query specification to match and rank the media objects. A visualization tool presents these query results for further inspection to the user. Also for this part of the generic system many paradigms are available: the results may be shown as clusters in a multidimensional space [vLdLW00] or the user can browse through them [CL96]. Other senses than the user's eyes may also be used to present the query results, *e. g.* when the media type is audio or a score the musical theme is played [MB01].

In most cases the user will have to refine the query to zoom in on the relevant set of multimedia objects [MM99, VWS01]. This relies on a better understanding by the user of the database content thus allowing a better formulation of the information need. Query refinement is supported by a relevance feedback mechanism [CMOY96, RHM98, RHOM98, RTG98, Roc71], which allows the user to indicate the positive and negative relevance of the objects retrieved. These indications are used by the system to adjust the query clues better to the user's query intention. Such a mechanism connects the visualization tool to the query specification tool and creates an interactive loop. The hypothesis is that when the user terminates the loop he or she will have found the media objects in the collection with a best match to the query intention.

In every system part the original media objects play a role. These media objects can be either stored directly in the database, or reside on a different storage medium, *e. g.* the file servers of the WWW. The information exchanged between the various subsystems will seldom contain the raw media objects. Instead database keys, filenames or *Uniform Resource Identifiers* (URIs) [BLFIM98] are passed along.

The information about the collection of media objects is produced by the annotation subsystem. Part of this system handles the interaction with the librarian, *i. e.* the person in the right part of Figure 1.1. This librarian uses his domain knowledge and standard conventions, *e. g.* in the vain of the traditional *Anglo-American Cataloguing Rules* (AACR2R) [GW98], to annotate the media objects. These annotations range from content-independent [DCM01], *e. g.* this image was added to the collection at July 1, 1998, to content-descriptive data [ISO01], *e. g.* this image is a portrait of Chet Baker [Gro94]. Apart from a manual part the annotation system also has an

automatic part. In the automatic part the system uses algorithms and additional information sources, like an ontology or a thesaurus, to automatically extract additional information. Interaction between the two parts may be used to complete and verify the annotation, *e. g.* automatic extracted concepts may be approved by the librarian.

The database functions as a persistent buffer between the off line produced annotation information and the on line use of this information to answer queries. This database is managed by a *Database Management System (DBMS)*. A DBMS offers not only persistent storage of the data, but also other functionality needed by a DMW. For example, to assure a consistent representation and to control mixed access, but also, one of the major research themes in database technology, query optimization. The search engine will profit from the last one in its search for matching media objects.

As the main focus of this thesis lies within the idea of automatic annotation extraction the coming section will further describe the role of this subsystem.

## 1.2 Annotations

As discussed in the walk through and shown in Figure 1.1 the annotation information is produced manually and/or automatically extracted. However, with the increasing size of media collections manual annotation of all media objects becomes unfeasible. Likewise, when the collection is unconstrained, *i. e.* contains media objects from various domains, manual annotation of the objects can never meet all possible query intentions. Even for domain and size restricted collections manual annotation remains hard, due to the fact that annotations tend to be subjective, *i. e.* they describe the personal perception of the librarian. These aspects increase the importance of the automatic part of the annotation subsystem.

### 1.2.1 The Semantic Gap

The holy grail for automatic annotation is to take over the content-descriptive part of the manual burden. To realize this, the *semantic gap* between raw sensor data and “real world” concepts has to be bridged. For visual data this gap is defined as follows [SWS<sup>+</sup>00]:

*The semantic gap is the lack of coincidence between the information that one can extract from the visual data and the interpretation that the same data have for a user in a given situation.*

This definition may be generalized to raw sensor data in general without loss of validity.

The semantic gap is visualized in Figure 1.2. The user with all his/her general knowledge will have many associations with this photo. These associations range from generic to specific ones, *e. g.* from “this is a portrait” to “this is a portrait of

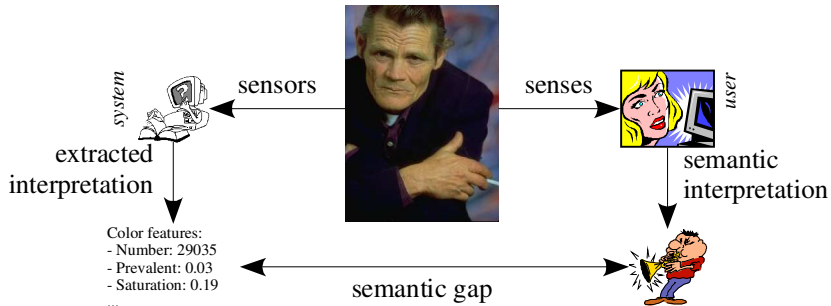


Figure 1.2: The semantic gap visualized

the jazz trumpeter Chet Baker”. Ideally, in the case where there is no semantic gap, the computer system can extract the same information from a digital version of this photo. Algorithms to classify this image as a photo and to detect the frontal face are available. Combining this basic information the validity of the generic semantic concept portrait can be induced. The validity of more specific concepts often depends on the availability of more contextual knowledge about the media object.

However, the semantic gap is still not filled and may never be. One of the reasons is the role of ambiguity. The more abstract a concept becomes the more subjective, due to *e. g.* cultural context-sensitivity, interpretations are possible. In [Eak96] the authors distinguish three image content levels:

**level 1** primitive features: color, texture, shape;

**level 2** derived (or logical) features: contains objects of a given type or contains individual objects;

**level 3** abstract attributes: named events or types of activities, or emotional or religious significance.

The higher the level the more subjective, and thus ambiguous, annotations become. State of the art annotation extraction algorithms reach level 2. Level 3 algorithms are only possible for clearly defined and distinguishable (narrow) domains. To provide enough support for an attack on the third level the annotation subsystem will need specialized constructs to handle this ambiguity, *e. g.* using probabilistic reasoning.

## 1.2.2 Annotation Extraction Algorithms

The predominant approach to try and bridge the semantic gap is the translation of the raw data into low-level features, which are subsequently mapped into high-level, *i. e.* semantic meaningful, concepts. This approach is reflected in frameworks like

ADMIRE [Vel98] and COBRA [PJ00] and the *compositional semantics* method used in [CDP99].

Low-level features (level 1) are directly extracted from the raw media data and relate to one or more feature domains embedded in the specific media type [Del99]. For images color, texture and shape features are well known examples. The choice of domains gets even bigger when several media types are combined into one multimedia object, *e. g.* a video which may be seen as a, time related, sequence of images with an audio track.

Rules, which may be implicit, map these low-level features into semantic concepts (level 2 and 3). An expert may hard-code these rules, *e. g.* a combination of boolean predicates, or they may be learned by a machine learning algorithm [Mit97]. Such an algorithm may result in human readable rules, as is the case with decision rules [Qui93], or the rules may be hidden inside a blackbox, *e. g.* in the case of a neural network [Fau94].

In fact there is a wealth of research on extraction algorithms for both features and concepts. When a subset of them are used to annotate a collection of media objects they depend on each other to create a coherent annotation.

### 1.2.3 Annotation Extraction Dependencies

Annotations of the example image of Chet Baker may be extracted by using these mappings (illustrated in Figure 1.3):

1. the image is classified as a photo: feature values, *e. g.* the number of colors and the saturation of these colors, are used in a boolean rule, which determines if the image is a photo or not [ASF97];
2. the photo contains a human face: the group of skin colors in the *c1c2c3* color space, are used to find skin areas and these areas form the input to a neural network which determines the presence of a human face in the photo [GAS00].

This example shows that concepts do not only depend on features, they may also depend on each other. In this example the face detection presupposes that the image is classified as a photo. This is a different kind of dependency. The dependency between feature and concept extraction is based on a direct output/input relation: the output of the feature detector is input for the photo decision rule. This type of dependencies is called an *output/input dependency*. However, the dependency between the two concepts is based on context: the photo concept functions as a contextual filter for the face concept.

This *context dependency* can be hardcoded as an output/input dependency. Unfortunately this will harm the generality of the face detector: it can not be reused in a different context, where there is no photo pre-filter. Context dependency is a design decision or domain restriction and is not enforced by the extraction algorithm. In this specific case the decision to use the photo classifier as a pre-filter is made because



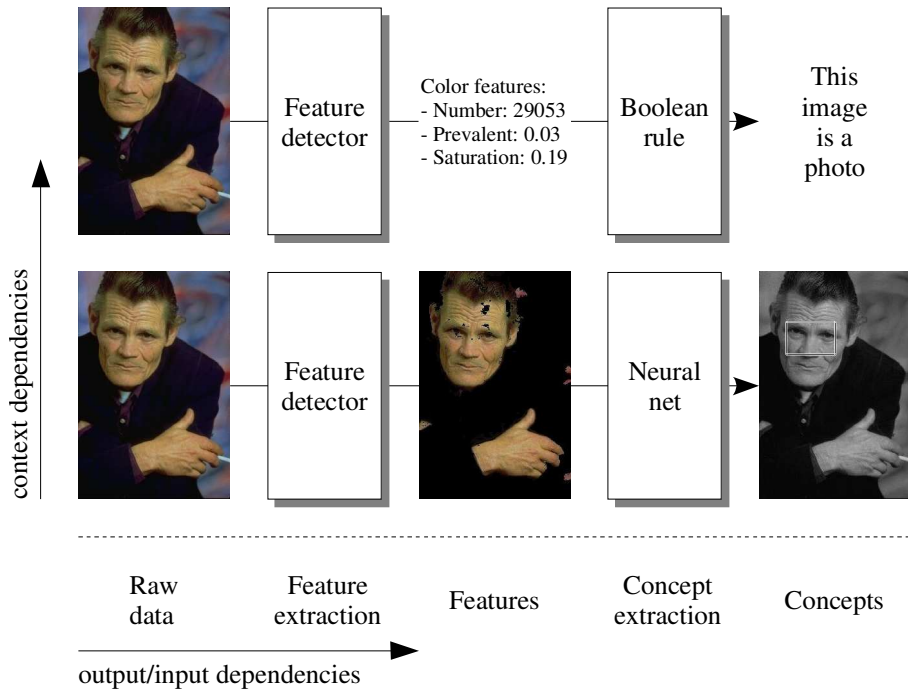


Figure 1.3: Automatic information extraction steps

the face detector is expensive, while the photo classifier is cheap. By using the photo classifier as a pre-filter only images with a high chance on the presence of a face will be passed on to the expensive face detector. Due to the explicit handling of this context dependency the face detector stays generic in nature and is able to be reused in a different context, *e. g.* black and white images.

The subsystem which controls the automatic information extraction has to take care of these dependencies and use them to call the algorithms, evaluate the rules and run the machine learning algorithms to produce the features and concepts to be stored in the database.

#### 1.2.4 Annotation Maintenance

Complicating the task of the annotation subsystem further, supporting multimedia information retrieval in a non-static environment, like the WWW, involves the maintenance of the annotations, features and concepts, stored in the database so they reflect the current status in this evolving environment.

There are several possible sources of change leading to the need of annotation

maintenance. Assuming that the media objects are not stored in the database, only the annotations are, the first source is an *external* one. The media objects themselves may be modified. Upon each modification the automatic (and manual) annotation has to be redone to guarantee that the database contains the correct and up-to-date data. Two other sources can be seen as *internal* to the system: changes in the extraction algorithms and in the dependencies between them. If an algorithm is improved (or a bug is fixed), the specific features or concepts have to be updated. Due to the output/input and context dependencies between features and concepts this change may trigger the need for reruns of many other extraction algorithms. Finally, the output/input and context dependencies may change. The addition or removal of a context dependency may, again, trigger the need for reruns of extraction algorithms.

When the dependencies and algorithms are embedded in a, hand crafted, special purpose program there is basically one option: rework the program and do a complete rerun of the annotation process for the affected multimedia objects. However, when (at least) the dependencies are described in a declarative manner, a supervisor program can take care of the maintenance process. Such a supervisor analyzes the dependencies and reruns only the extraction algorithms which are affected by the change. In this way a complete rerun, including unnecessary reruns of expensive algorithms, is prevented and the database is maintained incrementally.

## 1.3 The Acoi System

Although incremental maintenance of multimedia annotations has been identified as a key research topic [SK98], there has been little actual research to solve this problem and no satisfactory solution exists yet. This thesis describes the Acoi system architecture and its reference implementation, which provides a sound framework for the automatic part of the annotation subsystem, including incremental maintenance.

### 1.3.1 A Grammar-based Approach

Formal language theory forms the foundation of this framework. Its choice was based on the observation that proper management of annotations all involve context:

**the semantic gap** the more specific a concept, the more structural contextual knowledge is needed for validation (see Section 1.2.1);

**disambiguation** the more abstract a concept, the more user specific contextual knowledge is needed to disambiguate it (see Section 1.2.1);

**contextual dependency** to promote reuse of detectors, context dependencies should be explicitly handled (see Section 1.2.3);

**incremental maintenance** exact knowledge of the origins, *i. e.* the context, of an annotation is needed to localize the impact of internal or external changes and thus enable *incremental* maintenance (see Section 1.2.4).

The Acoi system would thus benefit from a dependency description or processing model which covers context knowledge for both annotations and extraction algorithms. Traversing the dependency description a path from the start of the extraction process to the actual extraction of a specific annotation can be maintained. A set of annotation paths can easily be described by a tree. Sets of valid trees, *i. e.* valid annotation paths, are naturally modeled by grammars. Grammars form a context preserving basis for a dependency description. However, the context descriptions should be underspecified enough to keep algorithms generic and enable, and even promote, reuse. The theoretical and practical implications of this intuition is investigated in this thesis.

### 1.3.2 System Architecture

Detailed descriptions of the Acoi system components, shown in Figure 1.4, and their relationships form the core of the thesis.

Chapter 2 starts with a description of the Acoi system foundation: the *feature grammar systems*. This foundation is based on a careful embedding of extraction algorithms into formal language theory and to formally describe both types of information extraction dependencies.

The next chapter introduces a non-mathematical notation for feature grammar systems: the *feature grammar language*. This language supports the core of a feature grammar system. Based on earlier experience extensions are added to conveniently support the various forms of feature and concept extraction.

In Chapter 4, the *Feature Detector Engine* (FDE) uses the execution semantics of feature grammar systems to produce the annotations. This involves traversing the dependencies described and execution of its associated extraction algorithms. The core is supplied by a practical algorithm taken from natural language research and compiler technology and adapted to handle the specific needs of feature grammar systems.

The impact of the system on the database is discussed in Chapter 5. The engine delivers its data, *i. e.* annotations and their complete context, in a format related to the semantics of the feature grammar language. This format is generic and can be mapped to the requirements of any DBMS. In this chapter a DBMS specific mapping for the Monet back-end and related optimization issues are discussed.

The *Feature Detector Scheduler* (FDS), described in Chapter 6, analyzes the dependencies, *i. e.* the possible contexts, in a specific feature grammar to localize the effect of changes in source data, algorithms or dependencies. When the parts affected are identified, the scheduler triggers incremental maintenance runs of the engine, which result in the propagation of changes to the database.

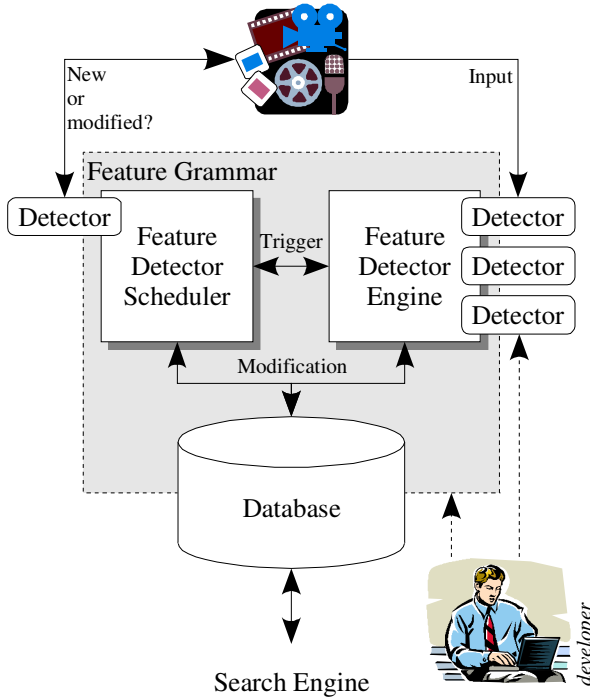


Figure 1.4: Acoi system architecture

### 1.3.3 Case Studies

Various real world applications have been used to identify and evaluate functional, performance and capacity requirements for the Acoi system architecture. The case studies will be entirely exposed in Chapter 7. But throughout the thesis they will also, just like the reference implementation, function as running examples to illustrate how specific requirements are met by the system architecture. Therefor the succeeding subsections will shortly introduce the case studies.

#### 1.3.3.1 The WWW Multimedia Search Engine

The WWW is probably the largest unconstrained collection of multimedia objects available. Search engines have extracted text-based entry points to function as road-signs to browse this vast collection. With the growing popularity of the web the search and retrieval of other media types is getting more attention, *e. g.* both AltaVista [Alt01] and Google [Goo01] are offering some support for retrieval of multimedia objects. However, this support is still based on text, *e. g.* keywords extracted from either the

URL of the object or from the web page it appears on. Content- or concept-based retrieval play only a significant role in research prototypes, like WebSeer [FSA96], WebSEEk [SC96] and ImageScape [Lew00]. These prototypes allow the retrieval of images on the basis of a limited set of, hardwired, concepts, *e. g.* faces or landscape elements.

The Acoi system architecture is used to build and maintain a multimedia search engine's index. With advances in computer vision and sensor informatics the number of automatic extractable features and concepts will gradually increase. Due to the system's ability to maintain its index incrementally (prototypes of) new features or concept extraction algorithms are easily added. This ability also makes it well suited to adapt to the dynamic behavior of the Internet, *i. e.* the index is continually updated instead of completely recreated.

The basis is a simple model of the web: web objects and their links. This model is then evolutionary enhanced with content-based feature and concept extraction algorithms.

### 1.3.3.2 The Australian Open Search Engine

This Australian Open case study also involves the maintenance of a search engine's index. But in this case the domain is restricted to the Australian Open tennis tournament. In the WWW case study the model contains multimedia objects and generic relations. This limited model makes it possible to extract only very generic features and concepts, *e. g.* this video contains 25 shots. However, in this case study the system also contains conceptual information and, combined with domain knowledge, more specific feature and concept extraction can be realized, *e. g.* this video of a tennis match between Monica Seles and Jennifer Capriati contains 25 shots of which 20 show the tennis court.

The prime benefit of the Australian Open case study is to test the flexibility and open character of the system architecture. The Acoi system is embedded in a larger application and has to interact with separate systems, which handle the conceptual data or function as distributed extraction algorithms.

### 1.3.3.3 Rijksmuseum Presentation Generation

The Rijksmuseum in Amsterdam, like many other museums, makes part of its collection available in digital format<sup>1</sup>. This gives the public alternative and interactive ways to browse the collection. The database underlying this interactive system contains manual annotations of the museum pieces.

The underlying database is semistructured in nature, *i. e.* the annotation is not always complete. The Acoi system is used, in this case, as a style database. If the annotator did not specify the style period of a painting the system tries to infer the

---

<sup>1</sup>[www.rijksmuseum.nl](http://www.rijksmuseum.nl)

correct style using the dependency description and associated extractors. The thus automatically augmented annotation may help in several ways. It may help the annotator in completing the annotation by providing useful hints. Furthermore, it may allow the museum visitor to retrieve possible matches.

The features and concepts extracted may also be used to influence and optimize the layout of the hypermedia presentation generated to browse a query result.

## **1.4 Discussion**

This introductory chapter surveyed the domain of digital media warehouses. A number of research challenges exist within this domain and are the focus of attention for a multidisciplinary research community. The research described in this thesis is dedicated to the problem of automatic extraction and (incremental) maintenance of multimedia annotations. To retain enough contextual knowledge a grammar-based approach is taken, which grounds the approach in a well-studied field of computer science. The subsequent chapters start with laying the formal basis and work towards a practical solution to the problem. Chapter 7 will showcase the solution in the form of the evaluation of several case studies in the problem domain, and may thus be of main interest to practical oriented readers.

## Chapter 2

# Feature Grammar Systems

*A wise man once said  
that everything could be explained with mathematics  
He has denied  
His feminine side  
Now where is the wisdom in that?*

Marillion – *This is the 21<sup>st</sup> century*

The core of incremental maintenance of the annotation index lies in the understanding and analysis of the dependency description. In this chapter the dependency descriptions used by the Acoi system architecture, called feature grammar systems, are introduced.

As the name suggests feature grammar systems are related to grammars known from natural languages, *e. g.* English or Dutch. Sentences in these languages are constructed from basic building blocks: words. Each language has a set of rules which describe how words can be put in a valid order, and what the semantic meaning of this order is. These rules form the grammar of the language, *e. g.* like the following grammar rules for a subset of the English language.

### Example 2.1.

$$S \rightarrow NP VP$$
$$NP \rightarrow John$$
$$NP \rightarrow Mary$$
$$VP \rightarrow V_i$$
$$VP \rightarrow V_i NP$$

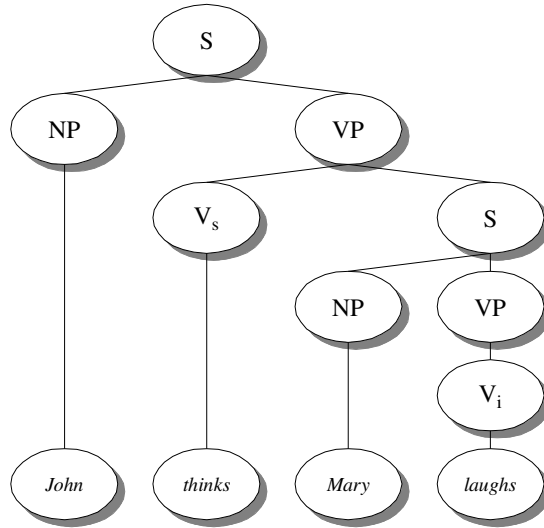


Figure 2.1: A parse tree

$$VP \rightarrow V_s S$$

$$V_i \rightarrow \textit{laughs}$$

$$V_t \rightarrow \textit{loves}$$

$$V_s \rightarrow \textit{thinks}$$

□

Sentences can now be tested for membership of the language induced by these specific grammar rules. A valid sentence  $S$  consists of two parts: a noun phrase  $NP$  and a verb phrase  $VP$ . Each of these phrases may (again) consist of other parts, *e. g.* a specific verb type. This can be repeated until the individual words of the sentence are reached. The result of such a process, also known as *parsing*, is the *parse tree* of a sentence. Figure 2.1 shows the parse tree for this sentence: *John thinks Mary laughs*. Notice that the complete parsing context is captured by this tree.

The fundamental idea behind feature grammar systems is that the same process of deriving a parse tree, and thus the underlying formal theories and practices, can be used as a driving force to produce the annotation of a multimedia object. To start once more with the basis: annotation items are seen as the words. Sentences formed by combinations of these words should be valid in a feature grammar system, *i. e.* a specialized dependency description.



As stated in the previous chapter two kinds of dependencies should be captured in these descriptions: output/input and contextual dependencies. Looking at the structure of the grammar rules it is easy to see that the right-hand sides of the rules capture contextual dependencies: in the context of a specific verb phrase  $VP$  a verb  $V_i$  should always be followed by a noun phrase  $NP$ . Output/input dependencies are directly related to annotation extraction algorithms. But those are not found in the grammar rules. The addition of feature grammar systems is that these algorithms are bound to specific symbols in the grammar. Upon encountering such a special symbol during the derivation process the output/input dependencies can be resolved by using the contexts stored in the gradually build parse tree. In fact the output/input dependencies are associated with the context of the left-hand side of a rule.

The first part of this chapter is devoted to embedding the additions of feature grammar systems into results of formal language theory. To support the understanding of this embedding the next section shortly recalls some relevant grammar theory. Readers which are familiar with grammar theory, *i. e.* the Chomsky hierarchy, the (regulated) rewriting process and grammar systems, may skip to Section 2.2.

A feature grammar itself is a valid sentence in a meta language: the *feature grammar language*. The next chapter describes how the expressive power of feature grammar systems is captured by the feature grammar language. Using this theoretical basis Chapters 4 and 6 will describe appropriate adaptations of formal language technologies used by the annotation subsystem to maintain the database. This database, as will be described in Chapter 5, stores a collection of the annotation sentences, *i. e.* a subset of all possible sentences in the language induced by the feature grammar, along with their parse trees.

## 2.1 A Grammar Primer

Grammars are a key concept in computer science and many theoretical and practical issues related to them have been studied extensively. The theoretical implications of grammars are studied in the field of formal language theory (see [Lin97, HMU01] for an introduction) and form the basis of this chapter. Parsing algorithms (see [GJ98] for an overview), *i. e.* how to efficiently determine if a sentence is a valid member of a language, is one of the more practical issues and will play an important role in Chapter 4.

### 2.1.1 A Formal Specification of Languages and Grammars

The formal specification starts with a language  $L$ .  $L$  consists of sentences constructed by concatenation of symbols from the alphabet  $\Sigma$ , *i. e.*  $L \subseteq \Sigma^*$ . A grammar  $G$  describes the language  $L(G)$  and is defined as follows.

**Definition 2.1.** A grammar  $G$  is defined as a quadruple  $G = (N, T, P, S)$ , where

1.  $N$  is a, non-empty, finite set of symbols called non-terminals or variables,
2.  $T$  is a, possibly empty, finite set of symbols called terminals, i. e.  $T \subseteq \Sigma$ ,
3.  $N \cap T = \emptyset$ ,
4.  $V = N \cup T$ ,
5.  $P$  is a finite set of rules of the form  $(L \rightarrow R)$ , called productions, such that
  - (a)  $L \in V^+$  is the left-hand side (LHS) of a production and
  - (b)  $R \in V^*$  is the right-hand side (RHS) of a production, and
6.  $S \in N$  is a special symbol called the start variable or axiom.

□

A production rule in  $P$  where the RHS is an empty string is written as:  $(L \rightarrow \lambda)$ , i. e.  $\lambda$  represents the empty string. Such a production rule is also called an erasing production.

### 2.1.1.1 The Chomsky Hierarchy

Grammars which apply to Definition 2.1 are called *recursively enumerable* (RE) or Type 0 grammars. These grammars permit the description of a large set of languages<sup>1</sup>, however, they are also unmanageable, e. g. there is no general efficient parsing algorithm. This problem led to a key work in formal language theory: the Chomsky hierarchy of grammars [Cho59]. In this hierarchy grammars are organized on the basis of their expressive power. The hierarchy starts with phrase structure grammars, and on each subsequent level restrictions are added. The restrictions result in gradually easier to “understand” or to parse grammars, but these grammars become also gradually less expressive. The following other grammar types belong to the Chomsky hierarchy [GJ98].

**Context-sensitive (CS) or Type 1 grammars** A grammar is context-sensitive if each production rule is context-sensitive. A rule is context-sensitive if actually only one (non-terminal) symbol in its LHS gets replaced by other symbols, while the others are found back undamaged and in the same order in the RHS. This rule is for example context-sensitive, where  $L$  is the left and  $R$  is the right context of  $S$ :

$$L S R \rightarrow L W R$$

---

<sup>1</sup>[Lin97] contains a proof that there are languages which are not in  $\mathcal{L}(RE)$ .

**Context-free (CF) or Type 2 grammars** A context-free grammar is like a context-sensitive grammar, except that it may contain only rules that have a single non-terminal in their LHS, *i. e.* there are no left and right contexts as shown in this rule:

$$S \rightarrow X Y Z$$

**Regular (REG) or Type 3 grammars** A regular grammar contains two kinds of production rules: (1) a non-terminal produces zero or more terminals and (2) a non-terminal produces zero or more terminals followed by one non-terminal. Examples for both kinds of rules are shown here:

$$S \rightarrow x y z$$

$$S \rightarrow v W$$

Due to the specific forms of rules in a REG grammar a more compact notation, a regular expression, is often used. Next to the alphabet  $T$  the notation supports: parentheses and the operators union (+), concatenation ( $\cdot$ ) and star-closure (\*). For example, the expression  $((a + b \cdot c)^*)$  stands for the star-closure of  $\{a\} \cup \{bc\}$ , that is, the language  $\{\lambda, a, bc, aa, abc, bca, bcbc, aaa, aabc, \dots\}$ . More extended regular expression languages, and thus more compact, are used in practice [Fri02]. As a convenience for later on the *period* operator ( $\cdot$ ) is already introduced. This operator matches any symbol of the alphabet, *i. e.*  $(\cdot a \cdot)$  describes the language where the second symbol is always  $a$ , while the first and third symbol may be any symbol (including  $a$ ).

### 2.1.1.2 Mild Context-sensitivity

Just like RE grammars, CS grammars turned out to be impractical. Although a generic parsing algorithm can be defined, *i. e.* in the construction of a linear bounded automaton (LBA) [Lin97, RS97a], specifying a CS grammar remains a non-trivial task, even for a small language, resulting in incomprehensible grammars (see for an example [GJ98]). For this reason most practical attention has gone to CF and REG grammars. However, it was early discovered that “the world is not context-free”, *i. e.* there are many circumstances where naturally non-CF languages appear. Linguists seem to agree [Man94] that “all” natural languages contain constructions which cannot be described by CF grammars. Three of such non-CF constructions are [RS97b]:

1. *reduplication*, leading to languages of the form  $\{xx|x \in V^*\}$ ;
2. *multiple agreements*, modeled by languages of the form  $\{a^n b^n c^n | n \geq 1\}$ ,  $\{a^n b^n c^n d^n | n \geq 1\}$ , etc.;

3. *crossed agreements*, as modeled by  $\{a^n b^m c^n d^m \mid n, m \geq 1\}$ .

Artificial languages, *e. g.* a programming language like *Algol 60*, have also non-CF properties [Flo62]. Seven examples of non-CF areas where languages are found are described in Section 0.4 of [DP89], and the section also concludes with the remark: “the world seems to be non-context-free ...”. The same book describes 25 different mechanisms for regulated rewriting. Using such a mechanism a “mild” subfamily of CS languages is created. A mild CS language has as many CF-like properties as possible, but is able to cover the required non-CF constructions.

Before some mechanisms of regulated rewriting are introduced the derivation process (also known as rewriting) for CF and REG grammars is formalized.

## 2.1.2 The Derivation Process

If the sentence  $w \in L(G)$ , then the following derivation exists:

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n \Rightarrow w$$

The strings  $S, w_1, w_2, \dots, w_n$ , which may contain variables as well as terminals, are called *sentential forms* of the derivation.

This application of the direct derivation can be defined as follows:

**Definition 2.2.** *The application of the direct derivation  $x_1 w x_2 \Rightarrow x_1 z x_2$  is allowed iff  $(w \rightarrow z) \in P$ , where  $P$  is the set of productions from grammar  $G$ .*

□

By using a subscript to the direct derivation ( $\Rightarrow$ ) a specific grammar or a (labeled) production rule may be indicated, *e. g.*  $\Rightarrow_G$ .

Using the transitive closure of the direct derivation the set of sentences in the language  $L$  can be defined:

$$L(G) = \{w \in T^* \mid S \xRightarrow{*} w\}$$

This set of sentences may be further limited by specific forms of derivation. A practical form, which will be encountered later on, is *leftmost* derivation. In this case each rule used in the derivation process rewrites the leftmost non-terminal in the current sentential form, *i. e.*  $x_1 \in T^*$ . This specific mode of derivation is indicated as follows:

$$L(G) = \{w \in T^* \mid S \xRightarrow[*]{lm} w\}$$

### 2.1.2.1 Bidirectional Grammars

Until now grammars are used in generating mode, *i. e.* by starting with the start symbol a specific sentence is gradually constructed. Using this process all sentences belonging to the language can be enumerated, hence all grammars are RE grammars. Grammars can also be used in accepting mode. In this mode a specific sentence is checked for membership of the language. This is done by flipping the left- and right-hand sides of the production rules. This derivation process then describes the acceptance of the sentence  $w$ :  $w \xRightarrow{*} S$ .

The mode of the grammar is indicated by  $G^{gen}$  or  $G^{acc}$  for respectively generative and accepting mode. When no mode is indicated ( $G$ ) the generative mode is used. A specific grammar may be usable in both modes and is then called *bidirectional* [App87, Neu91].

Notice that membership of a sentence can be resolved in either accepting or generating mode. In the latter case one enumerates all sentences until the sentence is found (or not), although this process may be endless. The optimization of this search process is the main target of parsing algorithms, which will be discussed in Chapter 4.

### 2.1.2.2 Parse Trees

By keeping track of the derivation steps a parse tree of the sentence, like the one in Figure 2.1, can be build. The parse trees for  $G$  are trees with the subsequent conditions [HMU01]:

1. each interior node is labeled by a non-terminal in  $N$ ;
2. each leaf is labeled by either a non-terminal, a terminal, or  $\lambda$ , however, if the leaf is labeled  $\lambda$ , then it must be the only child of its parent;
3. if an interior node is labeled  $A$ , and its children are labeled

$$X_1, X_2, \dots, X_k$$

respectively, from the left, then  $A \rightarrow X_1 X_2 \dots X_k$  is a production in  $P$ . Note that the only time one of the  $X$ 's can be  $\lambda$  is if that is the label of the only child, and  $(A \rightarrow \lambda)$  is a production of  $G$ .

### 2.1.2.3 The Delta Operation

Each sentential form  $w_i$  can now be associated with a parse tree  $t_i$ . The yield of this tree is defined as the corresponding sentential form, *i. e.*  $yield(t_i) = w_i$ . A tree can also be described by a set of paths, *i. e.* the result of  $path(t_i)$ . The in the beginning of the chapter shown parse tree (see Figure 2.1) contains, for example, the

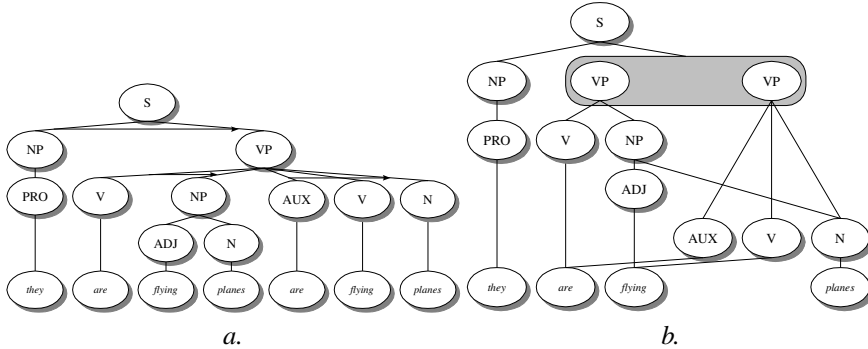


Figure 2.2: (a) An AND/OR-graph and (b) a packed shared forest

path  $S_1 \cdot VP_2 \cdot V_1 \cdot \text{thinks}_1$ . The subscripts indicate the order of the nodes among its siblings in the tree<sup>2</sup>.

The *path* and *yield* operations provide a simple relationship between REG and CF languages [Eng02]. For a language  $L$ , let the *delta* of  $L$ , denoted by  $\delta(L)$ , be the language of all yields of trees that have all their paths in  $L$ :

$$\delta(L) = \{\text{yield}(t) \mid \text{path}(t) \subseteq L\}$$

The relationship mentioned above is that the CF languages are exactly the deltas of the REG languages:  $\mathcal{L}(\text{CF}) = \{\delta(L) \mid L \in \mathcal{L}(\text{REG})\}$ .

#### 2.1.2.4 Parse Forests

Definition 2.1 allows multiple production rules to have the same symbol as their LHS. The RHSs of these rules are considered alternatives of each other. Grammars containing alternatives are called non-deterministic. If several of these alternatives lead to a valid parse tree for the input sentence, the grammar is also ambiguous. In these cases the sentence is not described by one parse tree, but by several, *i. e.* a parse forest. Such a forest can be represented by an AND/OR-graph [Nil98, Hal73]. In these graphs conjunctions of nodes are connected by an arc, see Figure 2.2.a. In this example the last three words of the input sentence can be explained by two (not shown in the simple example grammar for the English language) alternative production rules for the verb phrase  $VP$ . In one parse tree *flying* is an adjective to the noun *planes*, in the other structure *flying* is interpreted as a verb.

A packed shared forest [Tom86], see Figure 2.2.b, aims at structure sharing. Parsing algorithms (to be discussed in more detail in Chapter 4) do easily allow *sharing* of substructures by a well known dynamic programming concept: memoization [Mic68]. The *noun* node  $N$  and most of the terminal nodes are shared this way. This can be

<sup>2</sup>For brevity's sake the order information in paths will be omitted most of the time.

seen as sharing the bottom structures of the parse trees. *Packing* is targeted at sharing the upper structures. A *packed node* contains *sub-nodes* which have common leaf nodes and are labeled with the same non-terminal symbol. In the example structure the two alternatives for the verb phrase are packed together.

Packed shared forests are aimed at parse structures related to CF grammars. This grammar type only allows *local* ambiguity. The combination of ambiguity and the long distance dependencies of CS grammars calls for methods to indicate the global context and scope of a node. One method to achieve this is by naming the disjunctions and to annotate nodes in the same context with this name, see [DE90, Bla97, Bla98].

### 2.1.2.5 Disambiguation

In fact ambiguity may appear on many different levels in an application. In *natural language processing* (NLP) ambiguity can be found at these levels:

1. *lexical ambiguity*: a word has more than one meaning;
2. *syntactic or structural ambiguity*: a sentence has two or more parses as in Figure 2.2;
3. *semantic ambiguity*: may directly follow from syntactic ambiguity or from the semantic context.

The NLP community has introduced a vast amount of models and algorithms to *disambiguate* these ambiguities. For an overview see [JM00].

On the syntactic level disambiguation may be done by the use of *probabilistic parsing* [Boo69, Sol69]. In this case each alternative is assigned a probability [Bak79] and the parsing algorithm [Jel69] chooses the most probable parse tree. Another method is to ask for human interaction. Tomita describes a system where manual disambiguation is build into the parser [Tom86]. However, it is also possible to postpone disambiguation to a higher automatic or manual level. In which case the parser will have to deliver the complete ambiguous structure. For example, in [KV94, vdBKMOV03] the authors describe disambiguation filters, based on term rewriting, to prune a parse forest.

### 2.1.3 Regulated Rewriting

By regulating the applicability of a direct derivation step the set of valid sentences of a language can be decreased, while the basic rules keep close to their CF equivalents. In the monograph [DP89] the authors describe 25 regulation mechanism, *e. g.* matrix, programmed and random context grammars. In this section two mechanisms, which are closely related to the one applied by feature grammar systems, are described in detail.

### 2.1.3.1 Conditional Grammars

While other mechanisms work by restricting or explicitly stating the order in which productions may be applied the mechanism of *conditional* (C) grammars is based on conditions on the contextual sentential forms.

**Definition 2.3.** In a conditional grammar  $G = (N, T, P, S)$  the productions  $P$  are of the form  $(w \rightarrow z, Q)$  where

- $Q$  is a REG language over the alphabet  $V$ ,
- $N, T$  and  $S$  have their normal meaning.

The rule  $(w \rightarrow z, Q)$  is applicable to  $x = x_1wx_2$  yielding  $y = x_1zx_2$ , i. e.  $x \Rightarrow y$ , iff  $x \in Q$ . □

#### Example 2.2.

Take for example the following C grammar:

$$G = (\{S, S'\}, \{a\}, \{p_1, p_2, p_3\}, S)$$

with

$$\begin{aligned} p_1 &= (S \rightarrow S'S', (S')^*S^+) \\ p_2 &= (S' \rightarrow S, S^*(S')^+) \\ p_3 &= (S \rightarrow a, a^*S^+) \end{aligned}$$

[RS97b] shows that this grammar describes a known non-CF language:

$$L(G) = \{a^{2^n} \mid n \geq 0\}$$
□

In both [DP89] and [RS97b] it is shown that  $\mathcal{L}(C, CF - \lambda) = \mathcal{L}(CS)$ , i. e. C grammars with CF rules but no erasing productions are equivalent to CS grammars, and also that  $\mathcal{L}(C, CF) = \mathcal{L}(RE)$ . In [RS97b] this is done by giving rules for transforming a C grammar into a CS grammar, and vice versa.

### 2.1.3.2 Tree-controlled Grammars

Another regulated rewriting mechanism uses the parse tree to restrict applications of the derivation step. Those grammars, called *tree-controlled* (TC) grammars, are defined as follows [CM77, DP89]:



**Definition 2.4.** A tree-controlled grammar is a construct  $G = (N, T, P, S, R)$  where

- $G' = (N, T, P, S)$  is a CF grammar and
- $R \subseteq V^*$  is regular.

$L(G)$  consists of all words  $w$  generated by the underlying grammar  $G'$  such that there is a parse tree of  $w$  such that each word obtained by concatenating all symbols at any level (except the last one) from left to right is in  $R$ . □

All nodes of the parse tree with the same distance to the root of the tree are on the same level of the derivation tree.

**Example 2.3.**

Consider the following grammar:

$$G = (\{S, A, B, C\}, \{a, b, c\}, \{P_1, \dots, P_7\}, S, R)$$

with

$$P_1 = (S \rightarrow ABC)$$

$$P_2 = (A \rightarrow aA)$$

$$P_3 = (A \rightarrow a)$$

$$P_4 = (B \rightarrow bB)$$

$$P_5 = (B \rightarrow b)$$

$$P_6 = (C \rightarrow cC)$$

$$P_7 = (C \rightarrow c)$$

and

$$R = \{S, ABC, aAbBcC\}.$$

Evidently,

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}.$$

Which is a known CS language [CM77].

□

Also for this type of mildly CS languages it is shown in [DP89] that  $\mathcal{L}(\text{TC}, \text{CF} - \lambda) = \mathcal{L}(\text{CS})$ .

## 2.1.4 Grammar Systems

Until now only one grammar at a time is considered. However, grammars can cooperate in so called grammar systems [CVDKP94]. The theory of these systems was inspired by the will to model *multi-agent systems*. A common technique in the field of Artificial Intelligence (AI) is to structure those systems according to the blackboard architecture [Nil98]. In such a system various knowledge sources work together on solving a problem. The current state of the solution resides on the blackboard. A protocol of cooperation encodes the control over the knowledge sources.

In a grammar system the common sentential form is on the blackboard and the component grammars are the knowledge sources. Since its introduction in 1990 various forms of cooperation and control have been studied. Two basic classes are distinguished: *cooperating distributed* (CD) and *parallel communicating* (PC) grammar systems. In this section only CD grammar systems will be introduced, as they form the formal basis for feature grammar systems.

### 2.1.4.1 CD Grammar Systems

**Definition 2.5.** A CD grammar system is a  $(n + 2)$ -tuple

$$\Gamma = (T, G_1, G_2, \dots, G_n, S),$$

where,

1. for  $1 \leq i \leq n$ , each  $G_i = (N_i, T_i, P_i)$  is a (usual) CF grammar, called a component, with
  - (a) the set  $N_i$  of non-terminals,
  - (b) the set  $T_i$  of terminals,
  - (c)  $V_i = N_i \cup T_i$ ,
  - (d) the set  $P_i$  of CF rules, and
  - (e) without axiom,
2.  $T$  is a subset of  $\bigcup_{i=1}^n T_i$ ,
3.  $V = \bigcup_{i=1}^n V_i$ , and finally
4.  $S \in \bigcup_{i=1}^n N_i = N$ .

□

The components correspond to the knowledge sources solving the problem on the blackboard, where every rule represents a piece of knowledge which results in a possible change of the blackboard. The axiom, or start symbol,  $S$  corresponds with

the initial state of the problem on the blackboard. The alphabet  $T$  corresponds to knowledge pieces which are accepted as solutions or parts of solutions.

A derivation step in a grammar system is now defined as follows:

**Definition 2.6.** Let  $\Gamma$  be a CD grammar system as in Definition 2.5. Let  $x, y \in V_i^*$ . Then  $x \Rightarrow_{G_i}^k y$  is applicable iff there are words  $x_1, x_2, \dots, x_{k+1}$  such that:

1.  $x = x_1, y = x_{k+1}$ ,
2.  $x_j \Rightarrow_{G_i} x_{j+1}$ , i. e.  $x_j = x'_j A_j x''_j, x_{j+1} = x'_j w_j x''_j, (A_j \rightarrow w_j) \in P_i, 1 \leq j \leq k$ .

Moreover, this leads to the following other derivation modes:

- $x \Rightarrow_{G_i}^{<k} y$  iff  $x \Rightarrow_{G_i}^{k'} y$  for some  $k' < k$ ,
- $x \Rightarrow_{G_i}^{\geq k} y$  iff  $x \Rightarrow_{G_i}^{k'} y$  for some  $k' \geq k$ ,
- $x \Rightarrow_{G_i}^* y$  iff  $x \Rightarrow_{G_i}^k y$  for some  $k$ , and
- $x \Rightarrow_{G_i}^t y$  iff  $x \Rightarrow_{G_i}^* y$  and there is no  $z \neq y$  with  $y \Rightarrow_{G_i}^* z$ .

□

Any derivation  $x \Rightarrow_{G_i}^k y$  corresponds to  $k$  direct derivation steps in succession in the component grammar  $G_i$ . In a  $\leq k$ -derivation mode the component can perform at most  $k$  changes. The  $\geq k$ -mode requires the component to be competent enough to perform at least  $k$  steps. A component may work on the problem as long as it wants when the derivation is in  $*$ -mode. Finally, the  $t$ -mode corresponds to the case where the component should work on the problem as long as it can.

The language induced by a CD grammar system  $\Gamma$  is now defined as follows:

**Definition 2.7.** Let

$$f \in \{*, t, 1, 2, \dots, \leq 1, \leq 2, \dots, \geq 1, \geq 2, \dots\},$$

and let  $\Gamma$  be a CD grammar system. Then the language  $L_f(\Gamma)$  generated by  $\Gamma$  is defined as the set of all words  $z \in T^*$  for which there is a derivation

$$S = w_0 \Rightarrow_{G_{i_1}}^f w_1 \Rightarrow_{G_{i_2}}^f w_2 \Rightarrow_{G_{i_3}}^f \dots \Rightarrow_{G_{i_r}}^f w_r = z.$$

□

Finally, the choice of the “terminal” set of a CD grammar system may be restricted. A CD grammar system as specified in Definition 2.5 accepts in

$style(arb)$  iff  $T$  is an arbitrary subset of  $\bigcup_{i=1}^n T_i$ ,

$style(ex)$  iff  $T = \bigcup_{i=1}^n T_i$ ,

$style(all)$  iff  $T = \bigcap_{i=1}^n T_i$ ,

$style(one)$  iff  $T = T_i$  for some  $i, 1 \leq i \leq n$ .

Now  $(CD_nCF, f, A)$  denotes a class of CD grammar systems with at most  $n$  components working in the  $f$ -mode of derivation and accepting in style  $A$ , where CF denotes that CF component grammars are used.  $(CD_\infty CF, f, A)$  indicates a CD grammar system with an arbitrary number of components.

### 2.1.4.2 Internal Control

Just like with normal grammars regulation mechanisms can be added to restrict the application of derivation steps. This may take the form of either external or internal control. With external control a supervisor is added or a sequence of components is fixed in advance, *e.g.* by paths in a directed graph. Internal control makes use of conditions on the current state of the problem, *i.e.* the sentential form. These conditions can be used to either start or stop a component. As internal control is used by feature grammar systems only this form of control is further investigated.

**Definition 2.8.** A dynamically controlled CD grammar system  $\Gamma$  is a grammar system as in Definition 2.5 with  $G_i = (N_i, T_i, P_i, \pi_i, \rho_i)$  where

- $\pi_i$  is a start condition, and
- $\rho_i$  is a stop condition for component  $G_i$ .

Then the language  $L(\Gamma)$  generated by  $\Gamma$  is defined as the set of all words  $z \in T^*$  for which there is a derivation

$$S = w_0 \Rightarrow_{G_{i_1}}^* w_1 \Rightarrow_{G_{i_2}}^* w_2 \cdots \Rightarrow_{G_{i_r}}^* w_r = z$$

such that, for  $1 \leq j \leq r$ ,

$$\pi_{i_j}(w_{j-1}) = true \text{ and } \rho_{i_j}(w_j) = true$$

and for  $f \in \{t, 1, 2, \dots, \leq 1, \leq 2, \dots, \geq 1, \geq 2, \dots\}$ , the language  $L_f(\Gamma)$  generated by  $\Gamma$  in the  $f$ -mode as the set of all words  $z \in T^*$  such that there is a derivation

$$S = w_0 \Rightarrow_{G_{i_1}}^f w_1 \Rightarrow_{G_{i_2}}^f w_2 \cdots \Rightarrow_{P_{i_r}}^f w_r = z$$

such that, for  $1 \leq j \leq r$ ,  $\pi_{i_j}(w_{j-1}) = true$ .

□

Notice that when the derivation is not in \*-mode the stop condition  $\rho_{i_j}(w_j) = true$  is replaced by the stop condition which is naturally given by the  $f$ -mode.

Some special types of conditions have been studied for CD grammar systems. Condition  $\sigma$  may be of these types:

$type(a)$  iff  $\sigma(w) = true$  for all  $w \in V^*$ ,

$type(rc)$  iff there are two subsets  $R$  and  $Q$  of  $V$  and  $\sigma(w) = true$  iff  $w$  contains all letters of  $R$  and  $w$  contains no letter of  $Q$ ,

$type(K)$  iff there are two words  $x$  and  $x'$  over  $V$  and  $\sigma(w) = true$  iff  $x$  is a subword of  $w$  and  $x'$  is not a subword of  $w$ ,

$type(K')$  iff there are two finite subsets  $R$  and  $Q$  of  $V^*$  and  $\sigma(w) = true$  iff all words of  $R$  are subwords of  $w$  and no word of  $Q$  is a subword of  $w$ ,

$type(C)$  iff there is a regular set  $R$  over  $V$  and  $\sigma(w) = true$  iff  $w \in R$ .

Notice that  $type(C)$  corresponds with the conditions known from C grammars.

In the notation for grammar systems the  $f$  is now replaced by  $(X, Y)$ , where  $X$  indicates the start condition type and  $Y$  the same for the stop condition, when the grammar uses \*-mode. In the other derivation modes  $(X, f)$  is used, for example  $(CD_8CF, (rc, t), arb)$ .

Many more variants of CD grammar systems have been studied in the literature, including the use of bidirectional components [FH99], and the monograph [CVDKP94] discusses a lot of them. Some variants are combined to form the basis for the concept of feature grammar systems, and will be described during the formal definition of feature grammar systems in the next section.

## 2.2 Feature Grammar Systems

Lets, while gradually working toward a formal definition of feature grammar systems<sup>3</sup> using the building blocks from the previous section, return to the annotation example from Chapter 1. As stated in the introduction of this chapter the annotation items can be seen as words in a sentence, see Figure 2.3. A possible parse tree for this sentence is also shown in the same figure.

This parse tree is the result of a derivation process driven by the this grammar:

<sup>3</sup>In NLP the term feature grammars is sometimes used to indicate a type of grammar formalisms, e. g. HPSG [PS94] and LFG [Bre82], where next to a  $c$ -structure, i. e. a parse tree, also a  $f$ -structure is constructed. The  $f$ -structure contains (semantic) features which are propagated up in the tree using unification rules. Unification leads in the end to one combined  $f$ -structure for the root of the  $c$ -structure. Unfortunately this type of grammars was only encountered by the author when the concept of feature grammars, where the term feature refers to multimedia features, was already introduced to the multimedia and database research communities.

**Example 2.4.**

*Image* → *Location Color Class*

*Location* → *url*

*Color* → *Number Prevalent Saturation*

*Number* → *int*

*Prevalent* → *flt*

*Saturation* → *flt*

*Class* → *Graphic*

*Class* → *Photo Skin Faces*

*Graphic* → *bit*

*Photo* → *bit*

*Skin* → *bitmap*

*Faces* → *int*

*url* → *http://...*

*int* → *1*

*int* → *29053*

*flt* → *0.3*

*flt* → *0.19*

*bit* → *true*

*bitmap* → *00...*

□

In the upcoming sections this CF grammar will be extended until the full fledged power of a feature grammar system is reached.

**2.2.1 Detectors**

The dependency description evolves around the annotation extraction algorithms. Before introducing how feature grammar systems capture both output/input and contextual dependencies, annotation extraction algorithms are formally introduced into the grammar.

In a feature grammar these algorithms are bound to specific non-terminals, called (feature) detectors. In the example grammar the set of detectors is { *Color*, *Graphic*,

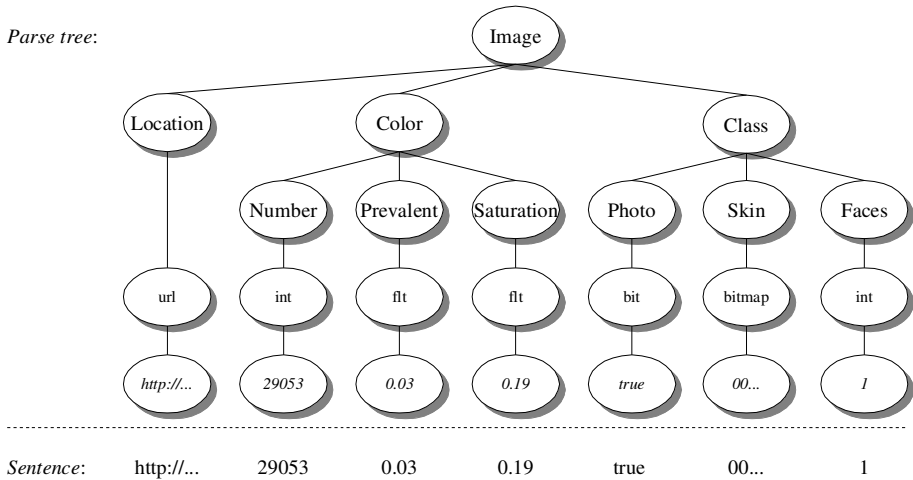


Figure 2.3: Annotation sentence

*Photo*, *Skin*, *Faces* }. Each of these detector algorithms can be seen as a function, which transforms its input sentence into an output sentence. The *Faces* detector, for example, transforms the sentence  $00\dots$  into the sentence  $1$ . The output sentence of each of these detectors is conveniently described by a set of CF grammar productions, *i. e.* each detector corresponds to a component in a CD grammar system.

The following definition describes the addition of feature detectors to CD grammar systems.

**Definition 2.9.** A basic feature grammar system is a  $(n + 6)$ -tuple

$$\Gamma = (D, N, T, P_N, G_1, G_2, \dots, G_n, G_S, S),$$

where,

1.  $V = (D \cup N \cup T)$  is the shared alphabet consisting of
  - (a) the set  $D$  of detectors, containing at least  $S_S$ ,
  - (b) the set  $N$  of non-terminals and
  - (c) the set  $T$  of terminals,
2.  $P_N$  contains productions of the form  $(N \rightarrow V^*)$ ,
3. for each  $d_i \in D$ , *i. e.*  $n = |D|$ , there is a  $G_i = (V, P_i = P_{d_i} \cup P_N, f_i)$  with
  - (a) the set  $P_{d_i}$  consisting of CF rules of the form  $(d_i \rightarrow V^+)$ ,

- (b) a partial detector function  $f_i : T^* \rightarrow (D \cup T)^+$ , and  
(c)  $\lambda \notin L(G_i)$ ,
4. the start component  $G_S = (V, P_S = \{(S_S \rightarrow D \cup N)\} \cup P_N, f_S)$  where  
(a)  $f_S$  is a function producing an initial sentence  $w_S \in (D \cup T)^+$ ,
5. and finally  $S = S_S$ .

□

All components in a feature grammar system always use the same alphabet  $V$ . In more practical terms: there is only one symbol table as all feature grammar system components share the same set of symbols. This variant of CD grammar systems is denoted as  $(CD'_\infty CF, f)$ , where the acceptance style is deleted as it is always *style(ex)*. This makes it possible to share semantically meaningful non-terminals. These non-terminals, which are not detectors, are used to group terminals. These semantic structures can be shared by different detectors and are put in the shared set of production rules  $P_N$ .

Each grammar component corresponds with one detector symbol. The detector function restricts the language accepted by a feature grammar system  $\Gamma$  in the following formal way:

**Definition 2.10.** *Let  $\Gamma$  be a feature grammar system as specified in Definition 2.9. Then the language  $L(\Gamma)$  generated by  $\Gamma$  is defined as the set of all words  $w \in T^+$  for which there is a derivation*

$$S \xRightarrow{*}_{\Gamma} w_l d_i w_r \xRightarrow{*}_{G_i} w_l f_i(w_l d_i w_r) w_r \xRightarrow{*}_{\Gamma}$$

Where  $f_i$  is the partial mapping function associated with detector  $d_i$ .

□

The moment a detector  $d_i$  is encountered the remainder of the derivation should replace this symbol by the output of the associated detector function  $f_i(w_l d_i w_r)$ . In other words  $f_i$  prescribes the sentence generated by this part of the grammar, and forms the stop condition of the component. The grammar component  $G_i$  mimics the unknown internal mapping of  $f_i$  and thus validates that the output of this detector conforms to the grammar<sup>4</sup>. This generated sentence functions as a stop condition of *type(C)*, i. e. a feature grammar system is a  $(CD'_\infty CF, (a, C))$  grammar system.

Notice that the empty sentence ( $\lambda$ ) is not allowed as output of a detector function, i. e.  $\lambda \notin L(G_i)$ . An empty sentence thus indicates the partiality of the detector function: the mapping is unknown for the input sentence.

<sup>4</sup> Notice that the grammar components are CF while the detector function itself may be more powerful, i. e. produce a sentence in a CS language like  $a^n b^n c^n$ . The grammar component will only be able to validate the CF envelope, i. e.  $a^* b^* c^*$ .



Definition 2.9 also introduces a dummy start symbol  $S_S$ . This special symbol is introduced to make it possible to have a “real” start symbol which is either a non-terminal or a detector. In the case of a detector the  $G_S$  component will stop directly after the application of the  $(S_S \rightarrow d_i)$  rule, as it does not contain any rules for  $d_i$ , and control will be transferred to  $G_i$ . In the case of a non-terminal  $P_N$  helps to parse the initial sentence produced by  $f_S$ . How  $f_S$  produces this initial sentence is an implementation issue and will be discussed in Chapter 4.

The feature grammar system for Example 2.4 is constructed from these building blocks (the dots (...) indicates some omitted production rules from the example):

**Example 2.5.**

$$\begin{aligned}
 \Gamma = ( & \\
 & D = \{S_S, Color, Graphic, Photo, Skin, Faces\}, \\
 & N = \{Image, Location, Number, Prevalent, Saturation, Class, \\
 & \quad url, int, flt, bit, bitmap\}, \\
 & T = \{http : // \dots, 1, 29053, 0.03, 0.19, true, 00 \dots\}, \\
 & P_N = \{(Image \rightarrow Location Color Class), \dots, (bitmap \rightarrow 00 \dots)\}, \\
 G_{Color} = (V, P_{Color} = \{(Color \rightarrow Number Prevalent Saturation)\} & \\
 & \cup P_N, f_{Color}), \\
 G_{Graphic} = (V, P_{Graphic} = \{(Graphic \rightarrow bit)\} \cup P_N, f_{Graphic}), & \\
 G_{Photo} = (V, P_{Photo} = \{(Photo \rightarrow bit)\} \cup P_N, f_{Photo}), & \\
 G_{Skin} = (V, P_{Skin} = \{(Skin \rightarrow bitmap)\} \cup P_N, f_{Skin}), & \\
 G_{Faces} = (V, P_{Faces} = \{(Faces \rightarrow int)\} \cup P_N, f_{Faces}), & \\
 G_S = (V, P_S = \{(S_S \rightarrow Image)\} \cup P_N, f_S), & \\
 & S = S_S \\
 & )
 \end{aligned}$$

□

The derivation process of the example sentence using this feature grammar looks as follows:

**Example 2.6.**

$$\begin{aligned}
 w_1 &= S_S \\
 &\Rightarrow_{G_S}^* \\
 w_2 &= f_S(w_1) \text{ Color Photo Skin Faces} \\
 &= http : // \dots \text{ Color Photo Skin Faces} \\
 &\Rightarrow_{G_{Color}}^*
 \end{aligned}$$

$$\begin{aligned}
w_3 &= \text{http} : // \dots f_{\text{Color}}(w_2) \text{ Photo Skin Faces} \\
&= \text{http} : // \dots 29053 \ 0.03 \ 0.19 \ \text{Photo Skin Faces} \\
&\Rightarrow^*_{G_{\text{Photo}}} \\
w_4 &= \text{http} : // \dots 29053 \ 0.03 \ 0.19 \ f_{\text{Photo}}(w_3) \ \text{Skin Faces} \\
&= \text{http} : // \dots 29053 \ 0.03 \ 0.19 \ \text{true Skin Faces} \\
&\Rightarrow^*_{G_{\text{Skin}}} \\
w_5 &= \text{http} : // \dots 29053 \ 0.03 \ 0.19 \ \text{true} \ f_{\text{Skin}}(w_4) \ \text{Faces} \\
&= \text{http} : // \dots 29053 \ 0.03 \ 0.19 \ \text{true} \ 00 \dots \ \text{Faces} \\
&\Rightarrow^*_{G_{\text{Faces}}} \\
w_6 &= \text{http} : // \dots 29053 \ 0.03 \ 0.19 \ \text{true} \ 00 \dots \ f_{\text{Faces}}(w_5) \\
&= \text{http} : // \dots 29053 \ 0.03 \ 0.19 \ \text{true} \ 00 \dots \ 1
\end{aligned}$$

□

The  $S_S$  start symbol allows for the non-terminal *Image* to be the “real” start symbol. The initial sentence  $\text{http} : // \dots$  is produced by  $f_S$  and triggers the mappings of the other detectors.

## 2.2.2 Atoms

Enumerating the complete terminal domain as illustrated in the example grammar is far from practical. But as this grammar is CF, and CF grammars are closed under substitution, a CF language  $L_a$  can be chosen to further describe each symbol  $a$  in  $\Sigma$ .

### Example 2.7.

$$\begin{aligned}
url &\rightarrow \{ \text{http}://([\text{^}:\text{/}]^*)(:[0-9]^*)?/?(\text{.}^*)\$ \} \\
int &\rightarrow \{ \text{^-}[0-9]^+\$ \} \\
flt &\rightarrow \{ \text{^-}[0-9]^+\.[0-9]^+([\text{Ee}][\text{-+}]?[0-9]^+)?\$ \} \\
bit &\rightarrow \{ \text{^}(0|1|(\text{true})|(\text{false}))\$ \} \\
bitmap &\rightarrow \{ \text{^}(0|1)^*\$ \}
\end{aligned}$$

□

In this case for *url* the regular expression (remember  $\mathcal{L}(\text{REG}) \subset \mathcal{L}(\text{CF})$ ) corresponds to  $L_{url}$ , and so on. The non-terminals which are the root of such a substitution language are called *atoms* (in parsing literature they are sometimes called *pre-terminals* [Tom86]). The yield of a specific (partial) parse tree rooted by an atom is called an *instantiation* of the atom, *i.e.*  $29053$  is an instantiation of the atom domain *int* or in short:  $\text{int}(29053)$ . The complete set of terminals described by the regular expressions are called the *lexicon* of the grammar.

In practice this phase is called *lexical analysis* [ASU86], in which the stream of characters or bytes is translated into pre-terminals. This abstraction process helps to

get the actual grammar rules closer to a *semantic grammar* [BB75]. In a semantic grammar the rules and symbols are designed to correspond directly to entities and relations from the domain being discussed. This makes the results associated with the grammar, *i. e.* the parse trees, better suited for (human) interaction in other system components of the DMW application.

### 2.2.3 Dependencies

Do feature grammar systems as defined until now capture the dependency types as identified in Chapter 1? Contextual dependencies, like a *Photo* which contains *Skin* colors, may also contain one or more *Faces*, are clearly described in grammars. But how about output/input dependencies?

In the first chapter it was stated that due to the explicit handling of context dependencies detectors stay generic. However, this property is limited again by the output/input dependency. The definition of the output/input dependency of a detector should be powerful enough to be precise, without being too restrictive on the context. This context knowledge influences both the input and output specification of a detector component, which will be investigated in the next two sections.

#### 2.2.3.1 Detector Input

In Definitions 2.9 and 2.10 detector functions depend on the whole sentential form as input, but in reality the mapping function only uses a part of this information. For example:  $f_{Faces}$  only needs the bitmap of skin pixels to find the number of faces, all the other information, like it is a photo, is irrelevant for its mapping. This form of mild context-sensitivity can be captured by adding a regulated rewriting mechanism as a start condition to a detector component.

In Section 2.1.3 C grammars were introduced. Using these conditions on the sentential form a detector component can only become active when its input is available. Take once more the *Faces* detector: it needs the skin *bitmap* derived from the image. The REG language  $R_{Faces} = (. * \cdot 00 \dots \cdot . *)$ , using the in Section 2.1.1.1 defined regular expression syntax, indicates that the sentential form should always contain a specific bitmap. However, as this condition works directly on the sentential form it does not give much control over the semantic context, *e. g.* the non-terminal level, where the desired input data resides in. TC grammars could give more grip on this context as they operate on the parse tree. But both mechanisms of the C and TC grammars restrict the context of the desired input in a horizontal fashion, *i. e.* C grammars limit the left and right contexts in the sentential form, while TC grammars limit the level sentences of the parse tree. To be able to create these conditions the developer of a detector needs to have complete knowledge about the context of the detector in a specific feature grammar system, which is just what the system should circumvent.

An easier and more context-free way is to look at the parse tree vertically, *i. e.* use the paths in the parse tree. This new regulated rewriting mechanism, *i. e.* leading to

*path-controlled (PC)* grammars, can be defined as follows:

**Definition 2.11.** In a path-controlled grammar  $G = (N, T, P, S)$  the productions  $P$  are of the form  $(w \rightarrow z, R)$  where  $R$  is a REG language over the alphabet  $V$ .  $N$ ,  $T$  and  $S$  have their normal meaning. The rule  $(w \rightarrow z, R)$  is applicable to  $x = x_l w x_r$  yielding  $y = x_l z x_r$ , i. e.  $x \Rightarrow y$ , iff the parse tree  $t$  associated to  $x$  satisfies  $R \subseteq \text{path}(t)$ . □

Notice that *PC* grammars are at least as powerful as *C* grammars, i. e. a *C* grammar can always be translated into a *PC* grammar.

A special variant of this type of grammars are *left path-controlled (LPC)* grammars. In their case only paths referring to the left context of  $w$ ,  $x_l$ , are allowed.

**Definition 2.12.** In a left path-controlled grammar  $G = (N, T, P, S)$  the productions  $P$  are of the form  $(w \rightarrow z, R)$  where  $R$  is a REG language over the alphabet  $V$ .  $N$ ,  $T$  and  $S$  have their normal meaning. The rule  $(w \rightarrow z, R)$  is applicable to  $x = x_l w x_r$  yielding  $y = x_l z x_r$ , i. e.  $x \Rightarrow y$ , iff the parse tree  $t$  associated to  $x$  satisfies  $\delta(R \subseteq \text{path}(t)) \subseteq x_l$ . □

The *delta* operation ( $\delta$ , see Section 2.1.2.3) creates a new sentence from the selection of paths from  $t$  which is created by the intersection between  $\text{path}(t)$  and  $R_i$ . This new sentence may only contain terminals from  $x_l$ .

Adding this specific version of the path control mechanism to Definition 2.9 gives the formal definition of a *LPC* feature grammar system:

**Definition 2.13.** A left path-controlled feature grammar system is a feature grammar system  $\Gamma$  as in Definition 2.9 with  $G_i = (V, P_i, R_i, f_i)$ , where  $R_i$  is a REG language over  $V$ . The start ( $R_i$ ) and stop ( $f_i$ ) conditions of the component  $G_i$  restrict a derivation in the following way:

$$w_j = w_l d_i w_r \Rightarrow_{G_i}^* w_l f_i (\delta(R_i \cap \text{path}(t_j))) w_r = w_{j+1}$$

Such that  $\delta(R_i \subseteq \text{path}(t_j)) \subseteq w_l$ . Where  $t_j$  is the parse tree associated with the sentential form  $w_j$ , i. e.  $\text{yield}(t_j) = w_j$ . □

The new sentence created by the *delta* operation contains exactly the information the detector  $f_i$  needs to perform its mapping, i. e. the context knowledge is optimally limited.

Using the atoms as terminal level, and adding this additional rewriting mechanism the example feature grammar looks formally as follows:

**Example 2.8.**

$$\begin{aligned}
\Gamma = (& \\
& D = \{S_S, Color, Graphic, Photo, Skin, Faces\}, \\
& N = \{Image, Location, Number, Prevalent, Saturation, Class\}, \\
& T = \{url, int, flt, bit, bitmap\}, \\
& P_N = \{(Image \rightarrow Location Color Class), \dots, \\
& \quad (Class \rightarrow Photo Skin Faces)\}, \\
G_{Color} = (& V, P_{Color} = \{(Color \rightarrow Number Prevalent Saturation)\} \\
& \cup P_N, (* \cdot Location \cdot url), f_{Color}), \\
G_{Graphic} = (& V, P_{Graphic} = \{(Graphic \rightarrow bit)\} \cup P_N, (* \cdot Number \cdot int) + \\
& (* \cdot Prevalent \cdot flt) + (* \cdot Saturation \cdot flt), f_{Graphic}), \\
G_{Photo} = (& V, P_{Photo} = \{(Photo \rightarrow bit)\} \cup P_N, (* \cdot Color \cdot *) , f_{Photo}), \\
G_{Skin} = (& V, P_{Skin} = \{(Skin \rightarrow bitmap)\} \cup P_N, (* \cdot Location \cdot url), f_{Skin}), \\
G_{Faces} = (& V, P_{Faces} = \{(Faces \rightarrow int)\} \cup P_N, (* \cdot bitmap), f_{Faces}), \\
G_S = (& V, P_S = \{(S_S \rightarrow Image)\} \cup P_N, \emptyset, f_S), \\
& S = S_S \\
& )
\end{aligned}$$

□

The derivation process of the example sentence using this *lPC* feature grammar looks as follows (see the parse tree in Figure 2.4 for the binding of the regular path expressions):

**Example 2.9.**

$$\begin{aligned}
w_1 &= S_S \\
&\Rightarrow_{G_S}^* \\
w_2 &= f_S(\lambda) Color Photo Skin Faces \\
&= url(http://...) Color Photo Skin Faces \\
&\Rightarrow_{G_{Color}}^* \\
w_3 &= url(http://...) f_{Color}(url(http://...)) Photo Skin Faces \\
&= url(http://...) int(29053) flt(0.03) flt(0.19) Photo Skin Faces \\
&\Rightarrow_{G_{Photo}}^* \\
w_4 &= url(http://...) int(29053) flt(0.03) flt(0.19) \\
&\quad f_{Photo}(int(29053) flt(0.03) flt(0.19)) Skin Faces \\
&= url(http://...) int(29053) flt(0.03) flt(0.19) bit(true) Skin Faces \\
&\Rightarrow_{G_{Skin}}^*
\end{aligned}$$

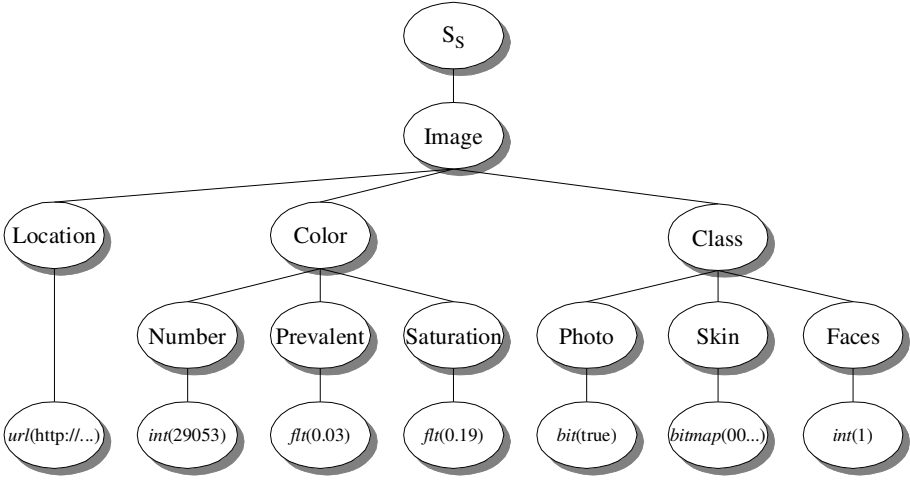


Figure 2.4: A parse tree constructed by a *LPC* feature grammar system

$$\begin{aligned}
 w_5 &= url(http://...) \ int(29053) \ flt(0.03) \ flt(0.19) \ bit(true) \\
 &\quad f_{Skin}(url(http://...)) \ Faces \\
 &= url(http://...) \ int(29053) \ flt(0.03) \ flt(0.19) \ bit(true) \ bitmap(00...) \ Faces \\
 &\Rightarrow_{G_{Faces}}^* \\
 w_6 &= url(http://...) \ int(29053) \ flt(0.03) \ flt(0.19) \ bit(true) \ bitmap(00...) \\
 &\quad f_{Faces}(bitmap(00...)) \\
 &= url(http://...) \ int(29053) \ flt(0.03) \ flt(0.19) \ bit(true) \ bitmap(00...) \ int(1)
 \end{aligned}$$

□

The derivation form in this example is unspecified: control may be transferred to any detector for which the input sentence is valid. For example, after derivation of  $w_2$  control can be transferred to both  $G_{Color}$  and  $G_{Skin}$ . The example derivation favors the leftmost symbol, *i. e.* resembles a *leftmost* derivation (see Section 2.1.2). The leftmost derivation of detector symbols is always applicable in a *LPC* grammar, as this rewrite mechanism enforces that the input sentence of the leftmost symbol is always available (when the sentence is valid). However, notice that normal non-terminals take precedence over detectors, *e. g.* the *Class* non-terminal is resolved before control is transferred to the *Color* detector. So the leftmost derivation takes place on the control level of the grammar system.

Using the *LPC* rewriting mechanism for a feature grammar system has a practical advantage: it implements a *deadlock* prevention mechanism. Deadlock situations occur when the regular expressions of two or more detectors depend on each others,

not yet available, parse trees. The start conditions of these detectors can thus never be satisfied. A common deadlock prevention strategy is *linear ordering* [Sta92]. *LPC* implements this strategy in a straightforward fashion: detectors only depend on the existence of preceding terminals, and can thus naturally be ordered from left to right.

The addition of left path-control to a feature grammar system turns it from a  $(CD'_\infty CF, (a, C))$  into a  $(CD'_\infty CF, (LPC, C))$  system.

### 2.2.3.2 Detector Output

Just like the detector function in Definition 2.9 depends on a sentential form for its input its result should also exactly match a sentential form, *i.e.* the stop condition of the component. In the example grammar this works fine as there are no nested detectors like:

#### Example 2.10.

*Color*  $\rightarrow$  *Number isBitmap Prevalent Saturation*

with

$$G_{isBitmap} = (V, P_{isBitmap} = \{(isBitmap \rightarrow bit)\} \cup P_N, (* \cdot Number \cdot *), f_{isBitmap})$$

□

This new *isBitmap* detector takes the *Number* of colors and maps it to *bit(true)* when there are exactly two colors, otherwise it returns *bit(false)*. Integrating this call into the derivation of Example 2.9:

#### Example 2.11.

$$\begin{aligned} w_{3a} &= url(http://...) f_{Color}(url(http://...)) Photo Skin Faces \\ &= url(http://...) int(29053) isBitmap flt(0.03) flt(0.19) Photo Skin Faces \\ &\xrightarrow{*} G_{isBitmap} \\ w_{3b} &= url(http://...) int(29053) f_{isBitmap}(int(29053)) flt(0.03) flt(0.19) Photo Skin \\ &\quad Faces \\ &= url(http://...) int(29053) bit(false) flt(0.03) flt(0.19) Photo Skin Faces \end{aligned}$$

□

This example shows that the *Color* detector now needs to know the exact position of the *isBitmap* detector in its output sentence. Definition 2.9 captures this by stating that  $f_i$  is a mapping from  $V^*$  into  $(D \cup T)^+$ . How to lift the burden of this context knowledge from the detector function? Optimally the range domain becomes a member of  $T^+$ . As stated in Section 2.2.1 the output sentence,  $z$ , can be seen as the only member of language described by a *REG* language. By enlarging  $z \in T^+$  to

a language containing all possible nestings of detectors, the detector implementation can become less context-sensitive.

$$\begin{aligned} f_i(w) &= a_1 \dots a_m = z_{d_i} \\ A &= (d_1 + \dots + d_n) \\ f_D(z_{d_i}) &= (A^* \cdot a_1 \cdot A^* \dots A^* \cdot a_q \cdot A^* \dots A^* \cdot a_m \cdot A^*) \end{aligned}$$

The function  $f_D$  takes the output of  $f_i$  and turns it into a REG language consisting of all words interleaved with arbitrary sequences of detectors, represented by the REG language  $A$ .

**Definition 2.14.** Let a conditional feature grammar system  $\Gamma$  be a feature grammar system as in Definition 2.9 where  $f_i$  is a mapping from  $w \in T^+$  into  $z \in T^+$ . The stop condition  $f_i$  of the component  $G_i$  restricts the derivation in the following way:

$$w_j = w_l d_i w_r \Rightarrow_{G_i}^* w_l z w_r = w_{j+1}$$

where

$$z \in f_D(f_i(w_j))$$

The function  $f_D : T^+ \rightarrow \mathcal{L}(REG)$  maps the output sentence,  $z_{d_i}$ , of  $f_i$  into a REG language where each terminal,  $a_q$  a word in  $z_{d_i}$ , is enveloped by an arbitrary sequence of detectors.

□

Notice that this adapted stop condition is also used for the output of the special “dummy” detector  $f_S$ .

Using this latest addition to the concept of feature grammar systems the (extended) example derivation can be (partially) rewritten:

**Example 2.12.**

$$\begin{aligned} w_2 &= \text{url}(\text{http://...}) \text{Color Photo Skin Faces} \\ &\Rightarrow_{G_{Color}}^* \\ w_{3a} &= \text{url}(\text{http://...}) \text{int}(29053) \text{isBitmap flt}(0.03) \text{flt}(0.19) \text{Photo Skin Faces} \\ &\Rightarrow_{G_{isBitmap}}^* \\ w_{3b} &= \text{url}(\text{http://...}) \text{int}(29053) \text{bit}(\text{false}) \text{flt}(0.03) \text{flt}(0.19) \text{Photo Skin Faces} \end{aligned}$$

where



$$w_{3a} \in f_D(f_{Color}(w_2)) = (A^* \cdot int(29053) \cdot A^* \cdot flt(0.03) \cdot A^* \cdot flt(0.19) \cdot A^*)$$

$$w_{3b} \in f_D(f_{isBitmap}(w_{3a})) = (A^* \cdot bit(false) \cdot A^*)$$

and

$$A = (S_S + Color + Graphic + Photo + Skin + Faces + isBitmap)$$

□

The derivation of the symbol *Color* from  $w_2$  into  $w_{3a}$  is a member of the language generated by  $f_D(f_{Color}(w_2))$  and thus satisfies the stop condition of the  $G_{Color}$  component. So the implementation of the *Color* detector function is now just as context-sensitive as needed, as it contains a precise specification of the input and produces a context-free sentence.

## 2.2.4 Ambiguous Feature Grammar Systems

Just like any other grammar feature grammar components may be ambiguous. In fact support for ambiguity is identified in Section 1.2.1 as a main requirement of the target application domain of Acoi. Take for example the classification of the *Image* into either a *Photo* or a *Graphic*. It is possible that both classes are valid, *i. e.* the  $f_{Photo}$  and  $f_{Graphic}$  detectors both return a valid output. This results in two parse trees combined in one forest as discussed in Section 2.1.2.4.

In such a forest the possibility exists that detector input can be bound to nodes from different parse trees, *i. e.* the binding is ambiguous. To illustrate this the example grammar is extended with an object recognizer, *e. g.* to find a vehicle.

### Example 2.13.

*Image* → *Location Color Class Object*

*Object* → *Vehicle*

with

$$G_{Vehicle} = (V, P_{Vehicle} = \{(Vehicle \rightarrow bit)\} \cup P_N,$$

$$(* \cdot Location \cdot url + * \cdot Class \cdot *), f_{Vehicle})$$

□

As shown in this extension the *Vehicle* detector constructs an input sentence containing the location of the image and the detected class. It uses the class to select between different image segmentation algorithms for photos and graphics. When the class detectors are confident enough, *i. e.* only one is valid, the input sentence of the *Vehicle*

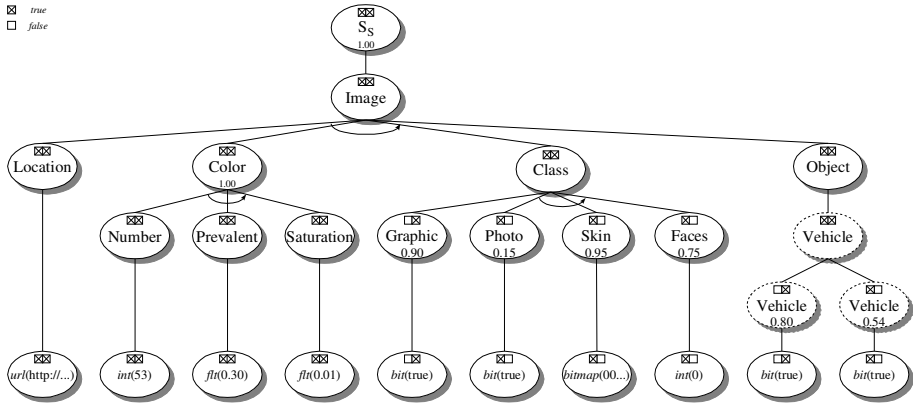
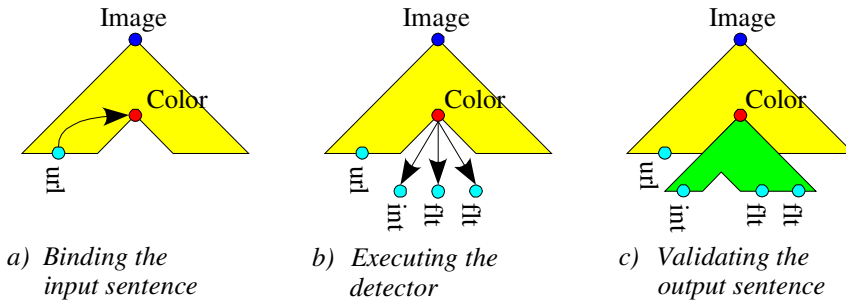


Figure 2.5: A parse forest containing quasi-nodes plus context and confidence annotations

detector can only be bound to one class. However, in the case of ambiguity two bindings are possible. The *Vehicle* detector cannot prefer one class over the other. Instead it needs to be executed twice: once for each class.

To reflect this in the parse forest the detector node is split in two levels: (1) a top *quasi-node*, the *quasi-root*, and (2) one or more bottom *quasi-nodes*, the *quasi-foots*. Quasi-nodes belong to quasi-trees which are inspired by D-Theory [MHF83]. [VS92] states: “ $vp_1$  [the quasi-root] and  $vp_2$  [the quasi-foot] can both refer to the same node, to avoid confusion, henceforth we will call them **quasi-nodes**.” In the feature grammar system case the quasi-root represents the detector symbol within its rule context. Each quasi-foot represents one possible execution of the detector. By gluing together the quasi-root with one of the quasi-foots one specific instantiation of the detector call is chosen. An example parse forest containing quasi-nodes, indicated with dashed outlines, for the *Vehicle* detector is shown in Figure 2.5. The figure also shows that when there is only one bottom node the quasi-nodes can be glued together into a normal tree node.

The parse forest in this figure contains additional information to cope with the long distance dependencies of a feature grammar system: nodes are explicitly labeled with a context. As Section 2.1.2.4 already said, CF grammars have only local ambiguity which makes it easy to locate the separate parse trees. However, due to the more advanced dependencies of detector nodes additional information is needed. In the NLP community named disjunctions (NDs) are used [Bla97]. However, NDs expect each disjunction to be of the same arity. Which means that when the third disjunction of a *controller* ND is valid the third disjunction in the *controlled* disjunction should also be valid. In the simple example of Figure 2.5 this is true: the *Class* disjunction controls the *Vehicle* disjunction, and within their parse trees the indices of the disjunctions

Figure 2.6: Execution of the *Color* detector

match. However, the input sentence of a detector may be constructed from terminal nodes obtained from several disjunctions, *i. e.* there will not be one controlling disjunction. So instead of naming disjunctions each node is provided with a context. This context consists of a list of binary flags, where each flag represents a parse tree and indicates if the node belongs to it (*true*) or not (*false*). In the example forest there are two trees. Most nodes belong to both trees, except for the different *Class* alternatives and their dependent *Vehicle* quasi-foots. Limiting the binding of detector parameters to the scope of the current quasi-root is now easily implemented by a binary operation (see Section 4.3.3.2).

Next to contexts the parse forest in Figure 2.5 also contains detector confidences. Although both *Class* detectors are valid, they are not both as confident. The *Graphic* detector is 90% sure that the image is a graphic, while the *Photo* detector has only a confidence of 15%. Providing this information may help the user or librarian in disambiguating the parse forest. The confidence values are in this case seen as node annotations, but may also be just modeled as ordinary leaf nodes.

Notice that the parse forest shown in Figure 2.5 describes several sentences, which may partially overlap. This is another reason why a packed shared parse forest as introduced in Section 2.1.2.4 is not usable in the case of ambiguous feature grammar systems. A packed shared forest describes always one sentence.

## 2.2.5 Mildly Context-sensitive Feature Grammar Systems

Combining the two forms of regulated rewriting as defined in Definitions 2.13 and 2.14 with Definition 2.9 the definition of *mildly context-sensitive feature grammar systems* is obtained. This form of feature grammar systems is used throughout the rest of this thesis.

Figure 2.6 illustrates the working of the *Color* detector. The *url* instantiation belongs to its input sentence, as specified by  $R_{Color} = (. * \cdot Location \cdot url)$ . The  $f_{Color}$  detector uses this information to load the image, analyze it, and produce the

output sentence: an integer and two floats. This sentence is parsed and matches the rules in the feature grammar system, so the stop condition is satisfied. The detector has only limited knowledge of the context it operates in: its input and its output. The CF grammar rules within the components can be used to place these detectors in different contexts. However, the start and stop conditions will enforce that these contexts match the (limited) knowledge of the detector.

## 2.3 Discussion

In this chapter the formal theory of feature grammar systems has been described. To achieve mild context-sensitivity this theory is based on a combination of CD grammar systems and a regulated rewriting mechanism. The major extensions to these existing theories include the addition of detector functions to produce (part of) the sentence just-in-time and the *PC* and *lPC* regulation mechanisms. A detector function directly determines, when the start condition is satisfied, the stop condition of the grammar component. This enables a very tight integration between user-defined functions and the grammar component, while staying formally sound. The *lPC* mechanism allows mildly context-sensitive specification of the start condition, while also implementing a deadlock prevention strategy.

Two kinds of dependencies were identified in Chapter 1 as key components of a declarative dependency description language. A feature grammar system captures them both. The RHSs of the CF production rules within each grammar component describe the context dependencies. While the regular expressions of the start conditions of the same components capture output/input dependencies.

Using a feature grammar system a parse tree/forest can be build which represents all (ambiguous) contextual knowledge about the extracted annotations, and may thus form a good basis for incremental maintenance.

Next to Acoi there are some systems which are also focus at controlling the flow of annotation extraction, or could be used for that purpose. MOODS [GYA97] is based on the extension of an object oriented schema with semantic objects. The novel aspect of a semantic object is its processing graph. In a processing graph extraction algorithms are linked together. The developer defines a split point in the processing graph. The first part is executed in a data-driven way when a new object instance is inserted into the database. The last part is executed in a demand-driven fashion during query processing. Additionally, the object can be extended with inference rules, which can be checked during query processing. There is no specific support in MOODS for context dependencies, unless they are hard-coded as an output/input dependency.

In the Mirror system [dVEK98, dV99] extraction algorithms are encapsulated in daemon (CORBA) objects. The daemons issue a *get\_work* query to the database, which functions as a data pool, extract the meta-data (*i. e.* annotations) for the returned objects and then issue a *finish\_work* query to commit their results. All knowledge about dependencies is embedded in the *get\_work* queries, *i. e.* there is no global

declarative specification. Context dependencies thus need to be tightly bound, *i. e.* hardcoded, and may become distributed over several daemon objects.

Linda tuple spaces [Gel95] can form an alternative for the daemon architecture. The daemon requests its input data once from the tuple space, instead of polling for it from time to time, blocks and becomes active when the tuple space notifies the availability of the data. However, although this offers a different implementation model it does not resolve the need for hardcoding the context dependencies.

Dataflow languages are also focused on output/input dependencies. For example, Microsoft DirectShow [Mic99] offers a way to tie algorithms together using filter graphs. This builds a pipeline of algorithms through which multimedia streams flow. The target of such systems is to produce mainly one final result, *e. g.* a video converted from color to black and white and from the MPEG format to AVI format. In a feature grammar system the result is a parse forest which contains all the annotation information, which may be considered intermediate data in the pipeline.

The ToolBus system [BK94, BK96, dJK] provides a coordination architecture for heterogeneous, distributed software systems. The coordination mechanism, *i. e.* the toolbus, is steered by T scripts, which are based on process algebra [BK86]. A T script contains one or more processes, while each process may control one or more external tools. Tools communicate, *i. e.* send messages or share data, with each other using the bus. To overcome different data formats the tools are encapsulated by adapters which transform the native formatted data into the general form used by the bus, *i. e.* ATerms [vdBdJKO00]. The ToolBus thus addresses separation of coordination (T scripts), representation (ATerms) and computation (the tools). This closely resembles feature grammar systems: where grammar components provide coordination, the parse tree provides representation and detectors computation. The main difference being the, in the case of feature grammar systems, by default constructed parse tree. This tree contains (semantic) contextual knowledge both for data and processes. This knowledge is, as shown in the introductory chapter, indispensable for the application domain considered in this thesis. However, ATerms are generic data types and can be used to describe trees or forests [Vis97]. But in the implementation the dependencies would become implicit in a, more generic, T script and possibly even hidden inside ATerm messages. To enable reasoning about these dependencies, *e. g.* to allow the FDS to steer incremental maintenance, this knowledge would have to be made explicit again. So once more a declarative dependency description, *i. e.* in the form of a feature grammar system, would be needed. However, this makes the ToolBus a possible, feature grammar driven, candidate to implement parts of the Acoi system, *e. g.* the FDE.



## Chapter 3

# Feature Grammar Language

*There is nothing that can be said by mathematical symbols and relations which cannot also be said by words. The converse, however, is false. Much that can be and is said by words cannot successfully be put into equations, because it is nonsense.*

C. Truesdell

The mathematical notation introduced and used in the previous chapter is, although precise, not very convenient in everyday life. In this chapter a more practical notation is introduced: the feature grammar language. This language describes how a feature grammar is specified, *i. e.* it is a *meta-language*.

In Appendix **A** the complete specification of the language is given using the *Extended Backus-Naur Form* (EBNF). The ancestor of EBNF, *Backus-Naur Form* (BNF), is a CF grammar notation mostly used for specifying programming languages, *i. e.* it was first used to define ALGOL 60. In the upcoming sections parts of the language are introduced by language snippets related to the example feature grammar. Appendix **B** contains the collection of feature grammars, which has been build for the various case studies (see Chapter **7**).

The next section introduces the core of the language which directly maps on the formalization of feature grammar systems. Subsequently additions to this language will be described whose main purpose is to provide shortcuts for developers or to steer the FDE or FDS in their analysis of the grammar.

### 3.1 The Basic Feature Grammar Language

The core of the feature grammar system is formed by the shared alphabet  $V = (D \cup N \cup T)$ , the production rules as distributed over the various grammar components, the start and stop conditions of these components, and the start symbol. In the subsequent subsections the notation for these core components is shown.

### 3.1.1 Production Rules

Production rules form the heart of every grammar. As Definition 2.1 and the section on ambiguous grammars showed there are alternative interpretations possible for one non-terminal. The extended notation (in this specific form also called *regular right part grammars* (RRPG) [LaL77]) makes it possible to combine these alternatives syntactically into one rule.

```

1 | Image      : Location ( Color Class )?;
2 | Color     : RGB* Number Prevalent Saturation;
3 | Class     : Graphic | Photo Skin Faces;
4 | RGB      : Red Green Blue;

5 | Location  : url;
6 | Number   : int;
7 | Prevalent : flt;
8 | Saturation : flt;
9 | Graphic  : bit;
10 | Photo   : bit;
11 | Skin    : bitmap;
12 | Faces   : int;
13 | Red     : int;
14 | Green   : int;
15 | Blue    : int;

```

The *RGB* non-terminal is introduced into the example to make it rich enough to illustrate some extended features.

In this notation a production rule's LHS and RHS are separated by a colon and the rule is terminated with a semicolon. Alternative representations for the same LHS are grouped together and then separated by vertical bars, | (see the *Class* rule). Symbol sequences, *i. e.* optional, star and positive closure, are indicated by respectively ?, \* and + occurrence indicators (see the *Image* and *Color* rules). Furthermore, symbols can be combined into groups using brackets. These groups can have their own occurrence indicators and embedded alternatives (see once more the *Image* rule).

All these extended constructs for production rules can be rewritten into the basic formal version of the production rules. For symbol sequences there are two methods. The first method of rewriting uses the *recursive* interpretation. In this interpretation the *RGB\** sequence is rewritten into the following formal rules:

$$\begin{aligned}
 \textit{Color} &\rightarrow \textit{Number Prevalent Saturation} \\
 \textit{Color} &\rightarrow \alpha \textit{ Number Prevalent Saturation} \\
 \alpha &\rightarrow \textit{RGB} \\
 \alpha &\rightarrow \textit{RGB } \alpha \\
 \textit{RGB} &\rightarrow \textit{Red Green Blue}
 \end{aligned}$$



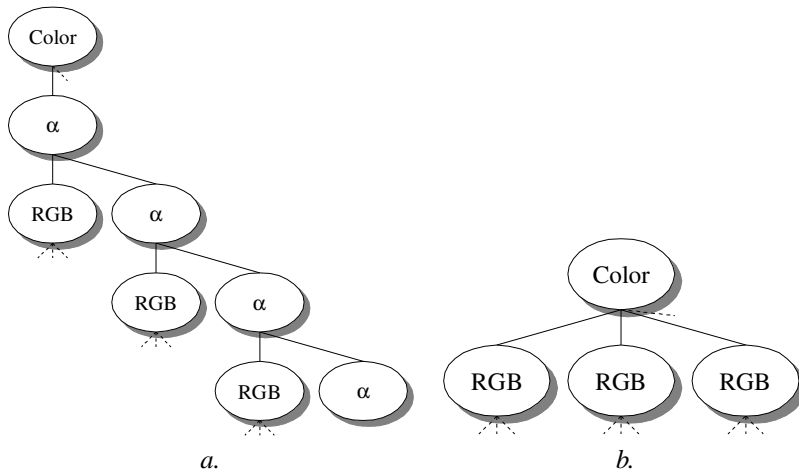


Figure 3.1: (a) A right-recursive and (b) an iterative interpretation of a list construct

This interpretation has an advantage that it is easy to explain because the transformation to a basic CF grammar is simple. Disadvantages are the introduction of anonymous variables (in this case  $\alpha$ ) and the lopsided parse tree (see Figure 3.1.a), which in general does not correspond to one's intuition.

The *iterative* interpretation of symbol sequences is more intuitive. It sees the  $RGB^*$  sequence as an abbreviation of:

*Color*  $\rightarrow$  *Number Prevalent Saturation*  
*Color*  $\rightarrow$  *RGB Number Prevalent Saturation*  
*Color*  $\rightarrow$  *RGB RGB Number Prevalent Saturation*  
*Color*  $\rightarrow$  *RGB RGB RGB Number Prevalent Saturation*  
*Color*  $\rightarrow$  ...  
 ...  
*RGB*  $\rightarrow$  *Red Green Blue*

The advantage of this interpretation is a beautiful parse tree (see Figure 3.1.b), but has disadvantages that it involves an infinite number of production rules and that the nodes in the parse tree have a varying fan-out.

Next to rewrite the occurrence indicators may also directly be interpreted by the parser implementation (see [GJ98]), *i. e.* using IF-statements for optional symbols, WHILE-statements for star closure and REPEAT-statements for positive closure. The resulting parse tree will adhere to the iterative interpretation. However, this implements these indicators in a greedy fashion and thus favors greedy alternatives. Notice

Atom name	Substitution language (as a regular expression)
bit	$^(0 1 (true) (false))\$$
chr	$^.\$$
int	$^-?[0-9]+\$$
flt	$^-?[0-9]+\.[0-9]+([Ee][+-]?[0-9]+)?\$$
str	$^.*\$$

Table 3.1: Default atom types

that is the case with most implementations of regular expressions [Fri02]. Only some languages support special constructs to make expressions non-greedy, *e. g.* the regular expression syntax used by *Tcl 8* and *Perl 5* supports  $*?$  for a non-greedy star-closure. A parser using this implementation strategy may only find one parse ( $a(b(c\ d))$ ) for the input sentence ( $c\ d$ ) using this grammar:

```

1 | a: b e? ;
2 | b: c d? ;
3 | e: d ;

```

A second parse tree, which postpones the consumption of  $d$ , *i. e.*  $a(b(c)\ e(d))$ , will *not* be found by this parser implementation.

Which of these alternative interpretations of symbol sequences is chosen is a FDE implementation decision. In Chapter 4 the actual implementation of the FDE will be discussed and in Section 4.3.1.1 this specific decision will be made and explained.

Symbol groups are rewritten by introducing anonymous symbols.

$$\begin{aligned}
 \textit{Image} &\rightarrow \textit{Location} \\
 \textit{Image} &\rightarrow \textit{Location} \beta \\
 \beta &\rightarrow \textit{Color Class}
 \end{aligned}$$

The anonymous symbol, in this case  $\beta$ , inherits the occurrence indicators and embedded alternatives of the symbol group. Notice that the *Image* rule is duplicated to eliminate the sequence once.

### 3.1.2 Atoms

The feature grammar language supports a default set of atom types, *e. g.* several numeric types and strings (see Table 3.1). However, additional types may be supported by the DBMS, for example by using an extension mechanism. To be able for a feature grammar to use these new types they have to be defined and a REG validation language can be added.

```
1 | %atom      image::bitmap {^[01]*$};
```

The *bitmap* type, available in the extension module *image*, is defined. The *bitmap* specific rule defines the REG substitution language from which a valid *bitmap* atom value is a member. Using this regular expression the FDE can validate a specific *bitmap* instantiation. When there is no regular expression the default validation code provided by the system will always accept each terminal of this atomic type.

Next to the atom definitions there are also atom declarations. Using an atom declaration several production rules can be summarized in one command.

```
1 | %atom flt   Prevalent, Saturation;
```

This declaration is a simple shortcut for these two rules:

```
1 | Prevalent  : flt;
2 | Saturation : flt;
```

Notice, once more (see also Section 2.2.2), that the application of this shortcut helps to make the production rules semantically more meaningful, *i. e.* more useful for human consumption.

### 3.1.3 Detectors

The production rules already provide insight in which symbol belongs to the set of terminals and non-terminals, *i. e.* non-terminals appear as the LHS of a rule. To get the set of detector symbols those are explicitly declared.

```
1 | %detector   Color(preceding::Location/url);
2 | %detector   Graphic(preceding::Number/int,
3 |             preceding::Prevalent/flt,
4 |             preceding::Saturation/flt);
5 | %detector   Photo(preceding::Color);
6 | %detector   Skin(preceding::Location/url);
7 | %detector   Faces(preceding::bitmap);
```

The output/input dependency is formally represented by the REG language  $R_i$  for detector  $d_i$ . In the detector declaration for  $d_i$   $R_i$  is represented as a set of regular path expressions. In Section 2.1.1 with the introduction of REG grammars it was already stated that there are many extended regular expression languages, *e. g.* like the one used for the specification of the atom substitution languages. For the specification of the regular path expressions the *XPath* language [W3C01d] is adopted by the feature grammar language.

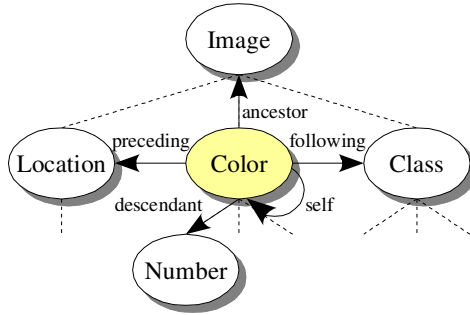


Figure 3.2: The XPath axes

### 3.1.3.1 The XPath Language

The XPath language is a W3C standard for describing paths in a document in the eXtensible Markup Language (XML) [W3C00]. XML, derived from the older Standard Generalized Markup Language (SGML) standard, is becoming a standard document format for data exchange between systems on the WWW. An XML document describes a tree structure of nodes. Each tree node is identified by an element name, encloses an area in the document identified by the opening and closing tags, may have zero or more textual values, may have a set of zero or more attributes, *i. e.* name/value pairs, and may be the parent of zero or more child nodes. The parse trees, as encountered until now, map naturally on this document format.

The basic XPath expression consist of a sequence of steps separated by / or //, where the first indicates “goto a matching child” and the latter “goto a matching descendant”. An example XPath expression is: `preceding::Location/url [xf:starts-with(., "http://") ]`.

The first step, `preceding::Location`, is evaluated in the context of the current node, for example the new *Color* detector node. The evaluation of this step may result in zero or more result nodes. In this case it leads to the *Location* node. The next step, `url[xf:starts-with(., "http://")]`, is evaluated for each of these result nodes. And selects the node *url* which satisfies the predicate. As this is the last step the result of this expression contains only this node. The upcoming paragraphs will describe the basic building blocks of each step: the optional axis, the nodetest and the set of zero or more step qualifiers.

The XPath specification defines a total of 13 axes to traverse from one node to another. These axes describe the whole parse tree from the view point of the context node: the *ancestor*, *descendant*, *following*, *preceding* and *self* axes (see also Figure 3.2). Other axes form sub- and supersets from these basic axes: the *child*, *parent*, *descendant-or-self*, *following-sibling*, *preceding-sibling* and *ancestor-or-self* axes. Most of these axes are forward axes, *i. e.* nodes are traversed in the order in

which they were added to the tree (the document order). In contrast, the *parent*, *ancestor*, *ancestor-or-self*, *preceding*, and *preceding-sibling* axes traverse the nodes in reverse document order.

The axis is used to select the set of possible nodes to move to. This set may be further reduced using a node test: either an exact match on the symbol name or a wildcard match.

The last part of the step specification is a, possibly empty, set of step qualifiers. A step qualifier takes the form of a predicate, *i. e.* a boolean expression on the context of the step. In this expression logical comparisons and an extensive set of functions may be used. The example expression uses the `xf:starts-with` function to check if the value of the terminal *url* uses the HTTP protocol. The value of the terminal is indicated by the single dot, which is the abbreviated syntax for `self::node()`.

All kinds of (user defined) functions are part of the XPath standard. Due to this a full fledged XPath expression may have a selective power that goes beyond a standard regular expression language. To stay in line with the formal feature grammar system of Chapter 2 the XPath expressions used for detector parameter binding are limited to axis steps and symbol name node tests<sup>1</sup>.

This subset of XPath expression language is adopted by the feature grammar language with (initially) two convenient changes:

1. as parameter path expressions should, enforced by the *LPC* rewriting mechanism, point backward into the tree the default axis in the first step of a path expression is `preceding::` instead of `child::`, notice that in all subsequent steps `child::` is still the default;
2. XPath expressions always return a node set, however, in a feature grammar the interest is mainly for, depending on the axis (forward or reverse), the first or last item in the nodeset (sorted on document order).

Using these path expressions the input sentence for a detector  $d_i$  can be selected. When one of the expressions results in an empty node set, *i. e.*  $R_i \not\subseteq \text{path}(t_j)$ , the start condition of the grammar component is not satisfied, and the detector function  $f_i$  will not be executed, which in the leftmost derivation process results in rejection of the detector symbol.

To retrieve the value of a node the XPath function `text()` is used. This function returns a concatenation of all values under the inner node. This allows the use of a non-terminal, which may carry more semantics, to indicate a terminal's value, *e. g.* the XPath expression `Location` is equal to `Location/url`. This helps once more to keep the feature grammar specification more semantic oriented (see Section 3.1.2).

---

<sup>1</sup>The extension to a more complete version of the XPath standard would need a reevaluation of the role of  $R_i$ .

### 3.1.3.2 Detector Confidence

All detectors should at least return one information token (see Definition 2.9). As indicated in Section 2.2.4 the disambiguation process would benefit from knowledge about the detector confidence. By enforcing the output of this confidence level, *i. e.* making it a default rewrite of a detector production rule, the developer of a feature grammar can conceptually write detectors which output an empty sentence. This can be used to allow the use of the detector symbol to model a binary decision, *i. e.* using the partiality of the detector function. The presence of the detector symbol in the parse forest then indicates the success of the function and thus the (maybe in-confident) validity of a concept. The absence of the confidence level, and thus of the symbol, is then used to model the failure.

These feature grammar declarations and rules:

```

1 | %detector      Color(Location);
2 | %detector      Graphic(Number, Prevalent, Saturation);
3 | %detector      Photo(Color);
4 | %detector      Skin(Location);
5 | %detector      Faces(bitmap);
6 | Color          : RGB* Number Prevalent Saturation;
7 | Class          : Graphic | Photo Skin Faces;
```

are rewritten into the following formal rules:

$$\begin{aligned}
 Color &\rightarrow \rho \text{ Number Prevalent Saturation} \\
 Color &\rightarrow \rho \alpha \text{ Number Prevalent Saturation} \\
 Graphic &\rightarrow \rho \\
 Photo &\rightarrow \rho \\
 Skin &\rightarrow \rho \text{ bitmap} \\
 Faces &\rightarrow \rho \text{ int}
 \end{aligned}$$

Notice that the feature grammar does not contain any explicit rules for the *Graphic* and *Photo* detectors, they are added to feature grammar system by the rewrite. The special terminal  $\rho$  describes the confidence value. This value may be seen as an annotation (or attribute) of the LHS (see also Figure 2.5).

### 3.1.4 The Start Symbol

The feature grammar language as defined until now only misses the declaration of the start symbol.

```
| %start      Image(child::Location);
```

This start declaration provides the information to construct the special grammar component  $G_S$ . First of all the dummy start symbol  $S_S$  directly passes control on to the *Image* symbol, in this case a non-terminal. Furthermore, the parameter defines which initial words the special detector function  $f_S$  should generate, *i. e.* in this case one word: a *Location* instantiation. How  $f_S$  determines this instantiation depends on the implementation, *e. g.* ask the user or look for the sentence on the command line of the FDE.

For specification of these parameters a special subset of XPath expressions is, once more, adopted. In this case only forward steps are allowed, *i. e.* the parse tree can only be traversed from the root to its descendants. This also means that the original default axis of XPath is used in these path expressions, *i. e.* `child::`.

## 3.2 The Extended Feature Grammar Language

The core of the language has been defined in the previous section. However, to make the life of a developer easier several shortcuts have been introduced. Other additions provide the developer with the possibility to influence the analysis and usage of the feature grammar by the tools in the Acoi system architecture.

### 3.2.1 Production Rules

#### 3.2.1.1 Additional Sequence Types

Symbol sequences, like the positive or star closure, lead to collections of symbols. The most natural form for storing this collection in a DBMS is a list, as it guarantees that the symbols can be reconstructed in exactly the same order.

```
| Color      : RGB[*] Number Prevalent Saturation;
```

Notice that this rule is equivalent to the original *Color* rule, as list is the default type. However, this may not always be needed and, as keeping information about the order of the symbols costs storage space, the feature grammar developer can give a hint that the collection type may be changed to a set.

```
| Color      : RGB{*} Number Prevalent Saturation;
```

When a list is limited in size another optimization may be to turn the type into a tuple, which has better selection properties.

```
| Color      : RGB<16:16> Number Prevalent Saturation;
```

This example shows another addition to the feature grammar language: it allows to exactly specify, *i. e.* with a lower and upper bound, how many symbols are allowed in the collection (which may be of any type). This case is equivalent to a rule where the *Color* rule contains exactly 16 *RGB* non-terminals. When the lower bound is omitted it defaults to zero.

These constructs are all very tight related to the storage model for the parse trees, and thus will be revisited in Chapter 5.

### 3.2.1.2 Constants

The extended language also allows a constant of a builtin type to be placed as a symbol in the right-hand side of a rule.

```

1 | Segment      : Scene*;
2 | Scene       : Begin End Type;
3 | Type        : "tennis" Tennis;
4 | Type        : "closeup";
5 | Type        : "audience";
6 | Type        : "other";

```

This set of example rules describes the type of scenes found in a video of a tennis match. For each scene the *Segment* detector finds it also determines the type and puts this as a string token in the output sentence. This token now determines which alternative rule of *Type* is validated, *i. e.* for a tennis scene the *Tennis* detector will be called. This spares the need for an explicit whitebox detector (to be discussed in the next section).

```

1 | %detector TennisType [ str = "tennis" ];
2 | Type          : str TennisType Tennis;

```

The other types can be handled in the same fashion. Furthermore, notice that another approach may be to add a normal detector for each type, which would implement the type detection algorithm now present in the *Segment* detector. The need for the constant would then disappear, and it would become possible to have several types for the same scene, *i. e.* ambiguity.

## 3.2.2 Detectors

### 3.2.2.1 Whitebox Detectors

One step in the annotation extraction process is the combination of the low-level features into high-level concepts. One way to do this combination is through a binary decision rule. Recall the description of the XPath language: the last part of the step specification consists of a, possibly empty, set of step qualifiers. Each step qualifier is



a predicate on the context of the step. *Whitebox detectors* provide the preferred way to embed such a predicate in the feature grammar.

In step qualifiers all XPath expressions are allowed, but not all of them will result in a boolean value. For these situations the XPath specification [W3C01d] provides these rules to derive the predicate truth value:

1. if the result of the expression is an empty node set, the predicate truth value is *false*;
2. if the result is one numeric value, this value is rounded and compared to the context position, *i. e.* the position of the context node in the processed sequence of nodes, this will always be *one* as the processed sequence contains only the detector node;
3. if the result is one boolean value, the predicate truth value is equal to this boolean value;
4. if the result node set contains at least one node, the predicate truth value is *true*;
5. otherwise there is a serious error and a runtime error will be raised.

When the predicate truth value equals *true* the detector symbol will be accepted, and otherwise rejected.

This partial feature grammar shows the simple decision rule for the *Photo* detector:

```

1 |%detector      Photo      [
2 |                Number      > 200
3 |                and Prevalent < 0.26
4 |                and Saturation < 0.67 ];

```

On the basis of the number of colors, the existence of a prevalent color and the average saturation of the colors the image is classified as a photo (or not). But XPath allows also more advanced expressions, *e. g.* these quantified expressions to extract a color map concept:

```

1 |%detector      ColorMap [
2 |                some $RGB in RGB satisfies
3 |                $RGB/Red != $RGB/Green
4 |                or $RGB/Red != $RGB/Blue ];
5 |%detector      GrayMap [
6 |                every $RGB in RGB satisfies
7 |                $RGB/Red = $RGB/Green
8 |                and $RGB/Red = $RGB/Blue ];
9 |Color          : RGB* Map Number Prevalent Saturation;
10|Map           : ColorMap | GrayMap;

```



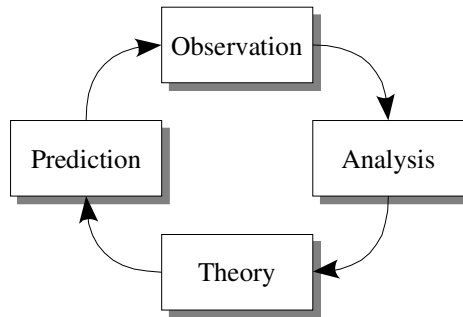


Figure 3.3: Empirical cycle

```

3 |         and Prevalent < 0.26
4 |         and Saturation < 0.67 ];
  
```

This code snippet is equivalent to the previous *Photo* whitebox detector declaration.

In the same vain other whitebox plugins can be developed. The Acoi system provides a default library function to instantiate templates with embedded XPath expressions.

### 3.2.2.3 Classifiers

*Classifiers* are an alternative to whitebox detectors to validate a concept. Instead of decision rules, most likely constructed by a human expert, they provide bridges to machine learning algorithms.

Machine learning algorithms can be characterized by the cycle shown in Figure 3.3. The process starts with a number of observations. These observations are analyzed to find patterns. If patterns are found a theory, or hypothesis, is formulated to explain the pattern. This theory is used to predict new phenomena that can be verified by new observations.

Take for example this classifier:

```

1 | %classifier bpnn::Faces(Skin);
  
```

The classifier *Faces* makes use of the plugin *bpnn*. The *bpnn* plugin, written by an expert, manages *back propagation neural networks*. The plugin can handle the generic steps of the classification cycle for this specific machine learning algorithm.

A back propagation neural network learns incrementally, *i. e.* the theory is updated after each observation is seen by the algorithm. The observations consist of a set of input sentences the developer feeds to the FDE. An observation for this classifier would contain the fact that the image contains one face. When the *Faces* detector is encountered its input, the *Skin/bitmap* terminal, is selected from the parse tree. The

Classifier plugin name	Description
decrules	Learn/use C4.5 decision rules [Qui93]
bpnn	Train/use a neural network [Mit97]

Table 3.3: Default classifier plugins

developer provided the expected output of the *Faces* detector: the sentence  $int(1)$ . The neural network is then trained by analyzing the actual and the expected output. On the basis of the error between these the internal parameters of neural network will be adapted.

When the developer did not provide an observation the theory is used to predict the output. The theory is not updated until new observations are fed to the classifier.

Other algorithms may learn per batch. In this case the algorithm will collect and store the observations. When there is no observation the theory will be build using this collected data. It will then use this, just-in-time, formulated theory to predict the actual output.

Formally the classifier is split into two detectors:

$$\begin{aligned}
 Faces &\rightarrow int\ Faces.analyze \\
 Faces &\rightarrow Faces.predict \\
 Faces.analyze &\rightarrow \rho \\
 Faces.predict &\rightarrow \rho\ int
 \end{aligned}$$

*Faces* is a non-terminal, while *Faces.analyze* and *Faces.predict* are both detectors. Due to the fact that an integer, *i. e.* the observation, is present in the parsed sentence a specific alternative is chosen. The first alternative will take this integer as input and train the neural network, while the second alternative will use the net to predict the value of the integer. The natural value for  $\rho$  in the case of an observation is 1.0, *i. e.* its confidence is high as its been provided by the developer.

The Acoi implementation provides a default set of classifier plugins. The *bpnn* plugin is an example of an incremental learning algorithm. *Decision rules*, a batch learning algorithm, are available through the *decrules* plugin.

### 3.2.3 The Start Symbol

#### 3.2.3.1 References

Until now there was always only one parse forest. But the main goal is to build and maintain a database of such parse forests, or in fact its building blocks: parse trees. It is possible to keep all these parse trees independent of each other in the construction phase, and to postpone resolving dependencies to the moment of insertion

into the database. In fact this was done in early versions of the Acoi system (see Chapter 7). However, as this knowledge would not be explicit and the system could thus not exploit it, the coding burden would be completely passed on to the developer.

Take for example the WWW multimedia search engine. On the WWW HTML pages contain the links which connect all the web objects together. By adding a detector which parses the HTML and extracts the links the structure of the WWW can be described by a feature grammar.

### Example 3.1.

```

1 | %start      WebObject(Location);
2 | %detector   HTML_type [ Location\[ends-with(., ".html")\] ];
3 | %detector   HTML(Location);
4 | %atom str   Title;
5 | WebObject  : Location WebBody;
6 | WebBody    : HTML_type HTML;
7 | HTML       : Title? WebObject*;

```

□

However, the WWW is not a tree but a graph. And this graph may contain cycles, which would lead to infinitely deep parse forests. This would make the feature grammar impractical. How to prevent the FDE from reanalyzing an object over and over again, thus storing redundant information in the parse tree? The developer may add detectors, which check if a web object has already been analyzed, and alter production rules, *i. e.* add optionality such that a partial parse tree containing foreign key information is also valid.

Once more the FDE can manage these dependencies better if they are made explicit. The HTML feature grammar illustrates when a cycle in the graph may be introduced: the start symbol *WebObject* is reused in a production rule. The start declaration already contains information on the data items an initial sentence should contain to start a parsing process. When this information is viewed as key information the FDE gets the ability to check the database for the existence of this specific (partial) parse tree. However, if the parse tree does not exist yet this key information is also enough to start building the parse tree.

This solves the case when the start symbol is reused, however, in other cases information would still be redundant. Keyword search can be added to the WWW application by extending the HTML module with these production rules:

```

1 | %atom str   Word;

```

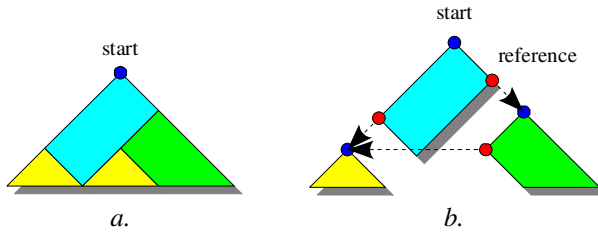


Figure 3.4: Parse tree structure (a) without and (b) with references

```

2 | HTML      : Title? Body WebObject*;
3 | Body     : KeyWord*;
4 | KeyWord  : Word Synset*;
5 | Synset   : Synonym* Hypernym* Hyponym*;
6 | Synonym  : Keyword;
7 | Hypernym : Keyword;
8 | Hyponym  : Keyword;

```

From the body of the HTML page keywords are extracted. These keywords are related with WordNet synsets [CSL01], which allow to expand the search with synonyms, hypernyms and hyponyms. If for each keyword encountered on the WWW a new partial parse tree would be added the database would explode in no time. Using the WWW case study a sample of 150,000 web objects lead to 850,000 keywords which were on average 40 times reused. This explosion is prevented by permitting multiple start symbols, *e. g.* a start declaration for the *Keyword* symbol is added.

```

1 | %start    Keyword(Word);

```

The FDE can now use the same strategy for binding a parse tree for this start symbol. A start symbol which reoccurs in the RHS of a production rule is also known as a *reference*. This emphasized in the language by adding a & prefix to the symbol occurrence. Figure 3.4 shows how this addition of references to start symbols affects the parse tree structure.

This conversion of a tree into a graph structure has also implications for the XPath expressions as used to specify either a detector input or a whitebox predicate. Resolving these paths could lead to endless loops, *e. g.* the XPath expression `preceding::Location` could reenter the same tree over and over again by encountering the same *WebObject* symbol again. To resolve this, references have to be crossed explicitly, which basically means that a XPath expression can not cross over to a start symbol. This splits the global parse tree, as shown in Figure 3.4.a, into several parse trees combined by references in a graph, as shown in

Figure 3.4.b. Take once more the XPath expression `preceding::Location`. Using this convention only *Location* symbols in the local parse tree can be found. To cross a reference, *i. e.* a start symbol, the XPath expression would look as follows: `preceding::&WebObject//Location`. The `&` prefix, which is not part of the XPath standard, allows the adapted XPath interpreter to resolve the `//Location` path in the parse tree to which a *WebObject* reference is bound. In the next chapter this feature grammar specific addition to the XPath language will be translated back into native XPath expressions, however, this depends on the implementation specific representation of feature grammar parse trees in XML documents.

A drawback of this is the reintroduction of deadlock situations when binding the detector parameters. The XPath expression may lead to a (indirect) self-reference, *i. e.* a detector needs access to  $x_r$  and thus violates the *lPC* regulated rewrite rules and thus the deadlock prevention strategy (see Section 2.2.3.1). As there is no general solution for this, an exception is passed on to the detectors implementation. The developer may, for example, know how to retrieve the requested token from  $x_r$  or which default value to pick.

The use of references is also related to the quasi-nodes introduced in Section 2.2.4. Quasi-nodes were introduced to handle the binding of ambiguous parameters to the right detector calls. Start symbols are always detectors, remember the introduction of  $S_S$ . The reference is the quasi-root containing the (ambiguous) binding information. The refereed  $S_S$  is the quasi-foot.

Notice also that with the addition of multiple start symbols the feature grammar now describes formally a set of feature grammar systems where one is instantiated on the moment a start symbol is chosen, *i. e.*  $S_S \rightarrow \dots$  is generated on-the-fly.

### 3.2.4 Feature Grammar Modules

One of the main goals of feature grammar systems is to keep detectors generic, so they are easily applicable in another context. This goal is further supported by the concept of feature grammar modules. In such a module related detectors, atoms and their production rules are grouped and available for reuse by other feature grammars.

```

1 | %module      WWW;
2 | %start      WebObject(Location);
3 | %detector   WebHeader(Location);
4 | %atom       www::url {^http://([^ :/*]*)(:[0-9]*)?/?(.*)$};
5 | %atom       temporal::date;
6 | %atom       url Location;
7 | %atom       date Modification;
8 | %atom       lng Length;
```

```

9 | WebObject   : Location WebHeader WebBody;
10| WebHeader   : Modification Length;

```

This WWW feature grammar module gives basic support for the WWW. The module offers a start symbol, *WebObject*, and a detector to retrieve basic HTTP header information, *WebHeader*. Furthermore, it contains the definition of two atomic types: *url* and *date*. These feature grammar statements show how the Image feature grammar uses the WWW module:

```

1 | %module     Image;
2 | %use       WWW;
3 | WebBody    : Image;
4 | Image      : Color Class;

```

In this example both the *Local* and *WebBody* symbols are unique for the union of the Image and WWW feature grammars. However, when this is not the case naming conflicts arise. Assuming that the module names are unique, they are used as namespaces to resolve such conflicts. An explicit namespace is added as a prefix to the symbol name, *e. g.* *WWW :: WebBody*.

Declarations and definitions always happen in the scope of the active namespace, *i. e.* the module in which the declaration or definition takes place. This means that atoms and detectors can not be redefined by another feature grammar module. However, additional production rules are allowed, *i. e.* to add an alternative view. The Image feature grammar uses this construction to add the Image view on a *WebBody* symbol. In this way feature grammars are easily extensible.

### 3.2.5 Change Detection

In Chapter 1 several sources of change were identified:

**internal sources** either the dependency description or the implementation of detector functions change;

**external sources** the source data changes, the system may check for these changes by itself (polling) or may be alerted by the librarian.

The upcoming sections will describe the features the feature grammar language offers to the Acoi system to detect some of the changes.

**Versions** One of the triggers for incremental maintenance of a set of parse trees are changes in detector implementations and the production rules. Changes in implementation may not always be reflected in the feature grammar, *e. g.* a bug fix does not have to lead to changes in the output, and thus the production rules, of a detector. To indicate these changes detectors have a version.



```
1 | %version      Skin 1.23.4;
```

Increments of the version indicate a change in implementation and forces the FDS to determine if any persistent stored parse trees are affected and should be updated. The priority which is assigned to this change is depending on the level of change: major (the first number), minor (the second number) or service (the last number).

**Polling** Detector versions handle the notification of internal detector changes. To poll for external changes an (optional) poll detector related to a start symbol is added. The arguments of this detector are at least the required initial sentence as part of the start declaration.

```
1 | %detector      WebObject.poll(Location,WebHeader/Modification);
```

An implementation of this poll detector will be called by the FDS (see Chapter 6) on a regular basis to check if a stored parse tree needs to be updated. When no poll detector is defined the default poll detector is used, which always returns *true*, *i. e.* always indicates an external change.

### 3.3 Discussion

This chapter described the feature grammar language: a convenient notation for feature grammar systems, including several syntactic shortcuts. The language aims at being readable, although a clear understanding of the use of CS grammars is still needed to construct one, *e. g.* understand the relationship between rules, trees and regular path expressions. This should not prevent the usability of the language as its target audience are DMW and annotation extraction algorithm developers, who will be schooled in basic computer science topics.

With the rise of XML more and more languages are expressed in XML and thus have the advantage of standardized tools like SAX parsing and DOM parse trees. But those languages tend to be verbose and non-transparent. However, an XML-ized version of the feature grammar language may be constructed as an intermediate format to gain those advantages. This language could be defined as an extension on one of the upcoming XML schema languages (see Section 5.2.1).



## Chapter 4

# Feature Detector Engine

... *Sharpen line forty-eight between twenty point twenty-seven.*  
... *Profile trace.*  
... *Stop. Back up.*  
... *Stop.*  
... *Enhance.*  
... *Seesaw.*  
... *Stop!*  
... *Enhance.*  
... *Enhance.*  
...  
... *Hey!*

Rick Deckard – *Blade Runner*

Grammars are mostly used to validate a sentence's membership of a specific language. This process of validation, called parsing, may lead to the construction of a parse tree, *i. e.* an internal representation of the structure of the sentence. The parsing process forms the heart of the *Feature Detector Engine* (FDE). During this process the FDE encounters detector symbols, binds their input sentence, executes the associated algorithms, and validates their output sentence (the actual parsing). There are many different parsing algorithms. Yet only a few of these algorithms are suited for processing a feature grammar system. The next section will review the predominant parsing algorithms for CF grammars. The remainder of the chapter is focused on the implementation decisions for the FDE, based on a parsing algorithm well suited for feature grammar systems.

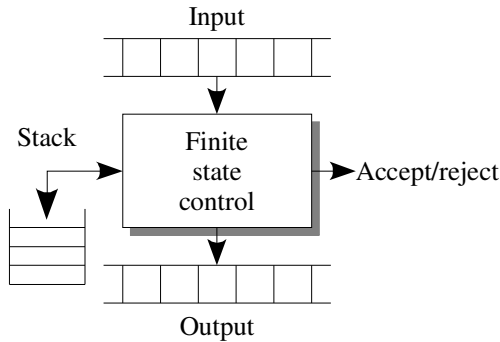


Figure 4.1: The basic PDT

## 4.1 A Parser Primer

The membership of a sentence  $w$  in a language  $L(G)$  can be checked using a parser, which is especially constructed for that grammar. Parsers are based on specific types of (finite) automata. For each grammar type or language family a specific type of (finite) automata is used. A REG language is parsed using a *finite automaton* (FA), while a CF language needs the more advanced functionality of a *push-down automaton* (PDA).

The basic automaton is an acceptor, *i. e.* accepts or rejects an input sentence. When the automaton also has additional output, *e. g.* a structural description of the symbols encountered, it is called a transducer.

The transducer version of the PDA, *i. e.* the *push-down transducer* (PDT), is basically a FA with a stack-based memory and an output tape, see Figure 4.1. The read head of the input tape can read one symbol at a time from the tape. The same goes for the write head of the output tape: it can write one symbol at the time. Both the read and write head advance to the next symbol after reading or writing. The end of both the input sentence and the stack is indicated by the special symbol  $\$$ . The stack-based memory allows the PDT to push and pop symbols on or from the stack in a last-in first-out (LIFO) way.

The transducer description includes a set of states. The states of the PDT are restricted to these types, which are directly related to its capabilities:

**start** the PDT;

**read** a single symbol from the input tape;

**write** a symbol on the output tape;

**push** a single symbol on top of the stack;

**pop** the topmost symbol from the stack;

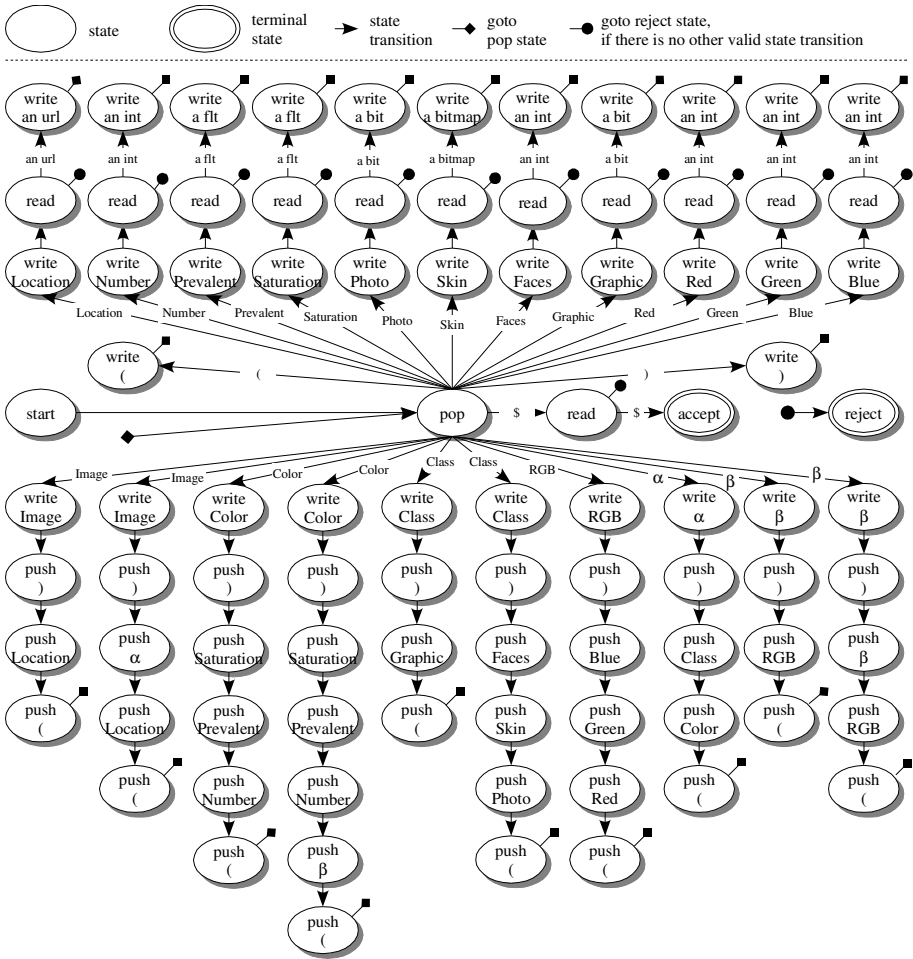


Figure 4.2: A non-deterministic PDT for the Image CF grammar

**accept** the input string and stop;

**reject** the input string and stop.

Using these states and a set of state transitions the PDT can be used to implement a parsing algorithm. To illustrate this Figure 4.2 shows a PDT for this example CF grammar:

**Example 4.1.**

*Image* → *Location*  
*Image* → *Location*  $\alpha$ ;  
 $\alpha$  → *Color Class*  
*Color* → *Number Prevalent Saturation*  
*Color* →  $\beta$  *Number Prevalent Saturation*  
 $\beta$  → *RGB*  
 $\beta$  → *RGB*  $\beta$   
*RGB* → *Red Green Blue*  
*Class* → *Graphic*  
*Class* → *Photo Skin Faces*  
*Location* → *url*  
*Number* → *int*  
*Prevalent* → *flt*  
*Saturation* → *flt*  
*Graphic* → *bit*  
*Photo* → *bit*  
*Skin* → *bitmap*  
*Faces* → *int*  
*Red* → *int*  
*Green* → *int*  
*Blue* → *int*

□

The PDT starts with the start symbol *Image* on its stack. After the *start* state the controller moves to the *pop* state, where the *Image* symbol is popped from the stack. Based on this symbol the PDT chooses a transition to a next state. Figure 4.3 shows the (possible) condition of the PDT the second time it visits the *pop* state. The *accept* state is reached when both the stack and the read tape are empty (reached by both popping and reading a \$ symbol). The output tape will then contain a textual description of the parse tree, e.g. *Image* ( *Location* *http://...*  $\alpha$  ( *Color* ( *Number* 29053 ... ) ... ) ).

On its way to the *accept* or *reject* state the controller has to choose a move to a next valid state. In the case of either a pop or a read state, the valid options are determined by respectively the symbol on top of the stack or under the read head of the input tape. If there is always precisely one valid move, the PDA or PDT is called *deterministic* (DPDA or DPDT), when there are more valid options the PDA or PDT is called *non-deterministic* (NPDA or NPDT). The PDT in Figure 4.2 is non-deterministic, e.g. after a pop of the *Image* symbol there are two valid moves (due to the alternative

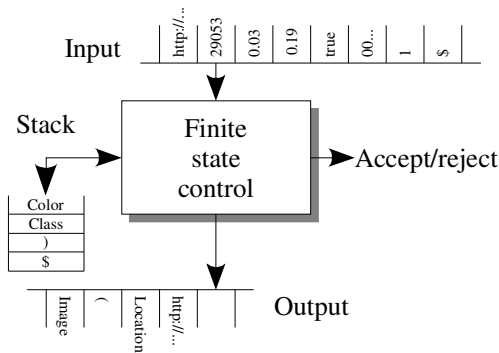


Figure 4.3: The condition of the PDT

production rules). In such a case the NPDT follows both choices, *i. e.* it is in several states at the same time. However, some of these choices will never lead to an accept state and will thus not yield a valid parse tree. On the other hand several choices may lead to valid parse trees. When more than one valid parse tree can describe the same input sentence the grammar is *ambiguous* (see also Section 2.1.2.4).

A PDA or PDT can be automatically generated from a CF grammar. The algorithm implemented in the PDT and sketched above performs a top-down parsing strategy – an intuitive method. In the next section a short overview of other methods and their most important properties are given.

### 4.1.1 More Parsing Algorithms for Context-free Grammars

A *top-down algorithm*, as has been sketched and implemented in a PDT in the previous section, starts with the start symbol and tries, by traversing the production rules in a smart way, to *produce* the input sentence. A *bottom-up algorithm* works just the other way around: it starts with the input sentence and tries to *reduce* it back to the start symbol.

Both algorithms can be used to parse the input sentence of Figure 2.3 using the example CF grammar. Due to space considerations a simplified version of the grammar is used (see Figure 4.4) to illustrate the behavior of the two basic algorithms.

Furthermore, the end of the input sentence is once more indicated with the special `$` terminal. This terminal reappears at the end of the rule for the start symbol, *Im*. This rule takes care for a correct detection of the end of the input sentence.

Figure 4.5 shows the steps taken by a specific top-down parsing algorithm, *i. e.* a depth-first algorithm. The information used by this algorithm consists of two parts: the active rules and the sentence. The current position within a rule (or the sentence) is indicated with a bullet (`•`). This bullet splits a rule in a *match* part and a *prediction*

part. The algorithm always follows one alternative. For example, in step  $j$  the  $Gr$  alternative of the  $Cl$  non-terminal is tried, only when this one fails the next alternative,  $Ph$ , is tried in step  $k$ . When none of the active rules has a prediction left and the input sentence is also completely consumed, and both these conditions are enforced by the start rules for  $Im$ , the input sentence can be accepted. By keeping track of the active rules the parse tree can be gradually build during the parsing process.

The application of a bottom-up algorithm, *i. e.* a breadth-first algorithm, is shown in Figure 4.6. The breadth-first algorithm inspects several possible parses in each step. Each parse under consideration is represented by a stack with attached partial parse trees. Each step in this algorithm consists of two phases. In the *shift* phase the next input symbol is appended to each stack. The following *reduce* phase then examines all stacks and if they allow one or more reductions copies of the stack are made and the reductions applied to them. These reductions produce the partial parse trees. The first reduction is applied in step  $d$ : the shift phase added the  $Sa$  token to the first stack, which enabled the reduction of the  $Nu Pr Sa$  symbol sequence to the  $Co$  non-terminal. This process continues until there is no input left. In the total of six (partial) trees left in step  $h$  there is only one which contains the start symbol,  $Im$ , which is also the root of the parse tree.

Both parsing algorithms process the input sentence from left to right, *i. e.* they are *directional*. However, there are also some algorithms which are *non-directional*. These methods may access the input sentence in any order they like. This requires the input sentence to be completely available upfront, while conventional algorithms work on a stream of tokens. To illustrate this: the breadth-first used algorithm in Figure 4.6 is well suited for on-line parsing where a source outside of the parser produces the input sentence gradually.

Table 4.1 shows a taxonomy of parsing algorithms (based on [GJ98], where these algorithms are described in more depth). The taxonomy shows that directional parsing algorithms can be further grouped. The description of top-down and bottom-up

	The simplified CF grammar:	The simplified input sentence:
1	$Im \rightarrow Lo \$$	$Lo Nu Pr Sa Ph Sk Fa \$$
2	$Im \rightarrow Lo \alpha \$$	
3	$\alpha \rightarrow Co Cl$	
4	$Co \rightarrow Nu Pr Sa$	
5	$Co \rightarrow \beta Nu Pr Sa$	
6	$\beta \rightarrow RGB$	
7	$\beta \rightarrow RGB \beta$	
8	$RGB \rightarrow R G B$	
9	$Cl \rightarrow Gr$	
10	$Cl \rightarrow Ph Sk Fa$	

Figure 4.4: The simplified CF grammar and input sentence



	<i>Active rules</i>	<i>Sentence</i>
a.	1. $Im \rightarrow \bullet Lo \$$	$\bullet Lo Nu Pr Sa Ph Sk Fa \$$
b.	1. $Im \rightarrow Lo \bullet \$$	$Lo \bullet Nu Pr Sa Ph Sk Fa \$$
c.	1. $Im \rightarrow \bullet Lo \alpha \$$	$\bullet Lo Nu Pr Sa Ph Sk Fa \$$
d.	1. $Im \rightarrow Lo \bullet \alpha \$$	$Lo \bullet Nu Pr Sa Ph Sk Fa \$$
e.	1. $Im \rightarrow Lo \alpha \bullet \$$ 2. $\alpha \rightarrow \bullet Co Cl$	$Lo \bullet Nu Pr Sa Ph Sk Fa \$$ $Lo \bullet Nu Pr Sa Ph Sk Fa \$$
f.	2. $\alpha \rightarrow Co \bullet Cl$ 3. $Co \rightarrow \bullet Nu Pr Sa$	$Lo \bullet Nu Pr Sa Ph Sk Fa \$$ $Lo \bullet Nu Pr Sa Ph Sk Fa \$$
g.	3. $Co \rightarrow Nu \bullet Pr Sa$	$Lo Nu \bullet Pr Sa Ph Sk Fa \$$
h.	3. $Co \rightarrow Nu Pr \bullet Sa$	$Lo Nu Pr \bullet Sa Ph Sk Fa \$$
i.	3. $Co \rightarrow Nu Pr Sa \bullet$	$Lo Nu Pr Sa \bullet Ph Sk Fa \$$
j.	2. $\alpha \rightarrow Co Cl \bullet$ 3. $Cl \rightarrow \bullet Gr$	$Lo Nu Pr Sa \bullet Ph Sk Fa \$$ $Lo Nu Pr Sa \bullet Ph Sk Fa \$$
k.	3. $Cl \rightarrow \bullet Ph Sk Fa$	$Lo Nu Pr Sa \bullet Ph Sk Fa \$$
l.	3. $Cl \rightarrow Ph \bullet Sk Fa$	$Lo Nu Pr Sa Ph \bullet Sk Fa \$$
m.	3. $Cl \rightarrow Ph Sk \bullet Fa$	$Lo Nu Pr Sa Ph Sk \bullet Fa \$$
n.	3. $Cl \rightarrow Ph Sk Fa \bullet$	$Lo Nu Pr Sa Ph Sk Fa \bullet \$$
o.	1. $Im \rightarrow Lo \alpha \$ \bullet$	$Lo Nu Pr Sa Ph Sk Fa \$ \bullet$

Figure 4.5: A top-down parse for the example CF grammar

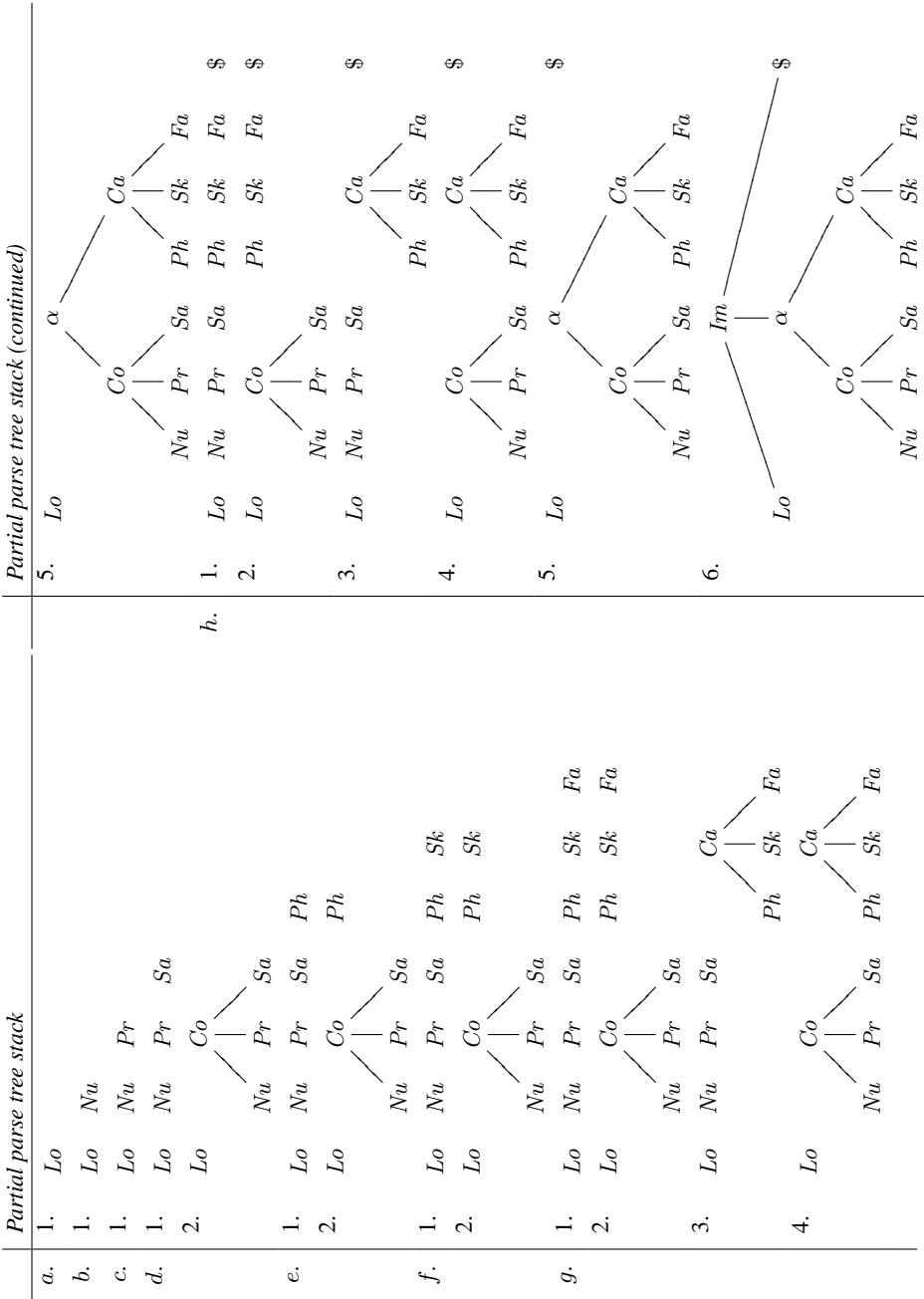


Figure 4.6: A bottom-up parse for the example CF grammar

	<i>top-down</i>	<i>bottom-up</i>
<i>non-directional</i>	Unger parser	CYK parser
<i>directional</i>	predict/match automaton 1. depth-first 1.a. backtracking 1.b. exhaustive backtracking 2. breadth-first 2.a. deterministic breadth-first 2.a.1. LL( $k$ )	shift/reduce automaton 1. depth-first  2. breadth-first 2.a. restricted breadth-first 2.a.1. Earley 2.a.2. Tomita 2.b. deterministic breadth-first 2.b.1. LR( $k$ ) 2.b.2. SLR(1) 2.b.3. LALR(1)

Table 4.1: A taxonomy of parsing algorithms

parsing already showed that either a depth-first or a breadth-first search strategy can be applied. Research on efficient algorithms, *i. e.* algorithms with linear complexity, has mainly focused on bottom-up, directional and deterministic methods. They use some form of look-ahead, *i. e.* one or more tokens of the input sentence, to decide which production rule to follow. Bottom-up parsers are more powerful for deterministic parsing as they will use more context, *i. e.* have seen more of the input sentence, before making a decision [Par93]. Although these variants are not shown in this table, deterministic algorithms can be generalized, *i. e.* made non-deterministic, by adding (pseudo-)parallel features [Lan74, Rek91].

## 4.2 Parsing Feature Grammar Systems

As summarized in the previous section, there exists a plethora of techniques to parse sentences and validate their membership of a CF language. However, are these parsing techniques also applicable to grammar systems and feature grammar systems in particular?

Grammar systems have been mostly studied in theory, however, some first steps have been taken to use them for practical purposes [PS98]. One step in this process is to investigate the use of (existing) parsing algorithms. In [MM96] the authors take a first step by investigating the deterministic subclass of grammar systems as a basis for parsing. However, as identified in the previous chapters the application domain of feature grammar systems benefits from non-determinism. In this section a suitable non-deterministic parsing algorithm for feature grammar systems is selected.

In a grammar system parsing operates on two different levels: the global grammar system, *i. e.* transfer of control between grammar components, and the local grammar

component, *i. e.* the actual parsing of a (partial) sentence. This is also reflected in the basic ingredients of the derivation process for a feature grammar system, as formally described in Chapter 2:

**bind** a grammar component  $G_i$  gathers its input sentence by binding its REG expression  $R_i$  with the *path* metamorphosis of the partial parse tree  $t_j$ ;

**detect** a detector function  $f_{d_i}$  maps the input into a, just-in-time produced, partial sentence;

**parse** the partial sentence  $z_{d_i}$  is parsed, and thus validated, by the corresponding grammar component  $G_i$ , resulting in an extended partial parse tree;

**(un)nest** the yield  $z$  of the partial parse tree derived using  $G_i$  contains the words in  $z_{d_i}$  enveloped by arbitrary sequences of detectors, as described by the REG language derived by  $f_D$  from  $z_{d_i}$ .

The just-in-time behavior determines where the control of the system lies initially: with the “dummy” detector  $S_S$ . This implies a top-down algorithm, which is confirmed by the needs of the binding step. As this last step depends on the availability of a (partial) parse tree which can be transformed into a set of neat paths in which the regular expression,  $R_i$ , can be resolved. The nesting of detector components asks for a component to hand over the control to another component. As stated in Section 2.2.3.1, the *lpc* rewriting mechanism has been added to prevent deadlock situations and prefers leftmost derivation on the control or grammar system level. So the grammar system level calls for a top-down leftmost, *i. e.* directional, parsing algorithm.

Within a component a complete sentence  $z_{d_i}$  is available, which in principle may be parsed with any of the non-deterministic parsing algorithms described in the previous section. What complicates this parsing process is the nesting of detectors. Upon encountering a detector there are two alternatives: (1) delay validation of the detector until the stop condition of the grammar component is satisfied, or (2) first validate the detector and then go on with validation of the output sentence. The first alternative closely follows the formal derivation method as described in Chapter 2, but does not fit within any CF parsing algorithm. The second alternative allows the use of a standard top-down algorithm, *i. e.* control is handed over to the detector and handed back after validation.

Both levels allow, and even favor, the use of an adapted top-down algorithm. There are even more, general, reasons for the use of a top-down instead of a bottom-up algorithm:

1. people parse sentences top-down [AS88, RJ99], *i. e.* debugging a top-down parse is thus more intuitive for a feature grammar developer;

2. these algorithms provide better support for the addition of semantic actions [Par93], *e. g.* detector functions, as they provide more context information, *i. e.* the same reason why detector parameters can be bound;
3. the same context gives also easy support for informative error reporting [GJ98], which, once more, helps during debugging.

A top-down algorithm has been implemented in the current version of the FDE and will be described in more detail in the next subsection.

### 4.2.1 Exhaustive Backtracking for Feature Grammar Systems

The top-down algorithm used within the FDE is based on an *exhaustive backtracking* algorithm. Backtracking indicates depth-first behavior: one alternative is chosen and followed until it either fails or succeeds. Upon failure the algorithm backtracks until an untried alternative is found and tries that one. The adjective exhaustive means that the algorithm used by the FDE also backtracks when the alternative is successful. By doing this the algorithm handles ambiguous feature grammars and constructs the parse forest.

To show the algorithm in action a basic feature grammar is constructed in relatively the same manner as the CF grammar in Figure 4.4. Figure 4.7 shows this simplified feature grammar. The same figure shows the formal feature grammar system derived from the grammar. The rewrite of the rules involves the introduction of anonymous symbols, *i. e.*  $\alpha$  and  $\beta$ , for the handling of a symbol group and sequences.

Figure 4.8 shows the various parsing actions, grouped per controlling grammar component. The actions are directly associated with the basic ingredients described before. A component which gets control starts with an empty output sentence. The REG expression associated with the detector is *binded* in the parse forest (see Figure 4.9 for the basic AND/OR graph) resulting in the input sentence. The output sentence is then filled by the *detect* action, *i. e.* the mapping function is applied. The parsing process of this sentence is then interleaved with control transfers to nested detectors. To be able to resume the parsing process the output sentence is pushed on the stack of sentences under inspection when control is transferred to a nested detector. This allows the delayed evaluation of the remainder of the stop condition by popping this stack when control is transferred back.

The exhaustive backtracking behavior of the algorithm is illustrated in step *a*, when the second alternative rule of *Im* is considered (and found valid in step *k*), even after the first rule has already been found valid.

Most algorithms pose limitations on the grammars they can parse. This is also true for a top-down parsing algorithm like exhaustive backtracking. The next subsection will investigate these limitations. The last two subsections will look at optimizations to make the algorithm more efficient by avoiding unnecessary backtracking and doing double work.

The simplified feature grammar:		The simplified feature grammar system:	
1	%start	Im(Lo) ;	$\Gamma = ($
2	%detector	Co(Lo) ;	$D = \{S_S, Co, Gr, Ph, Sk, Fa\},$
3	%detector	Gr(Nu, Pr, Sa) ;	$N = \{Im, \alpha, Cl, \beta, RGB\},$
4	%detector	Ph(Nu, Pr, Sa) ;	$T = \{\rho, Lo, R, G, B, Nu, Pr, Sa, \$\},$
5	%detector	Sk(Lo) ;	$P_N = \{(Im \rightarrow Lo), (Im \rightarrow Lo \alpha), (\alpha \rightarrow Co Cl),$
6	%detector	Fa(Sk) ;	$(\beta \rightarrow RGB), (\beta \rightarrow RGB \beta),$
7	%atom	Lo, R, G, B, Nu, Pr, Sa, Sk, Fa ;	$(RGB \rightarrow R G B), (Cl \rightarrow Gr), (Cl \rightarrow Ph Sk Fa)\},$
8	Im	: Lo ( Co Cl ) ? ;	$G_{Co} = (V, P_{Co} = \{(Co \rightarrow \rho \beta Nu Pr Sa \$), (Co \rightarrow \rho Nu Pr Sa \$)\}$
9	Co	: RGB* Nu Pr Sa ;	$\cup P_N, R_{Co} = ( \cdot \cdot Lo), f_{Co}),$
10	RGB	: R G B ;	$G_{Gr} = (V, P_{Gr} = \{(Gr \rightarrow \rho \$)\} \cup P_N, R_{Gr} = ( \cdot \cdot Nu) + ( \cdot \cdot Pr) + ( \cdot \cdot Sa),$
11	Cl	: Gr   Ph Sk Fa ;	$f_{Gr}),$
			$G_{Ph} = (V, P_{Ph} = \{(Ph \rightarrow \rho \$)\} \cup P_N, R_{Ph} = ( \cdot \cdot Nu) + ( \cdot \cdot Pr) + ( \cdot \cdot Sa),$
			$f_{Ph}),$
			$G_{Sk} = (V, P_{Sk} = \{(Sk \rightarrow \rho \$)\} \cup P_N, R_{Sk} = ( \cdot \cdot Lo), f_{Sk}),$
			$G_{Fa} = (V, P_{Fa} = \{(Fa \rightarrow \rho \$)\} \cup P_N, R_{Fa} = ( \cdot \cdot Sk), f_{Fa}),$
			$G_S = (V, P_S = \{(S_S \rightarrow \rho Im \$)\} \cup P_N, R_S = \emptyset, f_S),$
			$S = S_S$
			)

Figure 4.7: The simplified feature grammar (system)



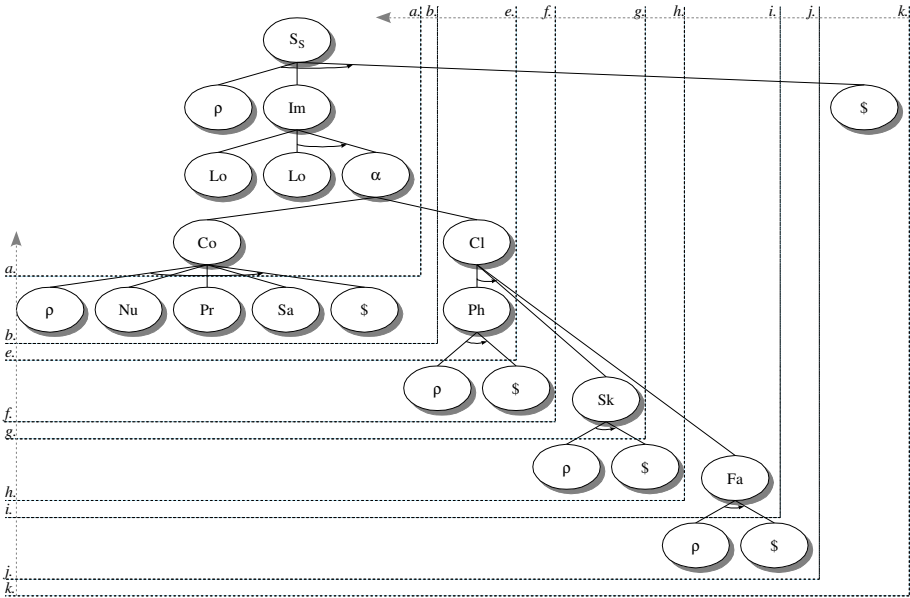


Figure 4.9: Partial parse forests for the steps in Figure 4.8

**4.2.1.1 Left-recursion**

The major limitation of top-down methods are their inability to handle left-recursive grammars. To illustrate this problem consider this, direct left-recursive, grammar:

$$S \rightarrow S a$$

$$S \rightarrow b$$

To validate the production rule of  $S$  the parser will try to validate  $S$  over and over again, thus entering an endless loop. Fortunately standard rewrite rules are available for left-recursion elimination. For example this grammar generates the equivalent language:

$$S \rightarrow b \alpha$$

$$S \rightarrow b$$

$$\alpha \rightarrow a \alpha$$

$$\alpha \rightarrow a$$



This is the same result as when the *right-recursive* interpretation for symbol sequences is used, *i. e.* both grammars are equivalent with this rule in the extended notation of the feature grammar language (see Section 3.1.1):

$| S \quad : b a^*$

These rewritten grammars show that any finite valid input sentence will have to start with a  $b$  terminal, followed by an optional tail of  $a$  terminals.

Indirect left-recursion is the case where left-recursion takes place after encountering several other non-terminals, *e. g.* as is the case in this grammar:

$$\begin{aligned} S &\rightarrow A B c \\ B &\rightarrow C d \\ B &\rightarrow A B f \\ C &\rightarrow S e \\ A &\rightarrow \lambda \end{aligned}$$

Recursion elimination in this grammar takes extensive rewrites (see for the algorithm pages 176 – 178 in [ASU86]): elimination of empty rules, elimination of unit rules and finally flattening of the rules interleaved with elimination of direct recursion. This whole process (using the rewrite rules based on the basic CF grammar notation) results in this grammar:

$$\begin{aligned} S &\rightarrow B c \\ B &\rightarrow \alpha \beta \\ B &\rightarrow \alpha \\ \alpha &\rightarrow \gamma \\ \gamma &\rightarrow c e d \\ \gamma &\rightarrow f c e d \\ \gamma &\rightarrow f \beta c e d \\ \beta &\rightarrow f \\ \beta &\rightarrow f \beta \end{aligned}$$

Notice that during the application of the rewrite rules symbols disappear and new anonymous symbols are added. Unfortunately this hinders the automatic application of the rewrite rules, especially when detector symbols are involved. The FDE can on one hand not decide to call the detector just once, as is the case with the rewrite rule for direct recursion which uses the extended notation. And on the other hand it can also not split the detector in two: one detector which produces the head ( $S$ ) and one which produces the tail ( $\alpha$ ). The same goes for indirect left-recursion elimination.

This moves the burden of removing left-recursion of a detector symbol to the developer, *i. e.* the developer should manually decide when the detector fails and end the infinite production. The need for explicit rewrites by the developer is not uncommon in the world of grammar driven tools, *e. g.* a parser generator like Yacc [LMB92] does not rewrite the grammar rules, but only warns the developer. The main reason for this is that, not unlike the detector functions in a feature grammar system, actions are associated to the grammar rules. And the developer has to modify these actions along with the grammar rules. However, the FDE offers support by warning the developer when left-recursion appears.

#### 4.2.1.2 Lookahead

Deterministic top-down parsing algorithms, and also some bottom-up variants, depend on lookahead. The algorithm looks ahead in the stream of tokens to be parsed to determine which alternative of a rule to choose. Depending on the lookahead depth the alternatives can share longer prefixes. In theory a lookahead of more than one token ( $k > 1$ ) has been studied [RS70, PQ95], however, due to the exponential explosion in time and space ( $|T|^k$ ) practical parsers have almost always implemented a lookahead of only one token.

The most common form of lookahead is implemented by two sets:  $FIRST_k$  and  $FOLLOW_k$ . Both are based on the  $k$ -prefix of a string,  $w = a_1 \dots a_n$ :

$$k : w = \begin{cases} w & |w| \leq k \\ a_1 \dots a_k & |w| > k \end{cases}$$

Using this prefix operation the  $FIRST_k$  and  $FOLLOW_k$  sets are defined as follows:

$$\begin{aligned} FIRST_k(\alpha) &= \{k : w | \alpha \xRightarrow{*} w\} \\ FOLLOW_k(A) &= \{FIRST_k(\beta) | S \xRightarrow{*} \beta A \gamma\} \end{aligned}$$

where

$$w \in T^*, A \in N, \alpha \in V^*, \beta \in T^*, \gamma \in V^*$$

The parse table is now constructed as follows: for every ( $A \rightarrow \alpha$ )  $\alpha$  is added to the ( $A, w$ ) entry of the table for every  $w$  in  $FIRST_k(\alpha FOLLOW_k(A))$  (see [GJ98]).

In [PQ96] the authors argue for the use of more lookahead to make grammars more natural. The rewrite from a  $LL(k)$  or  $LR(k)$  grammar to a  $LL(1)$  or  $LR(1)$  grammar may involve the introduction of many new (anonymous) symbols, *i. e.* to left-factor the rules, thus leading to obfuscation of the semantic meaning. The penalty for the use of more lookahead is the extra space needed for the lookahead table and the extra

time spent to check this table and make the decision. In [Par93] the author describes a linear, approximate, lookahead operation,  $LOOK_k^1$ , which should minimize this penalty ( $|T| * k$ ). This operation is defined as follows:

$$\begin{aligned} FIRST_k^1(\alpha) &= \{a | \alpha \xrightarrow{*} w \wedge w = xay \wedge x \in T^{k-1}\} \\ FOLLOW_k^1(A) &= \{FIRST_k^1(\beta) | S \xrightarrow{*} \alpha A \beta\} \\ LOOK_k^1(A \rightarrow \alpha \bullet \beta) &= \{FIRST_k^1(\beta FOLLOW_k^1(A))\} \end{aligned}$$

where

$$a \in T, y \in V^*, \alpha, \beta \in V^*$$

A set of  $LOOK_k^1$  tables now allows to look at just the discriminating token  $\tau_i$ , instead of having to inspect up to all  $k$  tokens.

In the FDE non-determinism is allowed. However, lookahead is still useful to prevent time consuming parsing and superfluous execution of detectors. By augmenting the exhaustive backtracking algorithm with some form of lookahead the FDE will be able to skip (many of) these dead alleyways.

In a feature grammar system the lookahead is restricted to the sentence belonging to one grammar component. So the sets and the table are constructed on a per component basis. To simulate a complete grammar a default erasing production is added for each detector symbol, including the component detector itself, appearing within the grammar component.

Using the  $LOOK_k^1$  operation the parser can skip the validation of a *Co* alternative (see step *b* in Figure 4.8) by looking at the second token in the lookahead. When this token is *R* choose alternative ( $Co \rightarrow \rho \beta \dots$ ), when it is *Nu* validate the rule ( $Co \rightarrow \rho Nu \dots$ ).

Normally the lookahead depth is determined by steadily incrementing  $k$  until all decisions have become deterministic. In a feature grammar system two or more alternatives may completely overlap within the grammar component, *i. e.* the terminals are only interleaved with (at least one) different detectors. This may result in a  $LOOK_k^1$  table which will still contain two or more rules for one set of lookahead values.

### 4.2.1.3 Memoization

Several parsing algorithms, *e. g.* chart parsers, depend for their efficiency on a well-know technique from dynamic programming: memoization [Mic68]. This technique basically means that each part of the input sentence is only parsed once. When, due to for example backtracking, the same partial sentence is reparsed the memoized parse tree is returned, thus saving processing time. In [Nor91] the author shows that by using this technique in a simple (deterministic) top-down parser the efficiency becomes equivalent to the much more advanced Earley parser, *i. e.*  $O(n^3)$  (where  $n$  is the length of the sentence).

The same technique can be applied within the FDE, but it can also be taken one step further. Remember that the target of the Acoi system is to store the constructed parse trees persistently in a database. Also, references were added to the language in Section 3.2.3.1. These references make it possible to share (partial) parse trees. This can be generalized even more by sharing detector executions as stored in the database. This is possible as, stated in Chapter 2, detectors are (deterministic) functions, *i. e.* the same input always results in the same output. Once a detector has been called with a certain input the output may be memoized and reused, thus preventing superfluous execution. However, memoized detector functions should really be side effect free. Memoization will, for example, spoil the value of an internal counter which needs to be incremented to reflect the actual number of symbol instances.

When detector parse trees are memoized the storage will contain two kind of trees: *elementary trees* and *auxiliary trees*. Elementary trees are rooted by start symbols, they exist individually. Auxiliary trees are rooted by other detectors, they always need to be (indirectly) associated to a elementary tree. This distinction is also known in a NLP technique: *tree adjoining grammars* (TAG) [AR01]. In some variants of TAG trees are also described by D-Theory and quasi-nodes are used to perform substitution and adjoining. Substitution is, in the case of feature grammar systems, the binding of a specific auxiliary detector tree to an elementary tree.

Memoization may also partially resolve the problems with left-recursion (see Section 4.2.1.1), depending on the type of repetition. If the recursive structure also repeats the instantiations, this instantiation will be memoized, be referenced the next time it is encountered and thus break the recursion in the parser. The recursion in the constructed graph will be retained by the memoization reference.

## 4.3 The Feature Detector Engine

This section will describe the actual implementation of the exhaustive backtracking algorithm in the FDE. Before going into the details of the various components within the FDE, the actual form of the FDE needs to be determined.

A grammar can be used in two basic ways: (1) it can be interpreted by a generic parser, or (2) it can be input to a generator which produces a specific parser. These two ways lead to two basic architectures as shown in Figure 4.10. Of course both architectures have their advantages and disadvantages.

The main advantage of the generic parser is its adaptability. A change in the grammar leads to updates of its internal bookkeeping structures, and because those are not hardcoded the changes can be done during runtime [HKR90]. This adaptability comes at a loss of performance, which is the main advantage of a specialized, generated, parser. But in this case changes to the grammar can only be reflected by regeneration and recompilation. To prevent the FDS from having to manage these (possibly complicated) steps the FDE is implemented as a feature grammar driven parser.

Figure 4.10 shows that the parser is preceded by a lexer. In traditional parsers the

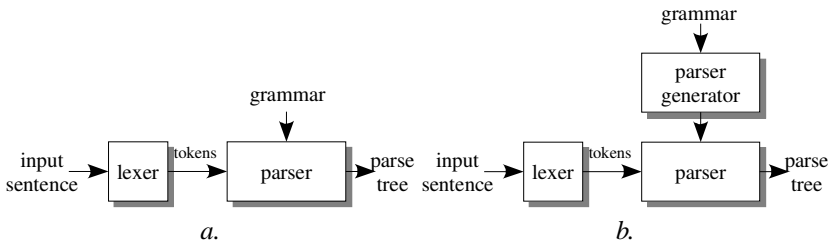


Figure 4.10: (a) A generic parser and (b) a specialized parser

lexer, which performs the lexical analysis, splits the input byte stream into meaningful tokens. In the FDE only a subset of the lexical analysis is needed, as the initial sentence and the output sentences produced by the detectors are already split into tokens. However, their validity is still checked using the specific atom validation rules (see Sections 2.2.2 and 3.1.2).

The internal architecture of the FDE is shown in Figure 4.11 and contains these components:

**the symbol table** is filled by a specific parser (based on the EBNF grammar in Appendix A) for the feature grammar language and contains all the information derived from the specific feature grammar, which is constructed by the developer;

**the set of detectors** are implemented by the developer and each of them can dynamically be loaded into the FDE;

**the set of plugins** are implemented by an expert and can take over the role of a detector, they can also be dynamically loaded into the FDE;

**the set of tokens** is gradually extended with the output of detectors, in fact multiple sets of tokens exist concurrently (one for each grammar component);

**the controller** uses the symbol table to call the detectors, to parse the tokens, and to gradually build the parse forest, *i.e.* implements the exhaustive backtracking parsing algorithm;

**the parse forest** is a DOM tree and can, when the parsing process has ended successfully, be dumped as an XML document containing all valid parse trees.

In the next subsections these components will be revisited and their specific implementation and optimization will be discussed.

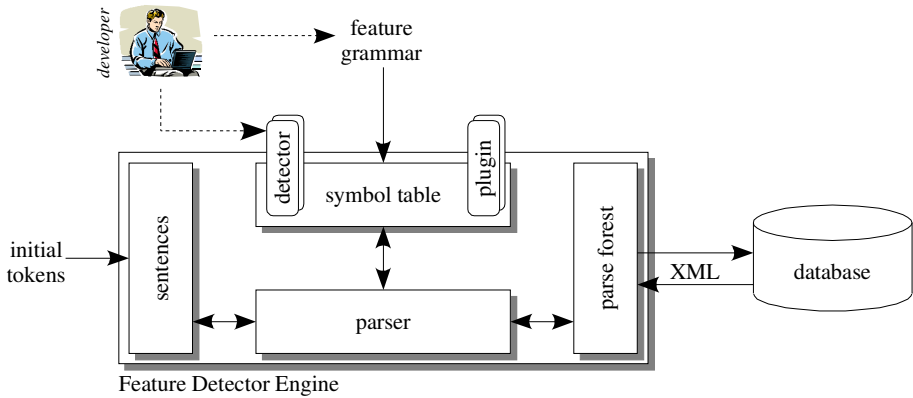


Figure 4.11: The FDE components

### 4.3.1 The Symbol Table

The symbol table is the basic bookkeeping structure of the FDE. It contains all information derived by parsing a specific feature grammar (which conforms to the language of Appendix A). This parsing step ensures the syntactic validity of the grammar. As shown in Chapter 3 some of the language constructs need additional semantic completion, *i. e.* rewrites. When the feature grammar system is complete a semantic check is needed to validate some additional constraints and warn the developer of some (unwanted) properties of the grammar. The rewrites and semantic checks are the topics of the upcoming subsections.

#### 4.3.1.1 Rewriting

The use of the feature grammar language allows a developer to describe a feature grammar system in an intuitive fashion. However, to achieve this some symbols and rules have become implicit. At some points during the parsing of a feature grammar these symbols and rules are made explicit by applying specific rewrites or adding annotations to the symbol entry in the symbol table.

**Symbol sequences** In the FDE symbol sequences are not rewritten but the occurrence indicators are translated into a lower and upper bound. These bounds are checked by a WHILE-statement in the parser implementation (see Section 3.1.1), *i. e.* greedy alternatives are favored.

**Symbol groups** For each symbol group an anonymous is introduced, according to the rewrites shown in Section 3.1.1. Extra care is taken to prevent these symbols to clog up the parse forest by the use of *edge folding*.

**Detector confidences** The compulsory confidence value (see Section 3.1.3.2) is enforced by the implementation skeleton of detectors, this will be illustrated in Section 4.3.5.2.

**Classifiers** Once more the formal rewrite is embedded within the parser instead of applying the rewrite explicitly. Due to the specific entry in the symbol table the FDE knows when and how to call the *analyze* or the *predict* detectors (see Section 3.2.2.3).

Notice that the greediness of this implementation would not notice the ambiguity of the example parse in Section 4.2.1. Only one alternative of the *Image* rules will be found. The greedy implementation conforms more to the usual semantic meaning of optionality: the symbol exists or not, *i. e.* both alternatives are not considered at the same time. As indicated in Section 3.1.1 the greedy implementation results in an iterative interpretation of symbol sequences. This interpretation circumvents the introduction of anonymous symbols and keeps resolving the XPath expressions relatively easy.

#### 4.3.1.2 Semantic Checks

The semantic analysis of the grammar ensures that the symbol table and the embedded grammar rules are semantically consistent. Furthermore, a series of checks is performed on the grammar to warn the developer of “unwanted” properties:

**Check for unknown symbols** When a symbol appears in a RHS, which has no rules but is also not a terminal or a detector, the symbol table does not know it yet. These unknown symbols become non-terminals with an, implicit, empty rule.

**Check for naming conflicts** A naming conflict happens when there are several (imported) namespaces to which a symbol can be bound.

**Check for unique rules** A warning is issued when a non-terminal contains exactly the same production rule more than once.

**Check for factors** The rules are checked for possible shared pre- and suffixes, a warning is issued when such a possibility is found.

**Check for recursion** Left-recursive non-terminals may lead to infinite parses. The FDE issues a warning when left-recursion is found, however, only the developer can resolve these or may have already solved them in the detector implementation.

**Check for non-reachable symbols** This check issues warnings about symbols which may never be reached from a specific start symbol. Notice that these symbols may be reachable from another valid start symbol.

**Check for valid path expressions** Using the detector dependency graph (as will be discussed in Chapter 6) the FDE checks if all the paths point to one or a set of other nodes.

**Check for independent alternatives** Path expressions may not point into other alternatives of the same context node, as each alternative will belong to a different parse tree and this will make the alternatives order dependent.

**Check for possible deadlocks** Check if a reference crossing in a parameter path expression may lead to violation of the linear ordering of detectors.

During the parsing process the controller uses the production rules and symbol information from the table to adapt its generic implementation of the exhaustive backtracking algorithm to the specific feature grammar system.

### 4.3.2 The Parser

Recursive descent is a popular method to implement exhaustive backtracking. In this method specialized functions are generated for each non-terminal, which are recursively called according to the exact semantics of the production rules. In this case, where the FDE is a generic parser, the specialized function is replaced by a generic one which adapts its behavior on the basis of knowledge from the symbol table and the production rules. The implementation of this generic function is shown in pseudo code in Figure 4.12. The other parsing functions (see lines 10 to 21) are all variations on this function. For example the *parse-detector* function will create a local new sentence *s* by executing the detector function (after successfully binding the input sentence), and will check if it is empty before declaring itself valid.

The next sections will focus on the various components the parser interacts with: the set of sentences and the set of parse trees, *i. e.* the parse forest.

### 4.3.3 The Parse Forest

The parse forest is the main result of the FDE. Due to the, possible, ambiguous nature of a feature grammar system and its mild context-sensitivity the parse forest is a rather complex data structure. To manage this structure several control mechanisms have been introduced in Section 2.2.4. Before discussing the actual implementation and use of these mechanisms the global (standardized) data structure is introduced.

#### 4.3.3.1 XML and DOM

As has been shortly mentioned in Section 3.1.3.1 XML documents describe tree structures [W3C00]. Due to the fact that XML is very popular as an exchange format on the WWW it, and many related standards, has been quickly adopted and implemented



<pre> function parse-non-terminal input 1   T: the parse trees under construction 2   s: the remainder of the input sentence under inspection for <i>ctxt</i> 3   nt: the non-terminal being validated output when nt is valid: (T, S) 1   T: the extended parse trees 2   S: the input sentence remainders to be inspected output when nt is invalid: <math>\emptyset</math> implementation 1   T' = expand t in T with a node for nt in s.ctxt 2   valid = <math>\emptyset</math> 3   for each production rule r of symbol nt: 4     if s matches the lookahead requirements of rule r: 5       s' = copy s in (new) context s'.ctxt derived from s.ctxt 6       for each symbol r<sub>hs</sub> in production rule r: 7         for each s'' in s': 8           bound = 0 9           while bound &lt; the higher bound of r<sub>hs</sub>: 10             if the r<sub>hs</sub> symbol is a start symbol: 11               res = parse-start(T', s'', the r<sub>hs</sub> symbol) 12               else if the r<sub>hs</sub> symbol is a classifier: 13                 res = parse-classifier(T', s'', the r<sub>hs</sub> symbol) 14               else if the r<sub>hs</sub> symbol is a detector: 15                 res = parse-detector(T', s'', the r<sub>hs</sub> symbol) 16               else if the r<sub>hs</sub> symbol is anonymous: 17                 res = parse-anonymous(T', s'', the r<sub>hs</sub> symbol) 18               else if the r<sub>hs</sub> symbol is a non-terminal: </pre>	<pre> 19               res = parse-non-terminal(T', s'', the r<sub>hs</sub> symbol) 20               else if the r<sub>hs</sub> symbol is a terminal: 21                 res = consume-terminal(T', s'', the r<sub>hs</sub> symbol) 22               if res == <math>\emptyset</math>: 23                 break 24               bound = bound + 1 25             end 26           if bound &lt; the lower bound of r<sub>hs</sub>: 27             rule r is invalid 28             break 29           end 30         if rule r is valid: 31           s' = res.s 32         end 33       end 34     if rule r is valid: 35       add s' to valid 36     else: 37       delete all nodes related to r from T' 38     end 39   end 40   if valid == <math>\emptyset</math>: 41     delete T' from T 42     return <math>\emptyset</math> 43   end 44   return (T', valid) </pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 4.12: Implementation of the generic parsing function

for a wide range of operating systems and programming languages. The FDE implementation uses an implementation [Vei03] of the Document Object Model (DOM) standard [W3C01a] as an internal representation of the parse tree. This DOM tree can be easily accessed by XPath expressions, *i. e.* whitebox detectors and parameter expressions are easily resolved.

### 4.3.3.2 Labeling Parse Trees

In the parse forest as introduced in Section 2.2.4 each node is labeled with a specific *context*, *i. e.* the parse trees the node is a member of. This context is a list of binary flags, where each flag represents a parse tree. When the flag is *true* the node belongs to the parse tree. The disadvantage of this rather simple scheme becomes clear when a new tree is added to the forest. All known nodes have to be revisited to indicate if they belong to the new tree (or not). To prevent these superfluous runs through the forest the context of a node should only be set when the parsing algorithm visits this node, *i. e.* in a *pre-* or *post-*visitation.

A pre-visitation takes place when the parser starts the validation of a non-terminal. At that moment the parser only knows the intermediate number of trees in the forest: this number is called the *scope* of the context. In principle the node is a possible member of all new parse trees which are added later on, however, those trees are outside its current scope. A new tree (except for the initial tree) always shares nodes with an older tree, *e. g.* its ancestors or the trees it took its detector parameters from. At least the root of the forest is shared by all trees.

After validation of the production rules of a non-terminal the node receives a post-visitation. At that moment the parser knows how many parse trees have been added by these rules and the scope of the context can be enlarged.

To illustrate the use of the context and scope in pre- and post-visitation this, rather artificial but highly ambiguous, feature grammar is used:

```

1 | %module          ambiguous;
2 | %start           S();
3 | %detector        b [ return i = 1 ];
4 | %detector        c [ return i = 10 ];
5 | %detector        d [ return j = 100 ];
6 | %detector        e [ return i = a//i * 2 ];
7 | %detector        g [ return i = a//i + 2 ];
8 | %detector        h [ return i = a//i - 2 ];
9 | %atom            i, j;
10| S                : a e?;
```

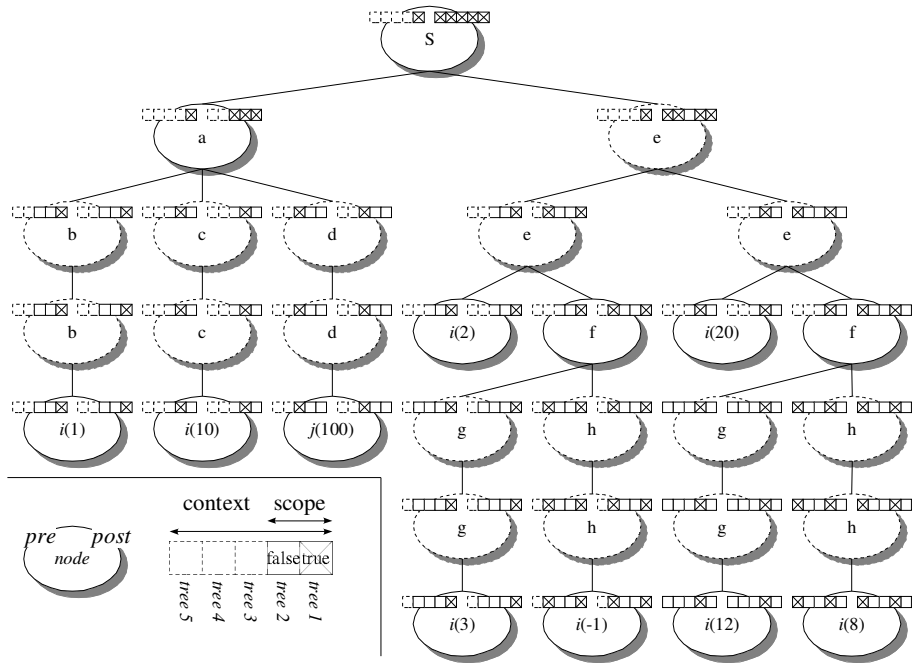


Figure 4.13: A parse forest

11	a	: b   c   d;
12	b	: i;
13	c	: i;
14	e	: i f;
15	d	: j;
16	f	: g   h;
17	g	: i;
18	h	: i;

A run of the FDE for this feature grammar (which has only one possible run) results in the parse forest shown in Figure 4.13. This forest contains 5 trees. The scope of the node contexts increases with the top-down left to right construction of the parse tree. The non-terminal  $a$  has three valid alternatives leading to the addition of two new trees, as the first alternative extends the existing tree. The parameter of the detector  $e$  has now an ambiguous binding: either  $i(1)$  or  $i(10)$ . This leads to two quasi-foots representing two executions of the detector function  $f_e$  in the two contexts. The non-terminal  $f$  has once more two alternatives leading to the addition of two new trees, each within their specific context. The  $g$  subtrees extend the existing trees, while

the  $h$  subtrees are derived new trees.

A node can determine which other nodes in the forest belong to its context by this binary operation ( $\$$  indicates the current node and  $@$  indicates the inspected node):

$$\begin{aligned} & npad(@scope, @context) \& npad(\$scope, \$context) \\ & \qquad \qquad \qquad = \\ & npad(max(@scope, \$scope), max(@context, \$context)) \end{aligned}$$

The  $npad$  function sets all flags outside of the context scope to the default value  $true$ . The  $max$  operations determine which of the nodes is deeper and further to the right of the forest, *i. e.* more specific as nodes higher and more to the left have a smaller scope and are shared more. Take for example the two possible  $h$  roots. The first one does not have  $i(10)$  in its scope and context (where  $t = true$  and  $f = false$ ):

$$\begin{aligned} & npad(3, ftf) \& npad(4, tfft) \neq npad(max(3, 4), max(ftf, tfft)) \\ & \qquad \qquad \qquad tfft \& tfft \neq npad(4, tfft) \\ & \qquad \qquad \qquad tfff \neq tfft \end{aligned}$$

Doing the same inspection for the second  $h$  root results in a positive match:

$$\begin{aligned} & npad(3, ftf) \& npad(5, tfft) = npad(max(3, 5), max(ftf, tfft)) \\ & \qquad \qquad \qquad tfft \& tfft = npad(5, tfft) \\ & \qquad \qquad \qquad tfft = tfft \end{aligned}$$

This also shows that the validity contexts of the  $h$  roots are in fact determined by their ancestor, the  $e$  quasi-foots.

In the post-visitation all contexts of the compulsory children of the node, *i. e.* those with a lower bound of one or more, are unified. See for example the quasi-root of  $e$ . The third tree does not contain an  $e$  node, however, this symbol is optional leading to a valid  $S$  node and thus to a valid third parse tree. The post context replaces the pre context.

This matching operation is used for resolving ambiguous parameter bindings by adding a feature grammar system specific nodetests to the XPath expression.

### 4.3.3.3 Memoized Parse Trees

Persistently memoized parse trees function for the FDE as a persistent lookup table of detector calls. Each detector call is identified by a quasi-foot which contains information about a specific input sentence. As a detector is a partial function this input

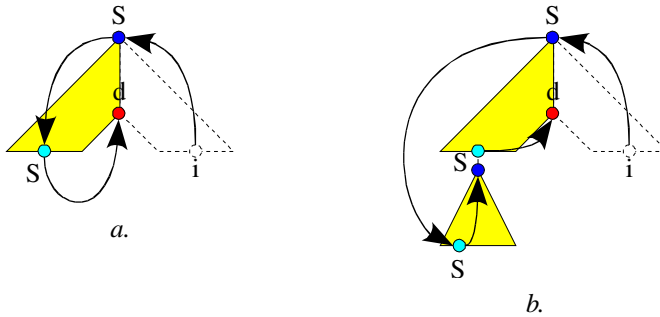


Figure 4.14: A deadlock situation due to (a) a direct and (b) an indirect self reference

sentence always maps to the same, stored, output sentence. The FDS, which will be discussed in detail in Chapter 6, manages the lookup table.

The moment the FDE has assembled a complete input sentence a request for the parse tree is send to the FDS. When there exists a mapping for this input sentence the FDS will return the unique identifier for the tree and its availability, the FDE will then take the appropriate action:

1. when the parse tree is *available*, the identifier is stored within the quasi-foot as a place holder;
2. when the parse tree is *under construction*, the FDE will have to wait till it is know if the mapping exists, *i. e.* the parse tree becomes available, is unknown or a deadlock situation occurs (which will be discussed in the next paragraph);
3. when the mapping *does not exist*, the detector symbol can be rejected by the parser;
4. when the mapping of a black- or whitebox detectors is *unknown* the FDE will inform the FDS that it will execute the detector to instantiate the parse tree, *i. e.* the parse tree becomes under construction.

In principle parse trees are not loaded from the lookup table, until a value is needed as part of an input sentence. The FDE then sends a request for the complete parse tree or the specific value, depending on the abilities of the underlying XML storage structure, to the FDS. When the parse tree is still under construction there may be a deadlock situation. Such a situation occurs when, by a reference, the linear ordering is violated. Figure 4.14 illustrates the two basic deadlock forms: due to a direct self-reference, *e. g.*  $d(\&S//i)$ , or an indirect self-reference, *e. g.*  $d(\&S//\&S//i)$ . As a global deadlock resolution strategy is not possible the detector is informed and expected to handle the situation leading to a memoizable parse tree (see Section 4.3.5.6).

In the previous section the trees within the parse forest have been labeled using a scope and context mechanism. However, these elementary and auxiliary parse trees

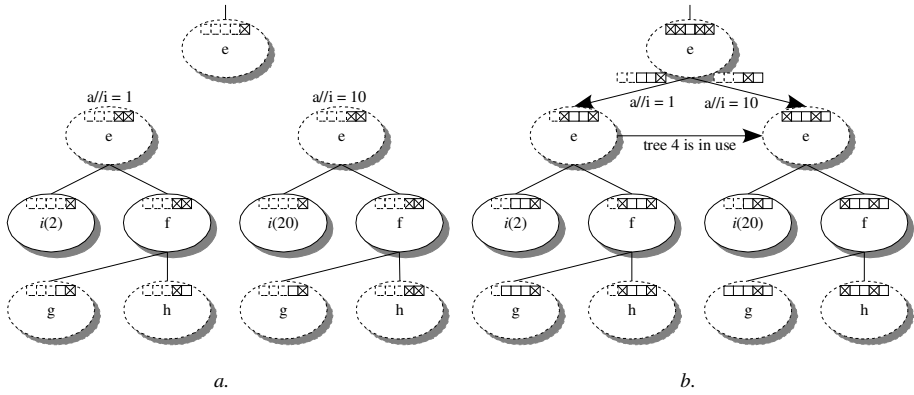


Figure 4.15: (a) Local parse forests are (b) combined in a global forest.

will be memoized. A memoized parse tree may be loaded in another forest with a different context. During saving, the context has to be localized, while during loading the context has to be globalized. Localization means that the parse tree loses the inherited global context, only the local context remains. Globalization then reinstates a, possibly different, global context.

Figure 4.15 illustrates this process. The local forests are derived from Figure 4.13. Notice that bits in use by siblings are stripped out, e. g. bit 4 for the second alternative of *e*. Figure 4.15.b globalizes the context once more by replacing bit 1 by the global context. As the first alternative claims another bit, i. e. creating a difference between the scopes of the quasi-root and the current scope, bit 4 is once more inserted for the second alternative. This dynamic behavior of the context bits makes it useless to persistently store the post-context as a change in one of the memoized trees may use up more bits.

### 4.3.4 The Sentences

Sentences are produced per grammar component by the detector function. Internally a sentence is a simple linked list of tokens. Figure 4.12 shows that for each alternative production rule of the sentence is made (*s'*). In fact only a copy of the token pointer is made, so each alternative points to its own current position in the sentence. Each copy is associated with a context, i. e. corresponds with a specific parse tree within the parse forest.

The stack of sentences under inspection, needed for resuming the validation of the sentence after control has been temporarily transferred to another grammar component (see the upcoming Section 4.2.1), is implicit, as each sentence is a local variable of a specific call of the *parse-detector* function.

## 4.3.5 Detectors

### 4.3.5.1 Detector Input

Detector parameters are identified by XPath expressions. These XPath expressions are normalized by the feature grammar parser. In this process these rewrites are applied:

1. The default axis for the first step is `preceding::`;
2. By default only the last match is returned, *i. e.* `add [fn:position() = 1]` for a reverse axis and `[fn:position() = fn:last()]` for a forward axis.
3. The feature grammar specific reference operation `&node` is translated into a `node[fg:bind(@id)]` call. This FDE specific XSLT extension function returns a nodeset containing the root node of the refereed (memoized) parse tree, *i. e.* this may have to be loaded just-in-time from the database.
4. The parse forest may contain several types of anonymous nodes, *e. g.* quasi-foots. The developer does not know about those nodes and thus will not take care of them within his XPath expressions. Between each two steps a skip expression is inserted in the vain of `/descendant::*[contains(@type, '.q.')] /`. This a rather expensive solution. It is cheaper to prevent creating these nodes at all. This can be done with anonymous nodes which do not contain additional information, *e. g.* group nodes. These parse forests stay closer to the semantic grammar and are also called *Reduced Derivation Trees* (RDT) [JS98].
5. Detector parameters may only be bound within the context of the current node. This XPath nodetest will only allow nodes which are within the current context scope:

```
[ fg:and(
    fg:npad( @scope, @ctxt),
    fg:npad( current()/@scope, current()/@ctxt ) )
=
fg:npad( fg:max( @scope, current()/@scope),
    fg:max( @ctxt, current()/@ctxt ) ) ]
```

The resulting XPath expressions can be resolved against the internal DOM tree. The result may be several sets of input parameters for different contexts. For each context a detector call will be bound to a quasi-foot.

```

1 | (Confidence, Sentence) Skin(Token myLocation) {
2 |     Sentence mySentence = newSentence();
3 |     Image myImage = openImage(getValue(myLocation));
4 |     Bitmap myBitmap = deriveBitmap(myImage, false);
5 |     Iterator myPixels = newIterator(getPixels(myImage));
6 |     while(hasMore(myPixels))
7 |         if (isSkin(nextElement(myPixels)))
8 |             nextBit(myBitmap, true);
9 |     putToken(mySentence, "Skin/bitmap", myBitmap);
10 |     return (0.95, mySentence);
11 | }

```

Figure 4.16: Implementation of the *Skin* blackbox detector in pseudo code

### 4.3.5.2 Blackbox Detectors

Blackbox detectors are implemented in the host language of the FDE, *i. e.* a *general purpose language* (GPL) like *C*. Figure 4.16 shows an implementation of the *Skin* detector in pseudo code.

The detector receives its input sentence as a set of tokens from the parse tree. It uses this information, *i. e.* the *Location* of the *Image*, to load the image. A new bitmap is created and filled by iterating over the pixels of the image and determining if they are a skin pixel or not. The new *bitmap* token is then added to the newly created output sentence which is returned to the FDE. Next to the sentence also the compulsory confidence information is returned: the *Skin* detector knows for 95% sure that these pixels are really skin.

### 4.3.5.3 Plugins

Plugins take over a large part of the coding burden from the developer by implementing a generic detector. Plugins come in the two basic variants of detectors: blackbox and whitebox. In the first case only the input parameters are provided, while in the latter case those are embedded within a template in a *domain specific language* (DSL), like XPath.

Figure 4.17 shows the implementation of the *matlab* plugin. The plugin receives a list of requested parameters belonging to one context. Using the symbols name, *e. g.* *Color*, a command call is constructed. When the command was successfully executed



```

1 | (Confidence, Sentence) matlab(Symbol mySymbol, List myParams) {
2 |     Engine myEngine = startEngine(getProperty("matlab"));
3 |     if (myEngine) {
4 |
5 |         String myCommand = getName(mySymbol) + "(";
6 |         Iterator myIterator = newIterator(myParams);
7 |         if (hasMore(myIterator))
8 |             myCommand += getValue(nextElement(myIterator));
9 |         while(hasMore(myIterator))
10 |             myCommand += "," + getValue(nextElement(myIterator));
11 |         myCommand += ")";
12 |
13 |         Sentence mySentence = runEngine(myEngine, myCommand);
14 |         if (closeEngine(myEngine) && mySentence)
15 |             return (1.0, mySentence);
16 |     }
17 |     return (0.0, newSentence());
18 | }

```

Figure 4.17: Implementation of the *matlab* plugin in pseudo code

the output sentence and a confidence of 100% is returned to the FDE. When the execution was unsuccessful a zero confidence is returned, which will lead to rejection of the symbol.

The same process happens for whitebox plugins although the FDE handles, instead of the list of parameters, the instantiated template over to the plugin implementation. So binding detector parameters is always done by the FDE, just like with blackbox detectors. But a plugin has additional access to the symbol table and can thus adapt its course on the actual rule context of the symbol.

#### 4.3.5.4 Classifiers

Classifiers are special in the sense that they imply two detectors, both are in fact implemented as a plugin. Figure 4.18 and Figure 4.19 show the implementation of these two detectors for the *bpmn* classifier.

#### 4.3.5.5 Start Symbols and References

Start symbols and references are once more implemented as plugins, *i. e.* the feature grammar developer does not have to provide any code for these detectors.

Only one start symbol is instantiated in a specific FDE run. This detector looks in the environment of the FDE for the required initial tokens. This environment consists of notifications of the FDS, the command line of the FDE or interaction with the

```

1 (Confidence, Sentence) bpnn.analyze(Symbol mySymbol, List myParams) {
2     Confidence myResult = 1.0;

3     bpnn myNN = openBPNN(getName(mySymbol) + ".net");
4     if (!myNN)
5         myNN = newBPNN(getLength(getParameters(mySymbol)), 4, 2);

6     Iterator myIterator = newIterator(myParams);

7     targetBPNN(myNN, 1, myResult);
8     targetBPNN(myNN, 2, atoi(getValue(nextElement(myIterator))));

9     integer i = 1;
10    while (hasMore(myIterator))
11        inputBPNN(myNN, i++, getValue(nextElement(myIterator)));

12    trainBPNN(myNN);
13    saveBPNN(myNN, getName(mySymbol) + ".net");
14    closeBPNN(myNN);

15    return (myResult, newSentence());
16 }

```

Figure 4.18: Implementation of the *bpnn.analyze* detector in pseudo code

librarian. When all tokens are available the parsing algorithm starts the validation process.

References take their required tokens from the sentence under inspection. Then they request the FDS for the identifier and status of the parse tree belonging to the sentence constructed from these tokens (see Section 4.3.3.3). If the parse tree is not yet known the FDE can build the parse tree, as the input sentence is available, and it needs to know if the tree is valid.

#### 4.3.5.6 Deadlock Resolution

Sections 3.2.3.1 and 4.3.3.3 identified that deadlocks have to be resolved by the developer within the detector implementation. For this the developer will have to check if one of the tokens received from the FDE is empty<sup>1</sup>. The developer has then three options: (1) use a default value, (2) know how to retrieve the value, which will only work when the token is part of the output sentence of this detector, or (3) let the detector fail. In the case of failure the detector symbol will not be accepted by the FDE.

<sup>1</sup>This means a self reference because when the token is just not available in the parse forest the detector would not have been executed, *i. e.* its start condition is not valid.

```

1 (Confidence, Sentence)bpnn.predict(Symbol mySymbol, List myParams) {
2     Confidence myResult = 0.0;

3     bpnn myNN = openBPNN(getName(mySymbol)+".net");
4     if (!myNN) {
5         myNN = newBPNN(getLength(getParameters(mySymbol)), 4, 2);
6         saveBPNN(myNN, getName(mySymbol)+".net");
7     }

8     integer i = 1;
9     Iterator myIterator = newIterator(myParams);
10    while (hasMore(myIterator))
11        inputBPNN(myNN, i++, getValue(nextElement(myIterator)));

12    feedforwardBPNN(myNN);
13    Confidence myResult = outputBPNN(myNN, 1);
14    Sentence mySentence = newSentence(outputBPNN(myNN, 2));

15    closeBPNN(myNN);

16    return (myResult, mySentence);
17 }

```

Figure 4.19: Implementation of the *bpnn.predict* detector in pseudo code

## 4.4 Discussion

This chapter contained a detailed description of the design and implementation decisions made for the FDE. The FDE steers the actual annotation extraction process by interpreting a specific feature grammar system described by a feature grammar. The top-down parsing algorithm, implemented in the FDE, is interrupted by the execution of detector algorithms.

This execution model may seem not too different from the way actions are associated to attribute grammars [GJ98] and interrupt the parser, *e. g.* as in parsers generated by Yacc [LMB92]. However, those actions can only intervene in a limited way in the parsed sentence, *e. g.* push a token back on the stack. The parsed sentence is completely available, while in the FDE the parsed sentence is extended just-in-time. This limits the parser severely in taking decisions based on lookahead. As discussed, lookahead can only be used within a grammar component, where the complete sentence is available. Bottom-up algorithms, like used in Yacc, may be used within individual components. However, the control transfer between components complicates this. Postponing this transfer may enable the use of, in general, more efficient bottom-up algorithms, and is thus an interesting topic for future research.

Performance can also be boosted by replacing the depth-first algorithm with a breadth-first algorithm, *i. e.* each parse tree gets its own parsing thread. Detectors should already be side-effect free, but shared data structures, like the parse forest, will have to be guarded by critical sections or replaced by localized copies. Investigation of the theory of PC grammar systems may also be of interest here.

The current implementation is in *C*. However, other implementation strategies are well possible, *e. g.* in a functional language or in the form of generation of ToolBus scripts or translating context dependencies into output/input dependency for a dataflow or a daemon architecture (see Section 2.3). However, the *C* implementation gave more freedom in staying close to a well known parsing algorithm and thus study the impact of the extensions of feature grammar systems. A future ToolBus or daemon implementation may allow to incorporate more concurrency, and may also allow relaxation of the deadlock prevention strategy.

# Chapter 5

## Feature Databases

*Systems have sub-systems and sub-systems have sub-systems and so on ad finitum - which is why we're always starting over.*

*Every program is a part of some other program and rarely fits.*

Alan J. Perlis – *Epigrams on Programming*

The FDE implementation described in the previous chapter produces a forest of parse trees, *i. e.* elementary and auxiliary trees. These trees are stored in a feature database for two reasons: primarily as a persistent buffer for the on line use by the DMW search engine and, secondarily, as a lookup table for memoized detector calls. The parse trees produced by the FDE are in fact XML documents. The mass storage of XML documents has been a major research topic since the rise of XML as *the* data exchange format for Internet-based applications. In this chapter the storage scheme for XML documents as used by the current Acoi implementation is described in more detail. The backend of this XML mapping is the Monet database kernel. The Acoi system functioned as a test case for many of its unique aspects. These unique aspects are introduced in the next section, while the other sections will reflect on the mapping used and the lessons learned.

### 5.1 The Monet Database Kernel

The Monet database kernel [BK95, Bon02] provides, for main memory optimized, access to *Binary Association Tables* (BATs). BATs are the actual implementation primitives for the *Decomposed Storage Model* (DSM) [CK85]. On top of this kernel several front-ends have been build. These front-ends use the extensibility features of Monet: the Monet Interpreter Language (MIL) [BK99] and its dynamic loading mechanism for accessing libraries of *C* code. In the case of the relational model

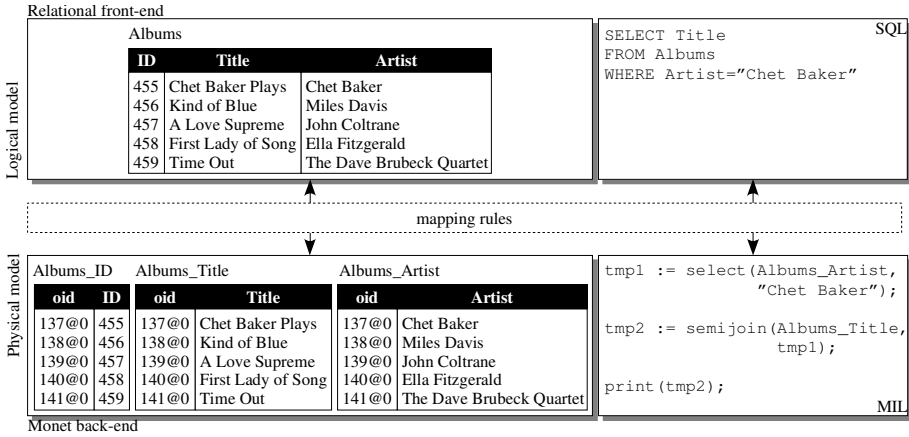


Figure 5.1: A relational front-end for Monet.

tables are vertically decomposed into binary tables, see Figure 5.1. SQL queries are translated into MIL commands which provide access to the appropriate BATs.

### 5.1.1 Monet and XML

Just as for the relational model an XML specific Monet front-end can be build. In fact several of such front-ends have been built, and they will be shortly described in this section. Where appropriate the mappings are illustrated with (parts of) the parse forest shown in Figure 4.9, assuming that confidences ( $\rho$ ) are stored as attributes, that end-of-sentence markers (\$) are not stored, and that all leafs contain a lexical instantiation.

An easy way to store XML documents in a database is into a *binary large object* (BLOB). This has been a popular way in the early days of the integration of XML into databases. However, its drawback is that to access the XML contents the XML document has to be (re)parsed. This approach prevents the use of the query optimization facilities of a DBMS. The solution to these problems is shredding. Shredding means that the XML document is parsed only once and the contents are directly exposed to the DBMS, which can thus optimize the access to it. All the methods described in the upcoming sections use a shredding approach.

#### 5.1.1.1 Semistructured Data

The Magnum Object Algebra (MOA) [BWK98] is an intermediate language between an object calculus, *e.g.* OQL, and the database execution language, *i.e.* MIL. In [vZAW99] the authors investigate an extension to MOA to also handle semistructured

data in the form of XML documents. The tree is represented by a set of binary associations. Each association describes a parent/child combination.

Taking the example parse forest the database stores these associations:

$$\begin{aligned}
 S_S[\rho] &= \{ \langle o_1, "1.00" \rangle \}, \\
 S_S/Im &= \{ \langle o_1, o_2 \rangle \}, \\
 Im/Lo &= \{ \langle o_2, o_3 \rangle, \langle o_2, o_4 \rangle \}, \\
 Im/\alpha &= \{ \langle o_2, o_7 \rangle \}, \\
 \alpha/Co &= \{ \langle o_7, o_8 \rangle \}, \\
 &\vdots \\
 Sk/cdata &= \{ \langle o_{18}, o_{19} \rangle \}, \\
 Fa/cdata &= \{ \langle o_{20}, o_{21} \rangle \}, \\
 cdata[string] &= \{ \langle o_5, "http://..." \rangle, \langle o_6, "http://..." \rangle, \\
 &\quad \langle o_{10}, "29053" \rangle, \langle o_{12}, "0.03" \rangle, \langle o_{14}, "0.19" \rangle, \\
 &\quad \langle o_{17}, "true" \rangle, \langle o_{19}, "00..." \rangle, \langle o_{21}, "1" \rangle \}
 \end{aligned}$$

In this case there is no large overlap in structure, but when instantiations of a parent/child relationship occur distributed over the document they will all end up in the same association, *e. g.* like the `cdata[string]` BAT.

This mapping provides a good on average query performance, even when used with an off-the-shelf DBMS, as has been benchmarked by [FK99].

### 5.1.1.2 Monet XML and XMark

Monet XML has been developed with the parent/child mapping from the MOA approach as starting point. Two basic features distinguish Monet XML [SKWW00, Sch02] from other XML to database tables mappings:

1. the decomposition method is independent of the presence of a *Document Type Definition* (DTD) or other schema, but explores the structure of the document at runtime;
2. it tries to minimize the volume of data to be processed during a query by storing associations according to their context in the tree.

This basically means that database tables are created upon need, and these tables are not only vertically decomposed, but also horizontal. The horizontal decomposition is administered by the path catalog which contains information about the specific context of the associations stored in the table. This leads to this specific database instantiation for the example parse forest:

$$\begin{aligned}
S_S[\rho] &= \{ \langle o_1, "1.00" \rangle \}, \\
S_S/Im &= \{ \langle o_1, o_2 \rangle \}, \\
S_S/Im/Lo &= \{ \langle o_2, o_3 \rangle, \langle o_2, o_4 \rangle \}, \\
S_S/Im/Lo/cdata &= \{ \langle o_3, o_5 \rangle, \langle o_4, o_6 \rangle \}, \\
S_S/Im/Lo/cdata[string] &= \{ \langle o_5, "http://..." \rangle, \langle o_6, "http://..." \rangle \}, \\
&\vdots \\
S_S/Im/\alpha/Cl/Fa &= \{ \langle o_{15}, o_{20} \rangle \}, \\
S_S/Im/\alpha/Cl/Fa[\rho] &= \{ \langle o_{20}, "0.77" \rangle \}, \\
S_S/Im/\alpha/Cl/Fa/cdata &= \{ \langle o_{20}, o_{21} \rangle \}, \\
S_S/Im/\alpha/Cl/Fa/cdata[string] &= \{ \langle o_{21}, "1" \rangle \}
\end{aligned}$$

It is clear that this approach uses a larger number of tables due to the use of more context in the distribution of the nodes, *i. e.* several tables contain `cdata[string]` information. This makes it possible to directly zoom in on a specific part of the XML document by resolving path expressions mainly in the path catalog. On the other hand complete reconstruction of an XML document is more expensive.

The Monet XML project also includes the definition of the XMark benchmark [SWK+01, SWK+02]. This benchmark is used to assess an XML database's abilities to cope with a broad spectrum of different queries, typically posted in real-world application scenarios. It is widely used to assess systems.

### 5.1.1.3 XQuery

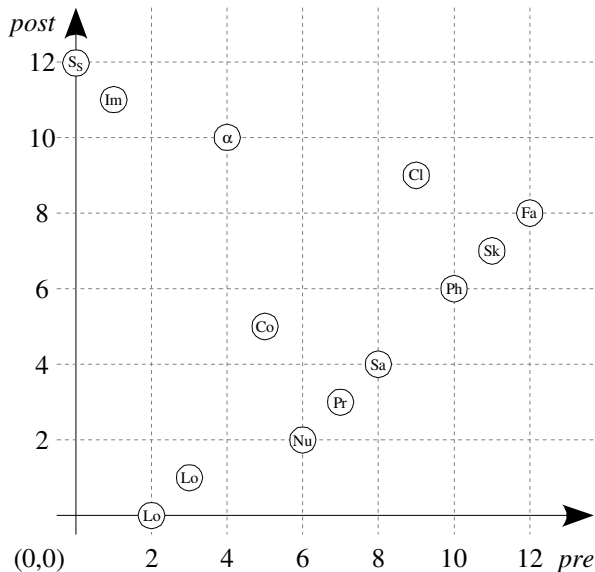
Based on [Gru02] an XQuery interface on top of Monet is currently under construction [GvKT03]. [GvK03] describes one of the major optimized operations: the staircase join. The optimizations in this implementation make use of a node numbering scheme. Each node is assigned a *pre*- and a *post*-order, *i. e.* resulting in a coordinate for a node in the *pre/post* plane. The staircase join uses extensive knowledge about the distribution of nodes in this plane with respect to a certain context node to prune areas from the search space.

Figure 5.2 shows the parse forest of Figure 4.9 in the *pre/post* plane. The information about these nodes is stored in a small number of BATs from which each has the unique and dense *pre*-orders as the head column.

## 5.2 A Feature Database

Most of the XML facilities for Monet were developed concurrently with Acoi. As such, an Acoi specific mapping had to be defined and implemented. In this section



Figure 5.2: The *pre/post* plane.

this mapping is described. It is based on the parent/child mapping from the MOA approach. Care has been taken to keep the interaction, from the FDE and FDS standpoints, purely XML and hence independent of the storage system and mapping. In this case the XML documents are transformed by an XSLT stylesheet [W3C01b] into a MIL script. This script inserts the data from the parse forest into the database.

Notice that this mapping is just a baseline implementation. Other mappings and systems, *i. e.* the discussed mappings for Monet or the XML support of an off-the-shelf DBMS, may prove to be a more effective and efficient XML storage alternative. Due to the XML exchange layer these alternatives can relatively easy replace the current storage backend.

### 5.2.1 A Database Schema

DTD-based or schema-less XML mappings support only one basic data type: character data (CDATA). However, a feature grammar contains information about the atomic types of the data leafs in the parse forest. To create a database schema which takes advantage of this information, *e. g.* the integer equivalent of *int(29053)* is cheaper to store than the corresponding character string, the grammar can be translated into a XML document providing schema information. Currently there are several competing XML schema languages. The major ones are: XML Schema [Fal01], Relax NG

[CM01] and Schematron [Jel02]. These languages can be partially intertwined. For example Schematron assertions may be embedded in an XML Schema [XFr01], while XML Schema datatypes can be reused inside Relax NG schemas [CK01]. All these languages have their strong and weak points. A favorable combination [vdV01] may look as follows: structures described by Relax NG, data types by XML Schema<sup>1</sup> and additional validation rules by Schematron.

Once more these schema languages were developed concurrently to the Acoi system. In the current implementation a straightforward propriety XML-based schema language is used. However, any other “standard” schema language may replace this language. The schema document contains a list of non-terminals, *i. e.* all the LHSs, and their possible children, *i. e.* all the RHSs. The symbols are all annotated with meta-information, *e. g.* the symbol type, and the lower and upper bound. Using an XSLT stylesheet this document is translated into a MIL script to create the database tables.

This part of the schema document (a complete version is found in Appendix C.1):

```

1 | ...
2 | <Image:Color type=".non-terminal.detector.blackbox.">
3 |   <Image:RGB type=".non-terminal." coll="list" lbnd="0"
4 |     hbnd="infinite"/>
5 |   ...
6 | </Image:Color>
7 | <Image:RGB type=".non-terminal.">
8 |   <Image:Red type=".non-terminal."/>
9 |   ...
10| </Image:RGB>
11| <Image:Red type=".non-terminal.">
12|   <fg:int type=".terminal.atom."/>
13| </Image:Red>
14| ...

```

is translated into these MIL statements:

```

1 | ...
2 | VAR Image_Color_Image_RGB_parent := new(void,oid);
3 | VAR Image_Color_Image_RGB_child := new(void,oid);
4 | VAR Image_RGB_Image_Red_parent := new(void,oid);
5 | VAR Image_RGB_Image_Red_child := new(void,oid);
6 | VAR Image_Red_fg_int_parent := new(void,oid);
7 | VAR Image_Red_fg_int_child := new(void,oid);
8 | VAR fg_int := new(oid,int);
9 | ...

```

<sup>1</sup>The data types of XML Schema lack a decent type system[Lew02], however, at least it provides an extension to the limited set of DTD data types.

The BATs with *void* head and *oid* tail will store the tree structure of the parse forest. Data from leaf nodes are stored in specific BATs which contain a tail column of the atomic type.

The *void* head produces a dense numbering scheme for a specific edge type, resulting in aligned array access of all the base and meta-data associated to the edge. This meta-data is stored in additional BATs. For example this BAT stores the position of a *RGB* instance in a specific list:

```
1 | VAR Image_Color_Image_RGB_list := new(void,int);
```

Other examples of needed meta-data are the context and scope of the nodes, the version and confidence of detectors and their input relations.

Next to these data BATs also information about the feature grammar system is stored. This enables the use of generic procedures which follow the dependencies between the various nodes, *e. g.* to reconstruct the original XML document.

## 5.2.2 A parse forest XML document

The FDE contains in memory a parse forest in the form of an XML document. This internal document contains more meta-data than needs to be stored in the database. Using an XSLT script this internal format is stripped down. Appendix C.2 contains an example of the final parse forest XML document. Some portions of this document will be described in this section.

```
1 | <?xml version="1.0"?>
2 | <fg:forest
3 |   xmlns:fg="http://www.cwi.nl/~acoi/fg/forest"
4 |   xmlns:WWW="http://www.cwi.nl/~acoi/WWW"
5 |   xmlns:Image="http://www.cwi.nl/~acoi/Image"
6 | >
7 |   <fg:elementary context="1:1" confidence="1.00" idrefs="2@1"
8 |     start="WWW:WebObject" date="20030625"
9 |   >
10 | ...
11 | </fg:elementary>
12 | <fg:auxiliary date="20030625">
13 | ...
14 | </fg:auxiliary>
15 | <fg:auxiliary date="20030625">
16 | ...
17 | </fg:auxiliary>
18 | </fg:forest>
```

The root of the document *fg : forest* contains information about the feature grammar modules used, *i. e.* they are mapped to XML namespaces. The root contains at least one *fg : elementary* child node and zero or more *fg : auxiliary* child nodes.

```

1 ...
2 <fg:elementary context="1:1" confidence="1.00" idrefs="2@1"
3   start="WWW:WebObject" date="20030625"
4   >
5     <WWW:WebObject id="5478@0" context="1:1">
6       <WWW:Location id="1" context="1:1">
7         <WWW:url id="2" context="1:1">
8           <![CDATA[http://...]]>
9         </WWW:url>
10        </WWW:Location>
11        <WWW:WebHeader idrefs="5479@0" context="1:1"/>
12        <WWW:WebBody id="7" context="1:1">
13          <Image:Image id="8" context="1:1">
14            <Image:Color idrefs="5480@0" context="1:1"/>
15            <Image:Class id="15" context="1:1">
16              <Image:Photo idrefs="5486@0" context="1:1"/>
17              <Image:Skin idrefs="5487@0" context="1:1"/>
18              <Image:Faces idrefs="5488@0" context="1:1"/>
19            </Image:Class>
20          </Image:Image>
21        </WWW:WebBody>
22      </WWW:WebObject>
23    </fg:elementary>
24 ...

```

Each parse forest is based on one start symbol, which roots the elementary trees. To these elementary trees auxiliary trees, which are rooted by detectors or references, may be attached. The *idrefs* attributes of inner nodes, *i. e.* a quasi-root, refer to specific instantiations of the auxiliary trees, *i. e.* the quasi-foot nodes. When there is more than one reference the node is ambiguous and each *idref* will point to a detector call for a different context. The *idrefs* attribute of *fg: elementary* refer to the initial tokens.

```

1 ...
2 <fg:auxiliary date="20030625">
3   <Image:Color id="5480@0" idrefs="2" context="1:1"
4     confidence="1.00" version="1.0.0"
5   >
6     <Image:Number id="9" context="1:1">
7       <fg:int id="10" context="1:1">
8         <![CDATA[29053]]>
9       </fg:int>
10    </Image:Number>
11    <Image:Prevalent id="11" context="1:1">
12      <fg:flt id="12" context="1:1">
13        <![CDATA[0.03]]>
14      </fg:flt>

```

```

15     </Image:Prevalent>
16     <Image:Saturation id="13" context="1:1">
17         <fg:flt id="14" context="1:1">
18             <![CDATA[0.19]]>
19         </fg:flt>
20     </Image:Saturation>
21 </Image:Color>
22 </fg:auxiliary>
23 ...

```

Auxiliary trees contain the output of a detector call or are placeholders for a reference to an elementary tree.

Notice that only root nodes contain an *id* with a @0 prefix, which indicates that it is database unique. The *id* attribute of an inner node is just a normal integer and needs to be turned into a database unique identifier upon insertion into the database. This minimizes the need for the FDE to request unique identifiers from the database when a node is added to the tree.

### 5.2.3 Inserting a Parse Forest

The insertion script for Monet is generated just as the schema script: by an XSLT stylesheet. For the example auxiliary tree these MIL statements are generated:

```

1 ...
2 Image_Color_idrefs.insert(id2oid("5480@0"),id2oid("2"));
3 Image_Color_context.insert(id2oid("5480@0"),context("1:1"));
4 Image_Color_confidence.insert(id2oid("5480@0"),flt("1.00"));
5 Image_Color_version.insert(id2oid("5480@0"),version("1.0.0"));
6 Image_Color_Image_Number_parent.insert(id2oid("5480@0"));
7 Image_Color_Image_Number_child.insert(id2oid("9"));
8 Image_Number_context.insert(id2oid("9"),context("1:1"));
9 Image_Number_fg_int_parent.insert(id2oid("9"));
10 Image_Number_fg_int_child.insert(id2oid("10"));
11 fg_int.insert(id2oid("10"),int("29053"));
12 fg_int_context.insert(id2oid("10"),context("1:1"));
13 ...

```

### 5.2.4 Replacing a (Partial) Parse Forest

The roots of the elementary and auxiliary (partial) parse forests contain database unique identifiers. Those are used to check if the forest is already stored in the database. If this is true the new forest will replace the old one. As this new forest may have a complete new shape and thus not neatly replace the old forest, the old forest is deleted from the database before the new forest is inserted. To support this a stored procedure, generated by XSLT from the schema document, is called. This

procedure knows which BATs are involved with this specific type of (partial) parse forests and pointer chases the specific forest or, in the case of a bulk operation, forests.

### 5.2.5 Query Facilities

As MIL is still the primary means to interact with Monet (the SQL and XQuery interfaces are still under development) an Acoi specific query interface has been developed. Once more this interface is based on the combination of an XML document and an XSLT style sheet.

The XML document contains zero or more selection trees and one projection tree. In the selection trees predicates on terminal values are specified. More than one selection tree is needed when the predicates are disjunctive or there are several conjunctive predicates on the same terminal. In the projection tree the nodes the user wants to be part of the answer XML document are marked. This query document requests all portraits from the database.

```

1 <?xml version="1.0"?>
2 <fg:query
3   xmlns:fg="http://www.cwi.nl/~acoi/fg/query"
4   xmlns:WWW="http://www.cwi.nl/~acoi/WWW"
5   xmlns:Image="http://www.cwi.nl/~acoi/Image"
6   grammar="video" start="WWW:WebObject"
7 >
8 <fg:select>
9   <WWW:WebObject>
10    <WWW:WebBody>
11     <Image:Image>
12      <Image:Class>
13       <Image:Faces>
14        <fg:int min="1" max="1"/>
15       </Image:Faces>
16      </Image:Class>
17     </Image:Image>
18    </WWW:WebBody>
19   </WWW:WebObject>
20 </fg:select>
21 <fg:project>
22   <WWW:WebObject>
23     <WWW:Location project="true">
24       <WWW:url project="true"/>
25     </WWW:Location>
26   </WWW:WebObject>
27 </fg:project>
28 </fg:query>

```

The XSLT sheet translates this query document into a MIL script which starts with

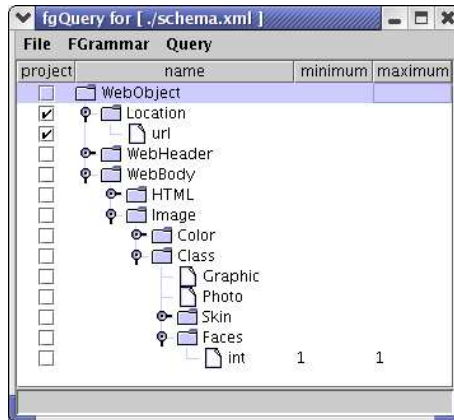


Figure 5.3: The query interface.

the predicates and traverses up the selection tree. After all selected trees are collected a traversal down the projection tree for each selected tree is started and each requested projection is printed. Some special measures are needed to check the contextual validity, *i. e.* a conjunction is only valid within the same context.

As query documents get quite verbose a simple *graphical user interface* GUI shows the tree derived from a specific schema document (see Figure 5.3). Using this tree control projection nodes can be marked and simple predicates can be defined. The query can then be stored as an XML document or directly be executed.

## 5.2.6 Adding Database Management to a Database Kernel

Monet is a database kernel, which means that it only provides the kernel primitives for a full fledged DBMS. The previous sections described how a feature grammar specific XML front-end was build on top of this kernel, however, there are still some key components lacking. To get a reasonable data throughput for a web crawler concurrent updates of the database are needed. BATs are by default not locked on read or write access, *i. e.* locking is left to the application programmer. The default extension modules offer the *fork* command and the *lock* atom type as building blocks for concurrency and a transaction mechanism. Using these a simple transaction system on the MIL level was realized, thus allowing concurrent access.

To allow asynchronous communication between the Acoi tools and the Monet backend a queuing mechanism was added. This enables a FDE to put its XML insertion request into the queue and request the next instructions from the FDS.

The bottom-line was achieved when all queries spend the major part of their idle time in waiting on the non reentrant MIL parser. The next major version of Monet will

contain a reentrant MIL parser.

### **5.3 Discussion**

This chapter described several possible XML storage schemes. This type of research has been developing rapidly over the past few years (see [Bou03] for an extensive overview of mappings). The current, rather ad-hoc, implementation used by Acoi is just a bottom line. Replacement by one of those, concurrently developed or newer, schemes, *e. g.* Monet XML, is now more of an engineering task than a research topic.



# Chapter 6

## Feature Detector Scheduler

*The library is a growing organism.*

S.R. Ranganathan - Law five of *The Five Laws of Library Science*

The FDE, as introduced in Chapter 4, can easily construct a huge collection of hierarchical structures, which are related by references and thus, on a higher level of abstraction, form a graph. The previous chapter introduced a persistent storage model for these structures based on Monet. However, the annotations contained in these structures should be kept synchronized with the (external) multimedia objects they describe. Next to those external changes, the annotation extraction algorithms may change over time, *i. e.* another reason for maintenance. This chapter describes the contour of the *Feature Detector Scheduler* (FDS), whose main goal is to steer the FDE to execute incremental parses and thus propagate the localized changes. Unfortunately, there is no complete FDS implementation experience, but the sketch is backed by prototypes of the core parts.

The core parts of the FDS are shown in Figure 6.1. In the subsequent sections these components will pass the revue and their relations to each other will be described in some depth. The last section will also contain a short discussion on the implementation.

### 6.1 The Dependency Graph

The basis of the FDS is its analysis of the dependency graph. The dependency graph describes how all the known symbols of a feature grammar relate to each other. Figure 6.2 shows the dependency graph based on the example HTML feature grammar (see Example 3.1).

The symbols play these roles (notice that symbols may play multiple roles):

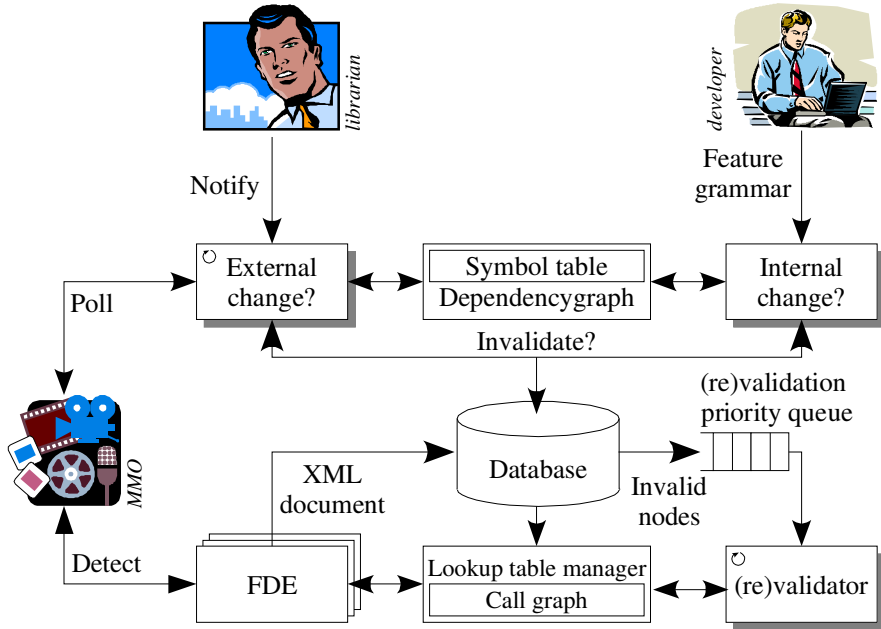


Figure 6.1: The FDS components

**start symbols** are the roots of the hierarchical structures, *e. g.* the *WebObject* symbol;

**detector symbols** are the symbols which are associated with an annotation extraction algorithm, *e. g.* the *WebHeader* symbol;

**transparent symbols** are anonymous symbols introduced by the rewrite rules, *e. g.* the *\_grp\_1\_* symbol, which is introduced to capture the grouped optionality of the *WebHeader* and *WebBody* symbols;

**terminal symbols** are instantiated with a value belonging to the symbols domain, *i. e.* they contain the real annotation information like the *date* atom;

**non-terminal symbols** are the symbols without any other specific role. They provide an intermediate semantic (grouping) level.

This information is stored in the symbol table (see Section 4.3.1). From the production rules the basic dependency graph is derived. Then the various path expressions are resolved on this meta-level, *i. e.* all instantiation (value) related predicates are skipped. For example for this whitebox predicate:

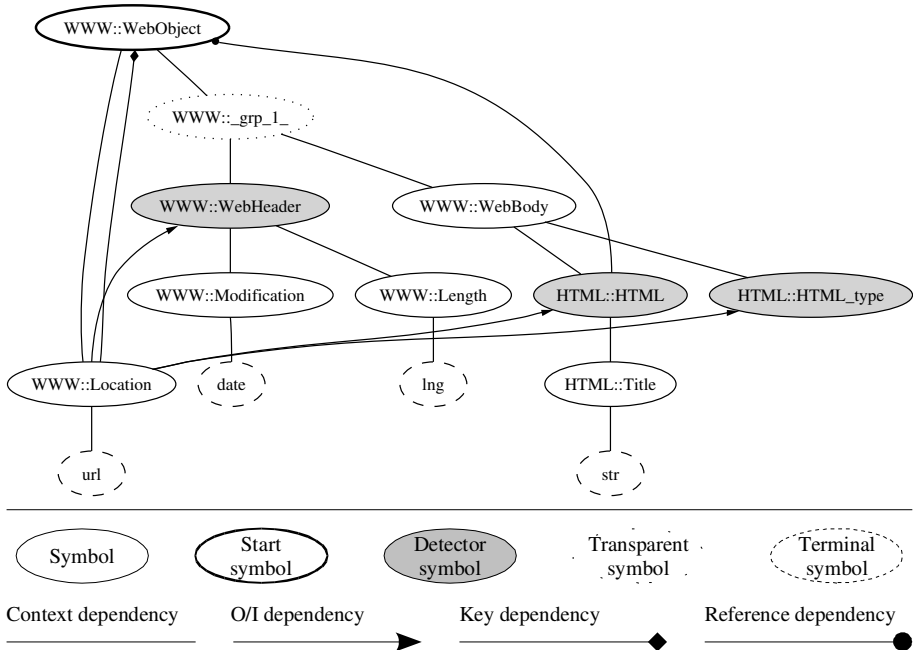


Figure 6.2: The HTML feature grammar dependency graph

```

1 |%detector   ColorMap [
2 |           some $RGB in RGB satisfies
3 |             $RGB/Red != $RGB/Green
4 |           or $RGB/Red != $RGB/Blue ];

```

these paths are resolved, with *ColorMap* as the context node, within the graph:

1. *self* :: \*/preceding :: RGB
2. *self* :: \*/preceding :: RGB/child :: Red
3. *self* :: \*/preceding :: RGB/child :: Green
4. *self* :: \*/preceding :: RGB/child :: Blue

This whole construction process results in these relationships between the nodes in the dependency graph:

**context dependencies** are the most basic parent/child relationship between the LHS and the symbols in the RHS of a rule. The child, *i.e.* the RHS symbol, always

depends for its validity on the validity of the parent, *i. e.* the LHS symbol. For parents the validity depends on all the mandatory children, *i. e.* those dependencies with a lower bound of one or more. The *WebObject* does not depend on the *\_grp\_1\_* symbol to be valid, however, *\_grp\_1\_* depends on *WebObject*. This way the parent/child dependency also enforces sibling dependencies;

**output/input dependencies** between annotation extraction algorithms: an algorithm takes its input parameter from the result parse tree of another algorithm. Assuming that the output of the algorithm directly depends on this input, changes in the values of the parameter symbols lead to the need to rerun the algorithm. For example: the *WebHeader* detector depends on the *Location* non-terminal as input parameter;

**key/reference dependencies** define the relationships between quasi-roots and foots. Start symbols may specify an initial set of required tokens. The same tokens are used to resolve references, *i. e.* they function as a composite key to an (unique) hierarchical structure. When those key values change the referential integrity of the relationships have to be revalidated. A tree rooted by a *WebObject* symbol is identified by its *Location*. Links found in a HTML page are represented by references to the corresponding *WebObject* trees. These references should be removed when specific *WebObject* trees are deleted.

Upon a change in the feature grammar system the dependency graph is recreated. However, to localize change and start updating the parse trees in the database the FDS needs one (or even several) starting points. The next section shows how these starting points are identified.

## 6.2 Identifying Change

Section 1.2.4 identified two sources, which are also reflected in the feature grammar language (see Section 3.2.5), of change in a DMW: external and internal. The FDS has to cope with both sources by using the handles specified in the feature grammar system and given by the developer.

### 6.2.1 External Changes

The first type of changes happen outside of the annotation system: the source multimedia data, which may or may not reside in the same database as the annotations, changes<sup>1</sup>. The FDS may be notified of such changes by four sources: (1) the librarian, (2) the FDE requests validation of a (new) multimedia object,(3) an (external) system, *e. g.* a database trigger, or (4) the exploration date of stored data passes a certain threshold.

---

<sup>1</sup>Notice that a new multimedia object is also considered a change.

The FDS reacts to these notifications by polling the source data. This polling is supported by a special detector, which is associated with a start symbol. The polling detector always receives the initial token set plus additional tokens needed to establish any modification of the source data since the annotations were extracted. For example the polling for web objects takes not only the location of the object but also the stored modification date (see the example in Section 3.2.5). The polling detector can then simply send a HTTP HEAD request to the web server and determine if the returned modification date is newer than the stored date.

When the object is considered modified its root node is invalidated in the database and added to the (re)validation queue.

## 6.2.2 Internal Changes

Updates to the feature grammar system and its associated detector implementations are considered internal to the annotation system, *i. e.* it has all the information about them. The stored annotations should reflect the current output of these implementations. These internal changes are always related to detector symbols, *i. e.* grammar components, and thus to the start condition, the detector function and the stop condition, *i. e.* the production rules. Notice that instead of single multimedia objects, as is the case with external change, internal changes refer to sets of multimedia objects.

### 6.2.2.1 The Start Condition

The start condition changes when one or more of the XPath expressions for binding of the parameters change. This invalidates the lookup table, *i. e.* the memoized parse trees, rooted by this detector as the input sentences may have changed. The consequence is that all these trees are deleted and the parse trees they were member of are scheduled for revalidation. During the revalidation process the new lookup table will be reconstructed using the new input sentences.

### 6.2.2.2 The Detector Function

Changes in the implementation of a detector function are reflected by the version declaration for the specific detector. Such a version consists of three levels. The lowest level indicates a *service revision*. These revisions will not lead to invalidation of any nodes in the current stored parse trees, so the FDS does not have to take any further action, *i. e.* the data is updated when an external change or more severe internal change triggers revalidation. Changes of the next level, the *minor revisions*, will lead to invalidation of the partial parse trees. However, the data may still be used to answer queries. Those revalidations are scheduled with a low priority. High priorities are used for invalidations caused by *major revisions*. In these cases the changes are so severe that the stored data has become unusable and are deleted from the database.

### 6.2.2.3 The Stop Condition

The production rules of the feature grammar component describe the valid output sentences. When these rules change the output sentences have to be revalidated. Changes to these rules can be found by a tree matching algorithm [ZS89, ZS90, CGM97, CAM01]. With the rise of XML these algorithms have found a new public, and several implementations are (freely) (on line) available (*e. g.* [Inc02, Mic02, IBM01, YW03]). From both feature grammar versions the derived XML schema document<sup>2</sup> (see Section 5.2.1) is fed into the diff implementation. For example, the *Delta* tool [Ltd03] will report these changes, when the web objects MIME type is extracted by *WebHeader*:

```

1  ...
2  <WWW:WebHeader deltaxml:delta="WFmodify">
3    <WWW:Modification deltaxml:delta="unchanged"/>
4    <WWW:Length deltaxml:delta="unchanged"/>
5    <WWW:MIME_type deltaxml:delta="add" type=".non-terminal."/>
6  </WWW:WebHeader>
7  ...
8  <WWW:MIME_type deltaxml:delta="add" type=".non-terminal.">
9    <WWW:Primary type=".non-terminal."/>
10   <WWW:Secondary type=".non-terminal."/>
11 </WWW:MIME_type>
12 <WWW:Primary deltaxml:delta="add" type=".non-terminal.">
13   <fg:str type=".terminal.atom."/>
14 </WWW:Primary>
15 <WWW:Secondary deltaxml:delta="add" type=".non-terminal.">
16   <fg:str type=".terminal.atom."/>
17 </WWW:Secondary>
18 ...

```

In this case the detector can directly be identified, in other cases the context dependencies in the dependency graph have to be traversed from child to parent to the first detector node(s)<sup>3</sup>.

## 6.3 Managing Change

Using the indicators of the previous section a priority queue has been filled with invalid (partial) parse trees. In the process of revalidation the FDS communicates about the parse trees with several FDEs and keeps knowledge about the global relationships between these trees. The next sections will describe this process and the use of this knowledge in some more detail.

<sup>2</sup>if the original grammar is not available anymore the schema can be derived from the meta information in the *Monet* database.

<sup>3</sup>When an intermediate non-terminal is reused in different contexts, there may be multiple detector roots.

### 6.3.1 The (Re)validation Process

The invalidation of nodes in the parse tree, and the scheduling of their revalidation, is handled by the FDS using these steps (the example assumes that the *WebHeader* detector implementation has changed, *i. e.* an internal change):

1. The FDS has invalidated all partial parse trees which have an instantiation of a *WebHeader* symbol as root. This involved *WebHeader*, *Modification* and *Length* nodes (and related terminal nodes), as can be derived by traversing the rule dependencies. For these parse trees incremental parsing processes are scheduled, which can be handled by the FDE. The followup action of the FDS is determined by the result returned by the FDE.
2. If the sub-tree is still valid, the output/input dependencies are checked for modification. If there has been a modification the dependent detector needs to be revalidated.
3. If the subtree has become invalid, the context dependencies are followed upward to the first detector or start symbol which is marked invalid. The FDS will repeat the whole procedure for these invalid symbols.
4. The referential integrity is checked using key/reference dependencies, when key values have been changed or parse trees have become invalid.

### 6.3.2 Lookup Table Management

During the revalidation task other (memoized) parse trees (see Section 4.3.3.3) may be encountered. The FDE provides the input sentence for the mapping described by the requested parse tree. The FDS will query the lookup table stored in the database for the existence of this mapping and return the appropriate information:

1. when the mapping is available: the unique identifier of the tree;
2. when the mapping does not exist: a *null* value;
3. when the mapping is unknown: a new unique identifier;
4. when the mapping is under construction: a timeout after which the FDE will have to resent its request;
5. when a deadlock is detected (see below): the unique identifier and the type of deadlock (direct or indirect).

Next to the requests to resolve or initialize a parse tree, the FDEs also inform the FDS about the validation results (independently of the database storage). Using this information the FDS maintains a call graph of parse trees currently under construction.

Detecting a cycle in this graph indicates a (in)direct deadlock, which is reported to the FDE. The FDE passes this information on to the detector (see Section 4.3.5.6), and will return the result, *i. e.* the mapping is available or does not exist (meaning that the deadlock could not be resolved).

## 6.4 Discussion

As stated in the introduction this chapter contains a sketch of the envisioned system architecture of the FDS, backed by a partial prototype implementation. Figure 6.1 shows the various components. Based on the detection and localization of internal and/or external changes incremental FDE runs are scheduled and controlled by the FDS.

The external change detection component is mainly an interface to the polling detectors. This is a continuous process. With a slight extension to the XML schema documents, *i. e.* to also contain the XPath expressions, the internal change detector component can be completely implemented around an implementation of a tree difference algorithm. The algorithm to derive the dependency graph from the symbol table has been implemented in the latest version of Acoi, as will be shown in the next chapter. The FDE has also been implemented (see Chapter 4) and experiments with incremental parsing have been conducted. However, adaptations, like submitting partial parse trees to both the database and the FDS, may be needed. The (re)validator and the lookup table manager have not been implemented, so the sketches of these components have to be validated by an actual implementation in the future.



# Chapter 7

## Case Studies

*De oplosmythe (1)*

Met een computer is elk probleem op te lossen.

*De oplosmythe (2)*

Met een computer is mijn probleem op te lossen.

*De oplosmythe (3)*

Met een computer is een of ander probleem op te lossen.

Joachim Graf - *De computerwetten van Murphy*

Throughout the development of the Acoi system and its major component, *feature grammars*, several case studies were conducted to assess its practical impact. In the next sections these case studies will be described together with the lessons learned. However, the Acoi system itself is also a case study in software engineering of a system based upon feature grammar systems. This chapter starts with a section describing the successive versions of the system. The case studies can then be related to the version being used, and to the changes they inspired.

### 7.1 The Acoi Implementation

The Acoi system is developed by the database research group of the Dutch Centre for Mathematics and Computer Science (CWI) from 1997 to 2003. This section contains a brief history of the system development so the case studies, which will follow later on in this chapter, can be placed in the right context.

#### 7.1.1 Acoi Prehistory

Before Acoi became a project subsidized by several national projects (AMIS, DMW and Waterland) the database research group already had laid some foundations. In

the late eighties the Grammatical Database Model was investigated in two unpublished manuscripts [Ker89, KS89] and a prototype implementation developed by a master student. This model is based on this quadruple:  $(L, F, G, T)$ .  $L$  is a language described by an unambiguous CF grammar,  $F$  is a set of transducers defined for  $L$  which can produce a new sentence from an existing sentence,  $G$  is a set of guardians defined for  $L$  which determine if a new sentence is valid and, finally,  $T$  is a collection of builtin and user defined types. The transducers and guardians from this model closely resemble the black- and whitebox feature detectors from the feature grammar systems. However, the design of this model was never completely finished.

Acoi stands for *Amsterdam Catalog of Images* and the system's first aim was to build and maintain a collection of images for research purposes. Feature detectors were identified as a basic building block for a database-based content-based image query system. The Acoi image algebra [NK98, Nes00] was a result of these efforts.

Both lines of (past) research flowed naturally into the Acoi project.

### 7.1.2 The Acoi Project

In the fall of 1997 Acoi became a CWI project for providing database support for the management of multimedia features. This internal project was mainly funded by the national DMW project, which will be described in some more detail in Section 7.3. Note that although not all the components were implemented in the various versions of the Acoi system, the general architecture shown in Figure 1.4 has been clear from the start. The exact details still needed to be filled in, which is the focus of this thesis.

### 7.1.3 Acoi 1998

The research started with the construction of a web robot in Java to gather images from the World Wide Web. This robot interacted with and helped debugging the Monet database server using the ODMG interface, which was under development at that moment. Concurrent to this robot a first version of a feature grammar based toolset was implemented [KNW98a]. This implementation would read in a specific feature grammar and generate the  $C$  source code for a grammar specific recursive descent parser.

### 7.1.4 Acoi 2000

The first rewrite, which was mainly targeted at a cleanup of the code base, of the Acoi system was still based on parser source code generation. The parse trees constructed by this generated FDE could first be dumped as a set of  $Mx$  macro calls [KSvdBB96]. The expansion of these macro calls would lead to a MIL script to insert the parse tree into the Monet database. Later on, this propriety setup was replaced by the combination of XML and XSLT (see Chapter 5).

The image robot was also rewritten into a feature grammar with an accompanying set of detectors. This provided the first experience with the general system

architecture. Parts of the previous implementation were not reused as mainly string handling, needed for the parsing of the HTML pages, is far too expensive in *Java*. Also controlling timeouts on HTTP connections turned out to be cumbersome. The implementation of HTML related detectors was therefore done in *Tcl* and later on in *C* [Vei03, W3C02a].

Based on the performance characteristics of the system during several case studies the internals of the system underwent several optimizations. First of all the token pool was hierarchically organized so larger portions were shared by different recursive descent levels in the parsing process. This allowed descending and ascending, *i. e.* backtracking, to be cheap and to cut away extensive copying of tokens.

The next optimization concerned the binding of detector parameters. In this binding process the internal tree had to be traversed. This tree could be traversed by a path expression language loosely based on XPath 1.0. However, when this tree grew big these traversals would visit too many nodes. By allowing additional hints in the path expression, *i. e.* indicating forward or backward traversal, these superfluous node visits could be prevented. For example:

```
1 | %detector shotlist(ancestor::video/child-forward::filename);
```

would ensure that traversal of the *child* axis would start at the first child of *video*. This in contrast to the default traversal strategy which implemented a backward depth-first search.

Feature grammars were seen as CF grammars with an limited amount of context-sensitivity [SWK99], *i. e.* not yet as a specific instantiation of CD grammar systems. Ambiguity was only allowed in a limited fashion: all alternatives should consume exactly the same tokens.

Around the FDE an extensible set of scripts in various languages, *e. g.* *Tcl*, *MIL* and *XSLT*, was build to implement the WWW search engine (see Figure 7.1). These scripts contained various hooks to insert knowledge not explicit in the feature grammar (at that time). This setup will be discussed in some more depth in Section 7.2.

This version of *Acoi* has been used extensively for case studies and has been described in [SWK99, WSK99, dVWAK00, NWH<sup>+</sup>01, WSvZ<sup>+</sup>03].

### 7.1.5 Acoi 2002

To accommodate the expected FDS implementation, the FDE was rewritten into an interpreter. As indicated in Chapter 4 an interpreter handles a changing grammar more easily and spares the FDS the hassle to manage a recompilation of a grammar specific FDE.

The feature grammar language, and also its parser, was redesigned to cope with modules, whitebox detectors, classifiers and plugins. The support for these features was added to the generic FDE. Detectors and plugins became dynamic loadable libraries. Furthermore, the implementation made use of more XML standards: DOM

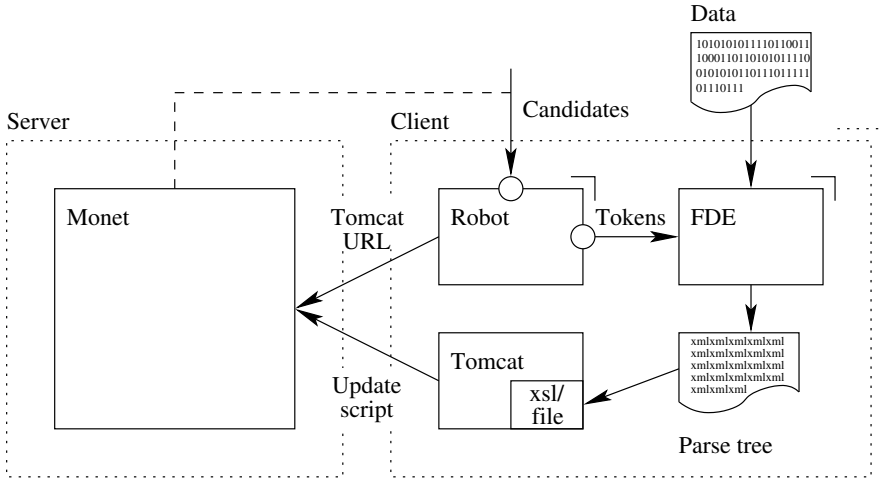


Figure 7.1: The WWW multimedia search engine architecture.

for the internal parse tree format and subsets of XPath for whitebox detectors and detector parameters [Vei03]. Feature grammar checks were added to warn for various (possible) weaknesses in the grammar. This included the semantic checks described in Section 4.3.1.2.

A first implementation of the FDS was able to construct and visualize the dependency graph. Also experiments were conducted with various implementations of XML diff algorithms.

The robot was once more reimplemented. In this case the monolithic grammar was cut up into several media type related feature grammar modules. Lessons on the patterns embedded in these modules will be described in the upcoming Section 7.2.

### 7.1.6 Acoi Future

The current implementation of Acoi still lacks some of the key aspects of the theory described in this thesis. The main absence is a complete FDS to replace the scripts which make up the WWW robot. The current FDS implementation lacks interaction with one or more FDEs and an interface to a tree diff algorithm to find some internal changes.

As described in Chapter 4 parse forests can be stored in one XML document by adding a scope and context attribute. At this point this level of ambiguity is not supported by the FDE. The FDE allows multiple alternatives to be true, but they all should describe the same subsentence. However, the scope/context scheme has been prototyped using a set of XML documents and XSLT templates.

Likewise, the view of feature grammars as feature grammar systems is not completely reflected in the implementation. This view makes a cleaner separation between subsentences as produced by different detector functions and thus belonging to different grammar components. This asks for these sentences to be only within the scope of their component and thus prevents the need for hierarchical sharing of tokens.

Once Monet completely supports an XML/XQuery front-end (see Section 5.1) the specific XSLT scripts can be deleted as XML documents can then be stored natively. This would also accommodate a closer, but still standardized, binding between the FDE and the database. Also allowing a clean addition of support for both memoization and references.

## 7.2 The WWW Multimedia Search Engine

One of the first targets for Acoi was the construction of an image index for the Dutch AMIS project. The size of the index aimed for was 1,000,000 images. To find these images the HTML pages containing their URLs had to be parsed and interpreted, so soon the index was extended with a full text indexing facility. As this case study played a major role over the years [KNW98b, WSK99, WSK00, BWvZ<sup>+</sup>01] annotation extraction algorithms were added for other multimedia types. In the upcoming sections the feature grammars and system architecture involved will be discussed in more detail.

### 7.2.1 The Feature Grammars

Moving away from the first monolithic feature grammar, the current set of feature grammar modules are very similar to the running examples in the previous chapters and showcased in Appendix B. The major decision points used in the grammars are based on the MIME type of the multimedia object under inspection. This MIME type is retrieved by the generic *WebHeader* detector which knows the HTTP protocol to retrieve this information. Using the *primary* and *secondary* MIME type, whitebox detectors in the feature grammar steer the FDE to the set of multimedia type specific detectors, *e.g.* language detection for HTML pages and face detection for images. Detectors for a specific multimedia type are grouped into one feature grammar module. The *Acoi* module combines all the modules into one grammar, which is used by the FDE to harvest links from the web. The complete set of detectors is listed in Table 7.1.

As all multimedia web objects are related through HTML pages it is possible to traverse these anchors and access a specific context of an object. Using this navigational information this, typical, query can be answered: *show me a web page about "Chet Baker" containing a portrait*. This query combines key words from the HTML page ("Chet Baker") with a high-level concept (portrait) extracted from the image object.

<i>Multimedia type</i>	<i>Detectors</i>
Generic web objects	HTTP header information allowance by the robot exclusion protocol
Text files	DRUID language classification
HTML pages	title, anchors and text extraction WordNet synsets [CSL01]
images	global color features graphic/photo classification [ASF97] skin coverage [GAS00] face detection [LH96] portrait classification thumbnail creation
MP3 audio files	ID3 tag extraction
MIDI audio files	MIDI fields
MPEG video files	animated video icon

Table 7.1: The WWW multimedia search engine detector set.

## 7.2.2 The System Architecture

The system architecture (shown in Figure 7.1) uses a set of shell, Tcl, MIL and XSLT scripts to explore the World Wide Web. The various system components provide hooks to plugin feature grammar specific scripts. These hooks are mainly used to implement knowledge about references, as those are not supported by the generic tools in Acoi 2002. The FDS as described in Chapter 6 exploits the explicit knowledge of multiple start symbols and references and can take over the role of these scripts in a future version of Acoi.

A small walk through will clarify the role of the various system parts. The user, *i. e.* the librarian, starts the database server and provides an initial set of candidate URLs. The user also starts one or more robots for corresponding Internet domains and/or multimedia types. Each robot contacts the database server for a subset of the candidates and starts a number of FDEs to index these. The FDE returns the location of the XML document containing the parse forest to the robot, which in its turn tells it to the database server. The database server contacts the Tomcat servlet engine [Pro03] to retrieve the XML document and transform it into a MIL script. The servlet engine provides several advantages: (1) it limits the startup time of the *Java*-based XSLT processor<sup>1</sup>, (2) it provides the possibility to run each robot on a different remote machine and thus makes the architecture more scalable.

<sup>1</sup>*XT* [Cla99] is, although written in *Java*, one of fastest XSLT processors [Dat01]. However, the startup time of the *Java* interpreter is still significant, but can be reduced to a one time affair by embedding *XT* in a servlet [Tch01]. The alternative to link in a XSLT processor with the FDE has been proved to be still slower than this *XT* setup when the parse forest grows into a XML document of several hundred megabytes.

The extensibility of the system was tested several times by the addition of new detectors. A detector basically means the addition of a new branch to a parse forest. For this the FDE supports incremental parses of existing parse trees. This parse tree is loaded from the database and a special command line option tells the FDE for which symbol to start (and stop) the detection process. The FDE will thus rebuild the internal parse tree using the retrieved tokens and will start detection when the new symbol is encountered. When the FDE returns to the symbol in the post traversal the detection is stopped again. In this way the extended parse tree (or forest) is build and sent to the database server. The database server will replace the old parse tree by this new one.

Using this architecture the robot harvested (within 2 weeks in 2000) links to 4,300,000 web objects from which it entirely indexed about 2,000,000. The index contained 750,000 images from which about 10% were classified as photographs. The major bottleneck of the system was the MIL parser in the Monet database server. This parser is non-reentrant and thus protected by a lock. Concurrent parse forest insertion scripts spent their time mainly waiting to obtain this lock.

The scalability of the index has been extended on the level of the Monet database server by passing the terms on to a specialized full text indexing service [Blo02]. This service uses several machines for horizontal fragments of the term index. A term is assigned to a fragment based on the *tf·idf* ranking model [BYRN99]. This integration has been described in [BWvZ<sup>+</sup>01] and [WSvZ<sup>+</sup>01].

This WWW indexing engine is different from traditional search engines at several points. Traditional search engines are mostly based on information retrieval (IR) theory and use technology common in this line of research [BYRN99], *e. g.* inverted files instead of a database system. These IR indexes are build from scratch with each new web crawl. In the feature grammar case updates to the index happen incrementally, *i. e.* queries (readers) and crawling (writers) find place on the same database using concurrent transactions. Furthermore, due to the feature grammar the index is easily extended with new multimedia types and new features.

### 7.2.3 Lessons Learned

The WWW multimedia search engine has been incrementally developed during the various version of Acoi. The extended language features – modules and references – were mainly inspired by this case study. Modules make it possible to easily reuse well defined parts of the feature grammar in a different context, *e. g.* the other case studies. The fact that the anchors between HTML pages form a graph complicated the annotation extraction from the start. Its possible to keep this knowledge implicit by embedding it in the detector implementations. However, making it explicit gives the Acoi tools the opportunity to handle referential integrity and to offer support for complicated recursive structures, *i. e.* deadlock detection. Ambiguity did not play a major role within this feature grammar system. Decision points are deterministic and mostly handled by whitebox detectors.

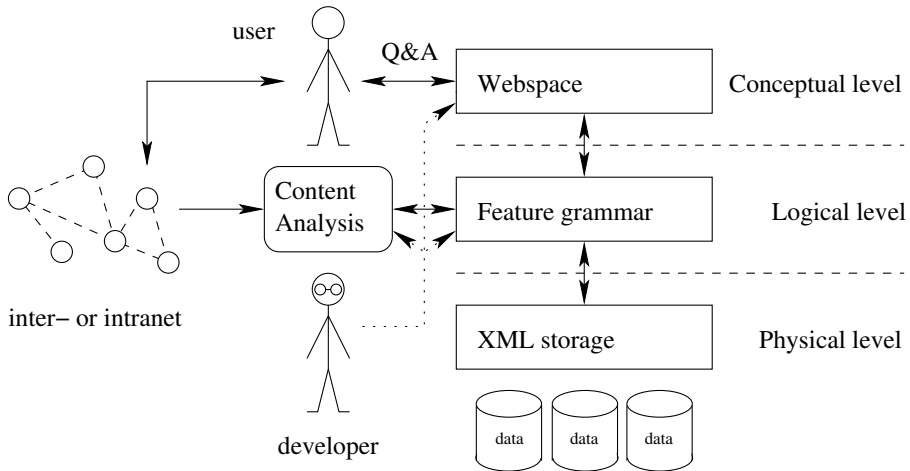


Figure 7.2: The DMW project levels.

### 7.3 The Australian Open Search Engine

Research on the Acoi system has been mainly carried out within the *Digital Media Warehouse* (DMW) project. This project's aim was to advance content-based retrieval techniques in large multimedia databases. To achieve this goal the project was split up in sub-projects for three levels (see Figure 7.2):

1. the conceptual level focuses on querying semi-structured data;
2. the logical level focuses on steering multimedia annotation extraction;
3. the physical level focuses on the storage of semi-structured data.

The logical level directly interacts with a collection of content analysis algorithms, also part of the research portfolio of the project.

Feature grammar systems and the accompanying Acoi system implement the logical level. The physical level was implemented by Monet XML (see Section 5.1.1.2). Both the content analysis algorithms and the conceptual level were developed at the University of Twente. Before describing the Australian Open case study in more detail the research of these project members is shortly described<sup>2</sup>.

<sup>2</sup>Parts of the subsequent sections are written by the co-authors of DMW related papers.



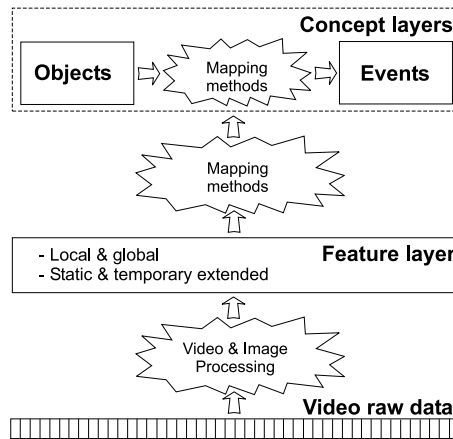


Figure 7.3: The COBRA video modeling framework.

### 7.3.1 The Webspaces Method

The conceptual level focuses on limited domains of the Internet, *i. e.* intranets and large web-sites. The content provided on such domains is often highly related and structured. This aspect makes it feasible to determine a set of concepts, which adequately describe the content of the document collection at a semantic level.

The Webspaces Method [vZ02] offers a methodology to model and search such a document collection, called a webspaces. The Webspaces Method defines concepts in a webspaces schema using an object-oriented data model. This collection is stored as XML documents in the XML storage level of the global system architecture, see Figure 7.2. A strong correlation between the persistent documents is achieved, since the structure of each XML document describes (a part of) the webspaces schema in turn. Actually each document contains a materialized view over the webspaces schema; it contains both content and schematic information.

The webspaces schema is used to formulate queries over the entire document collection. Novel within the scope of search engines and query formulation over document collections is that it allows an user to integrate information stored in different documents in a single query. Traditional search engines (*e. g.* AltaVista) are only capable to query the content of a single document at a time. Furthermore, using the Webspaces Method specific conceptual information can be fetched as the result of a query, rather than a bunch of relevant document URLs.

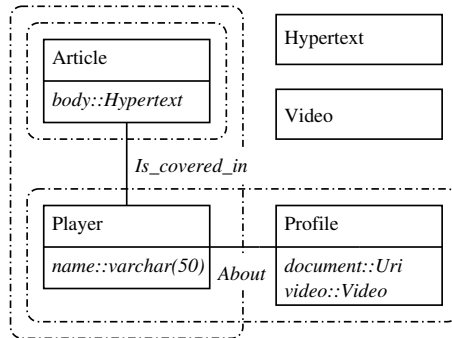


Figure 7.4: A fragment of the webspace schema for the Australian Open website.

### 7.3.2 COBRA

In line with Chapter 1 the COBRA video modeling framework [Pet03] recognizes four layers (see Figure 7.3): the raw data, the feature, the object, and the event layer. The object and event layers consist of entities characterized by prominent spatial and temporal dimensions respectively. In [Pet03] several instantiations of this model are constructed for different domains and using different machine learning techniques. As will be shown in the upcoming section, feature grammar systems provide a way to build domain specific instantiations of the COBRA model.

### 7.3.3 The Australian Open DMW Demonstrator

To demonstrate the power of the DMW system the Australian Open demonstrator was built [BWvZ<sup>+</sup>01, WsvZ<sup>+</sup>01, PwvZ<sup>+</sup>02, WsvZ<sup>+</sup>03, WvZ03]. The Australian Open<sup>3</sup> is a grand slam tennis tournament on a yearly basis. The demonstrator is based on the tournament of 2001.

The conceptual elements available in the structure of the website were modeled in a webspace schema. A fragment of this schema is shown in Figure 7.4. Using a set of special purpose feature grammars the HTML pages from the original website were transformed into the base XML documents of the webspace. These documents contain an instantiation of part of the schema (see the areas in Figure 7.4). As the website did not contain any video fragments from the matches, some matches were recorded and digitized. Then the index database for the base data, including the multimedia content, was built. For this the webspace tools extracted meta-data from the base documents, and triggered the FDE when a multimedia object was found. The FDE would then steer the video annotation extraction process. This process worked along the lines

<sup>3</sup>[www.ausopen.org](http://www.ausopen.org)

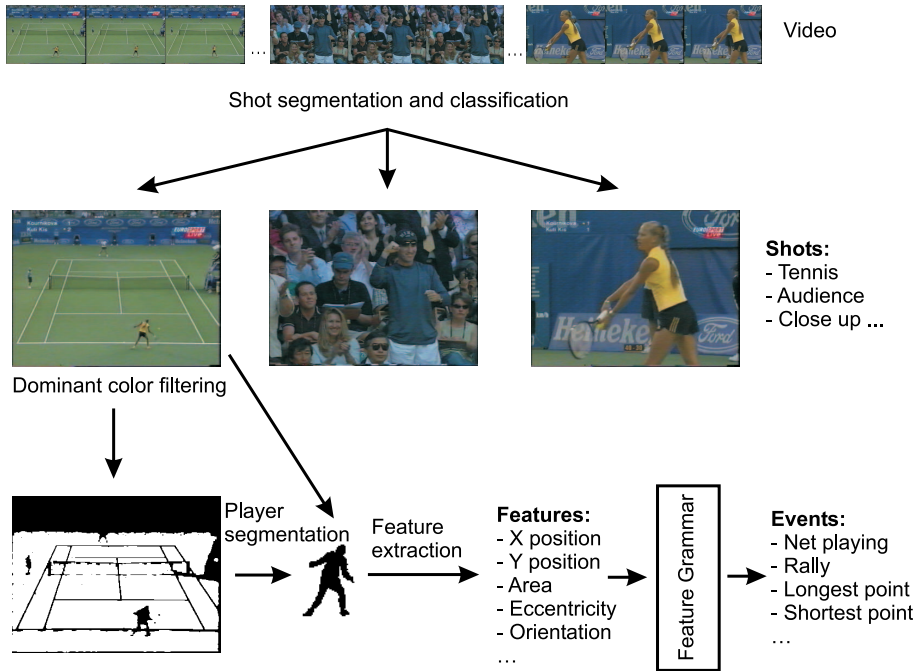


Figure 7.5: Tennis video annotation.

shown in Figure 7.5 and captured in the feature grammar in Appendix B.11. This feature grammar is a domain specific instantiation of the COBRA model. Notice that this grammar reuses feature grammar modules developed for the WWW multimedia search engine.

Finally a special purpose query interface was build. The formulation of a query in this GUI can be divided into three steps. During the first step, the query skeleton is constructed, using the visualization of the conceptual schema. Secondly, the constraints of the query are formulated, using the attributes of classes used in the query skeleton. In the last step, the user has to define the structure of the result of the query, which is generated as a materialized view on the conceptual schema.

Before continuing with the individual steps of the query formulation process, the queries presented below are used to illustrate the power of the search engine with respect to query formulation. The queries express the typical information need of an expert user querying the Australian Open document collection. It also shows, how after each query, the information need of the user can be refined, resulting in a more complex query over a document collection.

Q1. *'Search for left-handed female players, who have played a match in one of the*

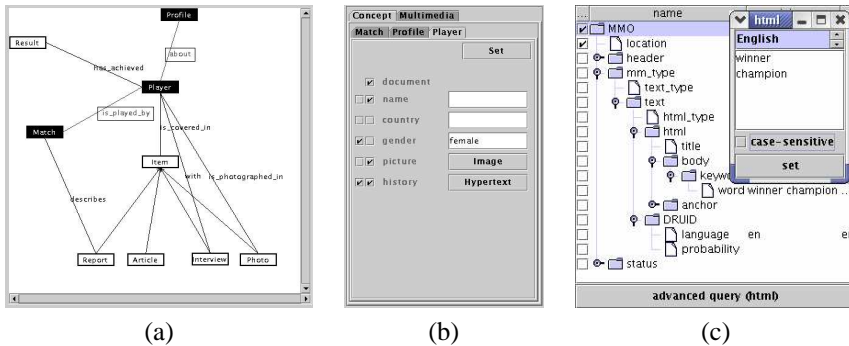


Figure 7.6: Formulating a query.

(quarter/semi) final rounds. For each of those players, show the player's name, picture, nationality, birth-date, and the document URLs containing information about this player. Also show category, round and court of the match'.

- Q2. 'Like query 1, with the extra constraint that the player has already won a previous Australian Open. Include that history in the result of the query'.
- Q3. 'Extend query 2 with the constraint that the result should also contain video-fragments, showing net-playing events'.

The first example query shows how conceptual search is used to obtain specific information originally stored in three different documents. The second example query extends the first query and provides an example illustrating the integration of content-based text retrieval in the conceptual framework. The third example query extends the complexity of the query even more, by integrating content-based video retrieval.

1. **Constructing the query skeleton.** The first step of the query formulation process involves the construction of the query skeleton. This skeleton is created, using a visualization of the webspace schema. This visualization consists of a simplified class diagram, and only contains the classes and associations between the classes, as defined in the webspace schema. The user simply composes the query skeleton, based on his information need, by selecting classes and related associations from the visualization. The (single) graph that is created represents the query skeleton.

In Figure 7.6.a a fragment taken from the GUI of the webspace search engine is presented, which shows the query skeleton (depicted in black-filled text boxes), that is used for the query formulation of the three example queries.

2. **Formulating the constraints.** In the second step of the query formulation process, the constraints of the query are defined. In Figure 7.6.b another fragment

of the GUI of the search engine is presented, showing the interface that is used for this purpose. For each class contained in the query skeleton a tab is activated, which allows a user to formulate the conceptual constraints of the query. As shown in the figure, a row is created for each attribute. Each row contains two check boxes, the name of the attribute, and either a text field or a button.

The first checkbox is used to indicate whether the attribute is used as a constraint of the query. The second checkbox indicates whether the results of the query should show the corresponding attribute. If the type of the attribute is a `BasicType`, a textfield is available that allows the user to specify the value of the constraint, if the first checkbox is checked. If the attribute is of type `WebClass`, a button is available, which, if pressed, activates the interface that is used to query that particular multimedia object.

Figure 7.6.c shows the interface that is used to formulate queries over `Hypertext-objects`, *i. e.* define content-based constraints. The figure shows both a low-level and advanced interface to the underlying feature grammar system. In the low-level interface projection and selection criteria can be filled in (see Section 5.2.5). The advanced interface is similar to the interfaces offered by the well-known search engines such as Google and Alta-Vista. The main part of the query-interface allows a user to formulate one or more terms, which are used to find relevant text-objects, The interface also allows the user to perform a case-sensitive search, and to select the language of the `Hypertext-object` in which the user is interested.

Figure 7.6.b shows the attributes of class `Player`. The constraints with respect to the player, specified in the first two example queries, are transposed in the selections depicted in the figure. Two constraints are formulated. The constraint that the user is only interested in *female* players is defined by selecting the constraint checkbox in front of gender, and by specifying the conceptual term '*female*'. The second constraint refers to the second example query, where an extra condition with regard to the player's history is formulated. Again, the corresponding checkbox is activated, and the interface of Figure 7.6.c is started, and completed. In this case, the query-terms '*winner*' and '*champion*' are used to find the relevant `Hypertext-objects` that are associated with the player's history.

3. **Defining the resulting view.** The second column of checkboxes is used to specify which attributes will be shown in the resulting views defined by the query. The XML document that is generated by the webspace search engine contains a (ranked) list of views on the webspace that is being queried. Besides selecting the attributes and the classes that will be shown as the result of the query, the user also has to determine which class is used as the *root* of the resulting view. In Figure 7.7 a screenshot of the result for the third query is shown. It includes a link to a tennis scene of the match played by Monica Seles

```


1.83827948486295
● Player
  gender:female
  name:monica seles
  history
    Australian Opens Played: 5
    Best Singles Performance: Winner (1991, 1992, 1993, 1996)
    Best Doubles Performance: Semi Finalist (1991)
  Events Entered:Women s SinglesWomen s Doubles (with Martina Hingis)
  documents
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]
  picture
    
  about
    ○ Profile
      birthdate:2 december, 1973
      nationality:united states of america
    is_won_by
      ○ Match
        category:women's singles
        court:rod laver arena
        round:qtr. finals
        video:fragment
          http://vzwol.org/susopen/video/c:apriatiselesao01.mpg
            1. tennis scene from frame 2016 to frame 2634
  
```

Figure 7.7: The result of example query 3.

in the quarter final round. The tennis scene shows a video-fragment in which Monica Seles plays near the net.

This DMW architecture consisting of webspaces, feature grammars, an instantiation of the COBRA video model and efficient XML storage resulted in a search engine which allows a combination of conceptual and content-based multimedia search [WvZ03], thus giving the user the power to post very specific queries to the database.

### 7.3.4 Lessons Learned

The feature grammar system for the Australian Open case study is a direct extension to the set of feature grammars for the WWW multimedia search engine. The extension did inspire two language features: constants within the production rules and plugins. The *Segment* detector not only detects scenes within the video but also their type. To prevent superfluous type detectors, this type was encoded as a string and matched by a string constant in the various alternatives. A remote procedure call (RPC) detector was developed which was quite generic and thus easily converted into the template-like approach of a plugin.

The main focus of this case study was the embedding of the Acoi system within the larger DMW application. In the WWW search engine the feature grammar is the main schema, but in this case the grammar is connected to the conceptual webspace schema.

Both the webspace tools and Acoi could have steered the meta-data and multimedia extraction process. The choice was made for a top-down implementation, *i. e.* the conceptual level triggers the logical level.

Looking back, the embedding of Acoi within the current DMW system could have been more tight. On the one hand by a tighter coupling with the webspace schema and thus with the conceptual data, allowing more specific semantic contexts for detectors. This coupling could be created by translating the concepts and their attributes into elementary feature grammar trees. Also on the side of the multimedia content analysis the current integration is shallow, *i. e.* the detector granularity is very coarse. By splitting the implementation of the *Segment* and *Tennis* detectors into smaller detectors, decision points can be made explicit and thus become manageable by the FDE and FDS. Drawbacks of a finer detector granularity is the possible many conversions needed from token to actual values, *i. e.* strings to integers or floats, and opening and closing of the media object. The latter drawback may be circumvented by adding a generic caching interface to the Acoi toolset.

## 7.4 Rijksmuseum Presentation Generation

As stated in Chapter 1 museums are digitizing their collection and making them available to the public. The Rijksmuseum, situated in Amsterdam, did the same. High resolution scans of photos made of paintings, statues etc. and the existing annotation database were made available to researchers in the Token2000 project. The Rijksmuseum Presentation Generation project developed an architecture for using (automatic extracted) annotation information for the automatic generation of user specific hypermedia presentations [NWP<sup>+</sup>01, NWH<sup>+</sup>01]<sup>4</sup>.

The architecture is shown in Figure 7.8 and consists of three major units:

- the style repository, which embodies style schemata, style grammars and rule-bases for different presentation styles;
- the data repository, containing the images and related meta-data, and the retrieval engine;
- the presentation environment, including a presentation generator and a hypermedia browser.

In the next sections these units will be introduced and the way in which they interact with each other will be described.

### 7.4.1 The Style Repository

The aim of the style repository is twofold. On the one hand it provides a collection of representations describing styles in fine art, such as *clair-obscur*, *impressionism* or

---

<sup>4</sup>Parts of the subsequent sections are written by the co-authors of these papers.

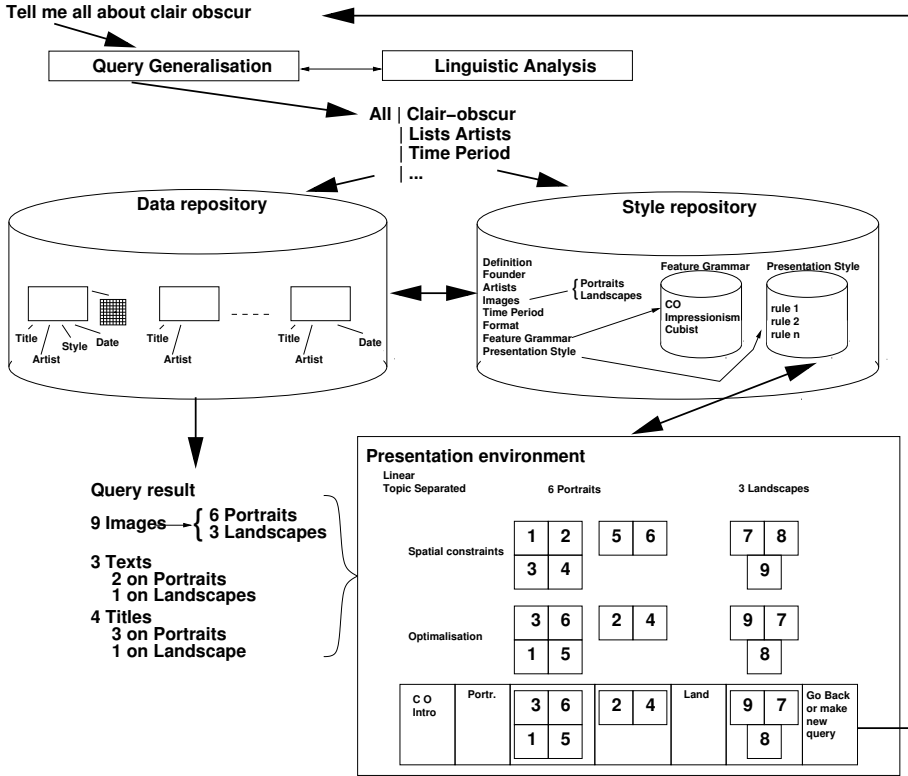


Figure 7.8: Hypermedia presentation generation framework.

*cubism*, in a structured way. These collections are designed to improve the retrieval of images or other meta-data in the data repository (see the data repository section below). On the other hand it provides a presentation rule-base in which rhetorical structures describe how retrieved material can be presented.

The collection of representations of fine art styles provides for each style mainly text-based schemata. A schema holds information about the definition, the main period, the inventor of this style, other artists using or improving it, etc.. The advantage of these style ontologies is that they allow an enlargement of the search-space, if the style plays a prominent part in the query. This is state-of-the-art technology within the retrieval community [SR00]. The development of text-based ontologies is still a mainly manual task, which is today quite well supported [GEF+99].

This information, while important, is insufficient, low level feature descriptions are also required. Rather than a random choice of features as a style description, features that represent the intrinsic characteristics of a particular style need to be col-



lected. In *clair-obscur* images, for example, a clear distinction of light and dark areas can be found. Usually there is one dominant light source, predominantly filled with high luminance colors, alongside dark areas with a high proportion of brown colors which can be blended with other objects [Arn74]. Thus, a collection of features such as color, shapes, brightness, either in the form of their extraction algorithms or as threshold values for a particular style, facilitate the automatic identification of relevant material. Such a collection can naturally be represented by a feature grammar.

Note that the development of feature-based representations also requires human effort, in particular by specialized experts who have an understanding of the compositional structures of an image [Pei60]. The collection of these features is, on the other hand, not too difficult, since tools for this particular task do exist [FCP00].

The third representation form in the style repository is of a different sort. Here rules are collected which describe rhetorical presentation structures, as addressed in the Rhetorical Structure Theory [MMT89] or Cognitive Coherence Relations [KD96], which might vary between general and specialized levels. If, for example, the presentation environment is educationally oriented, it can build presentations on a larger level of the form: Introduction Topic; Introduction Subtopic 1; Details Subtopic 1; Introduction Subtopic 2; Details Subtopic 2; Introduction Next Topic. A more detailed level specifies what an introduction means, *e. g.* show a definition of the topic in combination with a visual example of the topic. Another detailed description might be concerned with the sort of interaction, such as a linear presentation in the form of a slide show, or a more interactive way in the form of additional buttons for individual traversal. The combination of these rules form themselves schemata, which can then be connected to relevant styles. The design of these schemata and the connection to particular style representations again requires human effort, such as that of a graphical designer of a museum Web environment. Development environments which support such tasks are described in [ACC<sup>+</sup>99, NL00]. Once these presentation rules are in place, a system can react to the particular needs of a user.

## 7.4.2 The Data Repository

The repository, as shown in Figure 7.8, stores annotation schemata in the form of XML-based documents and media-based data, such as images in various formats (pic, gif, tiff, etc.). The repository itself can be realized using federated database technology.

The annotation documents are created by experts, using ontology-based environments for task-specific controlled vocabulary/subject indexing schemata for in-depth semantic-based indexing of various media [DAR03]. Note that annotation schemata are different from the style representations. Annotation schemata provide information about one particular image or artist. For example, they capture information about the title of an image, its painter, production date, a list of exhibitions where it was presented, reviews, and so forth. The annotation process follows a strata-oriented approach, which allows a fine-granulated space-oriented description of media con-

tent, where particular areas within an image can be especially annotated. The connection can be based on linking mechanisms as described in XML path and pointer [W3C01d, W3C02b] or MPEG-4 [ISO02].

As visualized in Figure 7.8, the annotations will hardly ever be completed. Most of the time only the most basic data will be available. Thus, even if the potential search space can be enlarged, as described earlier, there may still be a very limited information space to apply the query to.

Imagine that a user would like to know everything about Rembrandt and the different styles he painted his images in. With the textual representation the system might be able to find images by Rembrandt in the database. However, if these images have no further annotation attached than 'Artist = Rembrandt' it would not be able to classify the retrieval results according to the query. Having access to the style specific representation of intrinsic features, the image can now be analyzed during the retrieval process and decide based on the results which of the relevant styles is the most appropriate for this particular image. As a side effect, the feature information gained can be used as additional annotation for the image, not only in classifying it as a particular style, but also providing several different representations of it, which can be, for example, useful for presentational purposes. An image in the style of *clair-obscur* can be additionally represented in a grid that contains the light and dark areas. This grid can form the basis for a presentation of images, where the style of the images and the style of the presentation correspond. The FDE steers this automatic annotation extraction process based on the feature grammar from the style repository.

As the main goal of the suggested framework is to facilitate the automatic generation of user-centered multimedia presentations, the result space will not only contain the retrieved data, associated meta-data, and the relations between these different units but also information required for their presentation. Moreover, it also returns physical information about the retrieved data, *i. e.* image size and image file type.

### 7.4.3 The Presentation Environment

The presentation environment, as displayed in Figure 7.8, is basically a constraint-based planning system, which uses the definitions provided in the style representation schemata and the presentation styles [vOGC+01]. Since the system can access descriptions based on spatial, gradient and color features, the presentation generator is in the position to analyze the retrieved material based on the relevant presentation design, according to design issues such as graphic direction, scale, volume, depth, shapes (*i. e.* physical manipulation of the material for better integration into the presentation), temporal synchronization (interactive or linear presentation), etc., and provides a format that a hypermedia browser can interpret, *e. g.* SMIL or MPEG-4 [ISO02, W3C98, W3C01c].

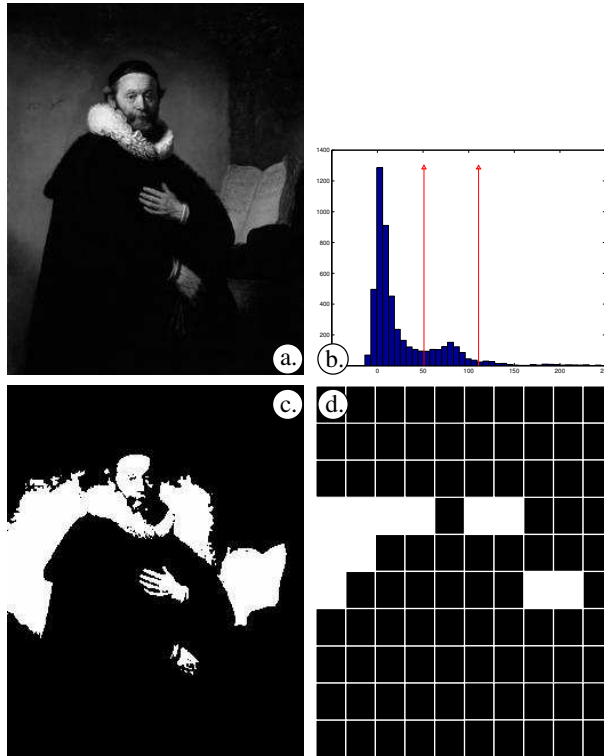


Figure 7.9: Feature detection steps.

#### 7.4.4 A Style Feature Grammar

Feature grammars play a role in both the style and data repositories. In both cases they extract the low-level feature descriptions, which can be used for the selection of relevant material and for the layout of the presentation. But in the latter case the features are also mapped to high-level concepts. These concepts correspond to manual annotations, and can thus replace these when they are not available.

This section will shortly describe the algorithms involved in detecting low-level, presentation oriented, features and high-level, manual annotation replacement, concepts. The grammar itself is available in Appendix B.13.

Since the design aims at an approach that is data-driven and can therefore operate unsupervised, it is important to incorporate adaptive decision-making algorithms. For instance, in the case of the *clair-obscur* style a vague high-level description of the style could be “a *brightly lit* object or person surrounded by a *dark* background”. To translate this vague conceptual description into an operational low-level feature-

extractor, precise values need to be assigned to fuzzy concepts such as *bright* and *dark*. However, these cannot be fixed in advance because these values depend on the context, *dark* and *bright* being defined relative to the rest of the painting.

The proposed approach is data-driven in that it inspects the data in search for natural thresholds, *i. e.* thresholds that are dictated by the structure apparent in the data [PF00]. To be more precise, assume a numerical image-feature  $x$  that can be computed at each of the  $n$  pixel in the image (*e. g.* hue, or brightness, see Figure 7.9.a). This gives rise to a numerical dataset  $x_1, x_2, \dots, x_n$ . The histogram gives an idea of how these values are distributed over the image. If, in terms of this feature, the image has a clear structure then a multi-modal histogram is expected, with peaks over the most-frequently occurring feature values.

For instance, in the case of *clair-obscur*, computing the brightness histogram (at least) two peaks are expected: one peak at low values created by the pixels in the dark regions, and one at high values corresponding to bright pixels, see Figure 7.9.b.

Locating the grey-value at the minimum in between these peaks determines a threshold that can be used to separate the bright from the dark regions in the image. This seemingly simple task is complicated by the fact that a data-histogram almost never has a clear-cut unimodal or multi modal structure, but exhibits many local maxima and minima due to statistical fluctuations. The challenge therefore is to devise a mathematically sound methodology that allows us to construct a smoothed version of the histogram, suppressing the spurious local extrema that unduly complicate the histogram structure.

For this the empirical distribution function  $F_n(x)$  is introduced, which for each feature-value  $x$  determines the fraction of observations  $x_i$  that are smaller than  $x$ . The reason for switching to the empirical distribution is that it allows to compute the precise probability that the given sample is drawn from a theoretically proposed distribution  $F(x)$ . The idea is simple: search for the *smoothest* distribution  $F$  that is compatible with the data, *i. e.* such that there is a high probability that the sample  $x_1, \dots, x_n$  has been obtained by sampling from  $F$ .

In mathematical parlance this amounts to solving the following constrained optimization problem: given  $F_n(x)$  find  $F(x)$  that minimizes the functional

$$\Psi(F) = \int (F''(x))^2 dx \text{ subject to } \sup_x |F_n(x) - F(x)| \leq \epsilon.$$

(The value for  $\epsilon$  is fixed in advance by specifying an acceptable level of statistical risk). This optimization problem can be solved using standard spline-fitting routines. Once the shape of the smoothest compatible distribution  $F$  is determined, its inflection points can be used to determine the genuine local minima in the histogram, thus yielding natural thresholds for the image-segmentation extractor.

The lowest of these thresholds is then used to segment the image into dark and light areas, see Figure 7.9.c. And as a next step information about the areas is localized by overlaying the image with a grid, see Figure 7.9.d.

In the feature grammar of Appendix B.13 these steps are distributed over several detectors and their dependencies are described. For example, the *light\_segment* detector calculates the brightness value of each pixel, this set of values is then taken as input by the *histo\_segment* detector to determine the segmentation thresholds.

The grammar defines several other (global) features. For example, the *co\_corr* detector computes the normalized correlation between the color histogram of the painting and two average normalized histograms, respectively for *clair-obscur* and non-*clair-obscur* paintings (see for a similar approach [ASF97]).

↓ classified as →	clair-obscur	cubism	impressionism	unknown
18 clair-obscur paintings	17	–	–	1
25 cubist paintings	6	21	3	1
56 impressionist paintings	1	5	31	20
83 unclassified paintings	70	1	26	–
182 paintings	94	27	60	22

Figure 7.10: Results for the style feature grammar

All these features form the input for the final step: determining if the painting is in the *clair-obscur* or one of the other styles. For this step style specific decision trees are derived using C4.5 [Qui93], resulting in the detectors *clair\_obscur*, *cubism* and *impressionism*. The performance of the feature grammar in annotating paintings with the various styles is shown in Figure 7.10. The last column corresponds with the event that there is no matching style found, *i. e.* the validity of the *style* rule is optional.

Notice also that more than one of the alternative *style* rules can be valid, which means that a painting can be annotated with multiple styles, *i. e.* ambiguous views on the same painting. So the support for ambiguity by the Acoi system comes in to play here. When a painting matches more styles multiple parse trees describe this one image. Each alternative, rooted by a detector, also contains a confidence value. In this case this confidence value is based on the support of the decision rule.

Furthermore, this grammar is mainly constructed to recognize paintings in the *clair-obscur* style. More features may be needed to fine tune the decision rules for the other styles, *e. g. impressionism*. The use of a feature grammar is well suited for this evolutionary approach as it supports incremental maintenance of the annotations.

## 7.4.5 Generating the Presentation

Part of the prototype implementation for the Rijksmuseum case study is a generation engine that is able to transform a high-level description of a presentation [RBvO<sup>+</sup>00] in the concrete final-form encoding that is readily playable on the end-user's system. In this system the final encoding form is SMIL [W3C98].



Figure 7.11: Ordered retrieval result before optimization.

The presentation generation engine of the system is a constraint-based planning system. The constraint system is used for solving the design-based constraints, such as:

- the overall presentation dynamics (e.g. linear or interactive) and the resulting subdivision of information blocks;
- organizing material for each information block, e.g. number of elements on a page and their spatial outline based on the actual size of each information unit;
- optimization of ordered material based on additional style criteria, such as color or brightness distribution, in particular to emphasize a particular style.

The use of the annotations produced by the FDE are mainly of interest in the last bullet. The inner details of the other parts of the system itself, especially the transformation of the presentation structures generated by the constraint engine into a SMIL presentation, have been discussed in [vOGC<sup>+</sup>01].

It is assumed that the system constructs a linear presentation for educational reasons and creates topic blocks to present the material. Finally, based on spatial constraints, it calculates how many images for each topic block can be presented on a page.

At this stage the generator tries to arrange the images in each topic block in such a way that the style criteria for *clair-obscur* are fulfilled. A decision rule could look

as follows:

```
style_order(Image_Style, List_Of_Images, Images_Per_Page,
  Presentation_List):
  gradient-match(Image_Style, List_Of_Images, Result_List),
  border-match(Result_List, Images_Per_Page,
  Presentation_List).
```

With this rule the system analyzes not the image itself but rather its grid abstraction, as shown in Figure 7.9.d. The system tries, for a particular style (*Image\_Style*), to order the images of one topic (*List\_of\_Images*) based on the pattern provided by those cells of the grid that represent light values. The analysis of these patterns is based on graphical shapes, such as triangle or rectangles. The direction of the light is derived from a number of criteria, such as solidness of a pattern (main light center), position in the grid (at the border indicates that the light source is outside the image), and the direction of the dissolve of this shape (direction of light beam). For Figure 7.9.d the result is that the light source is outside the image, that light is coming from the left side and dissolves towards the right side in a rectangular way. The *Result\_List* groups images in lists, where light follows similar directions, such as left, up-left, up, up-right, right, down-right, down, down-left, circular.

Once that is done, the system tries to align the images based on similar border pattern. Take once again Figure 7.9.d as the example, the system would try to find an image which shares a similar distribution of light and dark cells (up or down by one grid cell) but only on its right side. The combination of images is performed on the previous calculated maximum size of images per page.

Figure 7.11 shows the random image sequence. While the final presentation in Figure 7.12 uses the optimized order. This presentation is based on the rhetorics of an educational-oriented presentation, which requires introductions of topics and subtopics.

The temporal duration for every single page is calculated by the number of presented objects and their graphical complexity, the number of words for text elements, or temporal presentation qualifiers such as fade-in or out times for media units. The last screen offers choices for the next step.

## 7.4.6 Lessons Learned

The Rijksmuseum feature grammar has a much finer detector granularity than the previous grammars. It also called for the reuse of various detectors within a different context. The same feature detectors are used to determine global and local, *i. e.* per grid cell, features.

The use of decision rules for the various art styles inspired the addition of classifiers as a special instance of detector plugins. Furthermore, ambiguity plays a major role in this grammar as various styles may match concurrently. This triggered the addition of support for detector confidences and parse forests.



Figure 7.12: The optimized presentation.

## 7.5 Discussion

The case studies in this chapter showed the viability of the Acoi system and its formal basis feature grammar systems. Due to its focus on limited context-sensitivity basic building blocks for an multimedia annotation system can be constructed. Actual applications can then be build by putting these blocks, *i. e.* grammar components, together using the dependencies.

Next to being a description of the actual annotations, feature grammar systems can also be used in a more traditional way: to describe a workflow and its associated meta-data. This was done in a recent *Waterland* related case study. The Acoi system offers in this case the advantage that new workflow actions can be easily plugged in. Furthermore, the whole infrastructure, *e. g.* plugins, can be used to easily automate these actions.

Creating a semi-automatic feature grammar is one of the future research topics. Manual detectors may provide new annotations or validate automatically extracted annotations. A case study may provide insights in how conveniently the current implementation supports this mixed type of annotation extraction.

Comparison of the Acoi system with peer systems, and thus evaluation of implementation issues, *e. g.* performance, remains future work. As discussed in Sections 2.3 and 4.4 the explicit description and usage of context knowledge is unique to feature grammar systems, but can also be used to generate specifications, although probably more verbose and fragmented, for these peer systems. Applying such a translation of one or several of the case studies may enable comparable runs of these systems, and thus provide (more) insight into their specific strengths and weaknesses.



# Chapter 8

## Conclusion and Future Work

*If SETI@home works, for example, we'll need libraries for communicating with aliens. Unless of course they are sufficiently advanced that they already communicate in XML.*

Paul Graham – *The Hundred-Year Language*

This thesis describes many formal, architectural and implementation aspects of the Acoi system. They have provided the author with a wide scale of research topics and challenges during the past years. This final chapter concludes the description of this system by providing a look backward, into the past, and a look forward, into the future.

### 8.1 Conclusion

The aim of the Acoi system was to implement support for the complete life cycle of a DMW (*Digital Media Warehouse*), *i. e.* creation, storage and maintenance, by various interpreters of one declarative description. The model underlying this description would have to support these key requirements: (1) providing context for (possible) bridging of the semantic gap, (2) allowing ambiguous interpretations, (3) describing both contextual and output/input dependencies, (4) give enough context for incremental maintenance, and (5) keep the input specification of algorithms generic enough to enable and promote reuse. As all these requirements involve some form of context grammars, as known from formal language theory, were considered a good starting point.

Chapter 2 provided the formal basis for the system. Mildly context-sensitive feature grammar systems allow the embedding of annotation extraction algorithms, *i. e.* feature detector functions, on a low level inside the grammar formalism, while underspecifying their context. The feature grammar language of Chapter 3 introduced a more natural notation of this formalism. The next three chapters described the various interpreters, the FDE (*Feature Detector Engine*), the database schema and the

FDS (*Feature Detector Scheduler*), needed for the various life cycle stages. These core chapters of the thesis address the key requirements using formal language theory. This is mainly visible in the design of the FDE, which is directly related to practical algorithms from the computer science research fields of natural language processing and compiler technology. Where needed proper extensions of this theory and related practice have been defined.

The three case studies from Chapter 7 gave an impression of the practical impact of the Acoi system. Formal language theory in the form of feature grammar systems, with its focus on extending and in the same time limiting context-sensitivity, appears to be indeed well suited to meet the identified requirements of a DMW annotation subsystem. This, as this thesis already shows, makes a plethora of formal techniques and practical experience available to this application domain. Future experiments with a mature FDS implementation and further evaluation of the complete system may add additional support for this conclusion.

## 8.2 Future Work

The various components of the Acoi system provide a basis for the annotation subsystem of a DMW. However, there is always room for improvement, as has been indicated in various places throughout the thesis. The next sections will revisit these areas of future work and also describe some additional ones.

### 8.2.1 Feature Grammar Systems

The description of feature grammar systems is directly based on the formal theories of cooperating distributed grammar systems and regulated rewriting. Contributions of this thesis to these theories are *lPC* (*left path-controlled*) grammars (see Section 2.2.3.1). Intuition tells that these grammars are as powerful as conditional grammars, however, this should be backed up by a formal proof.

Detector functions can produce CS (*Context-Sensitive*) sentences, however, the feature grammar components are CF (*Context-Free*). In a future version of Acoi the CF components may be replaced by CS or mildly CS components, *e. g.* *lPC* (*left Path-Controlled*) components. This will also have to be reflected in the feature grammar language. In the case of *lPC* components a regular path expression could be associated to arbitrary non-terminals instead of only to detector symbols.

### 8.2.2 Feature Grammar Language

The core feature grammar language directly relates to the underlying feature grammar system. The extensions to the language provide shortcuts to the developer. One of the tasks of these extensions is to keep the grammar semantically rich and to avoid clutter with additional symbols which are only there to steer the extraction process. Ideally

these additional symbols are all anonymous so they are transparent to both the developer and the user. Additional language constructs may prove to be convenient, *e. g.* directly embedding of anonymous whitebox detectors in the production rules. Also the use of symbol or symbol specific scripts, in the vein of attribute grammars, for the propagation of confidence values may prove an interesting addition.

### 8.2.3 Feature Detector Engine

The current FDE implementation is based on a depth-first top-down parsing algorithm: exhaustive backtracking. Experiments with other parsing algorithms, and thus with other moments of control transfer between components, can provide alternatives for this algorithm. Another interesting experiment will be the use of a breadth-first algorithm to add some form of parallelism to the FDE. These alternative algorithms can also be realized by translating a specific feature grammar system into a specification for a coordination system, *e. g.* a T script for ToolBus.

A detector function can depend on some external functionality, which is (temporarily) unavailable. Adding an *unknown* result state next to *success* and *failure* may enable the FDE to proceed validation partially and to return later to the detector to retry its execution.

### 8.2.4 Feature Database

As already stated in Chapter 5 the current storage scheme can be replaced by another, probably better suited, XML storage scheme. The query process of the feature database is also still in its infancy. For example further experiments with the usage of the confidence values of detectors as input to a probabilistic reasoning scheme or a ranking formula will be useful. This is directly related to a proper use of the, possible, various alternative interpretations of one multimedia object.

At the moment all annotations are extracted during the building of the index. However, some annotations or features may be more dynamic, *i. e.* they are computed on demand and are not persistent. These derived annotations and features may be used to capture properties of the whole index at a specific moment in time, *i. e.* the moment of query execution. The impact of these type of symbols on the analysis of the dependency by the FDS will have to be investigated.

### 8.2.5 Feature Detector Scheduler

The implementation of the FDS has been sketched and some core components have been prototyped. Future work certainly contains an actual implementation of the complete FDS, accompanied by further experiments in the domain of, for example, the WWW multimedia search engine.

### **8.2.6 Digital Media Warehouses**

The case studies of Chapter 7 gave insight or hints in how to embed the system in a complete DMW system. But in most cases the Acoi system functioned as a blackbox. Future experiments could involve a further integration of the various system components. The manual annotation part is one of the first components which comes to mind. How well will the Acoi system cope with manual detectors and semi-automatic annotation extraction? If this turns out favorable for the Acoi system it will implement a complete annotation subsystem.

Comparison with the execution characteristics of other systems may become possible by implementing a common task, and may give insight in both modeling power and performance of the competing systems.

## **8.3 Discussion**

This thesis touched upon one of the key research challenges of a DMW: multimedia annotation extraction and (incremental) maintenance. Although there are still many open issues to be resolved, the Acoi system, with feature grammar systems as basis, rallied formal language theory and practice to meet this challenge with success.

# Appendix A

## The Feature Grammar Language

This appendix contains the feature grammar language in an EBNF notation, *i. e.* the notation as used by the W3C for the XML specification [W3C00].

```
1 #character classes
2 digit      ::= [0-9]
3 exponent   ::= [Ee][-+]?{D}+
4 letter     ::= [_a-zA-Z]
5 any       ::= [#x0-#xFFFE]

6 #literals or constants
7 float      ::= '-'? digit+ '.' digit+ exponent?
8 integer    ::= '-'? digit+
9 unsigned-integer ::= digit+
10 string    ::= '"' any* '"'
11 constant  ::= float | integer | string

12 #a symbol
13 symbol     ::= letter ( letter | digit )*

14 #a scope
15 scope      ::= letter ( letter | digit )*

16 #the prefix puts an symbol in a scope, this scope
17 #may refer to a feature grammar, an ADT module or
18 #a detector plugin
19 prefix     ::= scope ":@"

20 #simplified XPath expression
```

```

21 | xpath           ::= absolute | relative
22 | absolute       ::= '/' relative? | "//" relative
23 | relative       ::= step ( '/' step | "//" step ) *
24 | step           ::= axis? ( ( symbol | '*' ) | dereference )
25 |               | abbreviation
26 | axis           ::= "self::" | "parent::" | "child::"
27 |               | "ancestor::" | "ancestor-or-self::"
28 |               | "preceding::" | "preceding-sibling::"
29 |               | "descendant::" | "descendant-or-self::"
30 |               | "following::" | "following-sibling::"
31 | abbreviation   ::= '.' | '..'
32 | #a feature grammar specific addition
33 | dereference    ::= '&' symbol

34 | #a list of detector parameters, if no axis is specified for
35 | #the first step it defaults to preceding::
36 | detector-params ::= '(' ( detector-param ( ','
37 |                       detector-param ) * ) ')'
38 | detector-param  ::= constant | xpath

39 | #even more simplified XPath expression
40 | s-path          ::= s-absolute | s-relative
41 | s-absolute      ::= '/' s-relative? | "//" s-relative
42 | s-relative      ::= s-step ( '/' s-step | "//" s-step ) *
43 | s-step          ::= s-axis? ( symbol | '*' )
44 | s-axis          ::= "self::" | "child::"
45 |               | "descendant::" | "descendant-or-self::"

46 | #a list of start symbol parameters, if no axis is specified
47 | #for the first step it defaults to the standard child::
48 | start-params    ::= '(' ( start-param ( ','
49 |                       start-param ) * ) ')'
50 | start-param     ::= s-path

51 | #collection type and bounds specification for symbols on the
52 | #right-handside of a rule
53 | bounds          ::= list | set | tuple
54 | list            ::= '[' range ']' | range
55 | set             ::= '{' range '}'
56 | tuple          ::= '<' int-range '>'
57 | range           ::= wild-range | int-range
58 | int-range       ::= unsigned-integer
59 |                 ( ':' unsigned-integer ) ?
60 | wild-range      ::= '*' | '+' | '?'

61 | #the feature grammar language, i.e. the start symbol of this

```

```

62 #EBNF
63 feature-grammar ::= module-decl decl*
64 decl           ::= use-decl | start-decl
65                | poll-decl | atom-decl
66                | detector-decl | classifier-decl
67                | version-decl | rule-decl

68 #module declarations
69 module-decl    ::= "%module" scope ';'
70 use-decl       ::= "%use" scope ( ',' scope )* ';'

71 #start declaration
72 start-decl     ::= "%start" symbol start-params ';'

73 #poll declaration
74 poll-decl     ::= "%detector" symbol ".poll" start-params ';'

75 #atom and atom rule declarations
76 atom-decl      ::= "%atom" prefix? symbol
77                ( ( ( symbol ( ',' symbol )* )?
78                  | '{' any+ '}' ) )? ';'

79 #detector declarations
80 detector-decl  ::= "%detector" prefix? symbol detector-params ';'
81 detector-decl  ::= "%detector" prefix? symbol
82                '[' any+ ']' ';'

83 #classifier declaration
84 classifier-decl ::= "%classifier" prefix? symbol detector-params ';'

85 #version declaration
86 version-decl   ::= "%version" symbol
87                unsigned-integer '.'
88                unsigned-integer '.'
89                unsigned-integer ';'

90 #rule declaration
91 rule-decl      ::= rhs ':' lhs ';'
92 rhs            ::= prefix? symbol
93 lhs            ::= ( '&'? prefix? symbol bounds? | constant )+
94 lhs            ::= '(' lhs ')'
95 lhs            ::= lhs '|' lhs

```





# Appendix B

## Feature Grammars

### B.1 The WWW Feature Grammar

```
1 %module      WWW;
2 %start      WebObject(Location);
3 %detector   WebHeader(Location);
4 %detector   Robot(Location, "AcoiRobot");
5 %detector   Explored(MIME);
6 %detector   Tried(parent::Status);
7 %atom       www::url {(^http://([^ :/]*)(:[0-9]*)?/?(.*$)
8                |(^file://(.*)$)};
9 %atom       temporal::date;
10 %atom       url Location;
11 %atom       date Modification;
12 %atom       lng Length;
13 WebObject  : Location (Robot WebHeader WebBody)? Status;
14 WebHeader  : Redirect? MIME Modification? Length?;
15 MIME       : Primary Secondary;
16 Status     : Explored | Tried;
```

### B.2 The Text Feature Grammar

```
1 %module      Text;
2 %use         WWW;
```

```

3 %detector TextType [ MIME/Primary = "text" ];
4 %detector DRUID(Location);
5 %atom str Language;
6 WebBody      : TextType Text;
7 Text         : DRUID;
8 DRUID        : Language;

```

### B.3 The HTML Feature Grammar

```

1 %module      HTML;
2 %use        Text;
3 %detector   HTMLType [ MIME/Secondary = "html" ];
4 %detector   HTML(Location);
5 %atom str   Title, Word, Link, Alt;
6 %atom bit   Embedded;
7 WebBody     : HTMLType HTML;
8 HTML        : Title Body? Anchor*;
9 Body        : &Keyword+;
10 Anchor     : &WebObject Embedded Link? Alt?;
11 %start      Keyword(Word);
12 %detector   Synset(Word);
13 %start      Synonyms(Word);
14 %start      Hypernyms(Word);
15 %start      Hyponyms(Word);
16 Keyword     : Word Synset;
17 Synset      : &Synonyms &Hypernyms &Hyponyms;
18 Synonyms    : id str+;
19 Hypernyms  : id str+;
20 Hyponyms    : id str+;

```

## B.4 The Image Feature Grammar

```

1 %module      Image;
2 %use         WWW;
3 %detector    ImageType [ MIME/Primary = "image" ];
4 %detector    Portrait [ Faces/Number = 1 ];
5 %detector    Global(Location);
6 %detector    Icon(Location);
7 %classifier  decrules::Photo(Global);
8 %classifier  decrules::Graphic(Global);
9 %detector    Skin(Location);
10 %detector   Faces(Location);
11 %detector    exec::Histogram(Location);
12 %atom vector::flts;
13 %atom int    Number;
14 %atom flt    Prevalent, Far, NormalizedFar, Saturation, Percentage;
15 %atom flts   HSB, RGB;
16 %atom bit    Animated;
17 WebBody     : ImageType Image;
18 Image       : Global Icon Class;
19 Global      : Size Color Animated;
20 Size        : Width Height Ratio;
21 Color       : Number Prevalent Neighbor Saturation Histogram;
22 Neighbor    : Far NormalizedFar;
23 Histogram   : RGB HSB;
24 Icon        : Location;
25 Class       : Graphic | Photo (Skin Faces Portrait)?;
26 Skin        : Percentage;
27 Faces       : Number;

```

## B.5 The Audio Feature Grammar

```

1 %module      Audio;
2 %use         WWW
3 %detector    AudioType [ MIME/Primary = "audio" ];
4 WebBody     : AudioType Audio;

```

## B.6 The MIDI Feature Grammar

```

1 | %module      MIDI;
2 | %use        Audio;
3 | %detector   MIDIType [ MIME/Secondary = "midi" ];
4 | %detector   exec::MIDI(Location);
5 | %atom int   QuarterNode, Id, Channel, Track;
6 | %atom str   Lyrics, Name, Contour;
7 | Audio       : MIDIType MIDI;
8 | MIDI        : QuarterNode Musician* Lyrics* Profile*;
9 | Musician    : Instrument Channel;
10 | Instrument  : Id Name?;
11 | Profile     : Track Contour;

```

## B.7 The MP3 Feature Grammar

```

1 | %module      MP3;
2 | %use        Audio;
3 | %detector   MP3Type [ MIME/Secondary = "mpeg" ];
4 | %detector   ID3(Location);
5 | %atom str   Title, Performer, Album, Genre;
6 | %atom int   Year;
7 | Audio       : MP3Type MP3;
8 | MP3         : ID3?;
9 | ID3        : Title Performer Album Year Genre;

```

## B.8 The Video Feature Grammar

```

1 | %module      Video;
2 | %use        WWW
3 | %detector   VideoType [ MIME/Primary = "video" ];
4 | WebBody     : VideoType Video;

```

## B.9 The MPEG Feature Grammar

```

1 | %module      MPEG;
2 | %use        Video;
3 | %detector   MPEGType [ MIME/Secondary = "mpeg" ];
4 | %detector   Icon(Location);
5 | Video       : MPEGType MPEG;
6 | MPEG        : Icon;

```

## B.10 The Acoi Feature Grammar

```

1 | %module      Acoi;
2 | %use        WWW;
3 | %use        Text, HTML;
4 | %use        Image;
5 | %use        Audio, MIDI, MP3;
6 | %use        Video, MPEG;

```

## B.11 The Tennis Feature Grammar

```

1 | %module      Tennis;
2 | %use        Video;
3 | %atom flt   xPos, yPos, Ecc, Orient;
4 | %atom int   FrameNo, Area;
5 | %detector   xml-rpc::Segment(WebObject/Location);
6 | %detector   xml-rpc::Tennis(WebObject/Location,
7 |             ancestor::Scene/Begin/FrameNo,
8 |             ancestor::Scene/End/FrameNo);
9 | %detector   Netplay [ some $Player in Player satisfies
10 |                $Player.yPos <= 170
11 |                ];
12 | %detector   Rally(parent::Tennis);
13 | Video       : Segment;
14 | Segment     : Scene*;
15 | Scene       : Begin End Type;

```

```

16 | Type      : "tennis" Tennis;
17 | Type      : "closeup";
18 | Type      : "audience";
19 | Type      : "other";
20 | Begin     : FrameNo;
21 | End       : FrameNo;
22 | Tennis    : Frame+ Event;
23 | Frame     : FrameNo Player;
24 | Player    : xPos yPos Area Ecc Orient;
25 | Event     : Netplay? Rally?;

```

## B.12 The Australian Open Feature Grammar

```

1 | %module   AO;
2 | %use      Acoi;
3 | %use      Tennis;

```

## B.13 The Rijksmuseum Feature Grammar

```

1 | %module   Rijksmuseum;
2 | %use      Image;
3 | %atom flt threshold;
4 | %atom int  columns, rows;
5 | %atom int  column, row;
6 | %atom int  x, y, width, height;
7 | %atom flt  dark_coverage, light_coverage;
8 | %atom int  number;
9 | %atom flt  corr, non_corr, norm_corr;
10 | %atom flt  scalar;
11 | %atom bit  onoff;
12 | %detector  light(Location);
13 | %detector  global(WebObject/Location);
14 | %detector  contrast(WebObject/Location);
15 | %detector  grid(WebObject/Location);
16 | %detector  histo_segment(Location);
17 | %detector  segment(Location,
18 |           ancestor::light/histo_segment/threshold);
19 | %detector  light_dist(ancestor::image/general/light/segment/name,
20 |           shape);
21 | %detector  region(grid);
22 | %detector  co_histo(WebObject/Location);

```

```
23 | %detector   cu_histo(WebObject/Location);
24 | %detector   im_histo(WebObject/Location);

25 | %classifier decrules::clair_obscur(corr,non_corr,contrast/scalar);
26 | %classifier decrules::cubism(corr,non_corr,contrast/scalar);
27 | %classifier decrules::impressionism(corr,non_corr,contrast/scalar);

28 | Image      : general global local style;

29 | general    : light;
30 | light      : Location histo_segment segment;

31 | histo_segment: threshold+;
32 | segment    : Location;

33 | shape      : bbox;
34 | bbox       : x y width height;

35 | features   : light_dist;
36 | light_dist : light_coverage dark_coverage;

37 | global     : shape features contrast;
38 | local      : grid region;

39 | contrast   : scalar;

40 | grid       : columns rows cell*;
41 | cell       : column row shape features;

42 | region     : number;

43 | style      : co_histo clair_obscur;
44 | style      : cu_histo cubism;
45 | style      : im_histo impressionism;
46 | style      : ;

47 | co_histo   : corr non_corr norm_corr;
48 | cu_histo   : corr non_corr norm_corr;
49 | im_histo   : corr non_corr norm_corr;
```





# Appendix C

## XML documents

### C.1 A schema document

```
1 <?xml version="1.0"?>
2 <fg:grammar
3   xmlns:fg="http://www.cwi.nl/~acoi/fg/schema"
4   xmlns:WWW="http://www.cwi.nl/~acoi/WWW"
5   xmlns:Image="http://www.cwi.nl/~acoi/WWW"
6 >
7   <WWW:WebObject type=".non-terminal.start.">
8     <WWW:Location type=".non-terminal."/>
9     <WWW:WebHeader type=".non-terminal.detector.blackbox."/>
10    <WWW:WebBody type=".non-terminal."/>
11  </WWW:WebObject>
12  <WWW:WebHeader type=".non-terminal.detector.blackbox.">
13    <WWW:Modification type=".non-terminal."/>
14    <WWW:Length type=".non-terminal."/>
15  </WWW:WebHeader>
16  <WWW:Location type=".non-terminal.">
17    <WWW:url type=".terminal.atom." module="www"/>
18  </WWW:Location>
19  <WWW:Modification type=".non-terminal.">
20    <WWW:date type=".terminal.atom." module="temporal"/>
21  </WWW:Modification>
22  <WWW:Length type=".non-terminal.">
23    <Fg:lng type=".terminal.atom."/>
24  </WWW:Length>
25  <WWW:WebBody type=".non-terminal.">
26    <Image:Image type=".non-terminal."/>
27  </WWW:WebBody>
28  <Image:Skin type=".non-terminal.detector.blackbox.">
```

```
29     <Image:bitmap type=".terminal.atom." module="image"/>
30 </Image:Skin>
31 <Image:Color type=".non-terminal.detector.blackbox.">
32     <Image:RGB type=".non-terminal."
33         coll="list" lbnd="0" hbnd="infinite"/>
34     <Image:Number type=".non-terminal."/>
35     <Image:Prevalent type=".non-terminal."/>
36     <Image:Saturation type=".non-terminal."/>
37 </Image:Color>
38 <Image:Faces type=".non-terminal.detector.blackbox.">
39     <fg:int type=".terminal.atom."/>
40 </Image:Faces>
41 <Image:Red type=".non-terminal.">
42     <fg:int type=".terminal.atom."/>
43 </Image:Red>
44 <Image:Green type=".non-terminal.">
45     <fg:int type=".terminal.atom."/>
46 </Image:Green>
47 <Image:Blue type=".non-terminal.">
48     <fg:int type=".terminal.atom."/>
49 </Image:Blue>
50 <Image:Number type=".non-terminal.">
51     <fg:int type=".terminal.atom."/>
52 </Image:Number>
53 <Image:Prevalent type=".non-terminal.">
54     <fg:flt type=".terminal.atom."/>
55 </Image:Prevalent>
56 <Image:Saturation type=".non-terminal.">
57     <fg:flt type=".terminal.atom."/>
58 </Image:Saturation>
59 <Image:Image type=".non-terminal.">
60     <Image:Color type=".non-terminal.detector.blackbox."/>
61     <Image:Class type=".non-terminal."/>
62 </Image:Image>
63 <Image:Class type=".non-terminal.">
64     <Image:Graphic type=".non-terminal.detector.blackbox."/>
65     <Image:Photo type=".non-terminal.detector.whitebox."/>
66     <Image:Skin type=".non-terminal.detector.blackbox."/>
67     <Image:Faces type=".non-terminal.detector.blackbox."/>
68 </Image:Class>
69 <Image:RGB type=".non-terminal.">
70     <Image:Red type=".non-terminal."/>
71     <Image:Green type=".non-terminal."/>
72     <Image:Blue type=".non-terminal."/>
73 </Image:RGB>
74 </fg:grammar>
```

## C.2 A parse forest document

```

1 <?xml version="1.0"?>
2 <fg:forest
3   xmlns:fg="http://www.cwi.nl/~acoi/fg/forest"
4   xmlns:WWW="http://www.cwi.nl/~acoi/WWW"
5   xmlns:Image="http://www.cwi.nl/~acoi/Image"
6 >
7   <fg:elementary context="1:1" confidence="1.00" idrefs="2"
8     start="WWW:WebObject" date="20030625"
9   >
10    <WWW:WebObject id="5478@0" context="1:1">
11      <WWW:Location id="1" context="1:1">
12        <WWW:url id="2" context="1:1">
13          <![CDATA[http://...]]>
14        </WWW:url>
15      </WWW:Location>
16      <WWW:WebHeader idrefs="5479@0" context="1:1"/>
17      <WWW:WebBody id="7" context="1:1">
18        <Image:Image id="8" context="1:1">
19          <Image:Color idrefs="5480@0" context="1:1"/>
20          <Image:Class id="15" context="1:1">
21            <Image:Photo idrefs="5486@0" context="1:1"/>
22            <Image:Skin idrefs="5487@0" context="1:1"/>
23            <Image:Faces idrefs="5488@0" context="1:1"/>
24          </Image:Class>
25        </Image:Image>
26      </WWW:WebBody>
27    </WWW:WebObject>
28  </fg:elementary>
29  <fg:auxiliary date="20030625">
30    <WWW:WebHeader id="5479@0" idrefs="2" context="1:1"
31      confidence="1.00" version="1.0.0"
32    >
33      <WWW:Modification id="3" context="1:1">
34        <WWW:date id="4" context="1:1">
35          <![CDATA[Jul 23 2001]]>
36        </WWW:date>
37      </WWW:Modification>
38      <WWW:Length id="5" context="1:1">
39        <fg:lmg id="6" context="1:1">
40          <![CDATA[14197]]>
41        </fg:lmg>
42      </WWW:Length>
43    </WWW:WebHeader>
44  </fg:auxiliary>

```

```

45 <fg:auxiliary date="20030625">
46   <Image:Color id="5480@0" idrefs="2" context="1:1"
47     confidence="1.00" version="1.0.0"
48   >
49     <Image:Number id="9" context="1:1">
50       <fg:int id="10" context="1:1">
51         <![CDATA[29053]]>
52       </fg:int>
53     </Image:Number>
54     <Image:Prevalent id="11" context="1:1">
55       <fg:flt id="12" context="1:1">
56         <![CDATA[0.03]]>
57       </fg:flt>
58     </Image:Prevalent>
59     <Image:Saturation id="13" context="1:1">
60       <fg:flt id="14" context="1:1">
61         <![CDATA[0.19]]>
62       </fg:flt>
63     </Image:Saturation>
64   </Image:Color>
65 </fg:auxiliary>
66 <fg:auxiliary date="20030625">
67   <Image:Photo id="5486@0" idrefs="10 12 14"
68     context="1:1" confidence="0.85" version="1.0.0"
69   />
70 </fg:auxiliary>
71 <fg:auxiliary date="20030625">
72   <Image:Skin id="5487@0" idrefs="2" context="1:1"
73     confidence="0.95" version="1.0.0"
74   >
75     <Image:bitmap id="16" context="1:1">
76       <![CDATA[00...]]>
77     </Image:bitmap>
78   </Image:Skin>
79 </fg:auxiliary>
80 <fg:auxiliary date="20030625">
81   <Image:Faces id="5488@0" idrefs="16" context="1:1"
82     confidence="0.77" version="1.0.0"
83   >
84     <fg:int id="17" context="1:1">
85       <![CDATA[1]]>
86     </fg:int>
87   </Image:Faces>
88 </fg:auxiliary>
89 </fg:forest>

```

# Abbreviations

AACR	Anglo-American Cataloguing Rules
Acoi	Amsterdam catalog of images
ADMIRE	ADvanced MultiMedia Retrieval Model
AMIS	Advanced Multimedia Indexing and Searching
API	Application Programming Interface
BAT	Binary Association Table
BNF	Backus-Naur Form
C	Conditional (grammar)
CD	Cooperating Distributed (grammar system)
CF	Context-Free
CNF	Chomsky Normal Form
COBRA	COntent-Based Retrieval
CS	Context-Sensitive
DBMS	DataBase Management System
DMW	Digital Media Warehouse
DOM	Document Object Model
DPDA	Deterministic Push-Down Automata
DPDT	Deterministic Push-Down Transducer
DSL	Domain-Specific Language
EBNF	Extended Backus-Naur Form
FA	Finite Automata
FDE	Feature Detector Engine
FDS	Feature Detector Scheduler
GNF	Greibach Normal Form
GPL	General Purpose Language
GUI	Graphical User Interface
HPSG	Head-Driven Phrase Structure Grammar
HTML	HyperText Markup Language
IR	Information Retrieval
LBA	Linear Bounded Automata

---

LFG	Lexical-Functional Grammar
LHS	Left-Hand Side
LIFO	Last-In First-Out
IPC	left Path-Controlled (grammar)
MIL	Monet Interpreter Language
MIME	Multipurpose Internet Mail Extensions
MOA	Magnum Object Algebra
MPEG	Moving Picture Experts Group
ND	Named Disjunction
NLP	Natural Language Processing
NPDA	Non-deterministic Push-Down Automata
NPDT	Non-deterministic Push-Down Transducer
ODMG	Object Data Management Group
PC	Path-Controlled (grammar)
PC	Parallel Communicating (grammar system)
PDA	Push-Down Automata
PDT	Push-Down Transducer
QbS	Query by Sketch
QbT	Query by Text
RDT	Reduced Derivation Tree
RE	Recursively Enumerable
REG	REGular
RHS	Right-Hand Side
RPC	Remote Procedure Call
RRPG	Regular Right Part Grammar
SGML	Standard Generalized Markup Language
SMIL	Synchronized Multimedia Integration Language
SQL	Structured Query Language
TC	Tree-Controlled (grammar)
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	eXtensible Markup Language
XPath	XML Path language
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformations

# Bibliography

- [ACC<sup>+</sup>99] G. Auffret, J. Carrive, O. Chevet, T. Dechilly, R. Ronfard, and B. Bachimont. Audiovisual-based hypermedia authoring: using structured representations for efficient access to av documents. In Klaus Tochtermann, Jorg Westbomke, Uffe K. Will, and John J. Leggett, editors, *Proceedings of the 10th ACM conference on Hypertext and Hypermedia*, pages 169 – 178, Darmstadt, Germany, February 1999. ACM. 135
- [Alt01] AltaVista. *AltaVista - Image Search*. [www.altavista.com/sites/search/simage](http://www.altavista.com/sites/search/simage), 2001. 10
- [App87] D. E. Appelt. Bidirectional grammars and the design of natural language generation systems. In *Proceedings of Third Conference on Theoretical Issues in Natural Language Processing (TINLAP-3)*, pages 185 – 191, New Mexico State University, Las Cruces, New Mexico, January 1987. 19
- [AR01] Anne Abeillé and Owen Rambow. *Tree Adjoining Grammars: Mathematical, Computational and Linguistic Properties*. University of Chicago Press, first edition, January 2001. 82
- [Arn74] R. Arnheim. *Art and Visual Perception: A Psychology of the Creative Eye*. Faber and Faber, London, 1974. 135
- [AS88] G. T. M. Altmann and M. J. Steedman. Interaction with context during human sentence processing. *Cognition*, 30(3):191 – 238, 1988. 74
- [ASF97] Vassilis Athitsos, Michael J. Swain, and Charles Frankel. Distinguishing photographs and graphics on the world wide web. In *Workshop on Content-Based Access of Image and Video Libraries*, Puerto Rico, June 1997. 6, 124, 139
- [ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers – Principles, Techniques, and Tools*. Addison-Wesley, 1986. 32, 79
- [Bak79] J. Baker. Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547 – 550, Cambridge, Massachusetts, 1979. MIT Press. 21
- [BB75] John Seely Brown and Richard R. Burton. Multiple representations of knowledge for tutorial reasoning. In Daniel G. Bobrow and Allan Collins, editors, *Representation and Understanding, Language, Thought, and Culture*, pages 311 – 349. Academic Press, 1975. 33

- [BK86] J.A. Bergstra and J. W. Klop. Process algebra: specification and verification in bisimulation semantics. In M. Hazewinkel, J.K. Lenstra, and G.T.L. Meertens, editors, *Mathematics & Computer Science II*, volume 4 of *CWI Monograph*. North-Holland, 1986. 43
- [BK94] J.A. Bergstra and P. Klint. The toolbus - a component interconnection architecture. Technical Report P9408, Programming Research Group, University of Amsterdam, 1994. 43
- [BK95] P. A. Boncz and M. L. Kersten. Monet: An Impressionist Sketch of an Advanced Database System. In *Proceedings Basque International Workshop on Information Technology*, San Sebastian, Spain, July 1995. 99
- [BK96] J.A. Bergstra and P. Klint. The discrete time toolbus. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology (AMAST'96)*, volume 1101 of *Lecture Notes in Computer Science*, pages 286 – 305. Springer-Verlag, 1996. 43
- [BK99] P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. *The VLDB Journal*, 8(2):101–119, October 1999. The original publication is available in LINK, © Springer-Verlag. 99
- [Bla97] Philippe Blanche. Disambiguating with controlled disjunctions. In *Proceedings of the International Workshop on Parsing Technologies*, 1997. 21, 40
- [Bla98] Philippe Blanche. Parsing ambiguous structures using controlled disjunctions and unary quasi-trees. In *Proceedings of ACL-COLING'98*, 1998. 21
- [BLFIM98] T. Berners-Lee, R. Fielding, U.C. Irvine, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. The Internet Engineering Task Force, [www.ietf.org/rfc/rfc2396.txt](http://www.ietf.org/rfc/rfc2396.txt), August 1998. 3
- [Blo02] Henk Ernst Blok. *Database Optimization Aspects for Information Retrieval*. PhD thesis, Centre for Telematics and Information Technology, Enschede, The Netherlands, April 2002. 125
- [Bon02] P. A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002. 99
- [Boo69] T. L. Booth. Probabilistic representation of formal languages. In *IEEE Conference Recors of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, pages 74 – 81, Waterloo, Ontario, 1969. IEEE. 21
- [Bos99] Peter Bosch. *Mixed-Media File Systems*. PhD thesis, Centre for Telematics and Information Technology, Enschede, The Netherlands, Juni 1999. 1
- [Bou03] Ronald Bourret. *XML Database Products*. [www.rpbouret.com/xml/XMLDatabaseProds.htm](http://www.rpbouret.com/xml/XMLDatabaseProds.htm), 2003. 110
- [Bre82] J. Bresnan, editor. *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA, 1982. 27
- [BWK98] P. A. Boncz, A. N. Wilschut, and M. L. Kersten. Flattening an Object Algebra to Provide Performance. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 568–577, Orlando, FL, USA, February 1998. 100



- 
- [BWvZ<sup>+</sup>01] H. E. Blok, M. A. Windhouwer, R. van Zwol, M. Petkovic, P. M. G. Apers, M. L. Kersten, and W. Jonker. Flexible and scalable digital library search. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Rome, Italy, September 2001. 123, 125, 128
- [BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999. 125
- [CAM01] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In *Proceedings of ICDE*, 2001. 116
- [CDP99] Carlo Colombo, Alberto Del Bimbo, and Pietro Pala. Semantics in visual information retrieval. *IEEE MultiMedia*, 6(3):38 – 53, July 1999. 6
- [CGM97] S. S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of ACM SIGMOD*, 1997. 116
- [Cho59] Noam Chomsky. On certain formal properties of grammars. *Information Control*, 2:137–167, 1959. 16
- [CK85] A. Copeland and S. Khoshafian. A decomposition storage model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 268 – 279, Austin, Texas, USA, May 1985. 99
- [CK01] James Clark and Kohsuke Kawaguchi. *Guidelines for using W3C XML Schema Datatypes with RELAX NG*. Oasis, [www.oasis-open.org/committees/relax-ng/xsd.html](http://www.oasis-open.org/committees/relax-ng/xsd.html), 2001. 104
- [CL96] J. Cha and S. Lee. Comib: Composite icon browser for multimedia databases. *Multimedia Tools and Applications*, 3(3):203 – 223, 1996. 3
- [Cla99] James Clark. XT. [www.blnz.com/xt/index.html](http://www.blnz.com/xt/index.html), 1999. 124
- [CM77] K. Culik II and H.A. Maurer. Tree controlled grammars. *Computing*, 19:129 – 139, 1977. 22, 23
- [CM01] James Clark and Makoto Murata. *RELAX NG Specification*. Oasis, [www.oasis-open.org/committees/relax-ng/spec-20011203.html](http://www.oasis-open.org/committees/relax-ng/spec-20011203.html), 2001. 104
- [CMOY96] I.J. Cox, M.L. Miller, S.M. Omohundro, and P.N. Yianilos. Pichunter: Bayesian relevance feedback for image retrieval. In *Proceedings of ICPR 1996*, pages 361 – 369, Vienna, Austria, 1996. 3
- [CSL01] Princeton University Cognitive Science Laboratory. *WordNet - a Lexical Database for English*. [www.cogsci.princeton.edu/~wn/](http://www.cogsci.princeton.edu/~wn/), 2001. 60, 124
- [CVDKP94] Erzsébet Csuhanj-Varhú, Jürgen Dassow, Jozef Kelemen, and Gheorghe Păun. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*, volume 5 of *Topics in computer mathematics*. Gordon and Breach Science Publishers, 1994. 24, 27
- [DAR03] DARPA. *The DAML+OIL ontology markup language*. [www.daml.org](http://www.daml.org), 2003. 135
- [Dat01] Datapower. *XSLTMark*. [www.datapower.com/xml\\_community/xsltmark.html](http://www.datapower.com/xml_community/xsltmark.html), 2001. 124

- [DCM01] DCMI. *Dublin Core Metadata Initiative*. [dublincore.org](http://dublincore.org), 2001. 3
- [DE90] Jochen Dörre and Andreas Eisele. Feature logic with disjunctive unification. In *Proceedings of COLING'90*, pages 100 – 105, 1990. 21
- [Del99] Alberto Del Bimbo. *Visual Information Retrieval*. Morgan Kaufmann Publishers, Inc., 1999. 2, 6
- [dJGHN00] F. de Jong, J-L. Gauvain, D. Hiemstra, and K. Netter. Language-based multimedia information retrieval. In *Proceedings of RIAO*, pages 713–725, Paris, France, 2000. 2
- [dJK] H.A. de Jong and P. Klint. Toolbus: the next generation. In *Proceedings of Formal Methods Components and Objects 2002 (FMCO 2002)*. to appear. 43
- [DP89] J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1989. 18, 21, 22, 23
- [DP97] A. Del Bimbo and P. Pala. Retrieval by elastic matching of user sketches. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2):121 – 132, 1997. 2
- [dV99] Arjen P. de Vries. *Content and Multimedia Database Management Systems*. PhD thesis, Centre for Telematics and Information Technology, Enschede, The Netherlands, December 1999. 42
- [dVEK98] Arjen P. de Vries, Brian Eberman, and David E. Kovalcin. The design and implementation of an infrastructure for multimedia digital libraries. In *Proceedings of the 1998 International Database Engineering & Applications Symposium*, pages 103–110, Cardiff, UK, July 1998. 42
- [dVWAK00] A. P. de Vries, M. A. Windhouwer, P. M. G. Apers, and M. L. Kersten. Information Access in Multimedia Databases based on Feature Models. *New Generation Computing*, 18(4):323–339, October 2000. 121
- [Eak96] John P. Eakins. Automatic image content retrieval - are we getting anywhere? In *Proceedings of Third International Conference on Electronic Library and Visual Information Research*, pages 123 – 135, De Montfort University, Milton Keynes, May 1996. 5
- [Eng02] Joost Engelfriet. The delta operation: From strings to trees to strings. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal and Natural Computing*, volume 2300 of *Lecture Notes in Computer Science (LNCS)*, pages 39 – 56. Springer-Verlag, 2002. 20
- [Fal96] Christos Faloutsos. *Searching multimedia databases by content*. Kluwer Academic Publishers, Dordrecht, NL, 1996. 3
- [Fal01] David C. Fallside. *XML Schema Part 0: Primer*. W3C, [www.w3.org/TR/xmlschema-0/](http://www.w3.org/TR/xmlschema-0/), 2001. 103
- [Fau94] Laurene V. Fausett. *Fundamentals of Neural Networks*. Prentice Hall, 1994. 6
- [FCP00] G. Frederix, G. Caenen, and E. J. Pauwels. Pariss: Panoramic, adaptive and reconfigurable interface for similarity search. In *Proceedings of the ICIP 2000 International Conference on Image Processing*, pages 222 – 225, September 2000. 135

- [FH99] H. Fernau and M. Holzer. Bidirectional cooperating distributed grammar systems. *Publicationes Mathematicae Debrecen*, (54):787 – 806, 1999. 27
- [FK99] Daniela Florescu and Donald Kossman. A performance evaluation of alternative mapping schemes for storing xml data in a relational database. Technical Report 3680, INRIA, Rocquencourt, France, May 1999. 101
- [Flo62] R. W. Floyd. On the non-existence of a phrase-structure for algol 60. *Communications of the ACM*, (5):483 – 484, 1962. 18
- [Fri02] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. O’Reilly & Associates, Inc., second edition, 2002. 17, 48
- [FSA96] Charles Frankel, Michael J. Swain, and Vassilis Athitsos. WebSeer: An Image Search Engine for the World Wide Web. Technical Report 96-14, The University of Chicago, August 1996. 11
- [GAS00] Th. Gevers, F. Aldershoff, and A. W. M. Smeulders. Classification of images on internet by visual and textual information. In *Internet Imaging, SPIE*, San Jose, January 2000. 6, 124
- [GEF<sup>+</sup>99] W. Grosso, H. Eriksson, R. Fergerson, J. Gennari, S. Tu, and M. Musen. Knowledge modeling at the millennium (the design and evolution of protege-2000). Technical Report SMI-1999-0801, Stanford Medical Informatics (SMI), 1999. 134
- [Gel95] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80 – 112, January 1995. 43
- [GJ98] Dick Grune and Criel Jacobs. *Parsing Techniques - A Practical Guide*. Vrije Universiteit, Amsterdam, 1998. 15, 16, 17, 47, 70, 75, 80, 97
- [Goo01] Google. *Google Image Search*. [www.google.com/imghp](http://www.google.com/imghp), 2001. 10
- [Gro94] William I. Grosky. Multimedia information systems. *IEEE MultiMedia*, 1(1):12–24, 1994. 3
- [Gru02] Torsten Grust. Accelerating xpath location steps. In *Proceedings of the 21st ACM SIGMOD International Conference on Management of Data (SIGMOD 2002)*, pages 109–120, Madison, Wisconsin, June 2002. 102
- [GvK03] Torsten Grust and Maurice van Keulen. Tree Awareness for Relational DBMS Kernels: Staircase Join. In *Intelligent Search on XML*, Lecture Notes in Computer Science (LNCS), © Springer-Verlag, 2003. To appear. 102
- [GvKT03] Torsten Grust, Maurice van Keulen, and Jens Teubner. Accelerating xpath location steps in any rdbms (even if all you got is a binary table). *ACM Transactions on Database Systems (TODS)*, 2003. under revision. 102
- [GW98] Michael Gorman and Paul W. Winkler, editors. *Anglo-American Cataloguing Rules*. Amer Library Assn Editions, 2nd revision edition, 1998. 3
- [GYA97] James Griffioen, Raj Yavatkar, and Robert Adams. A framework for developing content-based retrieval systems. In Mark T. Maybury, editor, *Intelligent Multimedia Information Retrieval*. The AAAI Press and The MIT Press, 1997. 42
- [Hal73] Patrick A. V. Hall. Equivalence between and/or graphs and context-free grammars. *Communications of the ACM (CACM)*, 16(7):444 – 445, July 1973. 20

- [HKR90] J. Heering, P. Klint, and J. Rekers. Incremental generation of parsers. *IEEE Transactions on Software Engineering*, 16(12):1344 – 1350, 1990. 82
- [HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, second edition, 2001. 15, 19
- [IBM01] IBM. *XML Diff and Merge Tool*. [www.alphaworks.ibm.com/tech/xmldiffmerge](http://www.alphaworks.ibm.com/tech/xmldiffmerge), 2001. 116
- [Inc02] Dommitt Inc. *XML Diff and Merge Tool*. [www.dommitt.com](http://www.dommitt.com), 2002. 116
- [ISO01] ISO. *Overview of the MPEG-7 Standard*. [mpeg.telecomitalia.com/standards/mpeg-7/mpeg-7.htm](http://mpeg.telecomitalia.com/standards/mpeg-7/mpeg-7.htm), 2001. 3
- [ISO02] ISO. *Overview of the MPEG-4 Standard*. [mpeg.telecomitalia.com/standards/mpeg-4/mpeg-4.htm](http://mpeg.telecomitalia.com/standards/mpeg-4/mpeg-4.htm), 2002. 136
- [Jel69] F. Jelinek. Fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 64:532 – 556, 1969. 21
- [Jel02] Rick Jelliffe. *The Schematron Assertion Language 1.5*. Academia Sinica Computing Centre, [www.ascc.net/xml/resource/schematron/Schematron2000.html](http://www.ascc.net/xml/resource/schematron/Schematron2000.html), 2002. 104
- [JFS95] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. In *Proceedings of SIGGRAPH 95*, pages 277–286, August 1995. 2
- [JM00] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000. 21
- [JS98] Adrian Johnstone and Elizabeth Scott. Construction reduced derivation trees. Technical Report CSD-TR-98-09, Department of Computer Science, Royal Holloway, University of London, October 12 1998. 93
- [KD96] A. Knott and R. Dale. *Choosing a Set of Rhetorical Relations for Text Generation: A Data-Drive Approach*. Trends in Natural Language Generation: an Artificial Intelligence Perspective. 1996. 135
- [Ker89] M. L. Kersten. A grammatical database model: An informal introduction. Unpublished manuscript, May 1989. 120
- [KKK<sup>+</sup>91] P. Kophakis, A. Karmirantzou, Y. Kavaklis, E. Petrakis, and S. Orphanoudakis. Image archiving by content: An object oriented approach. In *SPIE Int. Conf. on Image Processing, Medical Imaging V*, pages 227 – 233, San Jose, CA, February 1991. 2
- [KNW98a] M. L. Kersten, N. Nes, and M. Windhouwer. A feature database for multimedia objects. In *ERCIM Database Research Group Workshop on Metadata for Web Databases*, pages 49–57, Bonn, Germany, 1998. 120
- [KNW98b] M. L. Kersten, N. Nes, and M. A. Windhouwer. A Feature Database for Multimedia Objects. Technical Report INS-R9807, CWI, Amsterdam, The Netherlands, July 1998. 123
- [KS89] M. L. Kersten and A. Siebes. The grammatical datamodel; its algebra. Unpublished manuscript, October 1989. 120

- 
- [KSvdBB96] M. L. Kersten, F. Schippers, C. A. van den Berg, and P. A. Boncz. Mx documentation tool. January 1996. [120](#)
- [KV94] Paul Klint and Eelco Visser. Using filters for the disambiguation of context-free grammars. In G. Pighizzini and P. San Pietro, editors, *Proceedings of the ASNICS Workshop on Parsing Theory*, pages 1–20, Milano, Italy, 1994. [21](#)
- [LaL77] W. R. LaLonde. Regular right part grammars and their parsers. *Communications of the ACM*, 20(10):731–741, 1977. [46](#)
- [Lan74] B. Lang. Deterministic techniques for efficient non-deterministic parsers. In J. Loeckx, editor, *Proceedings of the Second Colloquium on Automata, Languages and Programming*, volume 14 of *Lecture Notes in Computer Science*, pages 225 – 269. Springer Verlag, 1974. [73](#)
- [Lew00] Michael S. Lew. Next generation web searches for visual content. *IEEE Computer*, pages 46–53, November 2000. [11](#)
- [Lew02] Amelia Lewis. *Not My Type: Sizing Up W3C XML Schema Primitives*. [www.xml.com/pub/a/2002/07/31/wxstypes.html](http://www.xml.com/pub/a/2002/07/31/wxstypes.html), 2002. [104](#)
- [LH96] Michael S. Lew and Nies Huijsmans. Information theory and face detection. In *Proceedings of the International Conference on Pattern Recognition*, pages 601–605, Vienna, Austria, 1996. [124](#)
- [Lin97] Peter Linz. *An Introduction to Formal Languages and Automata - Second Edition*. Jones and Bartlett Publications, 1997. [15](#), [16](#), [17](#)
- [LMB92] John R. Levinne, Tony Mason, and Doug Brown. *lex & yacc*. O'Reilly & Associates, Inc., 1992. [80](#), [97](#)
- [Ltd03] Monsell EDM Ltd. *Change Control for XML, in XML*. [www.deltaxml.com](http://www.deltaxml.com), 2003. [116](#)
- [Man94] A. Manaster Ramer. Uses and misuses of mathematics in linguistics. In *Proceedings of Xth Congress on Natural and Formal Languages*, Sevilla, 1994. [17](#)
- [Mat01] The MathWorks, Inc. *The MathWorks Homepage*. [www.mathworks.com](http://www.mathworks.com), 2001. [56](#)
- [MB01] Colin Meek and William P. Birmingham. Thematic extractor. In *2nd Annual International Symposium on Music Information Retrieval*, 2001. [3](#)
- [MHF83] Mitchell Marcus, Donald Hincle, and Margaret M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 129 – 136, 1983. [40](#)
- [Mic68] D. Michie. Memo functions and machine learning. *Nature*, 218:19 – 22, 1968. [20](#), [81](#)
- [Mic99] Microsoft, [www.microsoft.com/Developer/PRODINFO/directx/dxm/help/ds/default.htm](http://www.microsoft.com/Developer/PRODINFO/directx/dxm/help/ds/default.htm). *DirectShow*, 1999. [43](#)
- [Mic02] Microsoft. *Microsoft XML Diff and Patch*. [apps.gotdotnet.com/xmltools/xmlldiff/](http://apps.gotdotnet.com/xmltools/xmlldiff/), 2002. [116](#)
- [Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. [6](#), [58](#)

- [MM96] Valeria Mihalache and Victor Mitran. Deterministic cooperating distributed grammar systems. Technical Report 63, Turku Centre for Computer Science, November 1996. [73](#)
- [MM99] J. Martinez and N. Mouaddib. Multimedia and Databases: A Survey. *Networking and Information Systems Journal*, 2(1):89–123, 1999. [3](#)
- [MMT89] W. C. Mann, CX. M. I. M. Matthiesen, and S. A. Thompson. Rhetorical structure theory and text analysis. Technical Report ISI/RR-89-242, Information Sciences Institute, University of Southern California, November 1989. [135](#)
- [Nes00] N. Nes. *Image Database Management System Design Considerations, Algorithms and Architecture*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, December 2000. [120](#)
- [Neu91] Günter Neumann. A bidirectional model for natural language processing. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics*, pages 245 – 250, Berlin, Germany, April 1991. [19](#)
- [Nil98] Nils J. Nilsson. *Artificial Intelligence – A New Synthesis*. Morgan Kaufmann, 1998. [20](#), [24](#)
- [NK98] N. Nes and M. L. Kersten. The Acoi Algebra: a Query Algebra for Image Retrieval Systems. In *Proceedings of the British National Conference on Databases (BNCOD)*, volume 1405 of *Lecture Notes in Computer Science (LNCS)*, © Springer-Verlag, pages 77–88, Cardiff, Wales, U.K., July 1998. [120](#)
- [NL00] F. Nack and C. Lindley. Production and maintenance environments for interactive audio-visual stories. In *Proceedings ACM MM 2000 Workshops - Bridging the Gap: Bringing Together New Media Artists and Multimedia Technologists*, pages 21 – 24, Los Angeles, CA, October 2000. [135](#)
- [Nor91] Peter Norvig. Techniques for automatic memoization with application to context-free parsing. *Computational Linguistics*, 17(1):91 – 98, 1991. [81](#)
- [NWH<sup>+</sup>01] F. Nack, M. A. Windhouwer, L. Hardman, E. Pauwels, and M. Huijberts. The Role of High-level and Low-level Features in Style-based Retrieval and Generation of Multimedia Presentations. *The New Review of Hypermedia and Multimedia*, 7:39–65, 2001. [121](#), [133](#)
- [NWP<sup>+</sup>01] F. Nack, M. A. Windhouwer, E. Pauwels, M. Huijberts, and L. Hardman. The Role of High-level and Low-level Features in Semi-automated Retrieval and Generation of Multimedia Presentations. Technical Report INS-R0108, CWI, Amsterdam, The Netherlands, June 2001. [133](#)
- [OS95] V.E. Ogle and M. Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, pages 40–48, September 1995. [2](#)
- [Par93] Terence John Parr. *Obtaining Practical Variants of  $LL(k)$  and  $LR(K)$  for  $k > 1$  by Splitting the Atomic  $k$ -tuple*. PhD thesis, Purdue University, August 1993. [73](#), [75](#), [81](#)
- [Pei60] C. Peirce. *The Collected Papers of Charles Sanders Peirce: (1) Principles of Philosophy and (2) Elements of Logic*. The Belknap Press of Harvard University Press, Cambridge, 1960. [135](#)

- [Pet03] Milan Petkovic. *Content-Based Video Retrieval Supported by Database Technology*. PhD thesis, Centre for Telematics and Information Technology, Enschede, The Netherlands, February 2003. 128
- [PF00] E. Pauwels and G. Frederix. Image segmentation by nonparametric clustering based on the kolmogorov-smirnov distance. In *Proceedings of the ECCV 2000, 6th European Conference on Computer Vision*, pages 85 – 99, Dublin, June 2000. 138
- [PJ00] M. Petkovic and W. Jonker. A framework for video modelling. In *proceedings of Intl. Conf. on Applied Informatics*, Innsbruck, Austria, February 2000. 6
- [PQ95] Terence J. Parr and Russell W. Quong. ANTLR: A predicated- $ll(k)$  parser generator. *Journal of Software, Practice & Experience*, 25:789, July 1995. 1995. 80
- [PQ96] Terence J. Parr and Russell W. Quong. LL and Lr translators need  $k > 1$  lookahead. *SIGPLAN Notices*, 31(2), 1996. 80
- [Pro03] The Apache Jakarta Project. *Apache Tomcat*. <http://jakarta.apache.org/tomcat/>, 2003. 124
- [PS94] C. Pollard and I.A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, 1994. 27
- [PS98] Gheorghe Paun and Arto Salomaa, editors. *Grammatical Models of Multi-Agent Systems*. Topics in Computer Mathematics. Taylor & Francis, 1998. 73
- [PWvZ<sup>+</sup>02] M. Petkovic, M. A. Windhouwer, R. van Zwol, H. E. Blok, P. M. G. Apers, M. L. Kersten, and W. Jonker. Content-based Video Indexing for the Support of Digital Library Search. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, San Jose, California, USA, February 2002. 128
- [Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. 6, 58, 139
- [RBvO<sup>+</sup>00] L. Rutledge, B. Bailey, J. van Ossenbrugge, L. Hardman, and J. Geurts. Generating presentation constraints from rhetorical structure. In *Proceedings of the 11th ACM Conference on Hypertext and Hypermedia*, pages 19 – 28, San Antonio, Texas, USA, May - June 2000. 139
- [Rek91] J. Rekers. Generalized Lr parsing for general context-free grammars. Technical Report CS-R9153, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 1991. 73
- [RHM98] Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Human perception subjectivity and relevance feedback in multimedia information retrieval. In *Proceedings of IS&T and SPIE Storage and Retrieval of Image and Video Databases VI*, San Jose, CA, January 1998. 3
- [RHOM98] Yong Rui, Thomas S. Huang, Michael Ortega, and Sharad Mehrotra. Relevance feedback: A power tool in interactive content-based image retrieval. *IEEE Trans on Circuits and Systems for Video Technology*, 8(5):644 – 655, 1998. 3
- [RJ99] Brian Roark and Mark Johnson. Efficient probabilistic top-down and left-corner parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 421 – 428, 1999. 74

- [Roc71] J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART retrieval system: Experiments in automatic document processing*, pages 313 – 323. Prentice-Hall, 1971. 3
- [RS70] D.J. Rosenkrantz and R.E. Stearns. Properties of deterministic top-down grammars. *Information and Control*, 17:225 – 256, 1970. 80
- [RS97a] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages – Vol. 1 Word, Language, Grammar*. Springer-Verlag, 1997. 17
- [RS97b] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages – Vol. 2 Linear Modelling: Background and Application*. Springer-Verlag, 1997. 17, 22
- [RTG98] Y. Rubner, C. Tomasi, and L. Guibas. Adaptive color-image embeddings for database navigation. In *Proceedings of the 3rd Asian Conference on Computer Vision (ACCV98)*, pages 104 – 111, Hong Kong, 1998. 3
- [SC96] John R. Smith and Shih-Fu Chang. Searching for Images and Videos on the World-Wide Web. Technical Report 459-96-25, Columbia University, August 1996. 11
- [Sch02] A. R. Schmidt. *Processing XML in Database Systems*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, November 2002. 101
- [SDWW01] A. Th. Schreiber, Barbara Dubbeldam, Jan Wielemaker, and Bob Wielinga. Ontology-Based Photo Annotation. *IEEE Intelligent Systems*, 16(3):66–74, May/June 2001. 3
- [SK98] Amit P. Sheth and Wolfgang Klas. *Multimedia Data Management: Using Metadata to Integrate and Apply Digital Media*. McGraw-Hill, 1998. 8
- [SKWW00] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents (Extended Version). In *The World Wide Web and Databases - Selected Papers of WebDB 2000*, volume 1997 of *Lecture Notes in Computer Science (LNCS)*, © Springer-Verlag, pages 137–150, 2000. 101
- [Sol69] A. Solomaa. Probabilistic and weighted grammars. *Information and Control*, 15:529 – 544, 1969. 21
- [SR00] S. Santini and J. Ramesh. Integrated browsing and querying for image databases. *IEEE MultiMedia*, 7(3):26–39, July - September 2000. 134
- [Sta92] William Stallings. *Operating Systems*. Maxwell Macmilan, 1992. 37
- [Sub97] V. S. Subrahmaniam. *Principles of Multimedia Database Systems*. Morgan Kaufmann Publishers, Inc., 1997. 1
- [SWK99] A. Schmidt, M. Windhouwer, and M. L. Kersten. Feature grammars. In *ISAS'99 The 5th. Int'l Conference on Information Systems Analysis and Synthesis*, Orlando, Florida, 1999. 121
- [SWK<sup>+</sup>01] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001. 102



- 
- [SWK<sup>+</sup>02] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 974–985, Hong Kong, China, August 2002. 102
- [SWS<sup>+</sup>00] Arnold W. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, December 2000. 4
- [Tch01] Paul A. Tchistopolskii. *PXSLServlet*. [www.pault.com/pault/old/Pxsl](http://www.pault.com/pault/old/Pxsl), 2001. 124
- [Tom86] Masaru Tomita. *Efficient parsing for natural language*. Kluwer Academic Publishers, 1986. 20, 21, 32
- [vdBdJKO00] M.G.J. van den Brand, H.A. de Jong, P. Klint, and P.A. Olivier. Efficient annotated terms. *Journal of Software, Practice & Experience*, 30(3):259 – 291, 2000. 43
- [vdBKMV03] M.G.J. van den Brand, A.S. Klusener, L. Moonen, and J.J. Vinju. Generalized parsing and term rewriting: Semantics driven disambiguation. In B.R. Bryant and J. Saraiva, editors, *Proceedings of the Third Workshop on Language Description, Tools and Applications (LDTA03)*, volume 82 of *Electronic Notes in Theoretical Computer Science*, 2003. 21
- [vdV01] Eric van der Vlist. *Comparing XML Schema Languages*. [www.xml.com/lpt/a/2001/12/12/schemacompare.html](http://www.xml.com/lpt/a/2001/12/12/schemacompare.html), 2001. 104
- [Vei03] Daniel Veillard. *The XML C library for Gnome*. [xmlsoft.org](http://xmlsoft.org), 2003. 88, 121, 122
- [Vel98] Daan Velthausz. *Cost-effective Network-based Multimedia Information Retrieval*. PhD thesis, Telematica Instituut, 1998. 6
- [Vis97] Eelco Visser. *Syntax Definition for Language Prototyping*. PhD thesis, University of Amsterdam, 1997. 43
- [vLdLW00] R. van Liere, W. de Leeuw, and F. Waas. Interactive Visualization of Multi-dimensional Feature Spaces. In *New Paradigms for Information Visualization*, pages 58–71, Washington, DC, USA, November 2000. 3
- [vOGC<sup>+</sup>01] J. van Ossenbruggen, J. Geurts, F. Cornelissen, L. Rutledge, and L. Hardman. Towards second and third generation web-based multimedia. In *Proceedings of the Tenth International World Wide Web Conference*, pages 279 – 488, Hong Kong, May 2001. 136, 140
- [VS92] K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):481 – 518, 1992. 40
- [VWS01] J. Vendrig, M. Worring, and A.W.M. Smeulders. Filter image browsing: Interactive image retrieval by using database overviews. *Multimedia Tools and Applications*, 15(1):83–103, September 2001. 3
- [vZ02] Roelof van Zwol. *Modelling and searching web-based document collections*. PhD thesis, Centre for Telematics and Information Technology, Enschede, The Netherlands, April 2002. 127

- [vZAW99] R. van Zwol, P.M.G. Apers, and A.N. Wilschut. Modelling and querying semistructured data with moa. In *Proceedings of the Workshop on Semi-Structured Data and Non-Standard Data Formats (SSDNSDF 1999)*, Jerusalem, Israel, 1999. 100
- [W3C98] W3C. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*. [www.w3.org/TR/REC-smil/](http://www.w3.org/TR/REC-smil/), June 1998. 136, 139
- [W3C00] W3C. *Extensible Markup Language (XML)*. [www.w3.org/TR/REC-xml](http://www.w3.org/TR/REC-xml), 2000. 50, 86, 147
- [W3C01a] W3C. *Document Object Model (DOM)*. [www.w3.org/DOM/](http://www.w3.org/DOM/), 2001. 88
- [W3C01b] W3C. *Extensible Stylesheet Language (XSL)*. [www.w3.org/Style/XSL/](http://www.w3.org/Style/XSL/), 2001. 103
- [W3C01c] W3C. *Synchronized Multimedia Integration Language (SMIL 2.0)*. [www.w3.org/TR/smil20/](http://www.w3.org/TR/smil20/), August 2001. 136
- [W3C01d] W3C. *XML Path Language (XPath) 2.0*. [www.w3.org/TR/xpath20](http://www.w3.org/TR/xpath20), December 2001. 49, 55, 136
- [W3C02a] W3C. *Libwww - the W3C Protocol Library*. [www.w3c.org/Library/](http://www.w3c.org/Library/), 2002. 121
- [W3C02b] W3C. *XML Pointer Language*. [www.w3c.org/TR/xptr/](http://www.w3c.org/TR/xptr/), August 2002. 136
- [WSK99] M. A. Windhouwer, A. R. Schmidt, and M. L. Kersten. Acoi: A System for Indexing Multimedia Objects. In *International Workshop on Information Integration and Web-based Applications & Services*, Yogyakarta, Indonesia, November 1999. 121, 123
- [WSK00] M. A. Windhouwer, A. R. Schmidt, and M. L. Kersten. Acoi: A System for Indexing Multimedia Objects. In *Proceedings of the International World Wide Web Conference*, Amsterdam, The Netherlands, May 2000. 123
- [WSvZ<sup>+</sup>01] M. A. Windhouwer, A. R. Schmidt, R. van Zwol, M. Petkovic, and H. E. Blok. Flexible and Scalable Digital Library Search. Technical Report INS-R0111, CWI, Amsterdam, The Netherlands, December 2001. 125, 128
- [WSvZ<sup>+</sup>03] M. A. Windhouwer, A. R. Schmidt, R. van Zwol, M. Petkovic, and H. E. Blok. Flexible Digital Library Search. In A. Dahanayake and W. Gerhardt, editors, *Web-enabled Systems Integration: Challenges and Practices*, pages 200–224. Idea Group Publishing, 2003. 121, 128
- [WvZ03] M. A. Windhouwer and R. van Zwol. Combining Concept- with Content-based Multimedia Retrieval. In H. M. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors, *Intelligent Search on XML Data*, volume 2818 of *Lecture Notes in Computer Science (LNCS)*, © Springer-Verlag, pages 217 – 230, 2003. 128, 132
- [XFr01] XFront, [www.xfront.com/ExtendingSchemas.html](http://www.xfront.com/ExtendingSchemas.html). *Extending XML Schemas*, 2001. 104
- [YW03] Jin-Yi Cai Yuan Wang, David J. DeWitt. *X-Diff – Detecting Changes in XML Documents*. [www.cs.wisc.edu/~yuanwang/xdiff.html](http://www.cs.wisc.edu/~yuanwang/xdiff.html), 2003. 116

- [ZS89] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18(6):1245 – 1262, 1989. 116
- [ZS90] K. Zhang and D. Shasha. Fast algorithms for unit cost editing between trees. *Journal of Algorithms*, 11(6):581 – 621, 1990. 116



# Index

- abstract attributes, 5
- acceptation style, 25
- accepting grammar mode, 19
- Acoi, 8
- Acoi image algebra, 120
- Admire, 5
- ALGOL 60, 45
- alternative production rules, 20, 46
- ambiguity, 5, 20, 68
- ambiguous feature grammar systems, 39
- analysis, 57
- AND/OR-graph, 20
- annotation extraction algorithm, 5
- annotation subsystem, 3
- annotations, 4
- anonymous symbol, 48
- artificial language, 18
- atom, 32, 48
- atom symbols, 32
- atom type, 48
- attribute grammars, 97
- Australian Open, 128
- automaton, 66
- auxiliary tree, 82
- axiom, 15
  
- back propagation neural network, 57
- basic feature grammar system, 29
- batch learning algorithms, 58
- bidirectional grammars, 19
- binary association table (BAT), 99
- blackboard architecture, 24
- blackbox detector, 56
- bottom-up parsing algorithm, 69
- bounds, 54
- breadth-first parsing algorithm, 70
  
- Chomsky hierarchy, 16
- classifier, 57, 95
- COBRA, 5, 128
- compositional semantics, 5
- concept, 6, 54
- conceptual level, 126
- conditional feature grammar system, 38
- conditional grammars, 22
- constant, 54
- context, 88
- context dependency, 6, 113
- context globalization, 91
- context localization, 91
- context-free (CF) grammars, 17
- context-sensitive (CS) grammars, 16
- controlled named disjunction, 40
- controller named disjunction, 40
- cooperating distributed (CD) grammar systems, 24
- crossed agreements, 17
  
- data repository, 135
- database management system (DBMS), 4
- dataflow languages, 43
- deadlock, 36, 61, 91, 96
- deadlock prevention, 36
- decision rule, 6, 54, 55
- decomposed storage model (DSM), 99
- delta operation, 19
- dependencies, 33
- dependency graph, 111
- depth-first parsing algorithm, 69
- derivation, 18
- derivation mode, 25
- derived features, 5
- detector, 28, 49
- detector confidence, 41, 52

- detector symbols, 28
- detector version, 62
- deterministic, 20, 68
- digital media warehouse (DMW), 1
- direct derivation, 18
- direct self-reference, 91
- directional parsing algorithm, 70
- disambiguation, 21
- document object model (DOM), 86
- document type definition (DTD), 101, 103
- domain-specific language, 14
- dynamically controlled CD grammar system, 26
  
- edge folding, 84
- elementary tree, 82
- erasing production, 16
- exhaustive backtracking parsing algorithm, 75
- extended Backus-Naur form (EBNF), 45
- extended regular expression languages, 17
- extensible markup language, 100
- extensible markup language (XML), 50, 86
- external change, 7, 62, 114
  
- feature, 6
- feature detector engine (FDE), 65
- feature detector scheduler (FDS), 111
- feature grammar systems, 27
- finite automaton (FA), 66
- forward axis, 50
  
- general purpose language (GPL), 94
- generating grammar mode, 19
- generic parser, 82
- grammar, 15
- grammar component, 24
- grammar systems, 24
- grammars, 15
- grammatical database model, 119
- greedy parsing, 47, 85
  
- high-level concept, 6, 54
- human interaction, 21
  
- indirect self-reference, 91
- instantiation, 32
  
- internal change, 7, 62, 115
- internal control, 26
- iterative interpretation, 47
  
- key/reference dependency, 114
  
- left path-controlled (IPC) grammar, 34
- left path-controlled feature grammar system, 34
- left-hand side, 15
- left-recursion, 78
- leftmost derivation, 18
- lexical ambiguity, 21
- lexical analysis, 32
- lexicon, 32
- Linda tuple spaces, 43
- linear bounded automaton, 17
- linear ordering, 36
- list, 53
- local ambiguity, 21
- logical features, 5
- logical level, 126
- lookahead, 80
- low-level feature, 6
- lower bound, 54
  
- machine learning, 6, 57
- Magnum, 100
- match part, 69
- Matlab, 56
- memoization, 20, 81, 90
- meta-language, 45
- Microsoft DirectShow, 43
- mild context-sensitivity, 17
- mildly context-sensitive feature grammar systems, 41
- Mirror, 42
- module, 61
- Monet, 99
- Monet interpreter language (MIL), 99
- MOODS, 42
- multi-agent systems, 24
- multimedia information retrieval, 2
- multiple agreements, 17
  
- named disjunctions, 21, 40
- namespace, 62

- naming conflict, 62
- natural language, 13
- neural network, 6, 57
- non-deterministic, 20, 68
- non-directional parsing algorithm, 70
- non-terminals, 15
  
- observation, 57
- optional, 46
- output/input dependency, 6, 114
  
- packed shared forest, 20
- parallel communicating (PC) grammar systems, 24
- parse forest, 20
- parse tree, 19
- path operation, 19
- path-controlled (PC) grammar, 34
- physical level, 126
- plugin, 56, 94
- polling, 63
- positive closure, 46
- post-order, 102
- post-visitation, 88
- pre-order, 102
- pre-terminals, 32
- pre-visitation, 88
- pre/post plane, 102
- predicate, 6, 54
- predict, 57
- prediction part, 69
- presentation environment, 136
- primitive features, 5
- probabilistic parsing, 21
- production rules, 15, 46
- productions, 15
- push-down automaton (PDA), 66
- push-down transducer (PDT), 66
  
- quasi-foot, 40, 61
- quasi-node, 40, 61
- quasi-root, 40, 61
- quasi-tree, 40
- query clues, 3
- query-by-sketch, 2
- query-by-text, 2
  
- recursive descent, 86
- recursive interpretation, 46, 78
- recursively enumerable (RE) grammars, 16
- reduce phase, 70
- reduced derivation tree (RDT), 93
- reduplication, 17
- reference, 58, 95
- regular (REG) grammars, 17
- regular expression language, 17
- regular right part grammars (RRPG), 46
- regulated rewriting, 21
- Relax NG, 103
- relevance feedback, 3
- revalidation, 117
- reverse axis, 50
- right-hand side, 15
- right-recursive interpretation, 78
- Rijksmuseum, 133
  
- Schematron, 103
- scope, 62, 88
- self-reference, 61, 91
- semantic ambiguity, 21
- semantic concept, 6, 54
- semantic gap, 4
- semantic grammar, 32
- sentential form, 18
- set, 53
- shift phase, 70
- shredding, 100
- similarity, 3
- specialized parser, 82
- staircase join, 102
- star closure, 46
- start condition, 26
- start symbol, 15, 52, 60, 95
- stop condition, 26
- stop condition types, 27
- structural ambiguity, 21
- style repository, 133
- subjectivity, 5
- substitution language, 32
- symbol table, 84
- symbols, 15
- syntactic ambiguity, 21
  
- terminals, 15

- theory, 57
- ToolBus, 43
- top-down parsing algorithm, 69
- transducer, 66
- transitive closure, 18
- tree adjoining grammars (TAG), 82
- tree-controlled grammars, 22
- tuple, 53
- Type 0 grammars, 16
- Type 1 grammars, 16
- Type 2 grammars, 17
- Type 3 grammars, 17
  
- upper bound, 54
  
- validation, 117
- variables, 15
  
- Webspace Method, 127
- webservice schema, 127
- whitebox detector, 54
  
- XMark, 101
- XML document, 50
- XML Schema, 103
- XPath axis, 50
- XPath expression, 50, 93
- XPath language, 50
- XPath node test, 51
- XPath step qualifier, 51, 55
- XQuery, 102
  
- yield operation, 19



# Samenvatting

Met de opmars van personal computers en allerlei vormen van randapparatuur om traditionele media te digitaliseren, met name onder de invloed van het als maar dalen van de aanschafkosten en het stijgen van de opslagcapaciteit, zijn grote collecties van digitale media (*digital media warehouses*) gemeengoed geworden. Maar het opslaan van digitale objecten is slechts één kant van de zaak, de gebruikers willen deze objecten ook weer terugvinden. Het terugvinden van deze objecten is echter geen sinecure en een omvangrijke en multidisciplinaire onderzoeksgemeenschap houdt zich daar dan ook mee bezig.

De focus van dit proefschrift wordt gevormd door één stap in het zoekproces. De media objecten in hun digitale vorm zijn namelijk niet gemakkelijk te vinden, daarvoor moeten ze geannoteerd worden. Deze annotaties kunnen zowel handmatig als automatisch gecreeërd worden. In het laatste geval worden de annotaties geproduceerd door uitgeprogrammeerde extractie-algoritmes, die op de objecten worden losgelaten.

De extractie-algoritmes zijn van elkaar en van elkaars annotaties afhankelijk. Allereerst kan er sprake zijn van een uit/invoer afhankelijkheid: de uitvoer van het ene algoritme is de invoer van een volgend algoritme. Een voorbeeld hiervan is het bepalen van het type van een afbeelding, een tekening of een foto. Dit gebeurt op basis van eerder geproduceerde annotaties, zoals het aantal en de gemiddelde verzadiging van de kleuren in de afbeelding. Daarnaast is er de mogelijkheid van een context afhankelijkheid. Hierbij wordt een extractie-algoritme alleen uitgevoerd als een eerder algoritme geslaagd is. Dit wordt geïllustreerd door de volgende afhankelijkheid: het bepalen of een afbeelding een of meerdere gezichten bevat is alleen nodig als eerst bepaald is dat de afbeelding een foto is.

Een complicatie is de semantiek van de annotaties. Eenvoudige annotaties, bijvoorbeeld de kleur geel komt in deze afbeelding voor, zijn eenduidig. Meer abstracte annotaties, zoals dit is een grimmige foto, zijn ambigu. Hun validiteit is afhankelijk van de context waarin het object zich bevindt, of de (cultureel bepaalde) context van de gebruiker. Een annotatie systeem moet dan ook de productie en het gebruik van alternatieve interpretaties ondersteunen.

In het proefschrift wordt een formele taal, kenmerk grammatica systemen (*feature grammar systems*), die voor het gelijktijdig beschrijven van de afhankelijkheden en de (alternatieve) contexten is ontwikkeld. In een natuurlijke taal worden valide zinnen beschreven door een grammatica. De grammatica bepaalt welke woorden samen, in een specifieke context, mogen voorkomen. Een kenmerk grammatica systeem doet hetzelfde voor annotaties en extractie-algoritmes. Daartoe bestaat een kenmerk grammatica systeem uit één of meerdere grammatica componenten. Elk component beschrijft het resultaat van een algoritme en de afhankelijkheid van andere componenten. Deze beschrijving kan alternatieve interpretaties bevatten.

Het extractie-proces kan nu gestuurd worden door een kenmerk grammatica systeem te interpreteren. Dit proces komt overeen met het parsen van zinnen in een natuurlijke of artificiële taal. Hiervoor zijn door de jaren heen veel efficiënte algoritmes ontwikkeld. Echter slechts enkele hiervan zijn geschikt voor kenmerk grammatica systemen. Een probleem is het dynamisch groeien van de annotatie zin: het activeren van een grammatica component leidt tot het uitvoeren van een extractie-algoritme en dus tot de productie van annotaties. Een ander probleem wordt gevormd door de afhankelijkheden: een extractie-algoritme kan pas worden uitgevoerd als zijn invoer, reeds eerder geproduceerde annotaties, beschikbaar is. Hierdoor komen alleen parseer algoritmes in aanmerking die van boven naar beneden werken. Een specifiek algoritme, die aan deze kenmerken voldoet, is geïmplementeerd en produceert de annotaties, beschreven door een of meerdere parseerbomen. Deze bomen worden opgeslagen in een database management systeem.

Verschillende factoren, zoals wijzigingen in de algoritmes, kunnen er echter toe leiden dat de opgeslagen bomen, en dus ook de annotaties, niet meer de werkelijkheid weerspiegelen. Om de invloed van deze wijzigingen te lokaliseren wordt er van het kenmerk grammatica systeem een afhankelijkheidsgraaf afgeleid. Een planningsproces kan dan het extractie-proces gedeeltelijk herstarten om daarmee de database te modificeren.

Het aldus ontworpen systeem, ACOI, is de afgelopen jaren aan het CWI ontwikkeld en ingezet bij verschillende praktijkstudies. Analyse van deze studies toont aan dat een kenmerk grammatica systeem een praktisch inzetbaar hulpmiddel is om (alternatieve) annotaties te produceren en te onderhouden.



- [4] A. R. Schmidt, M. L. Kersten, and M. A. Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 321–329, Heidelberg, Germany, April 2001.
- [5] F. Nack, M. A. Windhouwer, L. Hardman, E. Pauwels, and M. Huijberts. The Role of High-level and Low-level Features in Style-based Retrieval and Generation of Multimedia Presentations. *The New Review of Hypermedia and Multimedia*, 7:39–65, 2001.
- [6] A. P. de Vries, M. A. Windhouwer, P. M. G. Apers, and M. L. Kersten. Information Access in Multimedia Databases based on Feature Models. *New Generation Computing*, 18(4):323–339, October 2000.
- [7] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents (Extended Version). In D. Suciú and G. Vossen, editors, *The World Wide Web and Databases - Selected Papers of WebDB 2000*, volume 1997 of *Lecture Notes in Computer Science (LNCS)*, © Springer-Verlag, pages 137–150, 2000.
- [8] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *International Workshop on the Web and Databases (In conjunction with ACM SIGMOD)*, pages 47–52, Dallas, TX, USA, May 2000.
- [9] M. A. Windhouwer, A. R. Schmidt, and M. L. Kersten. Acoi: A System for Indexing Multimedia Objects. In *International Workshop on Information Integration and Web-based Applications & Services*, Yogyakarta, Indonesia, November 1999.
- [10] A. R. Schmidt, M. A. Windhouwer, and M. L. Kersten. Feature Grammars. In *Proceedings of the International Conference on Systems Analysis and Synthesis*, Orlando, FL, USA, August 1999.
- [11] M. L. Kersten, N. Nes, and M. A. Windhouwer. A Feature Database for Multimedia Objects. In *ERCIM Database Research Group Workshop on Metadata for Web Databases*, pages 49–57, Bonn, Germany, May 1998.

## Technical Reports

- [1] M. A. Windhouwer, A. R. Schmidt, R. van Zwol, M. Petkovic, and H. E. Blok. Flexible and Scalable Digital Library Search. Technical Report INS-R0111, CWI, Amsterdam, The Netherlands, December 2001.
- [2] F. Nack, M. A. Windhouwer, E. Pauwels, M. Huijberts, and L. Hardman. The Role of High-level and Low-level Features in Semi-automated Retrieval and Generation of Multimedia Presentations. Technical Report INS-R0108, CWI, Amsterdam, The Netherlands, June 2001.
- [3] A. R. Schmidt, M. A. Windhouwer, and M. L. Kersten. Indexing Real-world Data Using Semi-structured Documents. Technical Report INS-R9902, CWI, Amsterdam, The Netherlands, March 1999.
- [4] M. L. Kersten, N. Nes, and M. A. Windhouwer. A Feature Database for Multimedia Objects. Technical Report INS-R9807, CWI, Amsterdam, The Netherlands, July 1998.

---

## Demonstrations

- [1] M. Petkovic, M. A. Windhouwer, R. van Zwol, H. E. Blok, P. M. G. Apers, M. L. Kersten, and W. Jonker. Content-based Video Indexing for the Support of Digital Library Search. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, San Jose, California, USA, February 2002.
- [2] H. E. Blok, M. A. Windhouwer, R. van Zwol, M. Petkovic, P. M. G. Apers, M. L. Kersten, and W. Jonker. Flexible and scalable digital library search. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Rome, Italy, September 2001.
- [3] M. A. Windhouwer, A. R. Schmidt, and M. L. Kersten. Acoi: A System for Indexing Multimedia Objects. In *Proceedings of the International World Wide Web Conference*, Amsterdam, The Netherlands, May 2000.

## Other Publications

- [1] M. A. Windhouwer. *The Early Adaptors Guide to Acoi*. CWI, 2003.
- [2] CWI. *Digital Media Warehouses CD-Rom*, 2002.



# Acknowledgments

*"It's just a job. Grass grows, birds fly, waves pound the sand. I beat people up."*

Muhammad Ali

When I was a little boy a fascination for collecting and indexing collections already popped up. A small card-tray, a leftover from my dad's administration, was soon filled with a card for each book in my small, but steadily growing, library. And when the first computer entered my life, a borrowed old P2000, a catalog program was one of the first MBasic programs build. Throughout the years when older stuff exited and newer hard- and software entered replacing instantiations of this catalog program were devised, *e. g.* in Pascal and Clipper. So when, in 1996, I was looking for a master's project it was not too strange to accept a project in the area of database management.

This project on the parallelization of data mining queries brought me in, sometimes a bit hard, contact with the Monet database kernel prototype. Enjoying the interaction and freedom within the database group at CWI and the UvA, I was happy to take the opportunity, after finishing my master's thesis, to join.

Being tired of writing text, but not of code, I became a scientific system developer. My task was to build the ACOi image crawler for the AMIS project. However, soon it turned out that the developing ideas could function as the basis for a PhD. In the beginning of 1997 the DMW project was started, and I started as a PhD student at CWI. So writing text was, again, one, or even the, main tasks, and the results of my efforts are in your hands. However, many people had major influence on these results, and it is my honor to thank them all in this final part of the thesis.

First of all I like to thank my promotor Martin Kersten. Just like many of my colleagues he sent me on my research way with a, in this case developed together with Niels Nes, first prototype. Although there is not much of the source code left the underlying ideas proved a fertile basis. In the years not only many CVS messages asked Martin's attention, but also many versions of the various chapters and sections of this thesis. Thanks for the constructive comments and the unwearying attention for my English spelling and sentence structure.

The thesis was approved by the committee members: Peter Apers, Paul de Bra, Peter van Emde Boas and Paul Klint. I like to express my appreciation for their willingness to study this thesis, and thank them for their constructive comments.

Large parts, if not the complete thesis, would not have been written without the constructive environment the DMW project turned out to be. I thank Albrecht Schmidt, Roelof van Zwol, Milan Petkovic and Arno Knobbe, my fellow DMW PhD students, for their cooperation. Although it was a busy and stressful period, I have fun memories of the time we spent building the

Australian Open search engine and our trip to VLDB in Rome to demonstrate it. Thanks also go to the other DMW project members throughout the years. Special thanks and appreciation go to Arjen de Vries for giving this most fruitful period of the DMW project direction and continuous encouragement.

I was glad with several other opportunities to improve the ACOI system. Starting with a huge hand drawn “feature grammar”, which I cherish in my archive, The Rijksmuseum case study was dreamed up by Frank Nack. And joined by Eric Pauwels we tried to automatically classify digitized paintings. This was an entertaining activity with some success, and also turned out to be one of the first cross theme projects at CWI. Henk Ernst Blok stress tested the extensibility of the ACOI framework by extending the WWW search engine with distributed full text search capabilities. This also led to a demonstration at VLDB and a book chapter. And I also like to mention Jeroen Vendrig. We planned for years to build a video annotation feature grammar, unfortunately it never really happened. It has been really nice to work together with all of you.

Next to fellow project members and co-authors CWI provided many other nice colleagues. Stefan Manegold has been an ideal roommate for all these years. We had numerous nice conversations of a technical and cultural nature. Cultural difference keep on being interesting. I hope you did not mind the, sometimes (too) loud, music. Niels Nes and Peter Boncz I like to thank for helping me with many technical issues, and extensive debugging sessions. Also thanks to all the other colleagues, I am sorry I can not name you all here, but you all made CWI a nice working environment.

De wereld buiten het CWI werd vergroot door verschillende trips naar zuidelijk Afrika. Ronald, Patrick en Menno, ik ben benieuwd waar we, mag zich de mogelijkheid nogmaals voordoen, de volgende keer terecht komen. Elke safari werd weer spannender! Although Capetown is a beautiful city, it became a favorite holiday destination due to Dave, Colette and Jane. You have always made us feel welcome and at home and I hope to see you all in the near future.

Zonder mijn broer, Dick, had de omslag er niet zo professioneel uitgezien. Bedankt voor je hulp en de leuke, soms drukke, tijd die we samen in Amsterdam hebben doorgebracht.

Rest mij om mijn familie te danken voor hun ondersteuning en vertrouwen door de jaren heen. De afgelopen jaren waren (soms) zwaar voor ons allen, maar gelukkig brengt de toekomst altijd nieuwe situaties, met zijn eigen mogelijkheden en kansen, mee. Ik deel die toekomst nu samen met Mirjam, en ik kan geen woorden bedenken om haar daar voor te bedanken.

Menzo Windhouwer  
Amsterdam, September 2003



# SIKS Dissertation Series

- 1998-01 Johan van den Akker (CWI), *DEGAS - An Active, Temporal Database of Autonomous Objects*
- 1998-02 Floris Wiesman (UM), *Information Retrieval by Graphically Browsing Meta-Information*
- 1998-03 Ans Steuten (TUD), *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*
- 1998-04 Dennis Breuker (UM), *Memory versus Search in Games*
- 1998-05 E.W.Oskamp (RUL), *Computerondersteuning bij Straftoemeting*
- 1999-01 Mark Sloof (VU), *Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products*
- 1999-02 Rob Potharst (EUR), *Classification using decision trees and neural nets*
- 1999-03 Don Beal (UM), *The Nature of Minimax Search*
- 1999-04 Jacques Penders (UM), *The practical Art of Moving Physical Objects*
- 1999-05 Aldo de Moor (KUB), *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*
- 1999-06 Niek J.E. Wijngaards (VU), *Re-design of compositional systems*
- 1999-07 David Spelt (UT), *Verification support for object database design*
- 1999-08 Jacques H.J. Lenting (UM), *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.*
- 2000-01 Frank Niessink (VU), *Perspectives on Improving Software Maintenance*
- 2000-02 Koen Holtman (TUE), *Prototyping of CMS Storage Management*
- 2000-03 Carolien M.T. Metselaar (UvA), *Sociaal-organisatorische gevolgen van kennis-technologie; een procesbenadering en actorperspectief.*
- 2000-04 Geert de Haan (VU), *ETAG, A Formal Model of Competence Knowledge for User Interface Design*
- 2000-05 Ruud van der Pol (UM), *Knowledge-based Query Formulation in Information Retrieval.*
- 2000-06 Rogier van Eijk (UU), *Programming Languages for Agent Communication*
- 2000-07 Niels Peek (UU), *Decision-theoretic Planning of Clinical Patient Management*
- 2000-08 Veerle Coupé (EUR), *Sensitivity Analysis of Decision-Theoretic Networks*

- 2000-09 Florian Waas (CWI), *Principles of Probabilistic Query Optimization*
- 2000-10 Niels Nes (CWI), *Image Database Management System Design Considerations, Algorithms and Architecture*
- 2000-11 Jonas Karlsson (CWI), *Scalable Distributed Data Structures for Database Management*
- 2001-01 Silja Renooij (UU), *Qualitative Approaches to Quantifying Probabilistic Networks*
- 2001-02 Koen Hindriks (UU), *Agent Programming Languages: Programming with Mental Models*
- 2001-03 Maarten van Someren (UvA), *Learning as problem solving*
- 2001-04 Evgueni Smirnov (UM), *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*
- 2001-05 Jacco van Ossenbruggen (VU), *Processing Structured Hypermedia: A Matter of Style*
- 2001-06 Martijn van Welie (VU), *Task-based User Interface Design*
- 2001-07 Bastiaan Schonhage (VU), *Diva: Architectural Perspectives on Information Visualization*
- 2001-08 Pascal van Eck (VU), *A Compositional Semantic Structure for Multi-Agent Systems Dynamics.*
- 2001-09 Pieter Jan 't Hoen (RUL), *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*
- 2001-10 Maarten Sierhuis (UvA), *Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design*
- 2001-11 Tom M. van Engers (VUA), *Knowledge Management: The Role of Mental Models in Business Systems Design*
- 2002-01 Nico Lassing (VU), *Architecture-Level Modifiability Analysis*
- 2002-02 Roelof van Zwol (UT), *Modelling and searching web-based document collections*
- 2002-03 Henk Ernst Blok (UT), *Database Optimization Aspects for Information Retrieval*
- 2002-04 Juan Roberto Castelo Valdueza (UU), *The Discrete Acyclic Digraph Markov Model in Data Mining*
- 2002-05 Radu Serban (VU), *The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents*
- 2002-06 Laurens Mommers (UL), *Applied legal epistemology; Building a knowledge-based ontology of the legal domain*
- 2002-07 Peter Boncz (CWI), *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*
- 2002-08 Jaap Gordijn (VU), *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*

- 
- 2002-09 Willem-Jan van den Heuvel (KUB), *Integrating Modern Business Applications with Objectified Legacy Systems*
- 2002-10 Brian Sheppard (UM), *Towards Perfect Play of Scrabble*
- 2002-11 Wouter C.A. Wijngaards (VU), *Agent Based Modelling of Dynamics: Biological and Organisational Applications*
- 2002-12 Albrecht Schmidt (CWI), *Processing XML in Database Systems*
- 2002-13 Hongjing Wu (TUE), *A Reference Architecture for Adaptive Hypermedia Applications*
- 2002-14 Wieke de Vries (UU), *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*
- 2002-15 Rik Eshuis (UT), *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*
- 2002-16 Pieter van Langen (VU), *The Anatomy of Design: Foundations, Models and Applications*
- 2002-17 Stefan Manegold (CWI), *Understanding, Modeling, and Improving Main-Memory Database Performance*
- 2003-01 Heiner Stuckenschmidt (VU), *Ontology-Based Information Sharing in Weakly Structured Environments*
- 2003-02 Jan Broersen (VU), *Modal Action Logics for Reasoning About Reactive Systems*
- 2003-03 Martijn Schuemie (TUD), *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*
- 2003-04 Milan Petkovic (UT), *Content-Based Video Retrieval Supported by Database Technology*
- 2003-05 Jos Lehmann (UvA), *Causation in Artificial Intelligence and Law - A modelling approach*
- 2003-06 Boris van Schooten (UT), *Development and specification of virtual environments*
- 2003-07 Machiel Jansen (UvA), *Formal Explorations of Knowledge Intensive Tasks*
- 2003-08 Yongping Ran (UM), *Repair Based Scheduling*
- 2003-09 Rens Kortmann (UM), *The resolution of visually guided behaviour*
- 2003-10 Andreas Lincke (UvT), *Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture*
- 2003-11 Simon Keizer (UT), *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*
- 2003-12 Roeland Ordelman (UT), *Dutch speech recognition in multimedia information retrieval*
- 2003-13 Jeroen Donkers (UM), *Nosce Hostem - Searching with Opponent Models*
- 2003-14 Stijn Hoppenbrouwers (KUN), *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*
- 2003-15 Mathijs de Weerd (TUD), *Plan Merging in Multi-Agent Systems*