

MODEL-DRIVEN DEVELOPMENT OF CONTENT-BASED  
IMAGE RETRIEVAL SYSTEMS ON TOP OF  
OBJECT-RELATIONAL DATABASE MANAGEMENT  
SYSTEMS

Modellgetriebene Entwicklung inhaltsbasierter  
Bildretrieval-Systeme auf der Basis von objektrelationalen  
Datenbank-Management-Systeme

Dissertation  
zur  
Erlangung des akademischen Grades  
Doktor-Ingenieur (Dr.-Ing.)  
der Fakultät für Informatik und Elektrotechnik  
der Universität Rostock

vorgelegt von  
Temenushka Ignatova, geb. am 18.10.1976 in Kyustendil, Bulgarien  
aus Rostock

Rostock, den 02.07.2008

Gutachter:

Prof. Dr. Andreas Heuer, Universität Rostock

Prof. Dr. Klaus Meyer-Wegener, Friedrich-Alexander-Universität Erlangen-Nürnberg

Prof. Dr. Susanne Boll, Carl von Ossietzky Universität Oldenburg

Tag der Promotionsverteidigung: 18. September 2008



# Abstract

Digital images are increasingly used for capturing information. The growing amount of such data requires the development of adequate techniques for its reliable storage and efficient retrieval. Different computer science communities, such as computer vision, information retrieval, database systems, artificial intelligence etc., have devoted their efforts and expertise to providing a solution for this challenge. The search for meaningful visual characteristics of images, which can be used to automatically compare images by similarity, is an ongoing process. This is due to the fact that each application domain, each user and even each situation require different information from the image to be considered for the comparison. For these characteristics, suitable measures for calculating the similarity between them have been proposed. However, these measures must also be tested and parametrized for a particular application. Robust image processing algorithms for extracting the visual characteristics from the raw image data have to be implemented so that as few as possible information from the original image is lost. Data Mining techniques have been applied for deriving semantic data from the low-level image characteristics in order to bring the computer-based representation of the image content as close as possible to the human way of describing images. And finally, ways to associate the image data with other context dependent data in order to provide an integrated system to the user or gain more information about the content of the images have been investigated.

All these techniques have been implemented in so called Content-based Image Retrieval Systems (CBIRS). The prior idea of these systems was to support the similarity search for any kind of image collection. It was relatively fast recognized that this is not possible due to the different requirements of different application domains concerning the image characteristics and measures used for the retrieval. On the one hand, picking the right algorithms for a particular application is one of the challenges with which a developer of a CBIRS has to deal. On the other hand, building the application requires the definition of an architecture, data structures and functionality components. The basic architecture of these systems does not differ much, also in the specialized applications. Therefore, in order to help building such specialized applications by focusing mainly on choosing the right combination of algorithms an adequate development support mechanism is required. The first attempts to achieve this are based on software frameworks and source code libraries. However, these techniques do not provide enough flexibility with respect to platform independence and the resulting application are not compact specialized applications, but rather an extended version of the framework system.

In this thesis, the model-driven software development paradigm is employed as a development support technique for CBIRS. Therefore, two groups of techniques are elaborated, modeling and transformation techniques. Modeling techniques, based on a conceptual framework model

are proposed for modeling the components for image storage, feature extraction and image retrieval of a CBIRS architecture. Therefore, generic data structures and operations for the update, storage and retrieval of images are defined as a framework model, which can be used by developers to derive their own application specific conceptual models for CBIRS. In this thesis, transformation techniques for the automatized implementation of the model in an ORDBMS environment are defined. These are specified in terms of transformation rules for mapping the platform independent model concepts onto concepts of a platform specific model for object-relational database management systems (ORDBMS). Database management systems were chosen as a target platform for the implementation in this thesis because of the well established mechanisms for inserting and updating information which these systems provide, e.g. transaction management. Furthermore, database systems can be used to link other information to the image data in form of textual metadata, for which they can provide efficient access. However, since the conceptual model is platform independent, any other platform can be considered for the implementation of the model. In particular ORDBMS were used because of the insufficient support for multimedia data in relational database management system, and the need for such support in information integration applications, such as digital libraries, multimedia information systems.

The transformation techniques are evaluated by verifying the quality of the transformation rules. The criteria assuring the quality of the transformation were derived from the requirement for information capacity preservation of the transformation known in the database design theory. The investigation of the mapping rules showed that the derived quality requirements are fulfilled.

The elaborated modeling techniques are applied for the modeling of a CBIR system for storing images of music scores, and identifying their scribes, based on the visual handwriting characteristics of the images. In addition, a CBIRS application for the retrieval of similar images of 2D-electrophoresis Gels is derived from the CBIRS framework model. And finally, the model is evaluated for an application for the annotation of photos. These test cases showed that a large class of CBIRS fits well into the generic framework model, i.e. these applications can be easily modeled by making use of the proposed modeling techniques.

# Kurzfassung

Immer häufiger werden digitale Bilder zur Aufnahme verschiedenster Informationen eingesetzt. Für die effiziente Verwaltung dieser Informationen werden passende Techniken für ihre Speicherung und Suche benötigt. Verschiedene Forschungszweige der Informatik, wie beispielsweise die Bildverarbeitung, Information Retrieval, Datenbanksysteme und Künstliche Intelligenz, stellen sich diesen Herausforderungen. Im Zentrum dieser Lösungsversuche steht immer wieder das Finden relevanter visueller Bildmerkmale, mit deren Hilfe Ähnlichkeiten zwischen digitalen Bildern gefunden werden sollen. Die Frage, inwieweit sich Bilder ähnlich sind, muss für jedes neue Anwendungsgebiet und jede neue Nutzeranforderung anders gestellt werden. Die relevanten Informationen aus den Bildern sind für jedes einzelne Gebiet ebenso vielfältig wie die Ansätze, diese als Basis für einen Bildvergleich zu klassifizieren. Die verschiedenen relevanten Bildmerkmale in den zu vergleichenden Bildern werden immer wieder neu mit spezifischen Ähnlichkeitsmaßen versehen, um die Ähnlichkeit mathematisch erfassen zu können. Diese müssen getestet und parametrisiert werden, um die menschliche Wahrnehmung für Ähnlichkeit so genau wie möglich zu simulieren. Dazu werden komplexe Bildverarbeitungsalgorithmen entwickelt, um die gewünschten Merkmale fehlerfrei und effizient aus den Bildern zu extrahieren. Auf die Ergebnisse dieser Algorithmen werden Data Mining Techniken angewendet. Damit lassen sich einfachen, messbaren Merkmalen wie zum Beispiel der Farbe und der Textur semantische Konzepte, in etwa Sonne oder Himmel, zuordnen. Und schließlich werden Möglichkeiten untersucht, um andere kontextabhängige Informationen mit den Bildern zu assoziieren, um den Inhalt der Bilder zuverlässiger automatisch erkennen zu können.

Das Zusammenspiel dieser Techniken wird in so genannten inhaltsbasierten Bildretrieval-Systemen (auf Englisch: Content-based Image Retrieval Systems CBIRS) verwendet. Das ursprüngliche Ziel für den Einsatz dieser Systeme war es, die Ähnlichkeitssuche auf verschiedenste Bildinhalte gleichermaßen zu unterstützen. Jedoch wurde schon frühzeitig erkannt, dass diese Verschiedenheit der Bilder eine zu große Hürde für CBIRS darstellt. Daher muss der Entwickler solcher Systeme zum einen eine auf jede Bildanwendung zugeschnittene Auswahl an Algorithmen zur Merkmalsextraktion und zur Ähnlichkeitssuche verwenden. Zum anderen müssen vom Entwickler eine für die Bildanwendung passende Softwarearchitektur, Datenstrukturen und Funktionskomponenten entworfen werden. Die Softwarearchitektur selbst unterscheidet sich bei den verschiedensten Anwendungen jedoch nur geringfügig. Die Entwicklung einer solchen Architektur lässt sich daher mit übergreifenden Mechanismen unterstützen, die die jeweils passende Kombination von entsprechenden Algorithmen zur Verfügung stellen. Die ersten Versuche, solche Mechanismen zu erstellen, basieren auf Software Frameworks und Quellcode-Bibliotheken. Diese Techniken bieten jedoch nicht genügend Flexibilität in Bezug auf die Plattformunabhängigkeit und die Ergebnisse sind keine maßgeschneiderten, kompakten Anwendungen, sondern vielmehr erweiterte Versionen der Frameworks.

In dieser Dissertation wird das Paradigma des modellgetriebenen Softwareentwurfs als Hilfstech­nik für die Erstellung von CBIRS angesetzt. Hierfür werden zwei Gruppen von Techniken verwendet: Modellierungs- und Transformationstechniken. Die Modellierungstechniken basieren auf einem konzeptuellen Frameworkmodell und werden für die Modellierung der Bildspeicherung, der Merkmalsextraktion und der Komponenten für die Ähnlichkeitssuche in CBIRS eingesetzt. Das Frameworkmodell definiert die generische Datenstruktur und Funktionalität für die Speicherung, Aktualisierung und Suche in Bildern. Dieses kann vom Entwickler eines CBIRS für den Entwurf eines eigenen, auf die Anwendung zugeschnittenen, konzeptuellen Modells verwendet werden. In dieser Arbeit werden weiterhin Transformations­techniken für die automatische Generierung von Implementierungen in Objektorientierten Datenbank-Management-Systemen (ORDBMS) aus dem konzeptuellen Modell erarbeitet. Diese werden in Form von Transformationsregeln zur Abbildung plattformunabhängiger Modell-Konzepte auf Konzepte der Zielplattform dargestellt. In dieser Arbeit werden Objektorientierte Datenbank-Management-Systeme (ORDBMS) als Implementierungsplattform gewählt. Diese Systeme bieten ausgereifte Mechanismen zum Einfügen und Aktualisieren persistenter Informationen, zum Beispiel durch das Transaktionsmanagement. Weiterhin sind ORDBMS eine der häufigsten Anwendungen zur Integration von Daten aus verschiedensten Quellen, beispielsweise Bilddaten von Medienservern und Textdaten aus Datenbanken oder von Fileservern. Dennoch kommen aufgrund der Tatsache, dass die konzeptuellen CBIRS Modelle plattformunabhängig sind, weitere Zielplattformen in Betracht. Für die jeweilige Zielplattform müssen dementsprechende Transformationsregeln definiert werden. Objektorientierten Datenbank-Management-Systemen wird gegenüber Relationalen Datenbanken der Vorzug gegeben, weil Relationale Datenbanken keine hinreichende Unterstützung für Multimedia-Daten bieten. Eine solche Unterstützung ist jedoch für integrative Anwendungen wie Digitale Bibliotheken, Multimedia-Informationssysteme usw. essentiell.

Für die Evaluierung der Transformationstechniken wird die Qualität der definierten Transformationsregeln überprüft. Die aus der Theorie des Datenbankentwurfs bekannten Anforderungen zur Kapazitätserhaltung der Transformation dienen als Kriterien für die erforderliche Qualität der Transformation. Die Untersuchung der Transformationsregeln ergibt, dass die Qualitätsanforderungen erfüllt sind.

Die entwickelten Modellierungstechniken werden beispielhaft für die konzeptuelle Modellierung eines CBIRS zur Speicherung digitaler Musikhandschriften und zur Identifizierung deren Schreiber angewendet. Weiterhin kommt eine CBIR Anwendung zur Ähnlichkeitssuche in Bildern aus der 2D-Gelelektrophorese zum Einsatz. Eine dritte CBIR Testanwendung dient der Speicherung und automatischen Annotation von Fotos. Diese drei Anwendungen zeigen, dass verschiedenste CBIRS aus dem generischen CBIRS Frameworkmodell abgeleitet und dass diese mit Hilfe der dargestellten Modellierungstechniken ohne Schwierigkeiten umgesetzt werden können.

# Acknowledgments

I would like to express my gratitude to all the people who believed in me and gave me their support during the long years of researching and writing on this dissertation.

I wish to thank my adviser Prof. Dr. Andreas Heuer, who opened opportunities for me by giving me the right hints at the right time, by encouraging me in taking challenges, and by teaching me not to doubt my ideas.

Prof. Dr. Heidrun Schumann has also supported me with very important hints for the representation of my ideas. I thank her for her encouraging remarks.

My colleagues and friends have been a great assistance by sparing their time for fruitful discussions, by sharing their experience with me, and by being always glad to help or simply listen. In particular, I would like to thank, Ilvio Bruder, Andreas Finger and Matthias Rust for the productive and pleasant cooperation on the topic of multimedia information systems development, Dr. Meike Klettke and Dr. Holger Meyer for their valuable comments on the written work, and Sebastian Schick for his assistance in the design of the implementation.

I would like to thank all the students, whose projects and theses were supervised by me, for giving their best to understand and implement my ideas and for helping me come up with new ones.

Prof. Dr. Susanne Boll and Prof. Dr. Klaus Meyer-Wegener played also a decisive role not only at the end as reviewers of my thesis, but also by showing me their interest in the topic during the elaboration of the thesis.

I thank my family for all the warmth and optimism, and finally, I would like to thank my dear Christian for so much understanding and love.





# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>Kurzfassung</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>v</b>
<b>Table of Contents</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Underlying Problems . . . . .	3
1.3 Existing Solutions for CBIRS Development Support . . . . .	4
1.3.1 Software Frameworks for CBIRSs . . . . .	4
1.3.2 Alternative Support for the Development of CBIRSs . . . . .	6
1.3.3 Conclusions . . . . .	6
1.4 Employing the MDSD Paradigm for the Development of CBIRSs . . . . .	7
1.4.1 Modeling Techniques . . . . .	7
1.4.2 Generation Techniques . . . . .	9
1.5 Structure of the Thesis . . . . .	9
<b>2 Basic Principles and State of the Art of Used Technologies</b>	<b>11</b>
2.1 CBIR Systems - State of the Art . . . . .	11
2.1.1 CBIR Systems Architecture . . . . .	13
2.1.2 Implementation Paradigms for CBIR Systems . . . . .	18
2.1.3 Building CBIR Systems on Top of Extendable DBMSs . . . . .	19
2.1.4 Summary of CBIRS Technologies . . . . .	21
2.2 Model-Driven Software Development - Basic Principles . . . . .	22
2.2.1 Characteristics and Aims . . . . .	22

2.2.2	Models for Model-Driven Software Development . . . . .	22
2.2.3	Model Transformations and Code Generation . . . . .	25
2.2.4	Summary of MDSO Technology . . . . .	29
<b>3</b>	<b>Requirements Analysis and Conceptual Design</b>	<b>31</b>
3.1	Domain-Specific Modeling for CBIRSSs . . . . .	31
3.1.1	Requirements for a Domain-Specific Model for CBIRSSs . . . . .	33
3.1.2	Requirements Towards the Quality of the Model . . . . .	40
3.1.3	Evaluation of Existing Conceptual Image Models . . . . .	42
3.1.4	Conclusions . . . . .	63
3.2	Transforming the CBIRS PIM to a PSM . . . . .	68
3.2.1	Choosing a Software Architecture and an Implementation Platform for CBIRSSs . . . . .	69
3.2.2	A Platform Specific Model for ORDBMSs . . . . .	70
3.2.3	Model-to-Model Transformation . . . . .	74
3.3	Summary . . . . .	81
<b>4</b>	<b>A Generic and Adaptable Conceptual Model for Image Retrieval</b>	<b>83</b>
4.1	The Modeling Approach . . . . .	84
4.1.1	Framework Model . . . . .	84
4.1.2	UML for Frameworks . . . . .	85
4.2	Modeling the Data Structure of CBIRS Components . . . . .	86
4.2.1	StillImage . . . . .	86
4.2.2	Metadata . . . . .	87
4.2.3	Region . . . . .	87
4.2.4	Feature . . . . .	88
4.2.5	Key Attributes and OIDs . . . . .	88
4.2.6	Application-specific Classes . . . . .	88
4.2.7	CBIRS Data Types . . . . .	89
4.2.8	Instantiating the Framework . . . . .	90
4.3	Modeling Functionality of CBIRS Components . . . . .	92
4.3.1	Updates . . . . .	93
4.3.2	Queries . . . . .	96
4.3.3	Modeling Retrieval Functionality . . . . .	96
4.3.4	Implicit Object Behavior . . . . .	103

---

4.3.5	Instantiating the Framework . . . . .	103
4.4	Summary . . . . .	104
<b>5</b>	<b>Mapping Rules for Generating CBIRSs on Top of ORDBMSs</b>	<b>105</b>
5.1	Modeling Deployment . . . . .	105
5.2	Meta Models . . . . .	106
5.2.1	PIM Meta Model . . . . .	106
5.2.2	PSM Meta Model . . . . .	110
5.3	Mapping PIM onto PSM . . . . .	113
5.3.1	Class . . . . .	114
5.3.2	Associations . . . . .	117
5.3.3	Dependency . . . . .	119
5.3.4	Generalization, GeneralizationSet . . . . .	120
5.3.5	Interface . . . . .	121
5.3.6	Package . . . . .	121
5.3.7	DataType . . . . .	122
5.3.8	Applying the Mapping Rules . . . . .	122
5.4	Quality of the Transformation . . . . .	123
5.4.1	Direct mappings . . . . .	123
5.4.2	Not-directly-mappable concepts . . . . .	123
5.4.3	Multiple mapping possibilities . . . . .	124
5.4.4	Mappings resulting in the same PSM Concept . . . . .	124
5.4.5	Implementation specific concepts . . . . .	129
5.5	Implementation of an Image Database Generator . . . . .	130
5.6	Summary . . . . .	132
<b>6</b>	<b>Evaluation</b>	<b>133</b>
6.1	Test Case eNoteHistory . . . . .	134
6.1.1	Requirements Analysis . . . . .	135
6.1.2	Modeling the eNoteHistory CBIRS . . . . .	140
6.2	Test Case 2D-Gel Electrophoresis Images . . . . .	143
6.2.1	Requirements Analysis . . . . .	143
6.2.2	Modeling the 2D-Gel Electrophoresis CBIRS . . . . .	146
6.3	Test Case Photo Album . . . . .	148
6.3.1	Requirements Analysis . . . . .	148

6.3.2	Modeling the Image Annotation Application . . . . .	149
6.4	Estimating the Gain From the Result of the Transformation . . . . .	151
6.4.1	Mapping Data Structure . . . . .	151
6.4.2	Mapping Functionality . . . . .	151
6.5	Summary . . . . .	154
<b>7</b>	<b>Conclusions and Prospective Research Directions</b>	<b>155</b>
	<b>Bibliography</b>	<b>161</b>
	<b>List of Figures</b>	<b>177</b>
	<b>List of Tables</b>	<b>181</b>
	<b>List of Abbreviations</b>	<b>183</b>
	<b>Appendix</b>	<b>185</b>
<b>A</b>	<b>PIM Metamodel</b>	<b>187</b>
<b>B</b>	<b>PSM Metamodel</b>	<b>207</b>
<b>C</b>	<b>Large Versions of Selected Figures</b>	<b>235</b>
<b>D</b>	<b>Screenshots of the Image Database Generator Plug-In</b>	<b>253</b>

# Chapter 1

## Introduction

### 1.1 Motivation

Still images are a popular form of recording visual information and are associated with a wide variety of human activities. In order to organize and manage the growing amount of image data, adequate techniques for its digital storage and retrieval are needed.

Personal photo collections, for example, are currently stored mostly directly in the file system and are accessed either through file system navigation tools, e.g. Microsoft Explorer, directly, or through specialized tools for browsing and organizing image data found in the file system, such as Google's Picasa. In order to be able to search for specific photos, e.g. for a presentation on the home entertainment system, or for the digital photo frame, the user usually in advance arranges the photos according to the date, place and time they were taken. However, if she would like to include only photos made at the beach in her presentation, it would take a lot of time to pick the right photos. For this access scenario, content-based image retrieval techniques, which can sort the images automatically in predefined classes, such as "beach photos", "mountain photos" etc., can be applied. In order to achieve this, content-based image retrieval systems make use of image processing techniques to extract information from the images, such as color, object shapes, texture information and apply machine learning algorithms to associate these so called features to semantic concepts, such as sea, sand, sky etc. The difficulties of applying these approaches are mainly the choice of the right combination of features, similarity measures, and data mining algorithm parameterization which can reflect the human perception of similarity and semantic concepts.

Apart from content-based data which can be extracted from raw image data, image context related data is produced simultaneously with the image capture. This data can also be used to arrange or classify the information accordingly. In the case of digital photos, these could be for example GPS coordinates and camera technical data, such as aperture and exposure time. Additionally, such context data can be associated to the image at a later point of time, for example by matching the GPS coordinates with geographical name tags or the time stamp with a personal calendar event. These data are currently stored in the image headers, which poses a lot of difficulties for its updating. For example, there are no mechanisms in this case which can assure data integrity when updating the GPS coordinates or the geographical name tags. This problem is even more significant for images produced as results in the area of experimental sciences. The devices producing these images usually generate a lot of additional

experiment data, which do not even fit into the image header, but has to be saved in separate files. In order to integrate these data, to assure consistent update operations, and provide secure access to sensitive data, database management system techniques have to be used.

There are a number of other reasons for turning to databases when developing content-based image retrieval systems (CBIRSs), addressed in [VC02, AJO04]. However, existing database management systems (DBMSs) were neither originally designed to store complex data types such as images, nor could these systems foster content-based image retrieval. Proposed DBMS extensions (IBM DB2 Extenders, Oracle Options, Informix Datablades) and standards (SQL2003, SQL/MM) consider only a rudimentary set of functions for content extraction, content representation possibilities, similarity measures and indexing mechanisms. The customization or adaptation of these extensions for specialized CBIR applications is also limited. Therefore, in order to solve real-life CBIR problems each time a CBIR application is needed a new extension has to be developed.

The attempts to build universal CBIR applications have failed mostly because each specific domain requires adequate adaptation of the system components. This problem has been tackled on an implementation basis by offering adaptable software architectures, such as CBIR frameworks (e.g. GIFT [MSMP99], PicSOM [LKO02], VizIR [EB03], GRAID [GY00]), which can be extended and configured to support different CBIR applications. This approach has the disadvantage that the resulting applications are not tailored to the specific problem but contain much more functionality than needed and that the frameworks limit the implementation to a specific platform and software architecture.

The aim of this thesis is to create a more efficient development approach for CBIR applications, which builds upon the fact that CBIR applications use similar components that have to be adapted for specific applications. The proposed approach allows the adaptation of the components on a conceptual design level so that the resulting applications can be deployed on different platforms and system architectures. To achieve this aim, techniques for model-driven software development are adopted. First, an adaptable generic model is defined to allow the adaptation of the CBIR application on a conceptual design level. Second, the mapping of this model onto an ORDBMS Schema is automated, with the aim to provide assistance for the CBIR system developers to generate a platform-specific implementation of the conceptual application.

This approach can be very useful for building specialized CBIR applications, such as scientific image databases, which use particular image characteristics (features) and domain knowledge in order to derive new information or analyze the content of large image sets. The development of such systems requires the incorporation of domain-specific data and corresponding retrieval mechanisms. Therefore, universal CBIR systems cannot be used for these applications and new applications have to be implemented. The development of multidisciplinary applications, such as digital libraries or multimedia information systems, where images originate from various domains is also a target application field for the developed approach. In these applications, different collections of images have to be managed in different ways, which requires appropriate methods for their retrieval to be developed and implemented. Therefore, a tailor-made application has to be implemented and then integrated as part of the digital library or multimedia information system for each image collection.

## 1.2 Underlying Problems

The model-driven methodology for developing CBIRS represented in this thesis is elaborated systematically by analyzing and elaborating each of the underlying problems. These problems can be summaries in four question outlined in the following paragraphs.

Current research and development in the field of multimedia content-based retrieval systems focuses on integrating new aspects in such systems in order to improve their effectiveness as described in [LZLM07]. Low-level features, directly derived from the images, have turned out to be insufficient to solve the problem of retrieving or classifying images by their content. There are two main approaches to resolving this issue. On one side there is the endeavor to consider as much external information for the images as possible. Examples of this include the association of context data to images coming from camera devices, and associating textual descriptions from multimedia documents by identifying text related to the image. On the other side the aim is to bridge the semantic gap by deriving more meaningful information from the low-level features through applying machine learning techniques, using ontologies etc. The integration of these new aspects requires additional components to be implemented and interchanged in CBIR Systems. This leads to difficulties in determining a universal CBIR architecture. In order to develop a framework model for CBIR applications generic components have to be defined that can be adapted for a particular domain problem. The main question to which an answer is sought is “What should be modeled?”, i.e what are the components of a CBIR system, and which of these can be modeled conceptually?

At the same time, the different possibilities for combining the components of a CBIR system have to be investigated. These have to be considered when choosing a software architecture for the application and higher-level platforms. Thereby, an answer to the question “What should be generated?” is sought. For example, a choice between a two-tiered, three-tiered or distributed software architecture has to be made. This decision influences the answer to the question “What should be modeled?” because the software architecture also determines the system components which have to be modeled. For example, the choice of a persistence layer based on an ORDBMS would suggest that the CBIR System model does not have to additionally include a persistent management component.

After the components and the software architecture for the CBIR application are determined, a suitable modeling language has to be found for representing the concepts of a CBIR systems. A domain-specific modeling language can be evolved for this particular purpose or an existing CBIR model or universal modeling paradigm can be applied. Therefore, the expressive power of modeling languages has to be investigated based on the criteria for modeling CBIR applications. In this case, an answer to the question “How should it be modeled?” is sought. The modeling language should also allow the adaptation and extension of the conceptual design by the developer.

The final question which remains to be answered is “How should it be generated?”. In this case techniques for the mapping of the conceptual model onto an implementation platform have to be investigated and applied. Mapping rules for the chosen model language and platform language have to be determined. The preservation of the information capacity during mapping as well as other transformation characteristics such as bidirectionality, traceability etc. have to be considered. The application of these rules has to be automated as far as possible.

These questions are treated in detail later in this thesis.



### 1.3 Existing Solutions for CBIRS Development Support

Despite the large number of current CBIRSs, there is no universal solution which can handle different application domains. Generic CBIR Systems (e.g., QBIC [FBF<sup>+</sup>94], imgSeek [img06], IMatch [IMa06]) make use of generic low-level features such as color, texture and shape, which do not always represent the visual information used to estimate the similarity of images for a specific application. Specialized CBIR Systems, such as a system for recognizing similar images in a set of 2D-Electrophoresis Gel Images [AHF<sup>+</sup>88], identifying peoples' faces [JV03] or trademark recognition [Lew01] are implemented only for a certain application domain and are normally highly effective for this domain, but cannot be applied effectively in any other applications. Such specialized applications have to be designed to meet the specific application requirements. The first attempts to facilitate this task have been built as software frameworks (e.g., GIFT [MSMP99], VizIR [Viz06], PicSOM [LKO02], GRAID [GY00], CBIRFrame [CI03]). They offer extensible software architectures for developing domain-specific CBIR applications. The idea of these frameworks is to allow flexible addition or removal of features to the system, feature extraction functions or retrieval functions or using different back-end systems. Some of these frameworks, such as PicSOM and VizIR were developed as research frameworks in order to be able to test different retrieval algorithms, and not with the intention of making them available to the public to help building their own applications. However, they can be regarded as the first steps towards supporting the development of specialized CBIR systems.

The CBIR frameworks about which information could be found during the time in which this thesis is written are summarized below. This overview focuses on how these frameworks can be used and adapted to implement domain specific CBIR applications more easily. A detailed analysis of the frameworks is represented in [Thi08].

#### 1.3.1 Software Frameworks for CBIRSs

**GIFT** [MSMP99] (GNU Image-Finding Tool) is an open framework for content based image retrieval distributed under the GNU GPL. It has been developed by the Computer Vision Group of the University of Geneva and is under continuous improvement. It is implemented in C++ and uses MRML (Multimedia Retrieval Markup Language) for the communication between the CBIR clients and servers. It supports the extraction of local, global, simple color and texture features. These features are indexed in an inverted file. The system also supports relevance feedback techniques. For retrieval it uses separate normalization and classical inverse document frequency. The back-end used for storing the images and features is the file system.

GIFT is built around a kernel part and a collection of plug-ins. At the moment only the Viper plug-in is integrated which implements the currently available CBIR functionality. Developers can create their own plug-ins with the feature extraction and retrieval functionality they require. Detailed documentation for writing plug-ins is, however, not available for this project, so the developers have to learn about it through the source code of the application.

**VizIR** [EB03] (Visual Information Retrieval) is a framework of resources (mainly software components implemented in Java) that are needed to build visual information retrieval prototypes. The components include classes for media access, transportation and visualization,

for feature extraction (MPEG-7 descriptors), for querying and refinement based on a 3D retrieval and browsing panel, user-interface design and visualization of media data, evaluation and benchmarking. A data management layer for the feature vectors is used to manage the multidimensional data efficiently. The database management is based on object-oriented persistence management. The object/relational mapping to a specific DBMS is implemented with Hibernate.

The components of the framework can be interchanged freely. Each of them provides an extensible structure, which can be adapted for a specific application. For example, the query component of the framework provides a class for the formulation of the query from an XML string, which can contain an arbitrary number of parameters. This class should be adapted to support different query types. The corresponding query engine, which processes the query, must implement the interface `QueryEngine` of the framework and the methods for preparing the query and executing it. The result of the query processing can be passed to a specialization of the `QueryResult` class, which will take care of preparing the results for the delivery to the user-interface, for example. The other components of the framework are, however, not so well documented. Thus, adding new features to the framework would require more profound analysis of the source code. The communication between the framework components is implemented using the MRML communication protocol developed by the GIFT project team.

**GRAID [GY00]** (General Purpose Architecture for Image Databases) focuses on the implementation of a middleware layer on top of a DBMS which provides an infrastructure for constructing image retrieval applications. The architecture does not consider the conceptual design of the image database. Only the retrieval flow of an image database is part of the framework.

A simple implementation of the GRAID framework, named WIRED (Web-based Image Retrieval on Databases), is developed in Java. GRAID constitutes components such as Query Processor, DBMS Mediator, Image processing Pool, Image Interpreter. Additional developer tools for adapting these components are provided in WIRED:

- Configuration interface for adding new image processing algorithms to the pool
- Image-processing Pool interface for choosing available image-processing algorithms for an application
- Application-retrieval interface with tools for setting up the retrieval application according to the GRAID programming approach.

The query language used in the resulting application has to be defined by the developer. The WIRED implementation of GRAID is, however, not available as source code and thus cannot be evaluated.

**CBIRFrame [CI03]** (Content-based Image Retrieval Framework) is designed as an object-oriented framework which can be used as a basis for developing CBIR applications. It is based on C++ template classes and a five-tier architecture, which should allow the designed system to run in a distributed environment and make the components of the framework independent from each other. The authors of the framework define the following adaptable system components:

- domain component comprising of modules for feature extraction and image construction;
- user-interface component;
- data management component;
- user-interface facade and finally a data management facade.

In this way, the authors claim to provide the possibility to implement platform-independent domain components or integrate existing image databases in order to provide portability and interoperability, respectively.

In order to build a CBIR application using this framework, a developer has to subclass the template classes and provide implementations for the virtual functions. The parameters of the template classes also have to be specified. The framework provides some basic data types, such as color and container class of pixels, in order to support the developer. [CI03] describes the classes that have to be subclassed in order to add feature extraction or similarity measure functions. The source code of CBIRframe is not available to the public and, therefore, this framework cannot be evaluated completely.

### 1.3.2 Alternative Support for the Development of CBIRSs

**MetaXa** [BSST07] (Content-based and Context-driven metadata enhancement architecture) is described as a component-based architecture for the automatic feature extraction and meta data enhancement of photo collections. The feature extraction and enhancement algorithms are implemented as plug-ins in MetaXa. Depending on the domain of the photo collection different plug-ins can be used through a predefined workflow to extract the needed information from the images and their context data. The images and their features can be stored in the file system or in an Oracle DBMS. The retrieval components are, however, not part of this architecture. The developer has to build an own retrieval engine or use the query mechanisms of the DBMS to query the extracted features.

**Virage Image Search Engine**<sup>1</sup> [BFG<sup>+</sup>96] is a commercial CBIR engine which encapsulates the retrieval functionality of a CBIR system. It can be adapted to a specific application and integrated as a retrieval component into a CBIR system.

**LIRE**<sup>2</sup> (Lucene Image REtrieval) is a reusable CBIR source code library available under the GNU GPL license. It can be integrated into Java CBIR projects to extract features and create and search indexes of these features. This development aid, however, does not determine the overall system architecture of the CBIRS.

### 1.3.3 Conclusions

It is delusional to imagine a system that will meet the requirements of all possible domains of image retrieval. This is also not the aim when developing specialized applications. These have to be made suitable, effective, and efficient only for a particular domain. Therefore,

<sup>1</sup><http://www.virage.com/home/index.en.html>

<sup>2</sup><http://www.semanticmetadata.net/lire/>

developing new applications is an inevitable step towards effective software. The question is how to make this development easier so that it does not have to be done always from scratch. Most of the reviewed CBIR frameworks have well structured modular architectures, at least as far as it can be judged from their documentation, which intrinsically allow relatively easy extensions of the source code and exchange of components. Additional documentation about creating tailor-made applications, such as the one provided by VizIR enables even more efficient use of the frameworks. However, these frameworks are implemented for specific platforms and do not offer flexible data storage possibilities. This disadvantage is common for software frameworks in general [FSJ99]. Furthermore, regardless of the fact that only a part of their functionality is needed for a specific domain it is difficult to adapt the applications to contain only the required parts. The result of adapting these frameworks for a specific domain application is not a compact specialized application, but rather an extended version of the large framework application. Additionally, most of the frameworks are only conceptually described and no open source implementation is provided in order to practically apply them. The alternative development support solutions either consider only part of the CBIRS components, e.g. feature extraction in MetaXA, or provide a reusable code library without a system architecture design. These alternatives are also developed for concrete platforms. The combination of source code libraries with reusable framework architectures should be considered in future developments.

In this thesis, a new method for development support, which can abstract from the concrete platform, allow the generation of compact, tailor-made applications and combine the reuse of system architecture and algorithms, is proposed.

## 1.4 Employing the MDSD Paradigm for the Development of CBIRSs

Applying model-driven development techniques for generating CBIR Systems has not yet been considered as a means of providing a development support for CBIRSs. The recognition of the fact that CBIR systems share a common architectural design but require different functionality of the building blocks depending on the particular application domain has led to the development of so called CBIR-Frameworks which offer extensibility mechanisms at the source code level for adapting to the domain needs. In this thesis, the new idea of moving the adaptation processes to the modeling level and automation of the source code creation, by applying model-driven software development techniques, is investigated. The aims pursued by applying this development approach for CBIR systems are to enable a platform independent application design, domain-tailored applications and reuse of a system architecture design.

Therefore, two groups of MDSD techniques are designed for the model-driven development of CBIRSs as shown in Figure 1.1, modeling techniques for supporting the conceptual design of CBIRSs and generation techniques for mapping the conceptual model onto a specific implementation platform.

### 1.4.1 Modeling Techniques

These techniques have to support the developer in creating a platform independent model (PIM) for a specialized CBIR system. Therefore, a generic and adaptable model for CBIRSs

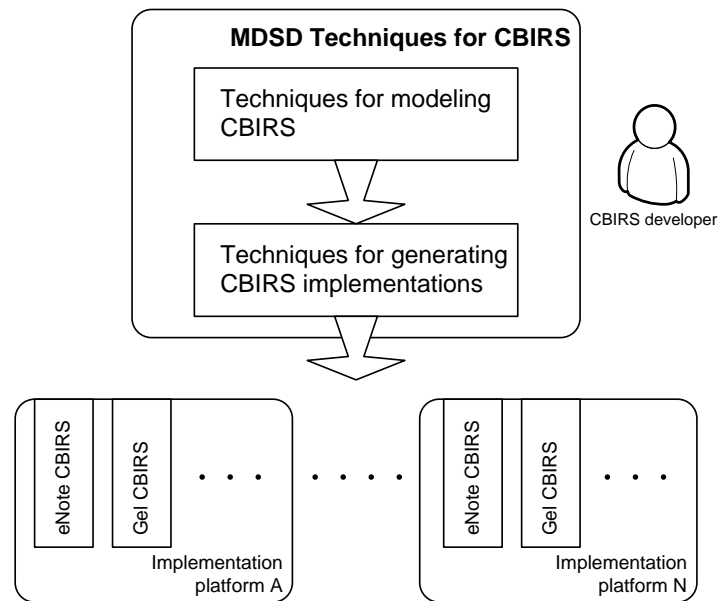


Figure 1.1: MDS techniques for the development of CBIRSs

- GiACoMo - is proposed. GiACoMo defines the generic functionality and classes of a CBIRS. It is defined as an instance of an extended UML class diagram meta model. The extensions of the UML class diagram meta model comprise mainly framework model stereotypes and data types for the CBIR application domain which can be used to derive application-specific CBIRS PIMs. The proposed modeling approach for CBIRSs is published in [IB05] for modeling multimedia documents, where images are only one kind of media. In [Ign06] preliminary ideas for the application of this approach for the modeling of CBIRSs are represented.

Due to the big variety of application fields that have been addressed by content-based image retrieval, such as medicine, biology, astronomy and geology etc. it is not trivial to abstract the generic components of a CBIRS. Each of these fields require quite different approaches to solve the domain-specific retrieval problems. In the current work, the generic components of CBIRSs have been defined based on generic CBIRS architectures, found in the literature [MF02, VC02, VT02, RHC99, Bim99]. For the detailed description of the structure and functionality of the generic components, in Chapter 3, an example application in the field of musicology is used. This reference application is applied for the recognition of scribes in historical music manuscripts, based on the visual characteristics of their handwritings. Due to the complexity of the features used for this retrieval task, and the two different retrieval mechanisms developed to solve the retrieval task, this application sets a good example for formulating the requirements of a generic CBIRS model. A system for solving this image retrieval problem was implemented as part of the efforts in the cross-disciplinary project eNoteHistory<sup>3</sup>. The eNoteHistory application is described in more detail in Chapter 6. The conceptual design and implementation issues for the CBIRS functionality in eNoteHistory are published in [BFHI03, BIM04].

<sup>3</sup>[www.enotehistory.de](http://www.enotehistory.de)

### 1.4.2 Generation Techniques

These techniques comprise mainly the mapping rules for the automatic transformation of the CBIRS PIM into a platform specific model (PSM). The mapping rules depend on the choice of a software architecture and an implementation platform for the application. In this work, the implementation of CBIRSs on top of ORDBMSs is taken into consideration for developing mapping rules. However, other platforms can also be used for the implementation of CBIRSs. Each platform requires the definition of new mapping rules. Some preliminary considerations for mapping the CBIR PIM onto the object-relational model are published in [Ign06]. Two existing approaches for mapping UML class diagrams onto ORDBMSs are compared in [Spi06]. In the same work, they are used for transforming a conceptual multimedia data model, defined in [IB05], onto an ORDBMS for building a multimedia document archive for digital libraries. In this thesis, these and some additional transformation approaches are evaluated. The informal mapping approaches described in [CT06, VVCM07, DU04] are used and extended to define the mapping rules for the CBIRS PIM. Requirements towards the transformation rules are defined to assure that the resulting target model (PSM) preserves the information capacity of the source model (PIM). The formulated rules are then evaluated against these quality requirements.

Not all the platform specific concepts can be modeled in the PIM. Therefore, either default values have to be considered during the transformation or the transformation has to make the decisions by interacting with the developer. In any case, it is desirable that the result of the transformation can be adapted by the developer in a modeling environment. Therefore, the generation techniques designed in this thesis provide also a UML-Profile for representing the object-relational concepts of SQL:2003 required to build a PSM of a CBIRS. This profile is designed as an extended compilation of existing SQL profiles [Rat03, Amb03, VVCM07].

The final task of the generation techniques is to interpret the platform-specific models and automatically create a platform-specific implementation of the model, e.g., for a specific database system SQL dialect. Therefore, some of the discrepancies between the SQL standard and the different DBMS implementations have to be considered. In [Czy05, Spi06] these differences are described for the IBM DB2 DBMS. In [VVCM07] the mapping rules are defined for the Oracle DBMS. Therefore, in this thesis, this problem is not further discussed.

## 1.5 Structure of the Thesis

The remainder of this thesis elaborates the idea of providing a more efficient development approach for building CBIRSs on top of ORDBMSs, and is structured as follows.

In **Chapter 2**, basics and current state of the art of the two technology fields blended in this thesis are described. In the first half of this chapter, the current state of the art of CBIR systems is discussed. Architectural aspects and different implementation paradigms for CBIRSs are also reviewed. The possibilities of using Database Management Systems as an implementation are considered. Challenges and advantages of using ORDBMSs as an implementation platform are discussed. Finally, the reasoning behind choosing an ORDBMS as a target platform in this thesis is given. The second half of this chapter outlines the basic principles of the model-driven software development (MDS) process. It represents the main concepts of MDS, such as meta modeling, platform independent and platform specific models, and transformation techniques.

In **Chapter 3**, the design of the two groups of MDS techniques is described. For the modeling techniques different ways of representing domain-specific CBIRS models are compared, and the requirements for a generic and adaptable image retrieval model are set. Furthermore, the functionality which has to be provided by the transformation techniques is discussed in detail. Decisions about the target architecture and platform for CBIRSs are made based on this discussion. The requirements for a platform specific model for ORDBMSs and transformation approaches are defined.

**Chapter 4** introduces a new Generic and Adaptable Conceptual Model for Image Retrieval Systems (GiACoMo-IRS), which can be used as a starting point and a conceptual framework for building domain-specific CBIR systems. The structural, as well as the functional aspects of modeling CBIR systems are abstracted in this generic model. Adaptability and extensibility concepts are introduced in order to provide the possibility to derive a tailor-made CBIR model from GiACoMo-IRS.

**Chapter 5** defines transformation rules for generating a platform specific implementation for the conceptual model. Furthermore, the quality of a transformation based on these transformation rules is analyzed. In particular, the ability of the transformation to preserve the information capacity of the conceptual model in the platform-specific model is evaluated.

In **Chapter 6**, the model-driven development techniques elaborated in the previous two chapters are applied for three test case applications. The eNoteHistory CBIR module for identifying scribes of historical manuscripts, introduced in this chapter, a CBIR system for the similarity-based retrieval of 2D-Gel electrophoresis images and a Photo annotation application are modeled using the GiACoMo-IRS framework model. Additionally, in this chapter, the effort which has to be performed by a developer of CBIRSs using these techniques is discussed.

**Chapter 7** concludes with a summary of the achievements of the thesis and suggests directions for further research.

## Chapter 2

# Basic Principles and State of the Art of Used Technologies

### 2.1 CBIR Systems - State of the Art

Content-based image retrieval systems are developed in order to support browsing, searching, classification, identification etc. of information represented as digital images. The information contained in digital images, which can be perceived by an observer is also referred to as visual content. Visual content can be represented in different ways, based on different levels of abstraction or depending on the point of view or background knowledge of the observer. A grouping of visual content representations according to their level of abstraction is shown in Figure 2.1. Pixel level characteristics of an image such as color, texture, and shape represent visual content which is most of the time similarly estimated by different observers. These kinds of characteristics are also called low-level features of an image. A question of subjective assessment, however, is visual content represented by more abstract characteristics such as semantic concepts and objects. The interpretation and recognition of such content reflects the more sophisticated knowledge and reasoning of the observer. Therefore, these groups of characteristics are also called high-level features.

One of the main aims of a CBIR system is to make human interpretation of visual content understandable for a machine. Therefore, a common language for describing visual content is sought. The direct approach for a person is to annotate images manually or semi automatically using text, since it is the natural way for people to express their information needs. The machine can subsequently apply text retrieval techniques to process queries. The problem with this approach is the low accuracy of the descriptions due to subjective perception. Therefore, the development of automatic annotation techniques was pursued. During the early years of CBIR, which are surveyed in studies like [RHC99], [SWS<sup>+</sup>00] continuous efforts in the content-based image retrieval community have focused on the development of techniques for representing the content of an image with high precision. As a result algorithms for the automatic extraction of pixel level image characteristics have been developed. These algorithms can achieve a very precise description of the low-level features in terms of complex numerical values. Retrieval techniques for image features, such as histogram methods, central moments etc. have been developed, most often using text retrieval techniques as a basis.

Since people cannot limit themselves only to the notion of color, texture or shape, when



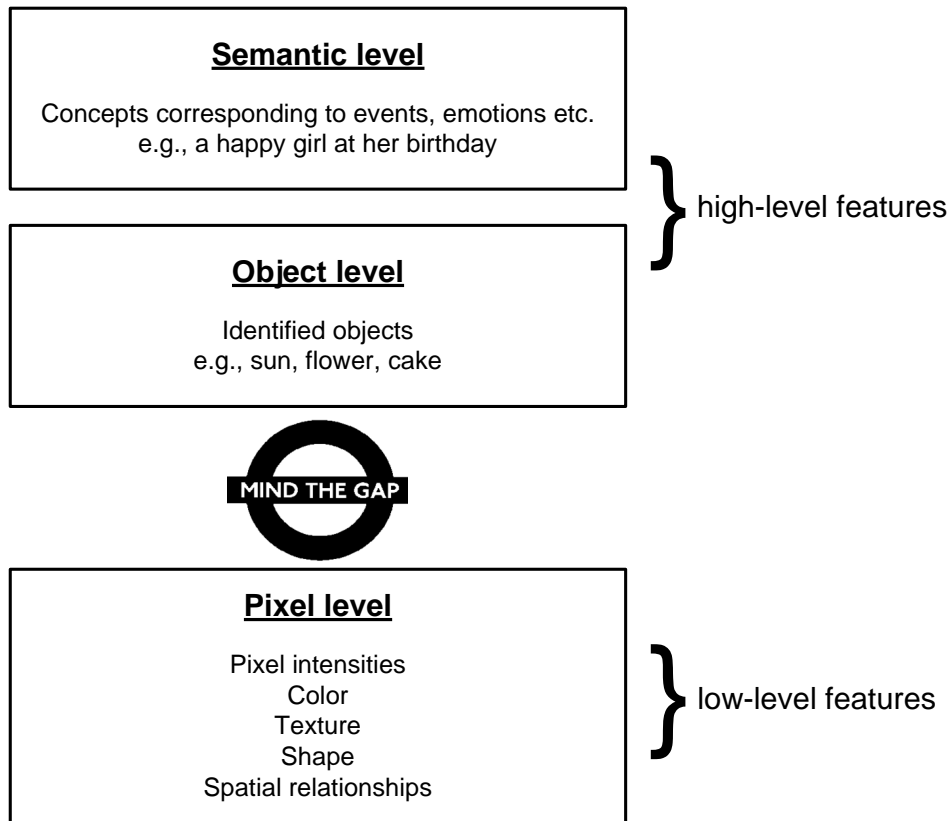


Figure 2.1: Levels of abstraction of visual content

searching for visual information, an automatic annotation on a higher-level of abstraction which reflects the image semantics has to be provided. This problem has received the name "semantic gap" and is the focus of numerous works in progress in different scientific fields, such as computer vision, machine learning, information retrieval, databases etc. In each of these fields the problem of the semantic gap is handled from a different perspective, either trying to derive as much as possible information from the image itself, to associate content-independent (context) data or create a domain knowledge representation and map the low-level image content to this domain knowledge. The latest results in these research directions have been summarized in [LSDJ06] and [LZLM07]. Techniques such as ontologies, relevance feedback, classification, ranking etc. have thus become an integral part of new generation CBIR Systems.

Currently, there is a large variety of visual information descriptors, retrieval measures, and techniques for bridging the semantic gap. However, bringing all these together in one application does not guarantee to lead to a universal application which can solve all domain-specific problems. A domain-specific problem requires normally only a small well chosen subset of these techniques in order to effectively fulfill the requirements of the application. Nevertheless, CBIR applications consist of similar basic components corresponding to one or more groups of techniques. In this chapter, basic architectural components of CBIR systems are identified and their realization for different applications are discussed.

### 2.1.1 CBIR Systems Architecture

CBIR systems often consist of similar basic building blocks which implement the required functionality. Almost each survey or introduction chapter on CBIR systems begins with a typical architecture of a CBIR system and a description of its components. In Chapter 2 of [MF02] a generic CBIR system comprises:

- user interfaces for querying the database and browsing and viewing the results;
- a collection of algorithms to perform the database search as a search engine;
- a digital image archive to store the images;
- visual summaries to represent the image content in a concise way;
- indexes to access visual summaries more efficiently;
- a component for the digitalization and compression of images;
- a cataloging component for extracting and indexing the features from the images.

Other representations of CBIR architectures can be found in Chapter 5 of [MF02] and in [VC02, VT02, RHC99, Bim99]. These consist of similar system components and claim to cover a broad range of CBIR applications. This fact leads to the conclusion that for a large class of CBIR systems these components are the same. In this chapter, an integrated view of a generic CBIR architecture is introduced and different approaches for the design and implementation of its components are discussed. The main components of a CBIR system and the interfaces between them are illustrated in Figure 2.2. This integrated view of CBIR architecture represents a class of CBIR systems which should be modeled and generated by the model-driven development techniques proposed in this thesis. It is used in the following chapter to identify the components of a CBIR system which can be modeled.

The processes which take place in such a CBIR system through the interaction of the components realize the adequate pre-processing, feature extraction and indexing of the data related to the image, which is depicted with the dotted arrows in Figure 2.2. Furthermore, the interaction with the user for the query formulation and the representation and exploration of the results is realized, which is depicted with the continuous arrows in Figure 2.2. The implementation of these generic components can, however, differ quite a lot depending on the chosen communication pattern, respectively the software architecture, the application domain, and the needs of the users.

**User Interfaces** in the diagram on Figure 2.2 represent the main functions of a CBIRS available to the user. These blocks can be realized by specific graphical user interfaces and input mechanisms for loading images. The setup of these user interfaces largely depends on the available underlying functionality in the CBIR system, but they can also implement additional functionality to support a more sophisticated interaction with the system. Many types of CBIR user interfaces exist, depending on factors such as system requirements, device requirements (e.g., support for mobile, ubiquitous devices), and user requirements (e.g., differentiate between expert and non-expert users). Functionality aspects such as visualization and exploration of the results, multimodal query formulation, interactive query enhancement,

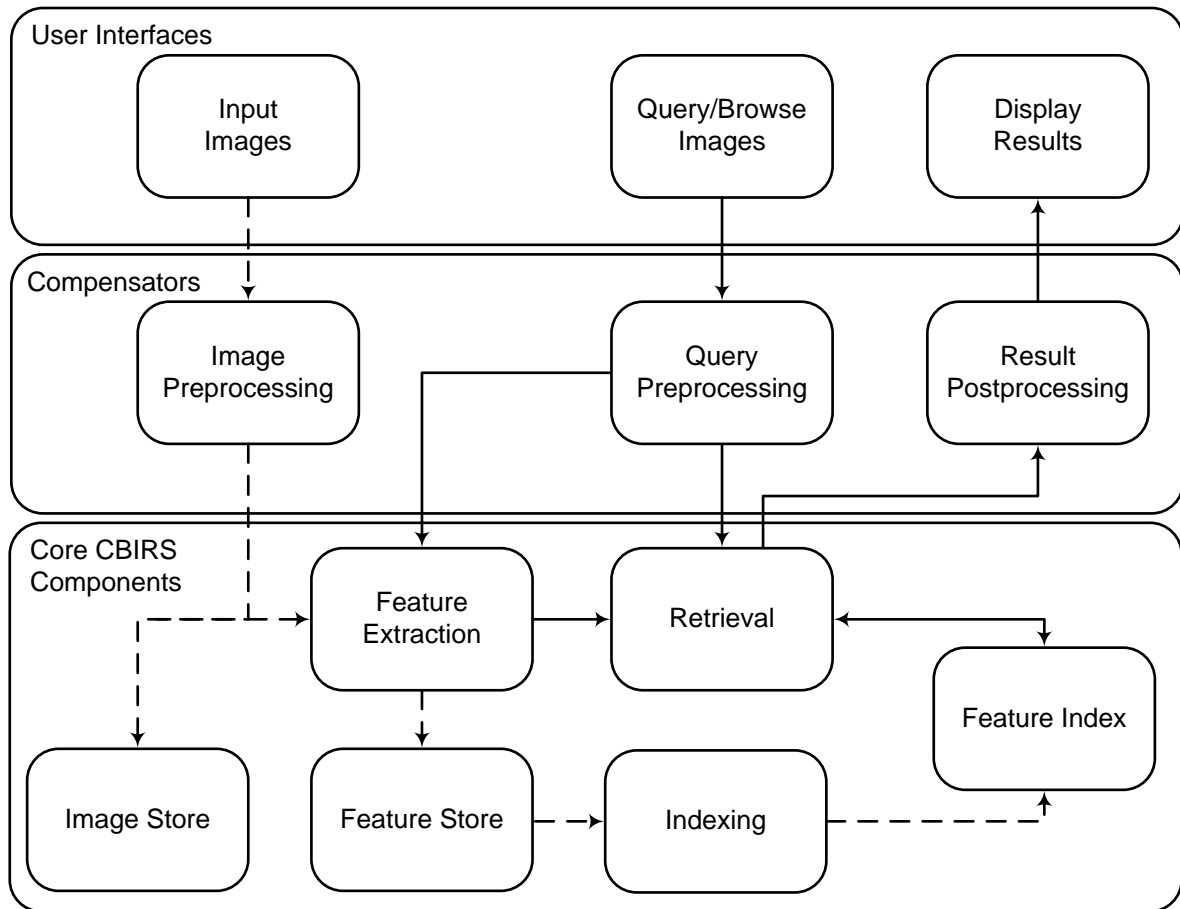


Figure 2.2: An integrated view of a CBIR system

adjustment and advanced user query formulation play an important role in building CBIR user interfaces.

A lot of work in the area of CBIR explicitly focuses on the development of user interfaces for CBIR. Techniques for the exploration of large query result sets or classifications have been proposed in [TSMR03, JWY<sup>+</sup>06, NMH03]. How to allow interactive query formulation and refinement through relevance feedback is discussed in [LPS<sup>+</sup>04]. Support for multimodal query formulation techniques has been studied in [BKPS03]. How to build such interfaces for mobile, web-based retrieval applications is discussed in [YTD04]. Usability is an important criterion in designing such interfaces. It is, however, not yet fully investigated for CBIRS user interfaces. In [vdBKV04] the usability of color selection interfaces in particular in CBIR is studied. Which kinds of user interfaces are suitable for particular kinds of CBIRs currently remains an open question.

**Compensators** represented by green blocks in Figure 2.2. They represent additional compensation or enhancement functionality of the CBIRS. These components can offer the end-user a better mapping of the user's query to the system's representation or vice versa. Furthermore, during the storage of the image data compression or enhancement functions can allow better integration of the images into the system. Thus, these components can be regarded as

an enhancement of both the user-interface and of the core system functionality.

**The Core CBIR components** can be grouped in storage and retrieval components. These components can vary depending on the domain of the application. Their implementation is discussed in more depth in the following sections.

### 2.1.1.1 Components for Extracting and Storing Image Content

This group of components fulfills tasks concerned with making the images persistent as well as extracting and persistently storing their content representations. The raw images are used mostly only as input and output of the retrieval process, but not during the retrieval process itself. However, a CBIRS should provide ways to store these raw images. Usually, specialized devices, such as optical discs are needed to physically store these data. Apart from the images themselves image properties and content abstraction need to be extracted from the raw images and stored in order to provide efficient ways for image retrieval. Which kinds of image characteristics should be extracted depends on the application domain and the retrieval task which has to be served. Color, texture and shape characteristics are used in almost any general purpose CBIR system, such as QBIC, Photobook, Virage, MARS, SIMPLicity, and VisualSEEK. Such systems can be used to help compare images from different domains according to some low-level characteristics, but cannot be applied to effectively solve particular search scenarios. Special purpose features, or purpose-oriented combination of generic features have been defined for particular applications, such as trademark recognition [Lew01]. Face recognition applications for example require specialized features, e.g., rectangle features [JV03] to compare images. A special kind of a feature - edge orientation autocorrelogram - is used to compare images in a patent database in [TB04].

Additionally, there is the content-independent data, sometimes referred to as metadata of an image, which cannot be derived from the visual characteristics of an image. Metadata is defined as the data describing the image, independent of the content. Some standards have suggested sets of metadata for describing images.

- Dublin Core Image Metadata: defines image metadata which can be used for retrieval, based on bibliographic data and document properties.
- ANSI/NISO Z39.87–2006 “Data Dictionary – Technical Metadata for Digital Still Images”, Approved December 18, 2006: these metadata can be used to describe technical properties of images.

Metadata are an important asset for content-based image retrieval because they can play a decisive role in the classification of images by helping the machine learning mechanisms to improve the final result. For example if a CBIR system has to decide if a photo of a sun above the ocean is a sunset or a sunrise, the timestamp or the location of the photo would make the decision more precise.

**“Image Store” and “Feature Store” Components** How to represent these image characteristics in order to store them for retrieval is the first question that arises when implementing the “Image Store” and “Feature Store” storage components. There are different classifications and taxonomies of image content used for image retrieval. A common view is

the grouping of different content characteristics in different levels of abstraction starting with the physical, through logical or structural to the semantic level of content data as shown in Figure 2.1. For the implementation of a CBIRS, however, a data model for storing such kinds of data is required. Since each CBIRS stores a different set of features, it is difficult to define a generic data model that can be applied for any arbitrary CBIR system. Existing works dealing with this problem are compared in Chapter 3. One of the most prominent examples of such a model is the MPEG-7 Visual Features<sup>1</sup>.

Another question that arises during the implementation of the storage components is the choice of a storage mechanism. Whereas some CBIR systems rely on the usage of relational or object-relational (e.g. QBIC, Virage, VizIR, Chabot) and even native XML database management systems (some techniques are presented in [Li05, WK03]) as storage mechanisms, there are others which prefer managing the data in proprietary databases (e.g. IMatch) or directly in the file system. The choice of a storage mechanism also depends to some extent on the model used to represent the data. For example, the MPEG-7 model would suggest using a native XML database for the storage of the XML descriptors, since it is based on an XML-Schema.

**“Feature Extraction” Component** The component for extracting the image related data also belongs to the group of storage components. Although this component is called simply “Feature Extraction” its realization can be very complex, for example if it is expected to support the automatic recognition of objects and scene description, i.e. to derive high-level features from digital images. In other domains it would be the task of the “Feature Extraction” component to only automatically determine the average color of an image or even just extract the EXIF header information from the image. Some feature extraction mechanism use machine learning techniques to classify the images in previously defined concepts, based on their perceptual characteristics (support vector machines, hidden markov models, neuronal networks). Others try to combine image and text descriptors, by using a training set of previously annotated images to achieve automatic annotation of images [DDL<sup>+</sup>90, ZG02, MGP03]. Finally, to make the a priori knowledge used by people for interpreting image content available to machines, different knowledge data structures can be used (ontologies, thesaurus, term lists). These approaches allow the representation and comparison (based on predefined rules) of relatively well structured content. These methods are especially used in scientific applications. However, for the correct assignment of concepts manual interaction is needed.

### 2.1.1.2 Components for Retrieving Images based on their Content

From an architectural point of view (see Figure 2.2) there are basically two ways to query the images by their content. Either querying by example - a query image is received which should be analyzed to extract the needed content information such as features, and then the content data would be compared against the content of a previously analyzed set of images. Or querying by content directly - where a query contains an explicit set of features that has to be compared against the features of the previously analyzed images. If query by example retrieval is chosen the first step is to extract the features from the image, using the “Feature Extraction” component from the storage group and then send the feature data to the “Retrieval” component.

---

<sup>1</sup><http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm>

**“Retrieval” Component** The component responsible for the processing of these queries is the “Retrieval” component. How a query should be processed depends on one hand on the types of features, based on which images have to be compared with each other. On the other hand different retrieval tasks require different retrieval mechanisms to be applied. For example, the task “find all images from a database similar to a given image by color histogram” would require the usage of a metric, such as the Euclidean or Manhattan to compare the color histograms of the images. The choice of the right metric to use is, however, a difficult task since the metric has to correspond to the human perception of similarity. This choice becomes even more difficult when more than one feature has to be considered in the comparison. In this case, in addition to the metrics used to compare the single features aggregation functions have to be also applied to combine the results from the single metrics meaningfully. In order to solve the task of the type “identify the person on the photo” the application of machine learning techniques, such as artificial neuronal networks, support vector machines etc., may be more appropriate than similarity metrics. These techniques include a preparation or training phase before the retrieval or classification phase can take place. This phase may need to be carried out again in case the images in the database change. These two groups of approaches for retrieving images by content are most common in CBIR applications. Both of them may require additional data representing domain knowledge and adjustable parameters.

**“Indexing” and “Feature Index” Components** In order to process retrieval tasks more efficiently CBIR systems use special indexing structures for storing the data used to compare images. The “Indexing” and “Feature Index” retrieval components represent the indexing mechanisms and structures responsible for creating, updating and storing the indexes. Since the data used for the comparison of images is represented by multiple features, objects and sometimes also relationships between them, the indexing structures applied in CBIR systems have to be able to deal with multidimensional data and still remain more efficient than the sequential search. Choosing the right multidimensional indexing mechanism is not a trivial task and depends a lot on the kind of data to be indexed. Usually the indexed data are represented as points in a multidimensional space spanned by the features used for the retrieval. Castelli and Bergman [VC02] differentiate between vector space indexes and metric space indexes for high-dimensional indexing of image data. The vector space indexes the regions or points representing image features and the metric space indexes are used to index the distances between feature vectors, representing image content. Indexing methods can also be grouped in nonhierachical (e.g., space-filling curves, Grid-File, G-Tree), recursively partitioning (e.g., quad-trees, k-d-trees and R-trees), projection based (e.g., pyramid tree, clustering with singular value decomposition) etc. according to [VC02].

There are different ways to implement these indexing mechanisms in a CBIR system. If the CBIR system is implemented without using a database management system as a basis, the indexing mechanisms are usually implemented straightforwardly as parts of the CBIR application. In this case, the index structures are stored directly in the file system. If a database management system is used as an implementation platform for the CBIR application there are different approaches to provide adequate indexing mechanisms. Database management systems usually offer their own indexing mechanisms, which should facilitate the work of the developers of CBIR systems. However, existing DBMSs support only basic indexing mechanisms (B-trees, R-trees), which cannot be applied to high-dimensional image data. Therefore, extendable DBMSs (ORDBMSs) offer extension mechanisms for implementing user-defined

indexing methods. Kriegel et. al [KPPS03] define three approaches for implementing user-defined indexing methods in DBMSs: the integrated approach, the generic approach and the relational indexing approach. Each of these approaches has its advantages and limitations and can be applied under given conditions. The most efficient approach is considered to be the integrated approach, which suggests that the indexing methods are implemented by extending the code of the DBMS directly. This approach requires full access to the source code of the DBMS and can thus lead to unexpected side effects. The relational approach suggests that the new indexing structures are implemented using the features of the relational database and thus leads to a more robust and secure implementation. It has the disadvantage that the new indexing mechanism builds on top of the database internal indexing mechanisms. The generic approach aims at providing a generic indexing mechanism in the DBMS, which can be adapted to any new user-defined indexing method. An example of such a generic indexing mechanism is GiST introduced in [HNP95]. It is, however, difficult to construct a mechanism that is so generic that it will support the development of any index mechanism. In [Söl07a] a generic indexing mechanism for creating user-defined high-dimensional indexes based on the relational approach is developed.

The components of a CBIRS reviewed in this section try to cover the whole functionality of a full-fledged CBIRS. Depending on the implementation platform some of these components might already be available in the platform, as in the case of a DBMS as an implementation platform where the storage components are partially already available to the developer. In the following sections different implementation approaches and platforms for CBIRS are represented.

### 2.1.2 Implementation Paradigms for CBIR Systems

There are three larger research communities dealing with building CBIR applications: Information Retrieval, Databases, and Computer Vision. These communities follow different implementation paradigms, respectively. According to Gudivada [Gud93] the following approaches for developing CBIR Systems exist:

- CDBMSs (Conventional Database Management Systems) as Image Retrieval Systems
- Image Processing Systems with Database Functionality as Image Retrieval Systems
- Extended/Extensible Database Systems as Image Retrieval Systems

In addition to this classification the information retrieval approach for building image retrieval systems should also be mentioned. Such systems are represented as search engines, e.g. the Virage Image Engine [BFG<sup>+</sup>96], which implement only the retrieval functionality and do not deal with the storage of the image data in particular. Virage has been used as a basis for the image retrieval functionality as an add-on to existing database management systems such as Oracle and Informix.

Until now, most applications that have been developed for searching in image collections are centralized applications. Recent works turn to the development of CBIR in distributed environments. A reason for investigating distributed environments is the need to integrate

different resources available on the Internet for processing CBIR queries. A survey of web-based CBIRSs is given in [KZB04]. Distributed environments also enable the use of different feature extraction and retrieval algorithms as services provided by different participants in a network, as in peer-to-peer CBIR applications like DISCOVER [KNS04]. Such systems have to provide additional CBIR system components, e.g., a gatherer in the web-based CBIRSs or modules for combining the results of different peers in the peer-to-peer CBIRSs.

It is difficult to provide a development methodology that can cover all these implementation approaches. Therefore, in this thesis the focus lies on using extensible database systems as a platform for building the core functionality of centralized CBIR applications. The reason for choosing to support this implementation architecture is the large number of CBIR applications based on it, which can be used as a reference. The reason for choosing extensible databases in particular as an implementation platform is the lack of methodologies and support to design domain-specific CBIR extensions for these.

### 2.1.3 Building CBIR Systems on Top of Extendable DBMSs

In order to build a CBIR system, at least the core components described in the beginning of this chapter have to be implemented. The core components require the implementation of some feature extraction mechanisms or algorithms and a persistence management mechanism for the images and the extracted features. The results of feature extraction algorithms are often complex structures, which have to be represented through appropriate data models. The second group of components is the retrieval components group. Therefore, query processing mechanisms for similarity search in image collections have to be provided. These query processing mechanisms are mostly based on information retrieval approaches such as the vector space model, but they can also implement data mining algorithms such as artificial neuronal networks. In order to provide efficient search mechanisms, robust indexing methods for multidimensional data have to be applied.

Since storage and retrieval functionality for managing large amounts of data are features of database management systems, CBIRSs are very often referred to as image databases. The idea behind image databases is to use database management systems as an implementation platform for domain specific CBIRSs. The main advantages of this implementation approach are the availability of storage management and retrieval optimization techniques, such as transaction management, recovery and backup, and query optimization. However, existing databases were not originally designed to handle complex data types and thus do not have functionality for management and retrieval of complex data types such as multimedia. Simple extensions of DBMSs are not sufficient to meet the requirements of multimedia data. For example, integrating information retrieval techniques such as ranking similarity query results is still a problem for DBMSs. The advantages and shortcomings of applying database management systems for storing multimedia data have been discussed in Chapter 1 of [KB94], in Chapter 3 of [ABH97] and Chapters 7 and 8 in [MW03]. Thus, different paths for solving these shortcomings have been followed. One of them is to further extend existing DBMSs with new mechanisms for managing multimedia data and the other is building specialized multimedia DBMSs to meet the specific requirements of these applications. The developments in these two directions are summarized below.



### 2.1.3.1 Multimedia Database Management Systems

The first works in the area of multimedia databases followed the intuitive idea of building specialized multimedia database management systems with the aim to provide more adequate support for multimedia data. Some of the results of this “first wave”, such as MediaWay, JASMINE, and ORION have been cited in [KD05]. Only a few of these systems have survived on the market, e.g., MediaWay.

A lot of theoretical work on the development of MMDBMS has been done to provide a more stable basis for building such systems. Architectural aspects, modeling, querying and indexing have all been considered. One of the first attempts to provide a complete theoretical background of MMDBMS is the work of Marcus and Subrahmanian summarized in [MS96]. The authors present a mathematical model of multimedia data and query processing operations based on this model. Architectural matters of MMDBMS have been tackled in [Gha95] and Chapter 5 of [KB94]. Later on Santini and Gupta [SG02a] describe the general architecture of a database system for multimedia (image) data, and in detail the components for schema design and a feature algebra. They propose an extensible feature management engine for image retrieval which uses the services of a traditional relational database with the addition of user-defined indexing schemes and with a bespoke database model.

This research leads to the recognition that existing DBMSs can be extended or adapted to support multimedia data types and more complex retrieval functionality. Therefore, bigger commercial DBMSs provided support for user-defined data types and functionality. Freeware, open source and research DBMSs have also been developed with the possibility of being applied to complex data types.

### 2.1.3.2 Object-Relational and Object-Oriented Databases

During the past years leading DBMS manufacturers such as IBM, Informix and Oracle have tried to offer support for managing multimedia data through database extensions. In response to this effort the SQL standard initiative has released new versions of the SQL Standard - SQL:1999 and SQL:2003 - which introduce object-oriented concepts into relational DBMSs to allow extensions of database functionality for different applications, such as multimedia applications. These new versions of the SQL standard are summarized in [Tür03]. How these new concepts can be used to build object-relational databases, is discussed in [CT06]. Design issues for object-relational database systems have been discussed in [DU04]. Additionally, the ISO SQL/MM Standard has been released, which proposes a way to support media types such as still images and spatial data in SQL-based DBMSs. A presentation and a discussion of the standard is given in [Sto01]. The SQL/MM standard, however, describes no more than what the earlier existing DBMS extensions already offered. Systems complying with this standard include Oracle interMedia and IBM DB2 Image Extender part of the AIV Extenders group. The extensions of DBMSs for still images have been reviewed in [Sto02].

The SQL/MM standard limits itself to the definition of user-defined types, methods and functions for supporting content-based storage and retrieval of images based on some of their perceived characteristics, such as global and local color, and texture. High-level features are not considered. The existing implementations are thus also limited and do not provide further possibilities for including other features or distance measures. The IBM DB2 Image Extender is no longer supported in the newest version V.9.1 of the DBMS. The manufacturer provides

a tutorial of how to build bespoke image extensions complying with the SQL/MM:Still Image Standard in DB2 [Sto05] for users interested in this functionality.

Neither the standard nor existing DBMSs extensions have met other requirements of multimedia databases, such as support for multidimensional indexing mechanisms, combining similarity query results or optimizing the processing of similarity queries. Therefore, these topics have since been the focus of database research. In [KPPS03] different approaches for integrating user-defined indexing methods in DBMSs have been compared. [Söl07b] shows how the relational indexing paradigm can be used to implement multidimensional indexing mechanisms in ORDBMSs. The support of information retrieval techniques, such as combining weighted vague results of sub queries in a SQL query, is tackled in [SSH05] by proposing a similarity calculus as a basis of a multimedia query language and extending query language based on the relational domain calculus. Finally, query optimization techniques for extended SQL queries have been proposed in Chapter 7.5 of [VC02].

Application considerations about image database management systems are noted in Nes' dissertation [Nes00]. On the basis of an image algebra, Nes defines the basic features of an image management system and implements an image model with its corresponding operations in the non-commercial, research DBMS MonetDB [Mon05]. MonetDB is a main-memory, object-oriented database, which supports various extensions and aims to bridge the gap between information retrieval and DBMSs [HZdVB07].

In the DISIMA project [OÖL<sup>+</sup>97] an image database system was built on top of the object-oriented DBMS ObjectStore. The data model is reminiscent of the VIMSYS model [GWJ91], with the representation of the different levels of abstraction and views of an image. The main aim of this model is to model the different views (interpretations) of image content. This model uses MOQL for query formulation and an ODMG Schema to implement the model.

In spite of the remaining problems of database extensions, object-relational DBMSs offer a sophisticated implementation platform for image databases. Existing example applications and development approaches can be used as guidelines. Therefore, these DBMSs have been chosen as target environments for deploying CBIR systems.

#### 2.1.4 Summary of CBIRS Technologies

In this section, the basic functional principles of a CBIR were described. Although CBIR systems are implemented differently depending on the domain that they have been created for, their functionality can be abstracted in common components. These generic components were represented above in an integrated CBRIS architecture which can be applied for a broad range of CBIRS applications. Various possibilities for the realization of the components for concrete applications have been summarized.

Different implementation paradigms and platforms can be used for the implementation of CBIRSs. For this thesis, a centralized architecture, based on an extensible database management system is considered to set the requirements towards the target implementation platform.

## 2.2 Model-Driven Software Development - Basic Principles

### 2.2.1 Characteristics and Aims

The Model-Driven Software Development (MDSO) approach aims to improve the software development process, in particular to increase the development efficiency through reuse and stable architectures and to provide a good basis for increasing software quality. The Object Management Group (OMG) standardizes this development approach with the Model-Driven Architecture (MDA) paradigm and focuses on two other major aims of MDSO, namely interoperability (better inter-software communication) and portability (platform independence). MDA also provides the basic terminology used for MDSO.

In order to achieve the above aims the development process is shifted from the source code level to the more abstract, but also more domain-specific modeling level. Model transformations are introduced to translate the abstract models into executable source code. In Figure 2.3 the MDA development process is illustrated. The starting point is a platform independent model (PIM). This model, describes only application domain specific artifacts and contains no information about the technology used for the implementation. Through one or more transformation steps the PIM is converted to a platform-specific model (PSM) which contains more specific details about the implementation. The second transformation step transforms the PSM into source code. The idea behind this development process is to provide an automated forward engineering pipeline for propagating changes made in the platform independent model to the source code or platform specific model. Thus, more than one iteration of this pipeline is possible. However, these iterations have to start from the PIM level in the general case, because it cannot be assumed that a reverse transformation, especially from PSM to PIM is always possible. In fact, most often these models are not isomorph. Despite of that, manual changes in the source code are often needed, and actually required to provide the final implementation of the application. Therefore, special techniques have to be applied in order to preserve manually edited source code or protect generated source code etc. In this chapter, the main terms and techniques in model-driven software development are briefly summarized.

### 2.2.2 Models for Model-Driven Software Development

Models in MDSO are used to describe particular aspects of an application domain, which are of interest for the development of a software system. The formalization of these models should allow the application of automated mechanisms for their interpretation, transformation and eventually execution. Therefore, software modeling has a lot of similarities with software programming. In fact, some authors as in [PM06, SVEH07] suggest that programs are also kinds of models, i.e. a programming language is a kind of modeling language. In that case, models also need a modeling language with corresponding syntax and semantics, in which they can be expressed, analogously to programming languages. Programming languages are usually represented textually, whereas models often use graphical representations. Behind each representation, which is referred to as concrete syntax, a robust abstract syntax should be defined. Modeling languages are also referred to as meta models. A meta model should be defined for each different type of model participating in the MDSO process, PIM and PSM. Moreover, different kinds of meta models may be suitable for different aspects of the application, such as persistence, business logic and graphical user interfaces. In MDA the

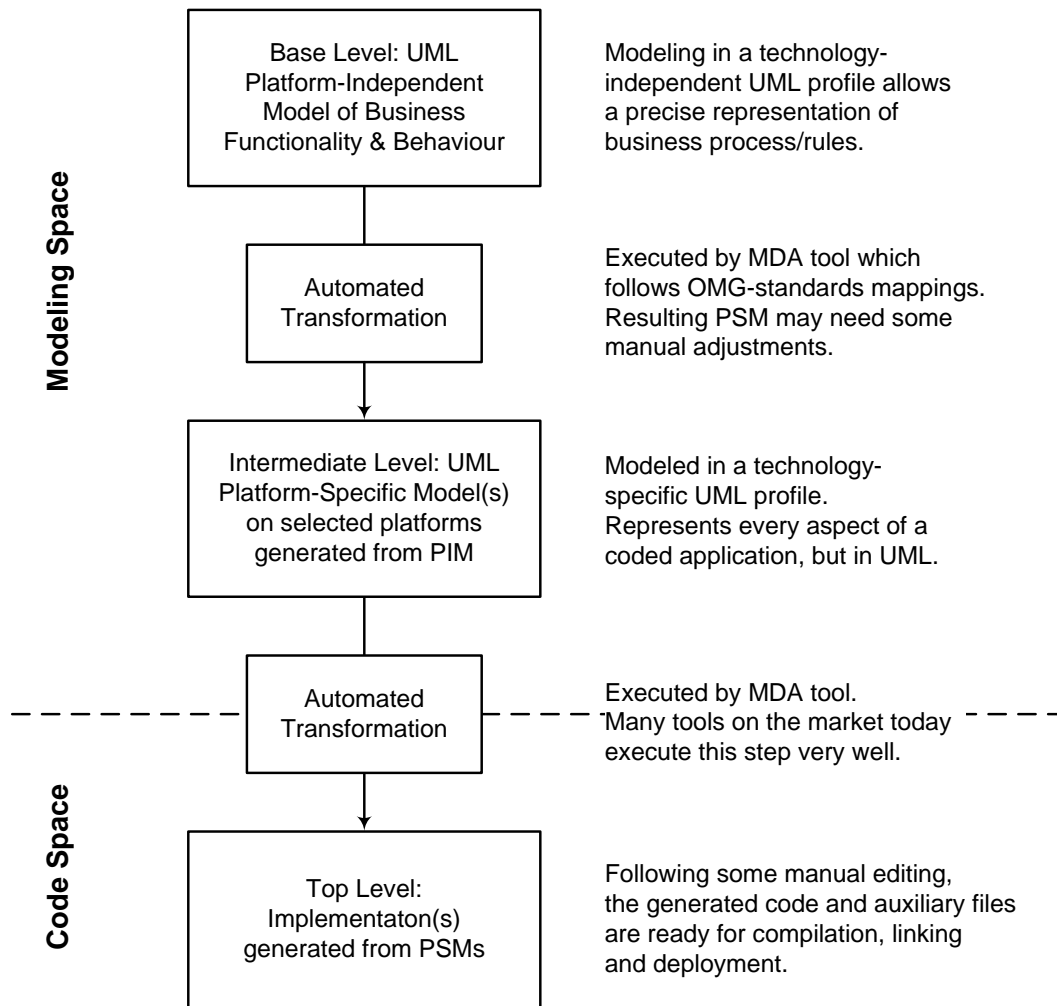


Figure 2.3: Basic structure of OMG's Model Driven Architecture (based on [Jon02])

meta models are defined as UML Profiles and thus are instances of the Meta Object Facility (MOF). However, also other modeling facilities are supported by existing modeling tools, such as Ecore - the meta meta model from the Eclipse Modeling Framework. This meta meta model can be used to define domain-specific meta models. Ecore is similar to the reduced version of MOF, known as essential MOF (EMOF). Other meta meta models, such as XML, abstract syntax trees and domain-specific languages have been reviewed in [SVEH07]. The choice of a meta model influences the implementation of the model parser, but also that of the transformer/code generator. In fact, model transformations are defined based on the meta model and executed on the concrete models.

In Figure 2.4 the different meta levels of MDA are shown. The PIM and the PSM are models of the level M1 in Figure 2.4. They can be instances of UML or of domain-specific extensions of UML (UML Profiles) from level M2 as shown in Figure 2.5. The PSM or PIM UML Profile is referred to as the meta model of the PSM or PIM, respectively. An example of a meta model for PIM would be a UML-Profile, representing the concepts of the Entity-Relationship model, such as entities, attributes, relationships. And a meta model for a corresponding PSM

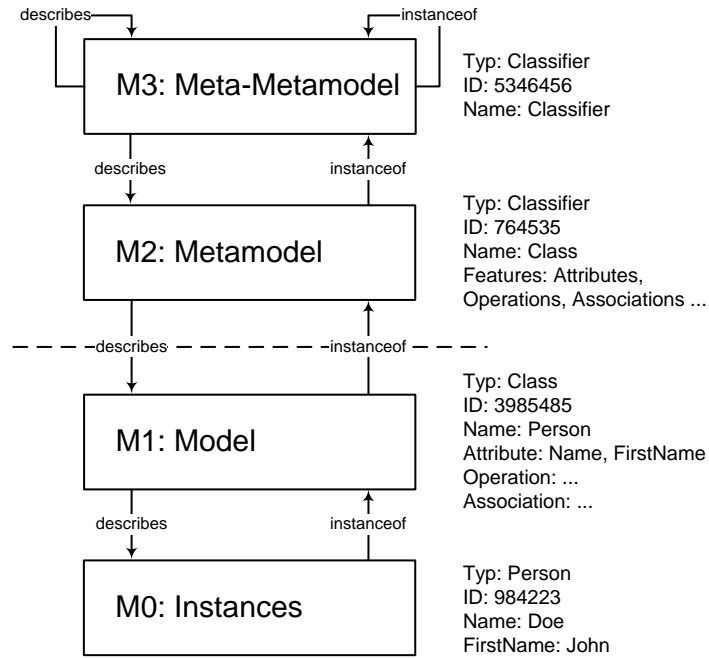


Figure 2.4: Meta levels in MDA (based on [SVEH07])

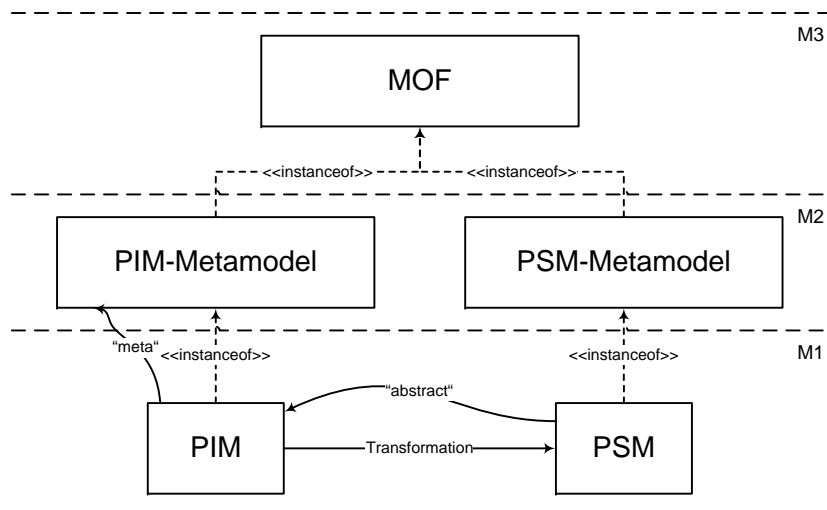


Figure 2.5: UML PIM and PSM meta models (based on [SVEH07])

would be a UML-Profile, describing relational concepts, such as tables, columns, primary key etc. This example is illustrated in [CH06].

### 2.2.3 Model Transformations and Code Generation

The next step in model-driven development after the design of the application-specific PIM is the transformation into a PSM. The PSM is a representation of the PIM in terms of the so called platform model (PM). This means that the PSM uses the interfaces and concepts provided by the particular platform, represented by the PM. In a straightforward transformation a PSM is the generated source code from the PIM. However, it is not always possible to generate source code from the PIM directly. Additional refinements of the model are often required in order to generate a platform specific implementation. Therefore, intermediate transformations can be carried out before generating the code. For example, the PIM can be transformed into a logical schema (PSM) in which some refinements can be made by the developer before generating the source code.

#### 2.2.3.1 Choosing a Platform

According to the authors of [PM06] the term *platform* in MDA refers to a specific hardware or application environment. Platforms usually build on top of one another to form a so called platform-stack. A platform-stack may consist of hardware and an operating system which resemble the first two platforms in the stack and an application environment platform on top, which is aware of the existence only of the operation system platform. In [OMG03] the following comment for a platform is given “...What counts as a platform is relative to the purpose of the modeler. For many MDA users, middleware is a platform, for a middleware developer an operating system is the platform. Thus, a platform-independent model of middleware might appear to be a highly platform-specific model from the point of view of an application developer...”. CBIR systems are implemented as applications on top of one or more specific application environments. Architecturally a CBIR system may consist of more than one component, which can be realized in different application environments. For example, the data storage and manipulation part could be implemented in a database system environment and the retrieval component as a database middleware in a C++ environment. The client applications for CBIR systems could be implemented in a Java Enterprise application environment.

#### 2.2.3.2 Model Transformations

After choosing a target platform for the implementation of the CBIR system the platform independent model has to be mapped onto the platform specific model. This transformation process is one of the main concepts discussed in Model-Driven Software Engineering as well as in database design. The aim of the transformation is to translate the concepts of the PIM (in database design the PIM corresponds to the conceptual model) into concepts of the platform. Therefore, in MDA the so called Platform Model has been introduced, which represents a meta model of the platform. The resulting PSM is, therefore, an instance of the Platform Model. In Figure 2.6 the transformation process in MDA is illustrated. The transformation of a PIM into a PSM is in the general case a translation of the concepts, represented in the PIM in terms of the abstract PIM meta model into terms of the concrete PSM meta model

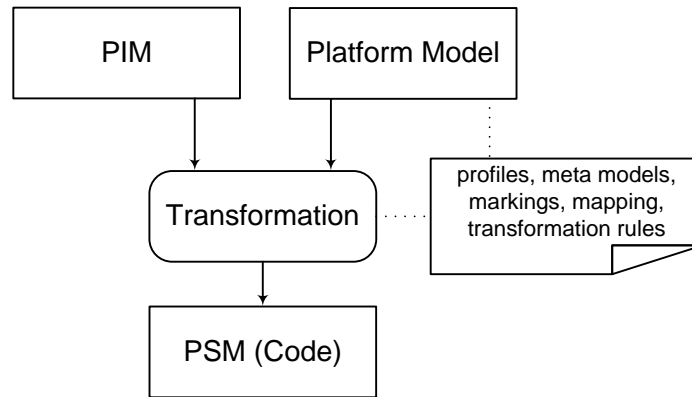


Figure 2.6: Transforming PIM to PSM (based on [PM06])

(PM). It is also referred to as an exogenous transformation. The opposite is the endogenous transformation which is defined between two representations of the model in the same meta model. This transformation is also referred to as rephrasing and is used for tasks, such as optimization, refactoring etc. If the transformation maps a concept from the source model (meta model PIM) to exactly one concept from the target model (meta model PSM) then the transformation can be regarded as a function and can have also further properties such as being injective, reverse, bijective.

The formalization of transformations is needed in order to support the automatic execution of the transformation. Therefore, a transformation language should be defined. A model transformation language should meet different requirements in order to assure quality of the transformation. The following requirements for transformation definition languages have been addressed in the literature:

- formalism [Kuz05] - the transformation language should be based on a well-formed grammar in order to be executed automatically.
- universality [Kuz05] - the transformation language should allow the definition of transformations of various platforms through tuning and parameterisation. This requirement is also considered in [CH06] by the tunability property of transformations.
- integrity preservation [Kuz05] - model-driven development often requires additional iterative changes of the source and generated models. Therefore, the mappings which take place during the generation step should be recorded so that the conformity between the source and target model can be checked when changes are undertaken. In [CH06] this requirement was referred by the traceability property of transformations.
- incrementality [CH06] - focuses on required properties of transformations, such as target incrementality (change propagation) for incrementing the target model with changes from the source, source incrementality assures that only the changed elements of a source are recompiled if needed and preservations of user edits in the target should be provided in case that user changes in the target model should not be overwritten by changes in the source model (e.g. by synchronisation).
- directionality [CH06] - depending on the application a transformation may need to be

executed not only in one direction, but in the inverse direction also. Therefore, bi- or multidirectional transformations should also be possible.

More practical properties of model transformations, such as scalability, simplicity and ease of adoption have been considered in [GGKH03]. Especially for the graph-transformation-based approach, semantic properties such as confluence (unique result) and termination (the transformation should terminate) play an important role.

What most of the works on model transformations for model-driven software engineering do not consider is the requirement to preserve the semantic equivalence between the source and the target model. Such a requirement towards the transformation function can be defined by means of information capacity [Hul86].

To what extent these requirements have to be fulfilled depends also on the application domain. In Chapter 3, the applicability of these requirements for the transformation of the CBIRS PIM onto an ORDBMS PSM is discussed.

**Model-to-Model** transformations are used not only for model-driven software development, i.e. for generating platform specific models from platform independent models. Schema matching, schema evolution, database design are other applications where model-to-model transformations are needed. We can distinguish between intra- and inter-model transformations, where in the first case a model expressed through a meta model is transformed into another representation or instance of the same meta model, as for example providing different views on the data stored in a database. Whereas in the second case the model is translated into the terms of another meta model, as for example translating a PIM into a PSM. For the transformation of the CBIR system PIM into an SQL model the intra-model transformations are of interest.

The Object Management Group (OMG) has acknowledged the need of formalizing such transformations by issuing a Request for Proposal in 2002 on Query/Views/Transformations (QVT). Formalizing transformations should enable their integration in MDA-tools, which can carry out the transformations automatically. Multiple responses to this request have led to the final adopted QVT specification in 2005 [OMG05] and to the development of numerous transformation techniques and tools. Existing transformation techniques have been categorized, based on a proposed feature model for their description in [CH03, CH06]. The following categories of transformation techniques, grouped according to their characteristics and design choices were proposed in [CH06].

- **Direct manipulation approach.** This is the most straightforward transformation approach. It provides an internal representation of the model and interfaces for its manipulation, as well as transformation classes implemented in a programming language. Object-oriented frameworks are used to provide a basic structure for the transformation, which has to be implemented for a particular transformation technique. One of the advantages of this approach is the freedom which the programmer has to define transformation exactly as he/she needs them, without being limited by a transformation language, as is the case with templates. One disadvantage is that due to this freedom, more time is required for the implementation of the transformations, and the implementation is more error prone.
- **Structure-driven approach.** The idea of this transformation approach is to copy all the



elements of the source model to the target and adapt them in the target model in order to achieve transformation. Implementations of this approach provide frameworks which take care of scheduling and copying. The users need to provide only the transformation rules.

- **Operational approach.** This approach is similar to the direct manipulation approach, but instead of a regular programming language, extensions of modeling languages with imperative constructs are used to implement the transformations, e.g. MOF with executable OCL. Adequate interpreters of these transformation models have to be provided.
- **Template-based approach.** In this approach, the target model is provided as a template with blanks, conditions, iterations etc. which have to be filled out depending on the source model. The routines for filling the blanks can be provided separately or within the template.
- **Relational approach.** Such an approach uses mathematical relations or constraints to declare the transformations which have to take place. Target model elements are created implicitly and do not allow in place transformations, i.e. there is always a source and a target model. This approach also supports multidirectional transformation rules.
- **Graph-transformation-based approach.** This approach is especially suitable for class model transformations, since it operates on typed, attributed, labeled graphs, which can be used as the formal representation of class diagrams. Therefore, a left hand side (LHS) pattern is matched in the source model to find the concepts which have to be transformed and is replaced by a right hand side (RHS) pattern in place. These transformations are performed in one direction.

Exact borders to differentiate between these techniques do not exist. The different techniques can also be combined in some applications. Some of them can be regarded as a specialization of others, for example the operational approach is a more specific direct manipulation. The feature model used to make this classification can be used to choose an appropriate technique for implementing the transformation. Before the features of the transformation can be formalized, the transformation rules or mappings have to be conceptually clearly defined. The target and source model and their presentation (textual, graphical, classes etc.) have to be determined. A transformation technique can be applied on the concrete syntax of a model, such as XSLT on XML. However, the transformations should be kept independent of the concrete syntax, because it may vary as suggested in [VS06]. For example, different tools support different versions of XML.

**Model-to-Code** The transformations needed to generate the source code of the designed application are a subset of the model-to-model transformations. This is due to the fact that code generation actually is a transformation onto a programming language, which can be regarded as a textually represented meta model. In [CH06] the following model-to-code approaches have been classified in the following groups.

- **Visitor-based approach.** This approach corresponds to the direct manipulation model-to-model transformation, and additionally states that the direct manipulation technique should use the visitor mechanism to traverse the internal representation.

- Template-based approach. The approach is analogous to the model-to-model template transformation approach.

These different implementation approaches for transformation rules are considered when choosing a way to define the transformation rules for the CBIRS application in Chapter 5. However, it is concluded that a textual description of the rules is more appropriate at the formalization, conceptual stage. The above approaches are more relevant for the implementation of the specified rules. In the students project of Andre Scheffe [Sch07] the direct manipulation was chosen to implement a generation plugin for the generation of an image database.

#### 2.2.4 Summary of MDSD Technology

MDSD describes a software development approach which aims at automating the generation of application implementations and allow developers to focus on domain-specific problems when designing the software. Therefore, a set of techniques have been suggested for developing domain-specific models and transforming them to platform-specific source code. Often for each application domain new modeling languages have to be specified in order to represent domain artifacts adequately. Platform-specific languages have been already specified for a large variety of platforms, e.g. RDBMS, but often they need to be extended to support more concrete or new features. These modeling languages can be created or adapted by using the UML or MOF meta modeling languages.

Regardless of which transformation technique is applied in the MDA tools, at first an adequate description of the mapping rules has to be provided. The characteristics of the mapping functions have to be analyzed in order to assure high quality of the transformation.



## Chapter 3

# Requirements Analysis and Conceptual Design of MDSD Techniques for CBIRSs

Model-driven software development requires two groups of techniques, namely modeling techniques and transformation techniques. Each of these techniques has to be adapted, i.e. designed, for the concrete application and technology domains. In this chapter, first the requirements towards the modeling techniques for each component of a CBIRS are described. Based on these requirements, existing generic models of image retrieval systems are evaluated. In the second section of the chapter, transformation techniques for mapping platform-independent models represented by the Unified Modeling Language onto a platform-specific model represented by the ORDBMS model, are reviewed. Existing meta models for the platform-specific model are also described. Furthermore, properties of the transformation, which can be used to prove its quality are defined. The concrete adaptation of these two groups of techniques is explained in the following two chapters, respectively.

### 3.1 Domain-Specific Modeling for CBIRSs

In order to design the modeling techniques for the first step of the MDSD process of CBIR, illustrated in Figure 3.1, the generic components of a CBIRS system which have to be modeled are identified and their properties and functionality are described.

A model of an application represents one or more common aspects of the application domain through some more or less formal modeling language. In model-driven design it is essential to use well formalized modeling languages, because these have to be transformed later on automatically into executable code. Choosing a suitable modeling language depends to a great extent on what exactly has to be modeled. The domain-specific modeling approach suggests that for each application or technology domain a specialized modeling language should be provided. Domain-specific modeling languages have been provided for technologies, such as web applications [NFG06] and mobile phone applications [DB07], as well as for applications in the field of ERP (Enterprise Resource Planning) [BLPR07], health care [MC07] etc. More examples of domain-specific modeling languages are represented on the website of the Domain-

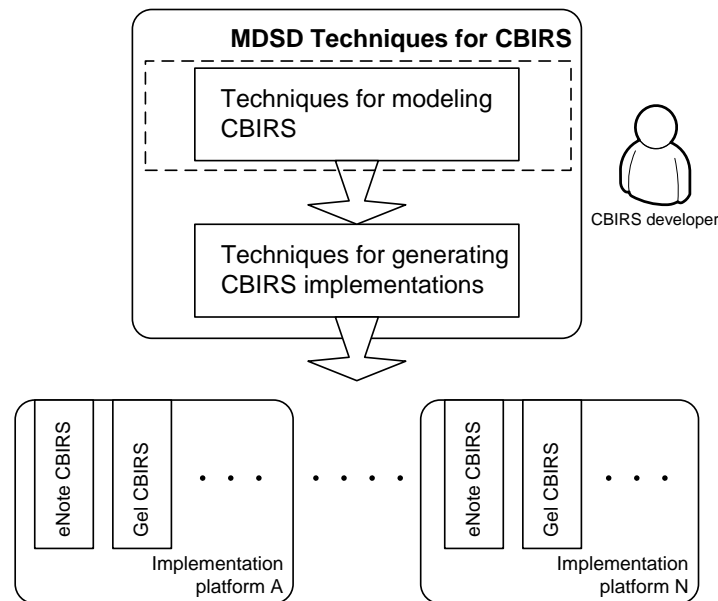


Figure 3.1: MDSD techniques for the development of CBIRSs - Modeling

Specific Modeling Forum<sup>1</sup>. Other modeling approaches prefer to adopt a generic modeling language, such as the Unified Modeling Language for representing their application design. However, generic languages usually have to be extended to support domain-specific concepts in a better way. Thus, generic modeling languages transform into domain-specific languages. A domain-specific modeling language would facilitate the design of CBIR systems. However, one of the aims of this thesis is to provide a reusable design methodology. Therefore, a generic model of a CBIRS has to be also elaborated, so that the developer can profit from reusing an architectural design. The generic model should allow flexible adaptation to the developer's requirements. To choose an appropriate modeling approach, at first a decision has to be made about which aspects of a CBIR system should be modeled and generated. Referring to the generic CBIRS architecture in Figure 2.2, three horizontal groups of components can be identified: the user interfaces, the compensators and the core components. The focus of this thesis is the modeling and generation of the core CBIRS components. The idea is to provide the core data structure and functionality of the CBIR system for use by different client applications. Multiple client applications may be later on implemented to meet diverse user needs. Therefore, the aim of the current work is not to provide a particular graphical user interface for the system. Usually CBIR systems require the design of complex user interfaces to support user-friendly interaction with the CBIR core. Furthermore, different technical environments, such as mobile devices, set special requirements for user-interfaces. Therefore, additional information apart from the basic functionality and data structure of the system has to be considered when modeling graphical user interfaces. For the model-driven design of advanced user interfaces, a technique based on task models has been proposed in [WFDR05]. User interfaces, however, play an important role when specifying the main functionality of the system, e.g. to define what kind of queries have to be supported. Therefore, they should be included in the model for the specification of the interfaces of the core components. In

<sup>1</sup><http://www.dsmforum.org/cases.html>

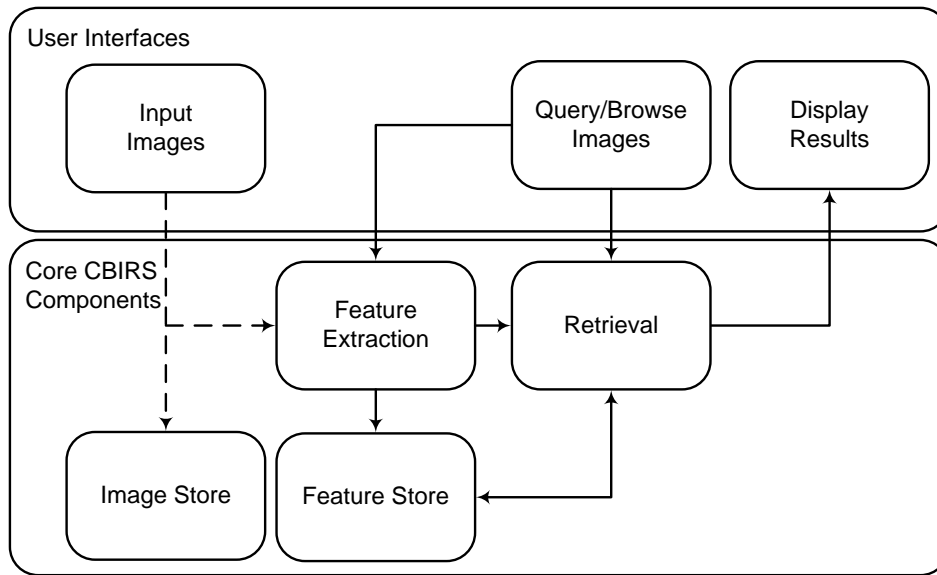


Figure 3.2: Core CBIR components to be modeled by the CBIRS Generator

the next section, the modeling requirements for the core components of a CBIR system are systematized. Their generic and varying characteristics are highlighted.

### 3.1.1 Requirements for a Domain-Specific Model for CBIRSs

The core CBIRS components, which have to be modeled and generated, are shown in Figure 3.2. These comprise the “Feature Extraction”, “Image Store”, “Feature Store” components for extracting and storing image content, and the “Retrieval” component for retrieving images based on their content. In Chapter 2, the “Feature Index” is also part of the core CBIR components. The data used for content-based image retrieval is multi-dimensional data which can lead to very inefficient retrieval when a sequential search is used for a large collection of images. Thus, appropriate multi-dimensional indexing mechanisms (R-tree, SS-tree) can be employed in a CBIR system to improve the efficiency of content-based image retrieval. Indexing mechanisms comprise usually an intermediate representation of the data used for the retrieval and corresponding update and access operations. Essentially, index structures are another representation of the image data which can be used to apply corresponding retrieval operations in order to answer specific queries more efficiently. This representation is platform dependent, and therefore, is not considered for the modeling of CBIRSs. The other core components have to be included in the CBIRS model. Each of these components sets different requirements to the model. These requirements are described in detail in the following paragraphs.

As mentioned in Chapter 1, the eNoteHistory scribe recognition application is used to define the scope of the CBIRS development approach proposed in this thesis. Therefore, in addition to a general description of the requirements towards a CBIRS component, a concrete application example from the eNoteHistory application is given. The eNoteHistory CBIRS is described in detail in Chapter 6.

### 3.1.1.1 “Image Store”, “Feature Store” Components

When designing the storage components of an image retrieval system, the aim is to determine and represent the characteristics of an image based on which an observer decides whether the images are similar or not. Two types of characteristics can be distinguished, those which can be directly derived from the raw image or visual representation of the image (content-dependent characteristics) and those which are independent of the information encoded in the image (content-independent characteristics).

#### Requirements for Modeling Content-Dependent Characteristics

One of the most common classification of image content is that of abstraction levels as illustrated in Chapter 2, Figure 2.1. In this classification, all content characteristics are grouped according to a so called level of abstraction which is often used in the image processing literature.

A single matrix of pixel intensities representing the raw image can be associated to each image. Although content-based retrieval on whole images is very rarely carried out based on all the pixels of an image, this information can be useful for comparing smaller regions of images. Furthermore, the image matrix can be used to derive compressed representations of the image using, for example, Fourier or Cosine transformations which can be used for image comparison. Although color distribution, texture, and shape are derived from the pixel-matrix, i.e. a higher level of abstraction is reached, these characteristics are classified as physical level characteristics because they can be regarded as a summary of the pixel information. They also are often referred to as low-level features in CBIR systems. They can be used directly to compare images or to derive higher-level characteristics, such as objects corresponding to regions in the image with the same color of pixels.

The characteristics at the logical level are related to the identification of objects in the image and the structural arrangement of these objects. In the case of large image collections this process cannot be carried out manually. Thus, the segmentation of an image has to be carried out by image processing algorithms which search for connected regions in the image with homogeneous physical-level characteristics. An object itself is nothing else but a set of pixels from the image which correspond to a certain 2-dimensional shape. Therefore, all characteristics that a whole image possesses can also be associated to a single object of that image. The structural layout of the objects is represented by spatial relationships, such as topological or directional relationships. For these relationships there are different representation possibilities: 2D-Strings [CSY87] and their variations [LC03] and attribute relational graphs (ARGs) [PF97] are some of these to name. The transition from the logical level to the semantic level is not so straightforward as from the physical to the logical level, because much more context and knowledge than available from the physical and logical characteristics is required in order to assign the correct semantic concepts to the image and objects. It should be mentioned that the transitions between different characteristics can be integrated into the data structure through relationships between the characteristics, similarly to the spatial relationships, for example, through an *interpreted* relationship between the characteristics. The semantics of such relationships depends on the transition functions which transforms the low-level characteristic into a higher level characteristic.

Semantic level characteristics can be textual descriptions or conceptual terms associated with

a whole image or with image objects. Different semantic interpretations are possible, for instance, depending on the users point of view or the time of the analysis. Extracting semantics from the image is a task which requires additional knowledge about the context and origin of the images. Therefore, reliable algorithms for automatic annotation of images are still under development. Most of these algorithms try to bridge the gap between the logical, the physical level, and the semantic level characteristics, by mapping lower-level features onto higher-level semantic concepts and thus defining relationships between semantic concepts and features used for their derivation. Content-independent information, if available, can also be used to derive semantic characteristics.

### Requirements for Modeling Content-Independent Characteristics

Although this type of characteristics has no direct connection to the content of the image, there are cases where indirect dependencies can be very useful for determining the content. Some authors refer to this data also as the context of an image [BSST07]. For example, if we have an image of a desert and have to decide whether it is an image of a wall or of a landscape, one entry in the meta data of the image: place="Sahara" would make it perhaps easier to make the decision. For each application domain there are different requirements for content-independent image data. In photography and medical imagery these data can have a completely different structure. However, it provides valuable information which can be used by the feature extraction or retrieval process later on. In Chapter 2, some standards aiming at classifying such metadata are mentioned.

### Example

In the eNoteHistory CBIR application the following characteristics of the image are needed in order to compare the handwriting of scribes.

1. *Region of Interest (ROI)*, which resembles a larger rectangular region of the image containing only the handwriting. If more than one scribe wrote the page then multiple ROIs can be expected in an image. At least one ROI should be detected in an image so that it can be used for comparison. In Figure 3.3 an example of such as ROI is illustrated.
2. *Music objects* each ROI can contain a set of smaller regions corresponding to music elements. Each region is characterized by its minimum bounding rectangle (MBR), i.e. two pairs of xy coordinates. Furthermore, the regions are assigned to concepts corresponding to note heads, note stems, bar lines. Note heads and stems can be combined into notes.
3. Content-independent characteristics, which can help in the retrieval task in this application are bibliographical data, such as the name of the music work or the name of the composer or publisher. At the time these manuscripts were copied (rewritten) by scribes who were employed by a particular composer or publisher for a longer time period. Thus, the identification of the scribe can be supported by such information.





Figure 3.3: A Region of Interest (ROI) from a music manuscript image and MBRs of the detected music objects

## Conclusion

A CBIRS model should be able to represent the data structures for different types of image characteristics. Each application may require a different combination of these characteristics. The generic parts of such data structures comprise some kind of an image identifier and a raw image representation. Optionally an image may be partitioned into several image regions with associated image characteristics. However, the exact types of regions or characteristics depend on the application. Mechanisms for managing the persistence of these data structures have to be modeled if there is no persistent storage management mechanism provided by the implementation platform. These mechanisms can be provided as a generic component in the CBIRS model.

### 3.1.1.2 “Feature Extraction” Component

Another task for the designer of an image retrieval system is to determine the way all image characteristics come into the system. By providing as much as possible automation for this process, a more efficient, scalable, and more objective extraction of the characteristics can be assured. However, the decreasing accuracy especially in the higher levels of abstraction can lead to imprecise characteristics and later on to bad retrieval results. Therefore, only well-performing algorithms should be considered for the automatic extraction of features.

### Requirements for Modeling Feature Extraction Functionality

Feature extraction refers to the process of transforming the images into representations in terms of characteristics. If this process is supposed to take place automatically or semi-automatically the algorithms responsible for these operations should be included in the image model. Manual feature extraction is regarded as part of the user-interface modeling. Therefore, it is not modeled in the “Feature Extraction” component. Each type of feature and each level of abstraction requires an adequate feature extraction algorithm. The image seg-

mentation can also be considered as a feature extraction algorithm, because its result is the identification of image objects, which are logical level characteristics of the image. These algorithms may be a subject of change, because more and more effective algorithms for the extraction of a feature may be implemented. In this case, the features gained from older algorithms could be replaced with the results of the new ones or they could be stored along with the new ones.

### Example

In the eNoteHistory application regions representing note heads, stems and bar lines are extracted by an image processing algorithm, which can also determine the minimum bounding rectangles (MBRs) and minimum bounding ellipses (MBEs) of the objects and other features as well as assign each region to a corresponding concept. Determining the ROI is a non-trivial task for image processing algorithms, thus it is determined manually by the user.

### Conclusion

It is difficult to define generic feature extraction algorithms, but placeholders for such functions can allow adapting these functions through framework development techniques. A useful help for the developer of such extraction algorithms can be source code libraries for extracting standard image characteristics.

#### 3.1.1.3 “Retrieval” Component

Apart from providing a data schema to store image data, a CBIRS model should provide means to model operations for processing image queries based on these characteristics. These operations should provide mechanisms to search for similarity in the image database. The authors of [ABH97, Sch04] have provided an overview of the recent research and techniques for the processing of content-based similarity queries for multimedia data.

The notion of similarity in queries has been formalized by Jagadish, Mendelzon and Milo in [JMM95]. They defined a context free pattern language for defining objects, a transformation rule language to specify the similarity between objects, and a general query language. This has been further on used as a basis for the development of a multi-similarity algebra [ABSS98], where similarity operators have been defined to address the need of query optimization in similarity-based retrieval and to combine different similarity implementations in one query. Further research on the definition of a similarity algebra and similarity operators has been represented in the works of Santini and Gupta [GS00b, SG01, SG02b], and by Atnafu et al. in [ABK01]. Atnafu et al. show also how their operators can be integrated in a DBMS environment.

Queries on image data can consist of multiple predicates which can be combined with each other. Algorithms for the processing of such combined queries have been offered: TA-Algorithm [FLN03], Algorithms for combining streams [PF95, HR03] as well as the Generalized VA-File Based Search (GeVAS) [BMSW01]. The first two have been implemented as middleware for ORDBMSs. In [AG01] the same problems have been studied especially for digital images.

What kind of algorithms should be applied to perform an automatic image similarity search is the second decision which the developer has to make with respect to the functionality of the system. Thereby, the kinds of queries which the system will support should be defined together with the characteristics which should be used to process these queries. The way these characteristics are combined is also an important question, in the case of queries over more than one characteristic, which is the way images are most often compared.

### Requirements for Modeling Retrieval Functionality

The retrieval algorithms represent the main functionality of a CBIR system. They depend, on one hand, on what kinds of retrieval tasks the system should be able to process according to the requirements of the users. On the other hand, they depend on the mechanisms which can be applied to interpret image content and measure image similarity.

In [SWS<sup>+</sup>00] content-based image retrieval tasks have been categorized in *exact queries*, where the query answer comprises the images from a given image set satisfying some set of criteria, and *approximate queries*, where the result from the query is a ranking of the images from a given image set with respect to the query, based on an appropriate similarity measure. Each of these categories has been further subdivided depending on whether the query relates to the global image characteristics, spatial image characteristics or to a group of images. Figure 3.4 from [SWS<sup>+</sup>00] shows examples for these types of queries and possible results from the Corel image database. Each of these three query types can in fact be assigned to one of the three levels of abstraction for representing image content. Spatial predicate queries are based on the spatial relationships and location of objects (salient regions) in the image. Queries on image predicates are queries using global image characteristics. In group predicate queries, the interpretation of the image content in terms of categories or classes is used for the retrieval. Furthermore, these queries can be combined in more complex queries, for example, querying the images based on local features combining spatial predicate queries with image predicate queries. It can be argued if the exact query example of the image predicate query is really exact. The features used in the predicate “sky” and “sand” are not exact, since they are subjective or difficult to measure exactly. However, the predicate represents an exact comparison.

**Similarity Measures:** Different mechanisms can be applied to answer such types of queries. Most of these mechanisms deal with proposing a representation of image similarity as perceived by the user in the domain of image characteristics. Therefore, for each type of characteristics a similarity operation should be defined. This operation can be used to compare the values of these types with respect to similarity. More often a similarity operation to measure the similarity between aggregated characteristics for processing complex queries is needed. One example of an aggregated characteristic is a feature vector comprising values for different types of features. A hierarchical aggregation is employed where local image characteristics referring to the features of image objects and their spatial relationships are used for comparing the images. To integrate retrieval functionality into the image model, the so defined similarity operations should be included in the model. Many different approaches which can be used to implement similarity operations exist. In [SWS<sup>+</sup>00] a systematization of similarity measure approaches is given where these approaches are grouped with respect to the types of image characteristics on which they can be applied.

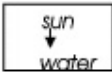




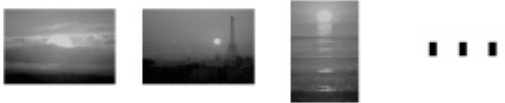






	Example query	Example query result
exact	Spatial predicate 	
	Image predicate <i>Amount of "sky" &gt; 20% and amount of "sand" &gt; 30%</i>	
	Group predicate <i>Location = "Africa"</i>	
approximate	Spatial example 	
	Image example 	
	Group example <i>pos</i>  <i>neg</i>  	

Figure 3.4: Example queries for the six query types and possible results from the Corel image database (from [SWS<sup>+</sup>00])

One of the most used similarity models is the metric model, where the similarity between images is represented by a distance function on their feature representations. This model requires that the feature space is a metric space. Other possible similarity models are the probabilistic model cited in [SWS<sup>+</sup>00], Tversky's model and the transformational distances model cited in [Bim99]. Measuring the similarity of aggregated features can also be performed by a distance function, such as the Minkowski distances for comparing color histograms, but also by nonparametric test statistics as mentioned in [SWS<sup>+</sup>00]. For comparing shape features, represented by points in space, the Hausdorff distance measure [HKRR93] has been employed. For processing spatial predicate queries different techniques for representing spatial similarity in images have also been proposed. Attributed Relational Graphs, different types of 2D-Strings, for example, have been used for representing spatial structures and different matching algorithms have been applied to compare these structures, e.g.  $L_p$  metric [PF97] or the Earth Movers Distance [KKYL04]. To compare images by similarity using their semantic level characteristics, techniques from the text retrieval field have been adopted, for example for comparing concept terms or annotations of images.

**Image Classification:** A second group of mechanisms for answering the types of queries defined above tackles the problem of interpreting the content of an image by assigning a meaning to an image through domain concepts. These mechanisms can be used to answer especially the group predicate queries. Their aim is to try to classify or cluster images, based on their characteristics, into classes of previously trained samples. Often an intermediate representation of the data is required apart from the retrieval operations in order to build these mechanisms. This task is similar to the feature extraction task of mapping low-level features onto semantic characteristics.

### Example

The retrieval task in the eNoteHistory CBIR application is to identify the scribe of a manuscript by comparing the handwriting of the given manuscript with the handwritings of manuscripts in the database with known scribes. The similarity between two between two manuscript images  $I$  and  $Q$  based on the handwriting characteristics can be represented by its opposite, i.e. the distance between these images. The distance between the whole images can be calculated by aggregating the distance measures on the image characteristics. Thus, the aggregated distance for the whole images can be represented with the following formula:

$$D(I, Q) = f_{i=1..l, j=1..m}(g_{k=1..n}(\delta_{f_k}(f_k^{r_i}, f_k^{r_j})))$$

where  $f_k^{r_i}$  is a feature, of a region  $i$  of a given query image  $I$  and  $f_k^{r_j}$  is a feature of the region  $j$  in an image  $Q$  from the database. The function  $g$  is an aggregation function which combines the distances from the distance measures  $\delta$  into a meaningful representation of the distance between the two regions  $i$  and  $j$ . The function  $f$  is an aggregation function which combines the distances between all regions of the images into a meaningful representation of the distance  $D$  between the images. This formula is explained in more details in section 4.3.3.1.

### Conclusion

Because of the big diversity of image retrieval mechanisms a generic retrieval functionality model can be induced only if it is abstracted from the concrete mechanisms. This means that a generic model needs to integrate retrieval functionality as concepts which can be adapted to support the different image retrieval mechanisms.

#### 3.1.2 Requirements Towards the Quality of the Model

The quality of the model is a very important requirement to consider in the development of a modeling approach, because the aim is to use the model for the automatic generation of the implementation. Quality goals should be defined in order to avoid ambiguousness and incompleteness of the model.

In the area of software engineering quality properties and metrics for software products have been studied extensively. A quality model for software products and metrics to achieve quality goals was issued as an ISO standard [ISO91]. In database research, models also play a very important role throughout the whole process of database design and implementation.

Models are used at different stages of the development: abstract models, such as the Entity-Relationship Model, are employed for the database design for capturing the requirements and thus is also referred to as a conceptual model; logical models, such as the Relational Model, are employed for the implementation logic of the conceptual models. In [Moo98] a set of metrics for evaluating the quality of Entity-Relationship Models has been defined. These metrics have been considered for the evaluation of conceptual data models in general. Some of them to name are: *completeness* - expresses whether the data model contains all information required to meet the user requirements; *integrity* - the extent to which the business rules (integrity constraints) which apply to the data are enforced by the model; *flexibility* - measures the effort for adapting the model according to changing requirements; *understandability* - reflects the ease with which data models can be understood by the users; *correctness* - ensures that the model adheres to the rules of the modeling technique being used, i.e. that it is syntactically and grammatically correct, and redundancy-free.

The evaluation of conceptual models with respect to quality is, however, still an often disputed research topic. Different frameworks have been proposed to determine quality goals and mechanisms to achieve them, but none has yet become a standard. For each specific purpose and requirement a different degree of fulfilling the quality goals may be desirable, thus quality has been regarded also in the sense of “fitness for use”. Therefore, the authors of [LSS94] have introduced a feasibility factor, which should be considered for each quality measure in such cases. They also offer a systematic classification of quality goals by introducing three kinds of quality:

- *Syntactic quality*: The more closely the model adheres to the language rules, the higher the syntactic quality. Syntactic errors occur when the statements in the model are not according to the syntax. These can be morphological errors or syntactic incompleteness.
- *Semantic quality*: It represents the similarity between the model and the domain (real world). Two semantic goals are *validity* and *completeness*. *Validity* expresses whether all statements made by the model are correct and relevant to the problem domain. In this case, *completeness* expresses whether the model contains all the statements about the domain. These goals are hard to achieve for complex domains, therefore the notions of feasible *validity* and *completeness* have been introduced, which consider the criteria satisfied as long as the costs for achieving them do not exceed the profit of having them. Other quality goals such as *consistence* and *unambiguity* have been considered as subsumed by the first two.
- *Pragmatic quality*: It shows how well a model can be understood or interpreted by its users. The only pragmatic goal is *comprehension*. Again the notion of feasibility was introduced to smoothen this requirement. The achievement of this goal is often related to additional support for annotation, extended visualization and other features supported by modeling tools.

The results from an empirical analysis of the framework published in [MSBS03], showed that syntactic quality has a 27% influence on the overall quality, semantic quality 39%, and pragmatic quality 34%, respectively.

The following quality criteria are considered for the modeling approach described in this thesis. These criteria are compiled from the quality goals for database models and conceptual

models in general, taking also into account the requirements towards an image model from the previous section.

- **Syntactic correctness** estimates to what extent the model adheres to all modeling language rules, in case these are known and well defined.
- **Validity** ensures that only concepts relevant to the problem are included in the model. These concepts are defined in the requirements from the previous section. The question is to what extent should domain-specific concepts be considered as relevant for the general model. Since one aim of the model is to support the designer in the development of domain-specific models any reusable concepts in the model may be useful. Therefore, including domain-dependent concepts in the model should not be regarded as violation of validity. However, such concepts must be defined as optional, so that they may be omitted in case that the application does not require them. The idea is to keep the resulting domain-specific model as compact and simple as possible.
- **Completeness** requires that the model contains all the statements about the domain. This quality feature is also based on the requirements for the model described in the previous section. The model should be able to represent all structural and operational features stated above. With regard to a general model this goal should be softened to assure that as few extensions as possible for the model are necessary when applying it to a specific model. The general model should provide more means for adaptation rather than for extending the model.
- **Flexibility** estimates the possibility to adapt or extend the model for a particular application domain. Well-defined interfaces and mechanisms for these processes should be offered by the model.
- **Understandability** estimates to which extent the model is understandable for users and if there are existing tools which can support the users in understanding the model, such as visualization tools. In this particular case of a general model, it is also important to provide well-defined mechanisms for using the model for deriving a domain-specific model. Usually, so called cookbooks, tutorials or supporting examples can be used for that.
- **Implementability** estimates how easy it is to map the model onto a specific implementation model (e.g., programming language). It proves whether there are existing mapping methodologies or do they still need to be defined.

### 3.1.3 Evaluation of Existing Conceptual Image Models

For the design of almost each existing CBIR system, a bespoke conceptual image data model has been used. These models have a lot in common, but very often they remain application specific, such as image models for the retrieval of medical or satellite images, images of human faces etc. Therefore, it has been an on-going aim for scientist to formalize a general image data model, which can be used for a broad range of application domains. In this section, a survey of some image models, which are considered as applicable for a broader range of applications and/or offer adaptability or extensibility possibilities is proposed. At first, models dealing

explicitly with the representation of images and their content are discussed. At the end, two multimedia models with extensive support for representing image content are considered. It is to note that a lot of the existing CBIR systems have not formalized their data models, that is why they are not mentioned in this survey.

Each of the listed models is represented through a detailed description, based on existing literature sources on the respective model. The eNoteHistory example application introduced in Chapter 1 and used as an example for formulating the requirements in the previous section, is used for the evaluation of the models. The idea is to check to what extent these models satisfy the quality criteria, defined in the previous section, by applying them for the modeling of a particular CBIR application. Completeness and validity are evaluated by proving whether all eNoteHistory data structures and functionality can be modeled. Flexibility is evaluated based on the effort which is required to extend the model in order to represent additional concepts or functionality etc. Finally, the evaluation of the models is summarized using the quality criteria defined in the previous section.

### 3.1.3.1 Conceptual Image Data Models

Several domain-independent image data models have been developed on a very abstract level in the early years of CBIR. A brief overview is provided in [GS00a] and [SB00].

#### 3.1.3.1.1 Adaptive Image Retrieval Data Model (AIR)

**Description:** Gudivada et. al proposed the AIR image data model in [GRV96] as a unified framework for retrieval in image databases which can be adapted for a class of image retrieval applications. The framework offers a domain independent data model and an associated retrieval model which can vary widely across domains. According to the authors, the image data model supports four types of retrieval:

- Retrieval by Browsing: the user searches for the needed information by narrowing the search space using visual query language techniques;
- Retrieval by Objective Attributes: in this case the queries use meta attributes (meta-data) and/or logical attributes to formulate the query which is later on evaluated as an exact match query;
- Retrieval by Spatial Constraints: this retrieval technique comprises exact and similarity queries based on spatial relations between objects found in the images;
- Retrieval by Semantic Attributes: the images in this case are matched based on their semantic attributes, which can be extracted automatically from the images or provided as domain specific annotations by a user.

However, the image processing and retrieval functions are outside the scope of the proposed framework. For the representation of the model a semantic data model proposed in [Urb87] is used. The semantic data model diagram of the AIR model is shown in Figure 3.5. The AIR image data model represents the image in several levels of abstraction - a physical representation and multiple corresponding logical representations, and transformations between them



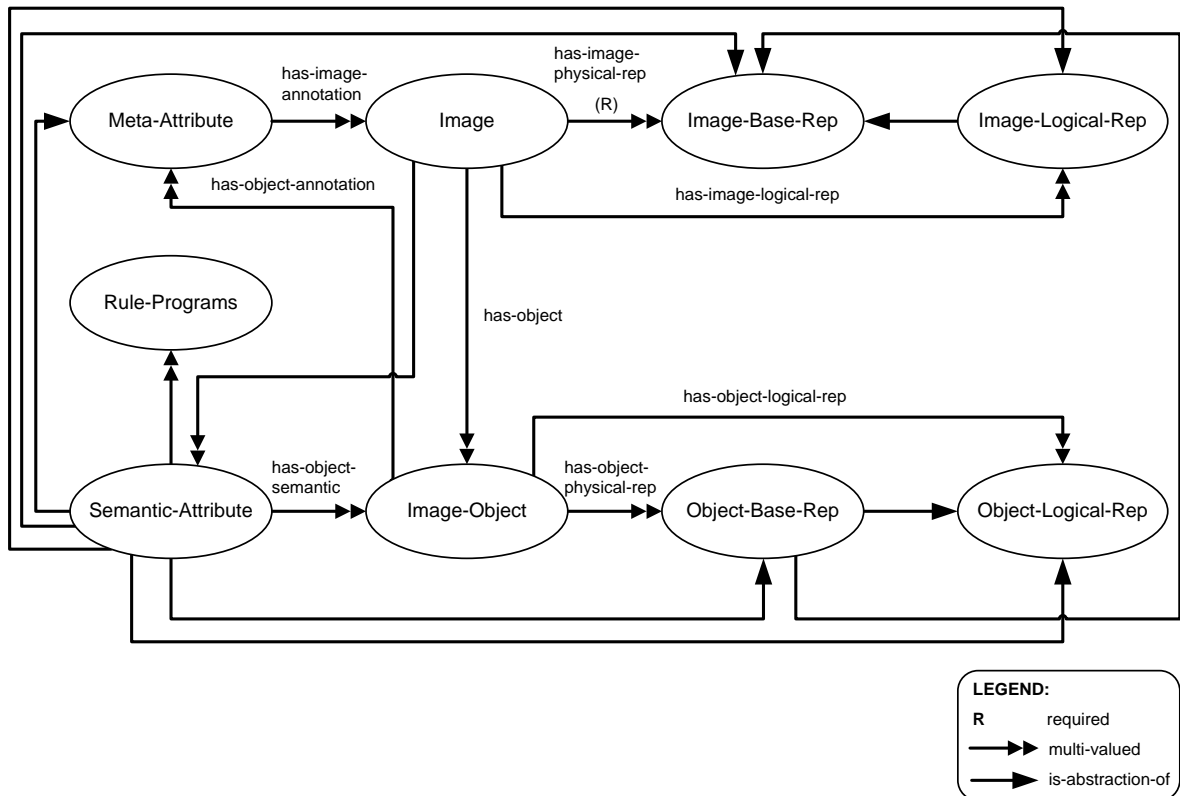


Figure 3.5: AIR Data Model (based on [GRV96])

in terms of domain-dependent image processing and interpretation techniques. The model consists of the following concepts:

**abstract classes:** *Image*, *Image-Object*, *Image-Base Representation*, *Image Object-Base Representation* are abstract classes corresponding to the images, the objects they comprise and their persistent storage presentations, respectively. Furthermore *Image Logical Representation (ILR)* and *Image-Object Logical Representation (OLR)* abstract classes are defined. *ILR* can be used to model spatial relationships and *OLR* to model geometrical attributes such as area, centroid, MBR, for example. The abstract class *Semantic Attribute* captures the high-level domain concepts and can be assigned to images and objects. These attributes can be derived by applying user-defined transformations, represented by the abstract class *Rule-Programs*, on the *Image-Base* and *Object-Base Representations*, logical attributes and also from instances of the *Meta-Attribute* class. Meta attributes are independent from the content of the image, they correspond to the content-independent image characteristics or metadata of an image described in section 3.1.1.1.

**properties:** Properties represent the relationships between classes. A *multivalued property* is represented by a double-headed arrow which corresponds to a set-valued functional relationship. A property may be defined as required **R** which indicates that the value set of the property must have at least one value. The original semantic model is extended with an additional construct - **isAbstractionOf** - which is used to model transformations between image representations. This construct represents the fact that an image representation is derived from another one by applying domain-dependent image-processing techniques.

**Evaluation:** The application of the AIR model was tested by the authors of [GRV96] for two application domains: real estate marketing and face retrieval. We will try now to apply this generic model for the music manuscripts example. We can represent the whole image and its physical storage structure as derived from the abstract classes *Image* and *Image-Base-Rep*, respectively. The ROIs and music objects can be represented as *Image-Objects*. However, the relationships between them cannot be represented in a straightforward way. Spatial relationships in the AIR model are represented by the *Image-Logical-Rep* class and have no relation to the *Image-Object* class. If we represent the objects in the *Image-Logical-Rep* we will not be able to associate with them any semantic or logical characteristics, such as music concept or shape. This relationship is not foreseen in the current generic AIR model. For determining the shape of these regions the **isAbstractionOf** relationship between *Object-Base-Rep* and *Object-Logical-Rep* may be used (In the image the arrow should be in the opposite direction). The concepts, which can be derived for each music object, can be represented by *Semantic Attributes* associated to the image-objects. However, semantic attributes can also be derived from other semantic attributes, e.g., a note can be derived from the combination of a note head and a note stem. This requires, as in the case of the spatial relationships between objects, a kind of relationship between the semantic attributes. *Meta-Attributes* can be used to represent bibliographical data related to the image. Feature extraction operations for extracting the objects of interest can be defined as **isAbstractionOf** between the *Image-Base-Rep* and *Object-Base-Rep*, but no further details about the operations can be given. Assigning a music concept to the objects can be represented by *Rule-Programs* for deriving semantic concepts from the logical attributes. For representing retrieval operations no concepts have been defined in the model.

In conclusion, it can be stated that the model fulfills the requirement for validity, however, it is not possible to represent some of the domain-specific concepts from the example application without further extensions to the model. In order to represent the specific requirements, such as spatial relationships between image objects, additional relations in the model are required. For representing retrieval operations new concepts have to be added to the generic model. Since the basis for the model is a semantic model, its implementation onto a specific platform, for example an object-oriented programming language, should be easily realizable. Understandability is also given only to a certain extent through the visualization possibility of the model. However, no guidelines for the application of the model for a specific domain are specified. It is not clear how the abstract classes should look like in a concrete model. No mechanisms for extending or adapting the model have been defined.

### 3.1.3.1.2 Visual Information Management System (VIMSYS)

**Description:** The VIMSYS [GWJ91] model proposed by Gupta, Weymouth, and Jain, combines object-oriented and functional models in a layered image model which represents the domain knowledge concepts corresponding to the image objects and their characteristics in four planes as shown in Figure 3.6. On the highest level the plane which represents the domain objects and the relations between them (DO) and the plane containing the domain events and their relations (DE) are defined. The intermediate plane consists of the image objects and their relations (IO). At the lowest level the plane comprising the so called representations and their relations (IR) is represented.

Representations in the IR plane are the underlying data types based upon which all the objects

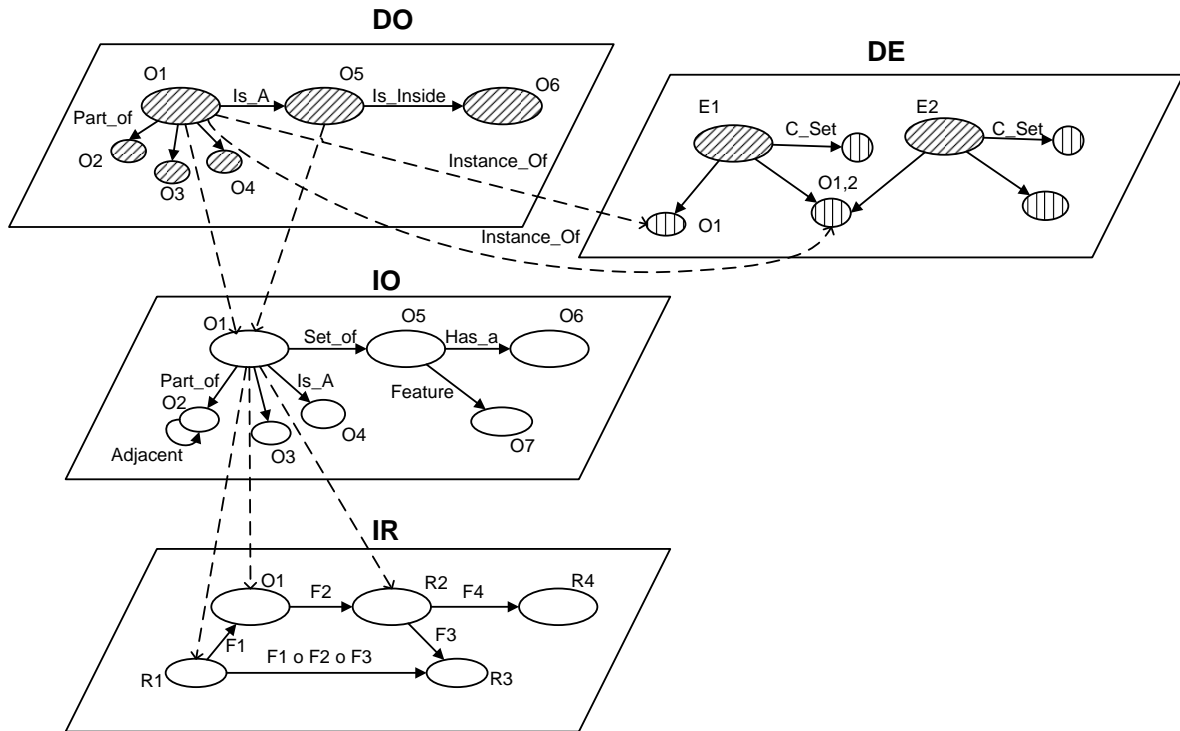


Figure 3.6: Layers of the VIMSIS image model (based on [GWJ91])

from the higher planes are constructed. They are interconnected in a functional model through functional relationships. Basic representations are, for example, *char*, *int*, *float*, *boolean* and *string* etc. The constructors are *set\_of*, *tuple\_of*, *vector\_of*, *matrix\_of*, *graph\_of*, *sequence\_of*. These constructors can also be used in the higher planes. A function is a separate data type, and can serve as the type of any relation. An image representation is an abstract data type for each image object in the IO plane. Multiple image representations for an image object are possible. New image representations and functions can be added without affecting the rest of the model. The IO plane comprises three basic classes of objects: image, image feature, and feature organization. These objects are interconnected in an object graph through *set\_of*, generalization (*is-a*), and *feature-of* relationships. Images are organized in image sets by the *set-of* relationship. An image consists of a set of regions which are members of the class *geometric feature*. Regions can themselves be related to other feature types, such as texture through the *feature\_of* relationship. The *geometric feature* allows organizing the regions of an image in an appropriate spatial structure. All features are members of the generic class *image feature* and can be further specified. Feature classes can also be combined in new ones by applying constructors defined in the IO plane or the additional *append* constructor, introduced in the IO plane. The DO plane consists of a semantic level specification of domain entities, built upon the two previous levels. The *objects* are connected through an object-relation graph. Any *object* in the DO plane may be a subclass or prototype of one or more *objects* in the IO plane. The DE plane is used to represent the *events* defined over image sequences. A *domain event* can be used to connect different *domain objects* by spatio-temporal relationships. For still images, however, it is not used to assign a temporal attribute to images, but to associate a description of an event represented in a sequence of

images, such as the rotation of ice blocks in the Arctic circle.

Query processing is not part of the VIMSYS model. It is designed in a separate module of the VIMSYS system. The data model is used to formulate the queries by allowing the users to determine which image characteristics should be used for processing the query. However, the query processing path is predefined and the end-user can influence the path by making certain decisions such as choosing a threshold or a distance function from a predefined library.

**Evaluation:** The model was elaborated as part of the VIMSYS project which examined the challenges for the design of three types of image retrieval systems: images of faces, newscast video clips, and satellite images of Arctic ice regions. The VIMSYS model offers a very extensive base which was used for the implementation of the Virage Image Search Engine. The Virage Image Search Engine is currently one of the leading players in the field of content-based video search. We can now try to build a model for our concrete example, based on the VIMSYS layered model. Since we have a concrete domain, we can start from the DO plane by defining the domain-specific objects we would like to represent with the model and then adapt or extend the lower level planes as needed. According to the requirements we need to represent a set of music manuscript images with their corresponding ROIs, where each image can consist of several ROIs. Each ROI can contain music objects, such as note heads, note stems, and bar lines. These objects can be organized in a hierarchical structure, such that one object can contain other objects. For each of these concepts, a corresponding object in the DO plane should be defined. They should be connected by arrows indicating that the source object contains a method which generates the destination object. These methods could be segmentation from music manuscript image to ROI, feature extraction from ROI to music object, i.e. to note head, note stem, bar line. In the IO plane images and their ROIs can be represented as the classes *image* and *region* connected by a *consists\_of* relationship. It is possible to represent the ROIs and the MBRs by the same class, but we should consider the fact that music object MBRs may have different attributes than the ROIs. Therefore, MBRs shall be regarded as features of ROIs. ROIs and MBRs need an associated organizational structure, for example a hierarchy. ROIs, MBRs, and their organization can be members of a generic geometric feature class. Each class contains its own attributes and methods which can be used to define segmentation and feature extraction algorithms. The content-independent data have to be represented as a kind of a feature which is associated to an image or an object of an image. At the IR plane we have to define the representation of the images, the ROIs and the music object MBRs as well as suitable organizations for each kind of regions, for example a hierarchy. Adaptable implementations for the methods can also be defined at the IR plane. Although the model does not foresee the integration of retrieval mechanisms an extension could be defined for incorporating these.

Concluding it can be stated that the VIMSYS satisfies to a great extent validity and completeness goals. It is, however, quite complex and requires a more detailed description which cannot be found in currently available articles on the topic. The existing articles provide a high-level overview of the model structure. Due to its complexity and usage of different types of modeling approaches at the different planes it is expected that the implementation of the model will not be straightforward. Flexibility with respect to adaptability is well supported through the independent-layers modeling approach. Extensibility to support retrieval operations, however, requires a more profound examination and understanding of the model which is difficult because of the lack of applicability examples or tutorials.

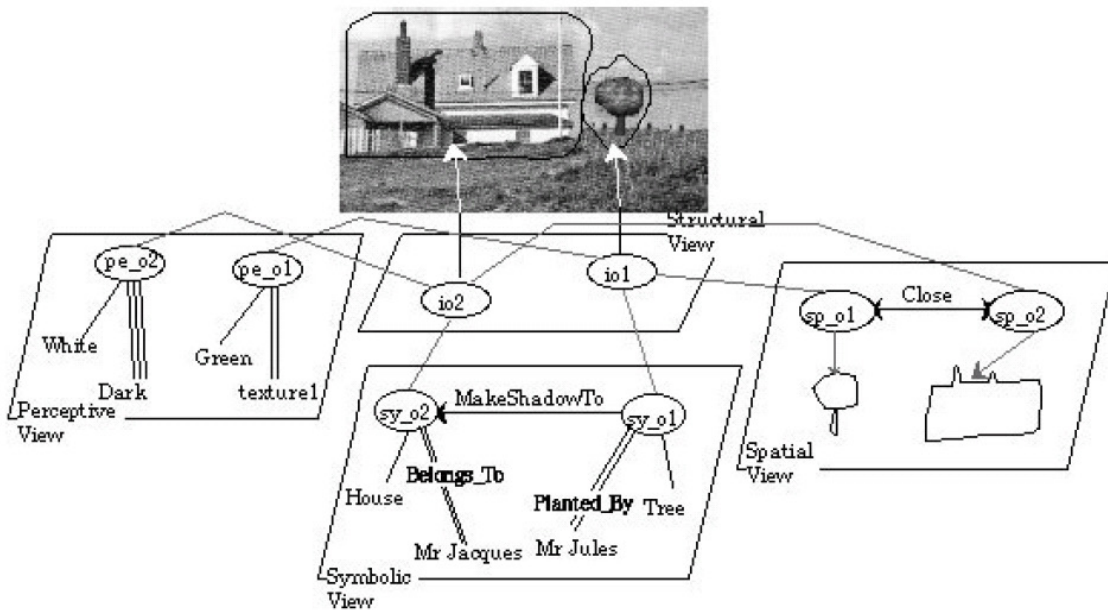


Figure 3.7: Example of describing an image with the EMIR<sup>2</sup> model (from [Mec95a])

### 3.1.3.1.3 Extended Model for Image Representation and Retrieval (EMIR<sup>2</sup>)

**Description:** Mourad Mechkour formalizes in EMIR<sup>2</sup> a general framework integrating all aspects of image content used for image retrieval. The aim of the framework model is to offer a generic structure that can be instantiated to fulfill application specificity. This framework is described in two subsequent articles [Mec95a], [Mec95b]. A general mathematical formalism and the Backus Normal Form are employed for the formalization of the framework in the articles, respectively. EMIR<sup>2</sup> combines different interpretations of an image building different views which correspond to different levels of abstraction: Physical view and Logical view (Structural, Spatial, Perceptive, Symbolic). Each view is based on a set of descriptors linked together by relationships specific to the view. In Figure 3.7, an example for applying the model for the description of an image of a house is shown. In [Mec95b], an implementation of the formal model as an extended conceptual graph is provided. The logical views are represented as canons in the conceptual graph formalism. The union of these canons constitutes the image model canon. The description of an image is then a canonical graph according to the defined canon. The physical view has not been transformed into the canonical graph model. A partial canonical graph of the house example is shown in Figure 3.8.

Here, in order to describe the different views the formalizations from [Mec95a] are used, because they express more details about the model than the ones in [Mec95b].

The **physical view** model is defined by the tuple:

$$\mathcal{M}_{ph} = (I_{ph}, \text{POINT}, EC, \text{TYPE}, h, w, tc, \text{pixels}, \text{type})$$

$I_{ph}$  is the set of physical view identifiers in EMIR<sup>2</sup>. POINT is the set of natural number pairs representing the cartesian coordinates of possible points. EC is the color set defined in

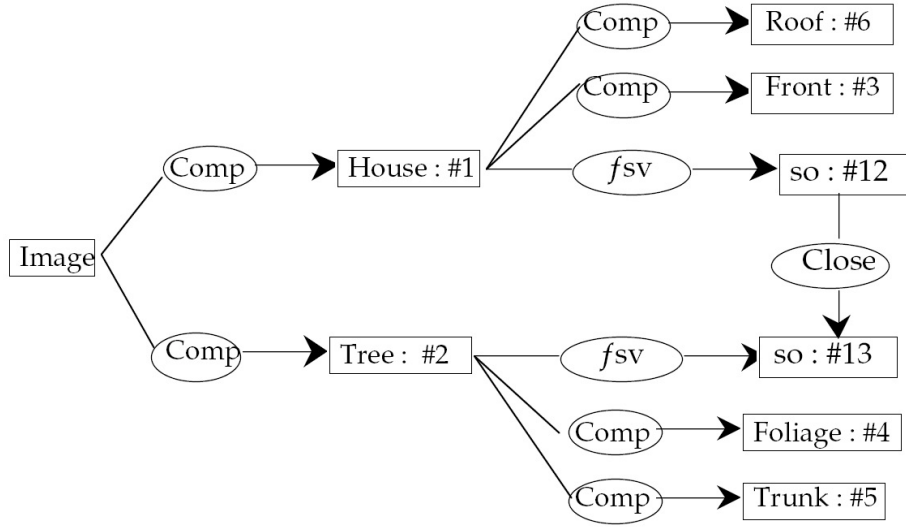


Figure 3.8: Image description example in EMIR<sup>2</sup>-CG (from [MBC95])

a particular color space. *TYPE* is the set of view types, such as Black&White, Grey Scale, Palette Color, True Color. *h* and *w* are functions which associate with each physical view identifier a number corresponding to the image height or width, respectively. *tc* is a function that associates with each  $I_{ph}$  a set of colors used in the corresponding image. *pixels* is a function which associates with each  $I_{ph}$  the set of pixels (an association of a point and a color) of the image. *type* is a function which associates the type of the image to each  $I_{ph}$ . Several constraints have been defined on the physical model, for example, the height and width of the image gives the number of pixels. The authors mention that two categories of operations to manipulate the physical view are defined, image processing functions and binary operations, however, these are missing in the formal representation.

The **structural view** of an image is represented by a directed graph where the nodes are the salient objects of the image and the arcs correspond to the composition relation between these basic objects. Formally, the structural view model is given by the tuple:

$$\mathcal{M}_{st} = (I_{IO}, CONT)$$

$I_{IO}$  is a set of image identifiers in the structural view and *CONT* is the composition relation between image objects which depends on the semantics associated to the image objects.

The **spatial view** of an image object represents the shape of the image objects and the spatial relationships that indicate their relative positions inside the image. It is defined by:

$$\mathcal{M}_{sp} = (I_{sp}, POINT, OS, RSPA, shape, R_{sp})$$

$I_{sp}$  is the set of spatial object identifiers. *POINT* is the set of integer pairs which represent the coordinates of all possible points. *OS* is the set of basic image objects that can be used to represent the shape of an image, for example, segment, polygon. *RSPA* is the set of spatial relations, for example, far, close, touch. *shape* associates with each  $I_{sp}$  its shape, for example, polygon, segment etc.  $R_{sp}$  is the relation that represents all possible spatial relations linking the spatial objects.

The **perceptive view** includes all the visual attributes of the image and/or image objects. It describes the appearance of the image components as perceived by an observer. In EMIR<sup>2</sup> three basic visual attributes are considered, the color, the brightness, and the texture. The perceptive view is defined by:

$$\mathcal{M}_{pe} = (I_{pe}, TX, BR, CL, tx, br, cl)$$

$I_{pe}$  is the set of perceptive object identifiers. TX, BR, CL are the sets of possible textures, brightness values and color values in the model, respectively. And the functions  $tx$ ,  $br$ ,  $cl$  associate a texture, brightness and color with the perceptive object identifier, respectively.

A **symbolic view** cannot be defined independent of the application, therefore, in EMIR<sup>2</sup> the symbolic view is defined as associations between an application semantic model and a set of abstractions representing the symbolic view. The application semantic model is defined by:

$$\mathcal{M}_{app} = (ID_{cl}, IS-A, ID_{pr}, ID_{rs}, VAL\_PROP, PROP, RSYMB, COMP, domain)$$

$ID_{cl}$  is the set of class identifiers organized by the *IS-A* relationship.  $ID_{pr}$  and  $ID_{rs}$  are the set of property identifiers and symbolic relation identifiers, respectively. VAL\_PROP is the set of possible values of the properties and *domain* the function which assigns these values to the properties. PROP and RSYMB are the set of property and symbolic relation definitions, respectively. COMP is the composition relation among classes.

The symbolic view model related to the above application semantic model associates with the set of symbolic objects their semantic interpretations:

$$\mathcal{M}_{sy} = (\mathcal{M}_{app}, I_{sy}, cl, RI, PI)$$

$I_{sy}$  is the set of symbolic objects identifiers.  $cl$  is the function that associates a class with a symbolic object identifier. RI represents the symbolic relations between the  $I_{sy}$ . PI is a relation that represents all properties associated with symbolic objects.

Relationships between the different views can be used to assign views to particular image objects from the structural view. As a result an **image model** is represented as an aggregation of all models and a set of relations that represent the inter-view dependencies:

$$\mathcal{M}_{im} = (I_{im}, \mathcal{M}_{ph}, \mathcal{M}_{st}, \mathcal{M}_{pe}, \mathcal{M}_{sp}, \mathcal{M}_{sy}, L_{sp}, L_{sy}, L_{pe})$$

In addition to representing the content of images in [Mec95a] a correspondence (similarity retrieval) model for EMIR<sup>2</sup> is proposed, which is given by the requirements of a query language and selection criteria for a correspondence function (similarity function). The selection criteria comprise an aggregation of constraints on the different views which resembles which objects of a view have to be considered in the similarity measure. The single constraints define the similarity measures for the objects in the different views.

**Evaluation:** The conceptual graph model has been implemented on top of the object-oriented DBMS (O2) as described in [MBC95]. According to the authors other operational models could also be used for the implementation of the formal model, such as object-oriented or first-order logic models. A model for the representation of music manuscript images can be defined as follows:

$$\mathcal{M}_{im\_ex} = (I_{im\_ex}, \mathcal{M}_{ph\_ex}, \mathcal{M}_{st\_ex}, \mathcal{M}_{sp\_ex}, \mathcal{M}_{sy\_ex}, L_{sp\_ex}, L_{sy\_ex})$$

The perceptual view is left out, because there is no need to store any perceptual characteristics for this type of images. The physical model is used to represent the pixel matrix of images on which image processing functions such as segmentation and feature extraction can be applied and included in the model. In the structural view model the ROIs and their associated MBRs can be represented by an identifier and composition relationships between them. Their spatial characteristics can be represented in the spatial view model and corresponding dependencies between both views. The higher level concepts associated to the objects can be represented by the symbolic view as follows:

$$\mathcal{M}_{app\_ex} = (ID_{cl\_ex}, IS - A, ID_{pr\_ex}, ID_{rs\_ex}, VAL\_PROP, PROP, RSYMB, COMP, domain)$$

$ID_{cl\_ex} = \{\text{note head, note stem, bar line, note}\}$

$ID_{pr\_ex} = \{\text{scribe}\}$

$ID_{rs\_ex} = \{\text{belong to the same note}\}$

$VAL\_PROP = \text{String}$

$domain(\text{scribe}) = \text{NScribe} \subset VAL\_PROP$ ,  $\text{NScribe} = \{\text{"Nicolai"}, \dots\}$  is the set of scribe names

$PROP = \{(\text{Scribe} \times \text{Image} \times \text{NScribe})\}$

$RSYMB = \{(\{\text{"belong to the same note"}\} \times \text{note head} \times \text{note stem})\}$

$COMP = \{(\text{note head, note}), (\text{note stem, note})\}$

For a music score image with one ROI, with one note consisting of a note stem and a note head, written by the scribe Nicolai the symbolic view model is defined as follows:

$$\mathcal{M}_{sy\_ex} = (\mathcal{M}_{app\_ex}, \{\text{sy\_o1, sy\_o2, sy\_o3, sy\_o4}\}, cl, PI, RI)$$

$cl(\text{sy\_o1}) = \text{Image}$ ,  $cl(\text{sy\_o2}) = \text{ROI}$ ,  $cl(\text{sy\_o3}) = \text{note head}$ ,  $cl(\text{sy\_o4}) = \text{note stem}$

$PI = \{(\text{scribe, sy\_o2, "Nicolai"})\}$

$RI = \{(\text{"belongs to the same note", sy\_o3, sy\_o4})\}$

The retrieval operations can be defined as constraints on the image model and the separate views. Content-independent data is not regarded as separate type of data in the model. It could be, however, represented through the concepts of the symbolic view.

With the expressiveness of the EMIR<sup>2</sup> formal framework, all structural and functional requirements of the example application can be met. The flexibility of the model with regard to adaptability and extensibility is well supported thanks to the fact that the model is based on a context-free description, close to a natural language description. However, no concrete suggestions for adapting the model to the special requirements of a special domain are given. Implementability with regard to other operational models such as an object-oriented model requires a more detailed investigation, as well.

#### 3.1.3.1.4 PIQ

**Description:** The Image Database Management System PIQ [SR96] designed by Shaft and Ramakrishnan, also recognizes the fact that existing image retrieval systems are either too specialized or too generic, and suggests a solution to the problem by introducing the idea



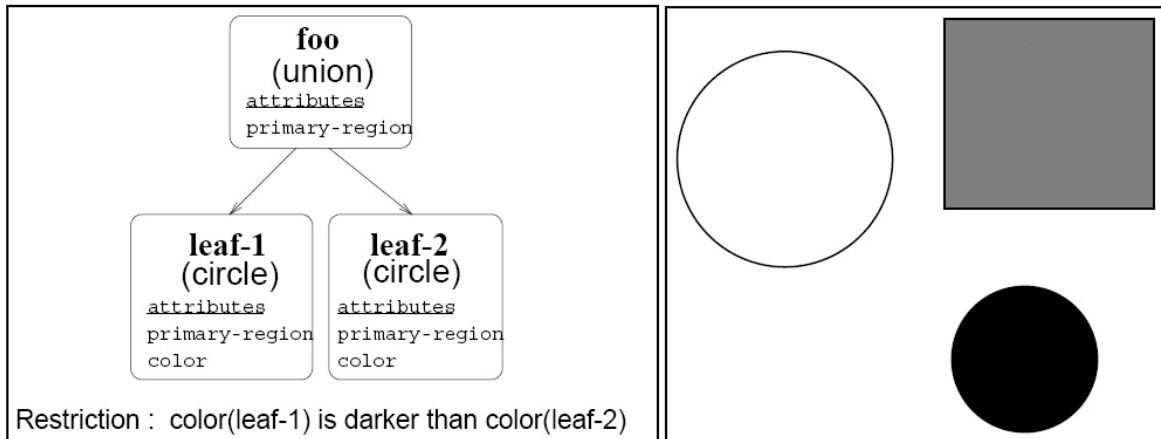


Figure 3.9: Example of a simple schema and an image (from [SR96])

of “image data modeling” based on an own Object Modeling Description Language(OMDL). The authors aim at using the modeling technique to provide more flexibility and guidelines in the feature extraction mechanisms and integrate semantics into the image descriptions in order to build a system which can deal with different types of collections simultaneously. A description of an image collection in terms of “image class” is represented by a so called *schema* - a tree structure where each node represents an object of the image. Each internal node has a type which determines how this node is derived (constructed) from its children. The types of internal nodes are **Union**, **Intersection**, **Difference**, **Set**, and **Or**. Each leaf node has a feature extraction algorithm associated with it which plays the role of a constructor for these kinds of nodes. The type of the feature extraction algorithm is the type of a leaf node. For a concrete application a summary tree is constructed for an image using the schema tree as a basis. Each node of the schema tree can have attributes of a predefined type which will correspond to values for these attributes in the summary tree. Restrictions can also be defined to specify legal summaries. Thus, a summary tree is always built for an image in the database in a bottom up manner and can be stored in an object-relational database such that each schema can be converted to a few relations and each summary can be converted to a set of tuples. The possible query types and mechanisms have been defined as SQL-queries on the object-relational presentation. In the left part of Figure 3.9, an example of a schema for describing a collection of images of the type illustrated in the right part of the figure is shown. Figure 3.10 depicts a summary of the example image, based on the given schema.

**Evaluation:** The main advantage of PIQ is that the model insures flexible adaptation to support new feature extraction algorithms and the resulting features. The bottom-up modeling approach corresponds to the application of image processing for the extraction and induction of image characteristics. However, retrieval algorithms are not considered in the model. They are defined for the first time at the implementation level in a database environment. For our test example we can define a schema which can be used to build summaries of music score images as shown in Figure 3.11. At first glance, the requirements for the representation of the data structures appear satisfied. However, there are problems representing a hierarchical structuring of objects from the same type. More important, it

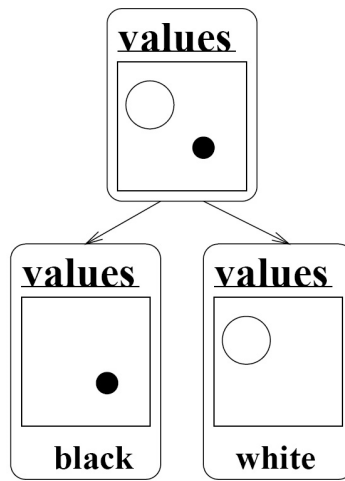


Figure 3.10: A summary tree of the example image from Figure 3.9(from [SR96])

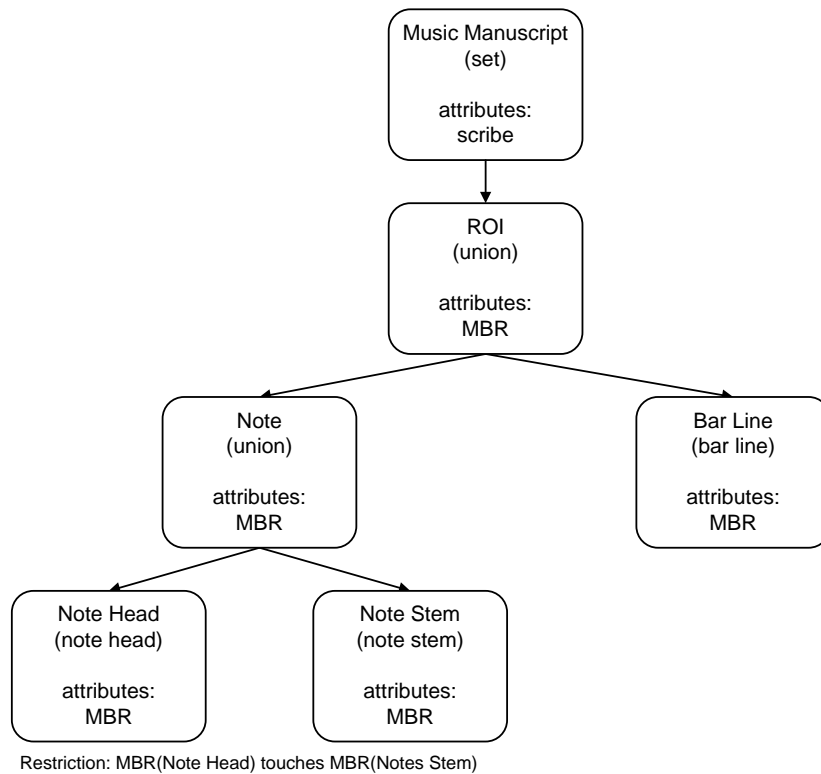


Figure 3.11: Example PIQ schema for music manuscripts

is not always a bottom-up approach for describing an image which has to be followed. For example the ROI of an image should be determined before the single objects from this ROI are extracted. The model is also difficult to extend or adapt for specific requirements of the application. Thus it does not fulfill the flexibility goal. Implementability in a relational database environment has been tested by the authors.

### 3.1.3.1.5 Distributed Image Database Management System (DISIMA)

**Description:** For the DISIMA project a mathematical formalization of a model was defined by the authors Oria, Özsu et al., in order to represent the content of an image which can support a wide range of queries [OÖL<sup>+</sup>97]. It has similarities with the VIMSYS in its representation of images and their content in layers as seen from Figure 3.12. The layers are grouped in two blocks: the image and the salient objects block. The image block consists of an image and an image representation layer which provide a so called representation independence in the model.

In the image layer an image type classification, such as a hierarchy of classes to which an image may be assigned (news image, medical image etc.), and functional relationships between the images can be defined by the user. The image representation layer defines the representation of the image in terms of format and type (raster, vector). An image in the DISIMA model is defined by the tuple:

$$\langle i, R(i), C(i), D(i) \rangle$$

where  $i$  is a unique image identifier,  $R(i)$  is the representation of the raw image,  $C(i)$  is the content of  $i$ ,  $D(i)$  is a set of descriptive alpha-numeric data associated with it. In DISIMA the content of an image is a set of salient objects and spatial relationships between them. The salient object block, which reflects this notion, consists of a physical, a logical salient objects layer, and a layer of salient objects representations. The physical layer contains the parts or regions (geometrical objects) of images with their properties such as shape, color, texture. A logical salient object from the logical layer is associated to a physical salient object and provides a semantic description for the physical object. Their representations are analog to the image representations and are stored in the salient object representation layer. The content of an image  $C(i)$  can then be defined as:  $\langle \mathcal{P}^i, s \rangle$ , where  $\mathcal{P}^i$  is a subset of  $\mathcal{P}$ , which is a set of physical salient objects and  $s$  is a function which associates  $\mathcal{P}^i$  with the corresponding logical salient objects from the set  $\mathcal{L}$ .

Furthermore, predicates were introduced to support queries for the different kinds of objects. For example, the *contains* predicate, which checks whether a salient object is found in an image, and the shape and color similarity predicates, which can also be combined and applied for comparing two images or salient objects with respect to their low-level features, are used. Additionally, spatial predicates on physical salient objects are supported.

In [OÖ03] the model was extended with the mechanism of image views. This extension enables the association of different logical salient objects to a physical salient object in order to allow the representations of different interpretations for the same image to support different context. A second extension allows an image to have different contents in terms of physical salient objects. For the implementation of the model an object-oriented database environment

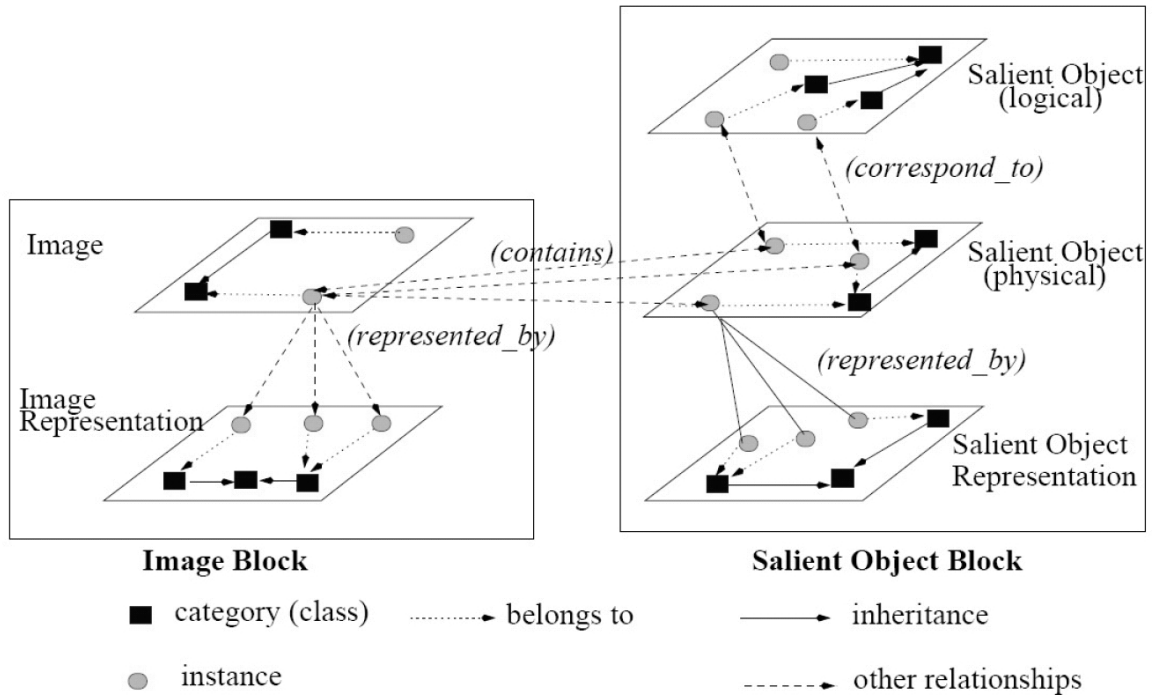


Figure 3.12: The DISIMA image model (from [OÖL<sup>+</sup>97])

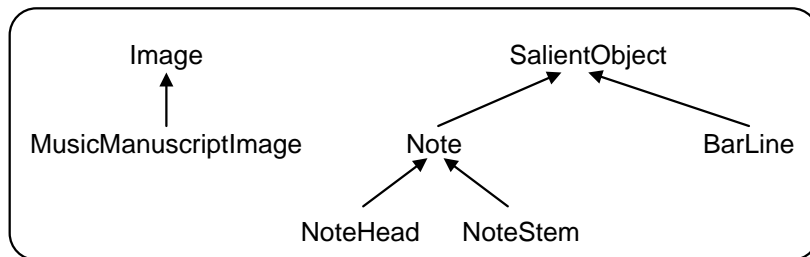


Figure 3.13: Image and Salient Object type classification for music manuscripts

was used. MOQL was used for query formulation and an ODMG Schema to implement the model.

**Evaluation:** The model has been implemented to support the storage, management and retrieval of images in the DISIMA system, which is built on top of an object-oriented database management system ObjectStore. However, the model does not cover the feature extraction functionality. This is a task of a separate module of the system. The retrieval module uses the MOQL language to query the image model, which on its turn uses the similarity predicates defined for different model components. We can now try to apply the model for representing our example as far as possible. At first, the type classifications for the image and logical salient objects layer are defined, as shown in Figure 3.13.

In the physical salient object layer objects corresponding to the MBRs of the ROI and the music elements can be defined as well as spatial relationships between them. The image and

salient object representation layers should contain objects corresponding to the JPEG format row images and salient objects, respectively. The automatic feature extraction operation can be defined as methods of the image and salient objects. Predicates for comparison by shape and for combining predicates can be defined to support the query mechanism.

### 3.1.3.1.6 Modeling Object-Oriented Data Semantics (MOODS)

**Description:** MOODS is an object-oriented modeling approach designed by Griffioen, Mehrotra, and Yavatkar, in order to support the modeling of a wide range of information types, such as multimedia data. In [GMY93b, GMY93a] the image information modeling related features are described. New features which this modeling technique, compared to the described models above, introduces are the integration of structures capturing the sequence of image transformations and the corresponding data. In addition, adaptable image processing functionality is integrated in the data model. To support these features in their object-oriented model, the authors have introduced new abstractions to the model: *dynamic data semantics*, *function groups*, respectively. Data semantics is an additional concept, which can be assigned to objects in the MOODS model, besides data structures and methods. It can change over the lifetime of an object depending on the current data in the data structures, and thus, allow to adjust the set of methods associated with the data. In this way, the model can accommodate different application domains through associating various semantics as well as storing dynamic semantics, corresponding to the changes of the semantics with time.

To model the image processing components of the system, the operations for image processing are grouped into classes corresponding to the different stages of the processing sequence: image enhancement, region segmentation, boundary detection, image display, primitive shape detection, domain dependent labeling, domain dependent analysis. Applying each of these function classes to the image produces new information which can be used to represent the image. Applying different sorts of algorithms corresponding to these classes can provide different views or interpretation of the image data. The groups of functions are defined as global to the model and are accessible by all classes in the model. A set of methods in a function group does not have to have the same input and produce the same output, however, they should provide similar logical operations. The binding of a specific function from the function group to a class is done at run time. This allows for the dynamic exchange of algorithms. With this concept the model tackles the problem of defining abstract operations to represent generic functionality.

The image structure derived from applying image processing functions can be represented by semantic classes in the model. These objects are connected by arrows representing the transformation functions between the different stages in the lifecycle of the image. An example for applying the model for the two different application domains “Industrial Images” and “Medical Images” is shown in Figure 3.14. Each circle represents a semantic class, with data structures, data semantics and specific semantic functions. The lines between circles represent the application of a particular function. The example is supposed to demonstrate the flexibility of the system to adapt the processing to each application domain. With this model, the semantics at each stage of the image processing can be saved and used by the retrieval or for new analysis of the images.

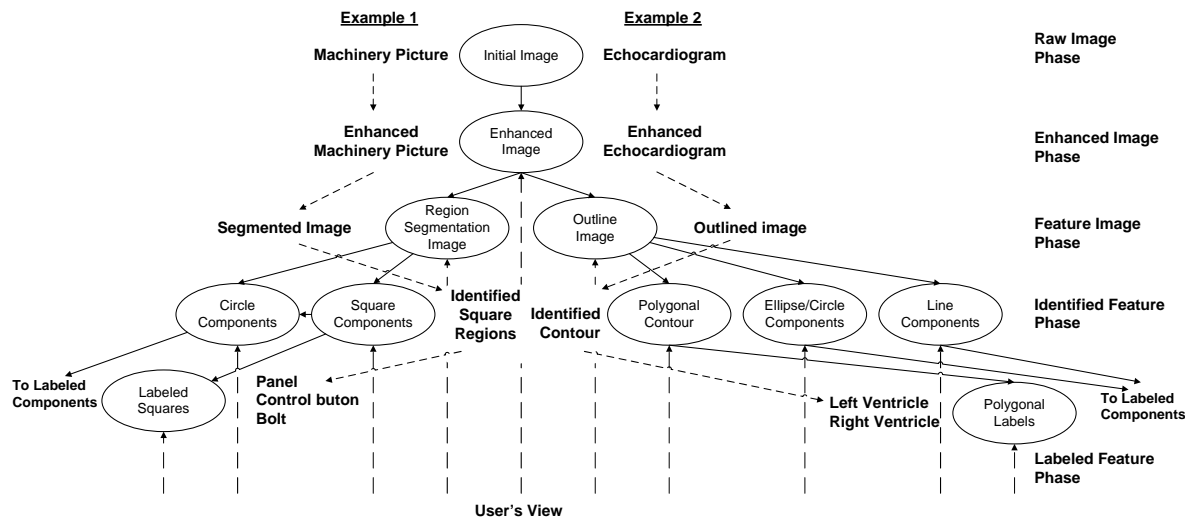


Figure 3.14: MOODS image information processing system (based on [GMY93b])

**Evaluation:** The description of the model in [GMY93b] focuses mostly on the newly introduced concepts and does not give an exhaustive definition of a generic image model structure as the models discussed before. Only the example shown in Figure 3.14 gives a hint about how an image model can look like. Contemporary object-oriented models have other mechanisms for defining abstract functionality, such as *interfaces*, but the newly introduced concepts to the MOODS object-oriented model demonstrate an interesting approach for solving the problem of integrating adaptability in the model. The semantic model of MOODS has been implemented in an extended C++ language with the same name - MOODS. By coincidence, the authors of the model describe in [GMY93a] the application of their model for the recognition of music manuscripts, which corresponds to the feature extraction part of our test example. This model is illustrated in Figure 3.15.

The model does not deal with the problem of retrieval at all. In fact it represents a model which can be used for supporting an image processing system, rather than an image retrieval system. Integrating retrieval functionality could be achieved by defining new function groups for the classes and perhaps applying them in the opposite direction of the image processing functions.

### 3.1.3.1.7 Other

In [ACB02], an image data model for medical image databases is proposed, that describes image data in several levels of abstraction in terms of External Space, that describes the general information associated with an image, which is not related to the content of the image, such as context-oriented, domain-oriented, and image-oriented data. The Content Space summarizes the content-related data such as physical, spatial and semantic features. The model considers the visual feature description for still images of the MPEG-7 and DICOM standards.

In [MSS97], an image retrieval model was defined using a kind of description logic, namely the *ALC* logic. The model represents images both at the physical level (form level) and at the

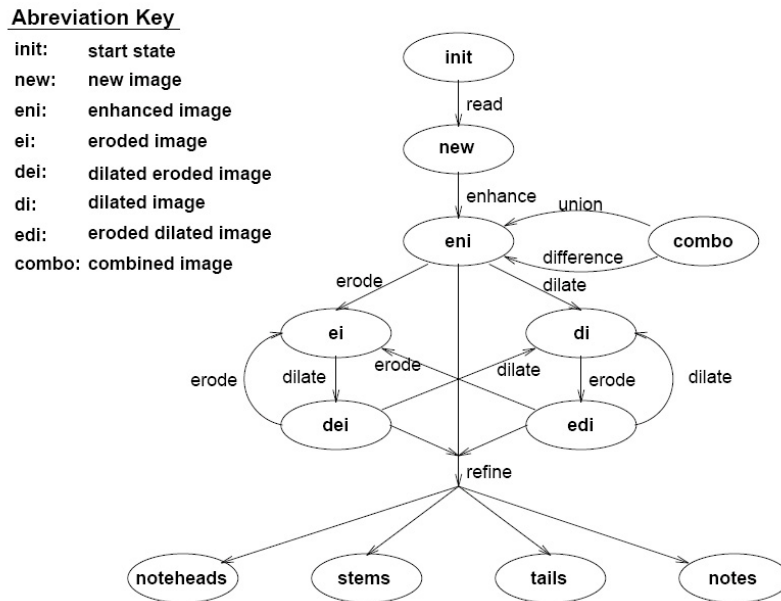


Figure 3.15: MOODS semantic model for music manuscript recognition (from [GM93a])

content level, as a set of logical assertions about the represented entities and about facts of the subject matter used for the retrieval. Physical features are not represented explicitly in the description logic, because it is better to manage them through some digital signal processing techniques. They are used in the logical reasoning through a mechanism of “procedural attachments”. For each feature an image layout is defined, which consists of a triple: a main region, a partition of this region into non-empty connected regions - (atomic regions), and a function assigning a value from the feature domain to each atomic region. The content of the image is represented through description logic formalisms: primitive, concepts, primitive roles, concepts and assertions, extended with fuzzy assertions to support uncertainty in retrieval. A content description for an image is a set of fuzzy assertions resulting from the combination of four components: the image identification, the object anchoring - identifying the region of the image and the object depicted by the region, scene anchoring - what the whole scene shown in an image is about, and the scene description - describes important facts about the image. The queries addressing an image form or content are formulated through special predicate symbols (SPS). SPS for global and local matching of images were defined. This modeling style is an application of the integrating of concrete domains in description logic, where certain symbols have an interpretation-independent meaning. Practically each occurrence of an SPS can be seen as a call to a routine which implements the needed image processing technique - procedural attachment. In [MSS01] this model was extended to a multimedia retrieval model. The General Image Data Model proposed by Stanchev and Grosky [GS00a] resembles a taxonomy summarizing existing data modeling approaches (EMIR<sup>2</sup>, AIR, VIMSYS) in a so called semantic schema. The taxonomy covers a physical and logical view of an image. The logical view is further specified in global and content-based view. The physical view can be an image header and/or an image matrix. The global view consists of meta- and semantic-attributes of an image. And the content-based view is represented by objects further specified by color, texture, shape and spatial. The relationships in this taxonomy are of the type multivalued-

abstraction. This model resembles a higher-level model, which aims to be applicable for the presentation of wide variation of image collections. However, it does not deal with modeling functionality.

Another modeling technique utilizing XML schema given in [TXRN06], describes still images using hierarchical tree structures similar to the summary trees of the PIQ model above [SR96] and object-relation graphs. Both of these graphical representation have been proposed earlier, but this work shows how to represent them in an XML Schema.

### 3.1.3.2 Multimedia Models with Extensive Support for Images

Digital images have been also modeled as part of broader data models, such as the multimedia data models. In [IB03] different kinds of multimedia models have been studied. External multimedia such as SMIL, MHEG, HyTime and HTML, ZyX [BK01] as well as logical multimedia document models reflecting aspects of different research fields, such as computer vision in Chapter 2 of [WHKL00], information retrieval [CMF96] and databases [MSS01] (which was already discussed in the previous section) have been reviewed. External models are used for representing the gathered data, before storing it into a system, or for displaying the results of a multimedia query. These models deal with multimedia data in raw formats. The logical models have the task to convert the multimedia data into a presentation which can be managed and queried efficiently by the multimedia information system. The logical models are the types which are of interest for the current study. In this section, two multimedia logical models are analyzed in respect to their ability to represent image data and support content-based retrieval on these data. The first model is a representative of the more theoretical approaches suggested by Marcus and Subrahmanian. The second approach is the one used by the MPEG-7 standard.

#### 3.1.3.2.1 Multimedia Database Systems Formalization - Marcus, Subrahmanian

**Description:** In [MS93], a framework for integrating individual media implementations in a multimedia information system in order to provide a unified logical query language for these data has been defined. One challenge which this framework tries to solve is to provide an answer to the following question: “What are multimedia database systems and can they be formally defined so that they are independent of any specific application domain?”. The basic concept which is introduced by the framework is a **media-instance**. A media-instance consists of a body of information represented using some storage mechanism in some storage medium, together with some functions and/or relations expressing various aspects, features and/or properties of this media-instance. A media-instance can be used to represent an image, a document, a video and even more general media types such as matrices, quad-trees, object-oriented media-instances etc. A multimedia system is defined to be a set of such media-instances.

The formal definition of a media-instance represents it as a 7-tuple:

$$mi = (\mathcal{ST}, fe, \lambda, \mathfrak{R}, \mathcal{F}, Var_1, Var_2)$$

where  $\mathcal{ST}$  is a set of objects called states,  $fe$  is a set of objects called features,  $\lambda$  is a map from  $\mathcal{ST}$  to  $2^{fe}$ ,  $Var_1$  is a set of objects called range variables ranging over states,  $Var_2$  is a



set of objects called feature variables ranging over features,  $\mathfrak{R}$  is a set of interstate relations on the set of  $ST$ , and  $\mathcal{F}$  is a set of feature-state relations. Each relation in  $F$  is a subset of  $fe^i \times ST$  where  $i \geq 1$ .

For this media representation a query language has been defined. Each query is an existentially closed conjunction of atoms. An atom is composed of predicates and variable-free terms. Each constant symbols, such as  $f \in fe^i$  or  $s \in ST^i$  and the combination of an n-ary function symbol  $\eta$  with other terms,  $\eta(t_1, \dots, t_n)$ , are terms in the query language. In Chapter 9 of [Sub98], the media-instance tuple consists of one additional component ATTR, which represents a set of objects called attribute values, which can be associated to a feature or a state object, for example to represent some content independent data for an image - the date on which the image was shot, by whom was it shot etc. In this book, the media-instance is called a media abstraction and the two relationship sets  $\mathfrak{R}$  and  $\mathcal{F}$  have exactly the opposite meaning:  $\mathfrak{R}$  is a set of relations between features, attributes and states and  $\mathcal{F}$  is a set of relations between states.

**Evaluation:** In Chapter 9 of [Sub98], image data is described as an instance of a media-abstraction. We will consider this version of the model for representing the music manuscripts example. The *states* which have to be considered as the smallest piece of media data in our example are image regions. These regions have to be organized in a hierarchy in order to resemble that a ROI consists of MBRs. We can introduce also an image region representing the whole image. The set of features, which we have to define is on the one hand the high-level features such as, note head, note stem, bar line, and on the other hand a set of integer pairs representing the shape of the ROIs and MBRs.  $\lambda$  shall contain the implementations for the extraction of features from the image regions. The meta data, such as the names of the scribes can be defined as attribute values. The relationships between regions, features and attribute values can be represented as the set  $\mathfrak{R}$ . However, retrieval methods are not supported by the model.

### 3.1.3.2.2 MPEG-7

**Description:** One of the most prominent multimedia conceptual data models is the one defined by the MPEG-7 standard [SS02]. The main focus of the MPEG-7 standard is to assure the interoperability between applications and devices by determining a set of features (multimedia characteristics) which have to be described and by providing a way to organize, structure and represent them with a common language. The MPEG-7 standard offers an extensive multimedia content description interface, for media-specific descriptors (such as visual descriptors), as well as media type independent Multimedia Description Schemes. The Description Definition Language (DDL) is used to define or extend the latter. “[...] The DDL is not a modeling language such as Unified Modeling Language (UML), but a *schema* language to represent the results of modeling audiovisual data, (i.e. descriptors and description schemes) [...] It also provides the syntactic rules by which users can combine, extend and refine existing description schemes and descriptors to create application-specific description definitions or schemes [...]” [SS02]. An MPEG-7 schema defines a class of MPEG-7 documents. The instances of this schema are XML documents which describe concrete multimedia documents and conform to the MPEG-7 schema. As a DDL, MPEG-7 employs an XML Schema Language with MPEG-7 specific extensions, such as the array and matrix data

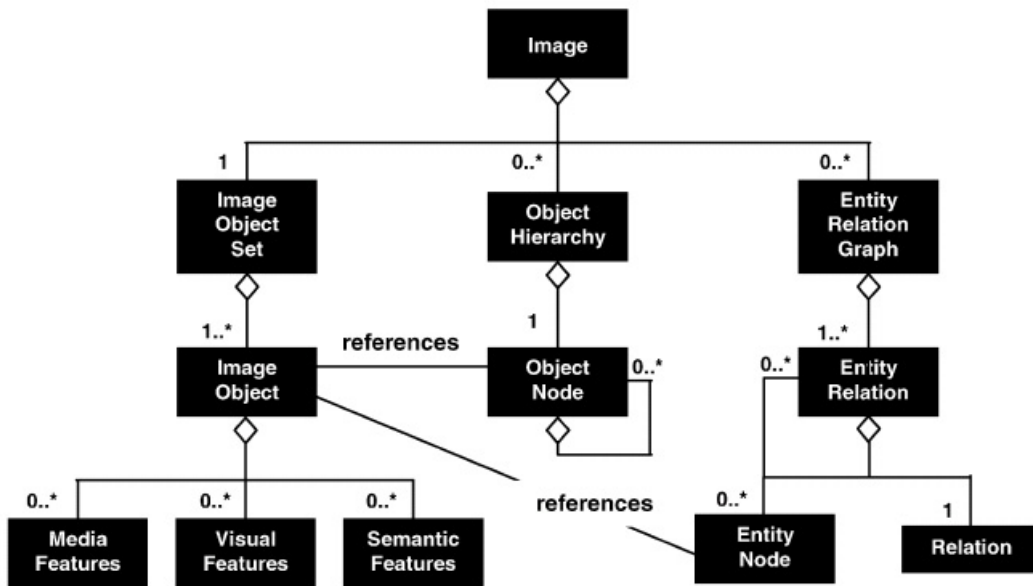


Figure 3.16: UML representation of the image description scheme (from [BPC<sup>+</sup>00])

types. Description schemes and descriptors are grouped in MPEG-7 tools. For the description of an image, description schemes from different tools may be required, such as creation information tools, still region tools, segment decomposition tools, semantic entity tools etc.

Most of the critics of the MPEG-7 model come from the usage of XML Schema as a DDL. XML Schema does not provide the right structure for managing the data efficiently in a database, as mentioned by Kosch in [Kos02]. A possible solution to the problem is offered in the doctoral thesis of Westerman [Wes04], in which the author represents a Typed Data Object Model as a generic data model for XML documents. In [TCLP04], the following further problems of MPEG-7 are pointed out:

- not possible to define new descriptors
- derivation of new types does not match the object-oriented concept
- lack of modularity
- no formal semantics.

During the elaboration of the standard, the employment of a conceptual modeling technology has been discussed in some publications. In [SB00], the usage of the Extended Entity-Relationship model has been applied to model the MPEG-7 concepts, with the perspective to use this model to generate an implementation as description schemes, database schemes or software classes. In [BPC<sup>+</sup>00], description schemes including one for still images have been elaborated as conceptual models, independent from the implementation using UML. The implementation has been then illustrated using XML. In Figure 3.16 an UML representation of the proposed image description scheme is shown.

**Evaluation:** For our test example we could give a description of a music manuscript document using the existing schemas for describing image content. This is not a trivial task, because these schemata are quite complex, since the aim is to support all possible application domains. New descriptors require very good understanding of the MPEG-7 schema. A more preferable approach would be to define a more compact and simpler schema for the concrete domain and then describe the documents according to the simple schema. An example of an MPEG-7 description of a music manuscript document is given below.

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
5   xmlns:xml="http://www.w3.org/XML/1998/namespace"
6   xsi:schemaLocation="urn:mpeg:mpeg7:schema:2001 .\Mpeg7-2001.xsd">
7 <Description xsi:type="ContentEntityType">
8   <MultimediaContent xsi:type="ImageType">
9     <Image id="I1">
10       <MediaLocator>
11         <MediaUri>file://musicmanuscript.jpg</MediaUri>
12       </MediaLocator>
13       <CreationInformation>
14         <Creation>
15           <Title> Musica Saec. XVIII 29.1 </Title>
16           <Creator>
17             <Role><Name>Scribe</Name></Role>
18             <Person><Name> <GivenName>Nicolai</GivenName> </Name> </Person>
19           </Creator>
20         </Creation>
21       </CreationInformation>
22       <SegmentRef idref="sem" />
23       <SpatialDecomposition gap="true" overlap="true">
24         <StillRegion id="SR0">
25           <SpatialLocator>
26             <Box><Coords dim="2 2"> x0 y0 w0 h0</Coords></Box>
27           </SpatialLocator>
28           <SpatialDecomposition gap="true" overlap="true">
29             <StillRegion id="SR1">
30               <SpatialLocator>
31                 <Box><Coords dim="2 2"> x1 y1 w1 h1</Coords></Box>
32               </SpatialLocator>
33             </StillRegion>
34             <StillRegion id="SR2">
35               <SpatialLocator>
36                 <Box><Coords dim="2 2"> x2 y2 w2 h2</Coords></Box>
37               </SpatialLocator>
38             </StillRegion>
39             <StillRegion id="SR3">
40               <SpatialLocator>
41                 <Box><Coords dim="2 2"> x3 y3 w3 h3</Coords></Box>
42               </SpatialLocator>
43               <Relation xsi:type="DirectionalSpatialSegmentRelationType" name="right" target="#SR2"/>
44             </StillRegion>
45           </SpatialDecomposition>
46         </StillRegion>
47       </SpatialDecomposition>
48     </Image>
49   </MultimediaContent>
50 </Description>
51 <Description xsi:type="SemanticDescriptionType">
52 <Semantics id="sem">
53   <Label>
54     <Name>Semantic description of a music score</Name>
55   </Label>
56 <SemanticBase xsi:type="ConceptType" id="C1">
57   <Label>
58     <Name> hote head</Name>
59   </Label>
60   <Relation type="urn:mpeg:mpeg7:cs:SemanticRelationCS:2001:symbol" target="#SR1"/>
61 </SemanticBase>
62 <SemanticBase xsi:type="note stem" id="C2">
63   <Label>
64     <Name> hote head</Name>
65   </Label>
66   <Relation type="urn:mpeg:mpeg7:cs:SemanticRelationCS:2001:symbol" target="#SR2"/>
67 </SemanticBase>
68 <SemanticBase xsi:type="bar line" id="C3">
69   <Label>
70     <Name> hote head</Name>
71   </Label>
72   <Relation type="urn:mpeg:mpeg7:cs:SemanticRelationCS:2001:symbol" target="#SR3"/>
73 </SemanticBase>
74 </Semantics>
75 </Description>
76 </Mpeg7>

```

To describe the structural content of the example image, the description of type `ContentEntityType` and the corresponding semantic view with the `SemanticDescriptionType` description are used. Even the content-independent data can be represented with the `CreationInformation` element. Thus, we can conclude that the MPEG-7 description schemes and descriptors are exhaustive enough to represent our example. In fact, they offer much more representation possibilities. This makes the effort to comprehend them too big in comparison with the gain that we have of building such a small example. Functionality is, however, not part of the MPEG-7 description schemes. Feature (descriptor) extraction functions and similarity metrics have been suggested for the experimentation initiative of the standard, but they are not standardized or included in the MPEG-7 model. Adaptability and extensibility are limited to adapting the schemas and descriptors. However, adding new descriptors is not possible.

### 3.1.4 Conclusions

The evaluation of the above models with regard to the predefined quality goals is summarized in the following paragraphs. A compact summary of the criteria for each model is provided at the end of this subsection in Table 3.1.

#### 3.1.4.1 Syntactic quality

This quality goal, depends on the modeling language used for representing the models. Each of the discussed models uses a different modeling paradigm. Sometimes they use standard modeling techniques, such as semantic models or XML schema, but there are also models which define their own mathematical formalisms or modeling languages. EMIR<sup>2</sup> and the model of Marcus and Subrahmanian, for example, use low-level modeling approaches such as mathematical formalisms and then map them onto an operational model, such as conceptual graphs. AIR and MOODS use a single modeling paradigm, in this case, a semantic model and an extended object-oriented model, respectively. VIMSYS and DISIMA follow a more sophisticated methodology by combining functional and object models and mathematical formalisms in one model. PIQ and MPEG-7, on the other hand, have chosen a schema language to define the structure of an image representation, employing an Object Modeling Description Language (OMDL) and XML Schema, respectively. The analysis did not find any concepts in the models which do not adhere to the corresponding syntactic rules of the modeling paradigms.

#### 3.1.4.2 Implementability

The possibilities to implement these models depend to a great extent on their corresponding modeling paradigms. The low-level modeling approaches need to adopt an intermediate operational model additionally. EMIR<sup>2</sup> has chosen conceptual graphs as an operational model and the object-oriented database system environment of O2 for the implementation platform. The model of Marcus and Subrahmanian is defined as a theoretical framework for the analysis of multimedia systems and no implementation for it is provided. The examples of the AIR model have been implemented using the object-relational database system POSTGRES as a backend. MOODS uses an extended version of the C++ programming language for the implementation of the model. The VIMSYS model has been implemented as the center of the Virage Image Search Engine. The Virage Engine is implemented as an extensible module,

which can be integrated in different applications. For example, it has been combined with the object-relational database *Illustra*. The *DISIMA* model has been implemented within the object-oriented database system *ObjectStore*. The authors of the *PIQ* model have demonstrated its usage based on a relational database implementation. And finally the model, which has the largest number of implementations, since it is also a standard, is the *MPEG-7* model. Its XML-based representation suggest that for best efficiency a native XML storage and querying environment such as an XML database system would be the best implementation platform. However, there have been also object-oriented implementations, for example in the *VizIR* framework. To determine to which extent the models can be implemented on other platforms, a more profound investigation is required.

#### 3.1.4.3 Validity and Completeness

All of the above models offer good support for representing image data and structure. There are, however, some missing concepts, for example, spatial relationships between images in the *AIR* model, explicit support for content-independent data in *PIQ*. These could be added to the models in a relatively straightforward way by appropriate extensions. Most of the models are kept compact by focusing only on the general characteristics of images. Only the *MPEG-7* model tries to be as exhaustive as possible by the definition of image descriptors. Nevertheless, these can be freely used or omitted from the concrete representations. With respect to functionality, however, most of the models have not integrated all necessary functionality for content-based image retrieval as corresponding modeling concepts. Exceptions make the mathematical formalisms in *EMIR*<sup>2</sup> and the model of Marcus and Subrahmanian, which support the definition of different functions on the sets of objects representing the image content. However, the mapping of these functions onto an operational model has not been discussed. *MOODS* and *PIQ* offer support for modeling feature extraction, functionality, but do not represent retrieval functionality at all. *AIR* and *VIMSYS* can represent feature extraction functionality through relationships between objects. These concepts, however, are described in few details in the existing articles. Extending the models to support all needed functionality is not a trivial task, compared to additionally adding structural elements, because some modeling paradigms, such as the XML Schema does not support the representation of functionality at all, and others need the integration of new structures, which have to fit in the existing data structures.

#### 3.1.4.4 Flexibility

The flexibility of the models can be examined with respect to two aspects - extensibility and adaptability. Only the *MPEG-7* model from the above defines concrete interfaces for extending and adapting the modeling structures, through refining description schemas and descriptors. However, there are some limitations and drawbacks of the extensibility mechanisms mentioned in [TCLP04], such as, no possibility to define new descriptors, the derivation of new types does not match the object-oriented concept, lack of modularity, no formal semantics. The mathematical formalism models also offer a relatively straightforward way for adapting or extending them. First, they define modeling concepts with the help of sets of objects, which can be further extended or refined, depending on the concrete application. Secondly, because of the natural language form of the modeling approach, new mathematical formalisms can always be added as long as certain constraints are met. *PIQ* and *MOODS*

use their own modeling languages, which are not in detail described in the available articles. They do not offer explicit extensibility interfaces and thus do not satisfy the flexibility goal. VIMSYS and DISIMA aim at providing easier extensibility and adaptability of the model through the layered architecture which separates the different modeling stages. Both models do not describe general models for the different layers, but rather give partial examples how the application specific models for the different layers may look like. Thus, the description of the general model consists of mainly only of the layered architecture and no derivation or reuse of a model at each layer is described. The AIR model provides such a general model consisting of abstract classes, but it is not clear how this model can be adapted or extended to represent an application-specific model. Therefore, it can be concluded that none of the discussed models provides fully fledged extensibility and adaptability possibility, thus, they do not fulfill the flexibility goal.

#### 3.1.4.5 Understandability

This goal depends on the one hand to a great extent on the modeling paradigm, used for the representation of the model. On the other hand, it depends on the availability and quality of definitions and descriptions for the usage of the model for deriving domain-specific models. Mathematical formalisms are difficult to comprehend, because there are no visualization tools and the mapping onto some machine-understandable information representation is still needed as an intermediate step towards implementation. Schema languages are easier to understand until they expand a certain size. Visualization tools for these kind of languages are available for example for XML-Schema. Semantic and object-oriented models also have their visualization and case-tools support, however, mixing object models and functional models such as is the case in the VIMSYS model additional support for dependencies between these models is required. Understanding a general model from the view point of the developer of CBIR systems includes not only understanding of the modeling paradigm, but also understanding how this general model can be applied to derive a concrete domain-specific model. MPEG-7 has been most often implemented, and therefore, there are a lot of examples where the application of the model can be understood. All other models have mentioned examples of employing the models for certain domain-specific applications, but have not shown how the concrete domain-specific models are derived from the general models. Thus, none of the models defines a methodology on how to apply (extend, adapt) them for domain-specific applications.

Table 3.1: Evaluation of the Image Retrieval Models

Criteria Model	Implementability	Validity and Completeness	Flexibility	Understandability
<b>AIR:</b> semantic data model	implemented on top of Postgres	<ul style="list-style-type: none"> <li>- structure: generic, has to be extended for the test application;</li> <li>- functionality: only feature extraction functionality can be modeled through relationships between objects;</li> </ul>	no extensibility interfaces explicitly defined	good, through visualization of semantic models
<b>VIMSYS:</b> combination of functional and object-oriented models, and mathematical formalisms	implemented as Virage Image Search Engine, integrated in Illustra	<ul style="list-style-type: none"> <li>- structure: generic, the structural part of the test application fits well into the model;</li> <li>- functionality: only feature extraction functionality can be modeled by relationships between models;</li> </ul>	no extensibility interfaces explicitly defined	difficult due to the combination of different types of modeling languages
<b>EMIR<sup>2</sup>:</b> mathematical formalisms	conceptual graph as intermediate model, implemented on top of O2	<ul style="list-style-type: none"> <li>- structure: generic, the structural part of the test application fits well into the model;</li> <li>- functionality: the mathematical formalisms supports the definition of different functions on the set of image content objects;</li> </ul>	extending and refining object sets, extending the mathematical formalism with new concepts	mathematical formalism difficult to comprehend
<b>PIQ:</b> object modeling description language	implemented on top of a relational DBMS	<ul style="list-style-type: none"> <li>- structure: generic, has to be extended for the test application;</li> <li>- functionality: support for modeling only feature extraction functionality;</li> </ul>	modeling language not explained in detail, no extensibility interfaces explicitly defined	difficult, because modeling language not explained in detail

Table 3.1: Evaluation of the Image Retrieval Models

Model	Criteria	Implementability	Validity and Completeness	Flexibility	Understandability
<b>DISIMA:</b> combination of functional and object-oriented models, and mathematical formalisms		implemented on top of ObjectStore	<ul style="list-style-type: none"> <li>- structure: generic, the structural part of the test application fits well into the model;</li> <li>- functionality: modeling retrieval functionality is supported separately as MOQL queries, feature extraction functionality is not part of the model;</li> </ul>	no extensibility interfaces explicitly defined	difficult through the combination of different types of modeling languages
<b>MOODS:</b> extended object-oriented model		implemented as C++ extension	<ul style="list-style-type: none"> <li>- structure: insufficiently described, focus lies on image processing functionality;</li> <li>- functionality: supports feature extraction, but no retrieval functionality;</li> </ul>	modeling language not explained in detail, no extensibility interfaces explicitly defined	difficult through the combination of different types of modeling languages
<b>MMDB:</b> mathematical formalisms		only formal, no implementation known	<ul style="list-style-type: none"> <li>- structure: generic, the structural part of the test application fits well into the model;</li> <li>- functionality: the mathematical formalisms supports the definition of different functions on the set of image content objects;</li> </ul>	extending and refining object sets, extending the mathematical formalism with new concepts	mathematical formalism difficult to comprehend
<b>MPEG-7:</b> XML Schema		implemented as object-oriented system, XMLDB	<ul style="list-style-type: none"> <li>- structure: exhaustive data structure with concrete image descriptors, adapted to the application through picking the needed descriptors;</li> <li>- functionality: not supported;</li> </ul>	extending and adapting modeling structures through refining description schemas and descriptors with few limitations	easy to comprehend, but above certain size becomes too unclear



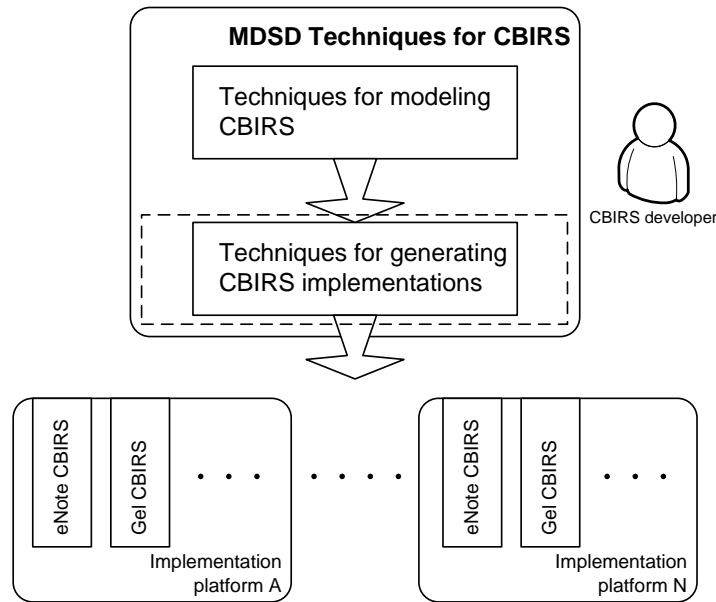


Figure 3.17: MDSD techniques for the development of CBIRSs - Mapping

Since none of the above models satisfies the requirements defined in the beginning of this section to a satisfactory extent, in this thesis a domain-specific model for CBIRSs is proposed. In Chapter 5, this adaptable conceptual image model, based on the object-oriented modeling paradigm to support the model-driven generation of a CBIR system is represented. The model aims at providing a general base for representing image data structure and retrieval functionality as defined in the requirements in the previous section. It has to support the implementability of the model on a large number of platforms and provide explicit mechanisms for extending and adapting the model for domain-specific applications. Finally, it has to achieve good understandability through a modeling paradigm supported by a wide range of modeling tools and a comprehensive tutorial for applying the model for deriving a domain-specific application model.

Having this model as a starting point in mind, in the following section the requirements towards the second type of MDSD techniques for CBIR, the generation techniques, are defined. Approaches for transforming the conceptual PIM into an implementation model, the PSM, are reviewed. Finally, criteria for the quality of the transformation rules are induced.

### 3.2 Transforming the CBIRS PIM to a PSM

In order to design the mapping techniques for the second step of the MDSD process of CBIR shown in Figure 3.17, the target platform and system architecture are determined.

The PSM in a straightforward sense is the generated source code of the application. However, usually the development process consists of more than one transformation and refinements before the final source code can be generated. Before generating the database schema for the implementation of a CBIR system an intermediate model can be used as shown in Figure 3.18. In the intermediate model, the developer of a CBIR system can make decisions concerning specific platform requirements, which cannot be made by an automatic transformation

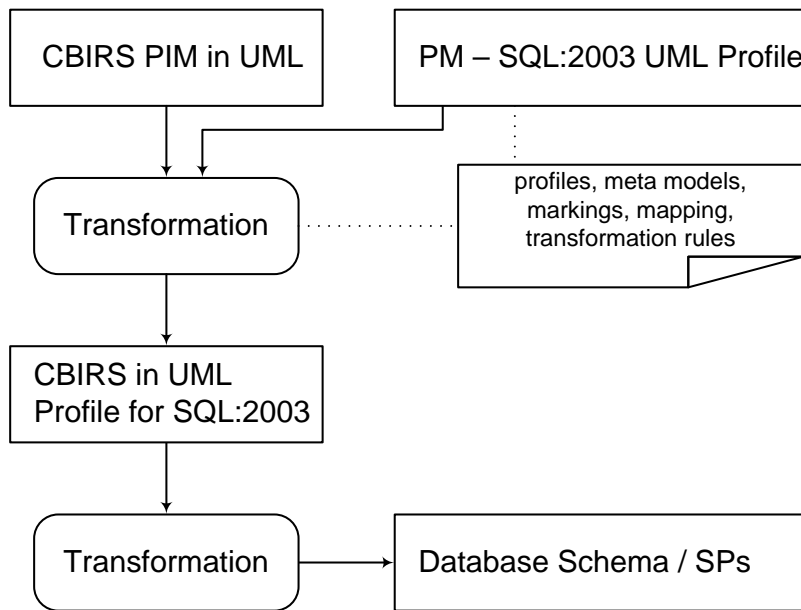


Figure 3.18: Transforming a CBIR System PIM to a Database Schema (based on [PM06])

algorithm, before moving to the source code.

As shown in Figure 3.18, two types of transformations are necessary to complete the model-based generation of the CBIR application. The first transformation maps the PIM onto a PSM and is thus referred to as model-to-model transformation. The second transformation translates the PSM into SQL statements, which can be used directly in the DBMS to create the schema of the CBIR system.

Since the platform specific model depends on the platform which will be used for the implementation, first, some decisions about the target platform have to be made. Moreover, these decision may have to be made for different components of the CBIRS. That means that different parts of the applications may require different platforms. For each different platform, first, a platform specific meta model is required, e.g., a UML Profile for the platform. Secondly, rules for transforming the platform independent model are needed.

In the next section, the possible software architectures and platforms for CBIRSs are reviewed. This discussion is used to choose the transformation approach which has to be supported for the generation of a CBIRS.

### 3.2.1 Choosing a Software Architecture and an Implementation Platform for CBIRSs

In Chapter 2, the architecture of a CBIR system is described. It is determined that the framework for model-driven development should deal with the design only of the core parts of the system, which can be used by different client applications. Therefore, the design of a concrete graphical user interface is not the aim of the framework. The resulting system should provide APIs for such client applications.

Mostly client/server software architectures are used for building CBIR systems. Different combinations of the data management, application logic and user interface layers in 2-, 3- or

n-tier architectures are used to support the requirements of CBIR applications. The large amount of data and complex algorithms for feature extraction and similarity matching require that the data management and application logic tasks are done by more powerful computers. The availability of the CBIR systems as web-applications is another good reason to use client/server architectures. Current research also reveals the possible use of other architectures for CBIR, such as Peer-to-Peer [KNS04] and Grid [RBPR06]. Major aims of these decentralized architectures are to increase the speed of query processing, provide access to very large and heterogeneous collections etc. These architectures still pose some difficulties for the proper functioning of the systems, such as availability of resources, trustworthiness of sources, and security. Moreover, these architectures normally comprise a number of client/server architectures under the control of a corresponding decentralization mechanism implemented in a middleware layer. Therefore, in this work we consider the client/server architecture as a target software architecture for implementing the modeled domain-specific CBIR system.

The components of a CBIRS can be grouped corresponding to the layers of an information system architectures defined in [ACKM04]: presentation layer, application logic layer and resource management layer. Each of these layers of the CBIR architecture could be implemented on separated platforms, but they can also be integrated in one platform. In Figure 3.19 common platform independent client/server architectures which group the three system layers in different number of tiers are shown. Each tier corresponds to a platform.

Different implementation platforms can be used for each layer of the CBIR system, respectively. Since the aim of the current framework is the implementation of the main-functionality of the system we will consider only the storage management and application layers. As mentioned in Chapter 2, for the implementation of the system the employment of an object-relational database management system as an implementation platform is chosen. ORDBMSs allow not only the management of the data resources but also the implementation of the main application functionality as stored procedures and user-defined functions. Thus, the 2-tier client/server architecture from Figure 3.19 is applied, where the server provides all necessary interfaces to the CBIR system through the database API. This choice can be made in favor for any other implementation platform if required since the conceptual model of a CBIRS should not pose any platform limitations.

### 3.2.2 A Platform Specific Model for ORDBMSs

A domain-specific model should be implementable on any specific platform. The PIM should not include any platform specific modeling concepts. In this thesis, the implementation onto an object-relational database management system (ORDBMS) is considered, based on the standard SQL:2003 [Tür03]. This environment is chosen in order to facilitate the possibility for image database developers to design and implement customized database extensions for storing and querying images by content in ORDBMSs. Existing database image extensions (e.g. IBM AIV-Extenders) as CBIR applications belong to the first group of CBIR systems from the classification in section 1.3 - generic CBIRSs. It is not possible to use or adapt these extensions for a specific application domain. Perhaps exactly because of this limitation IBM AIV Extenders in particular are not anymore supported in the newest version of the ORDBMS IBM DB2 V.9.1. Instead, the product documentation points out that the implementation of such extensions is left to the user. Therefore, the implementation onto an ORDBMS is an adequate example and test case for the developed concepts.

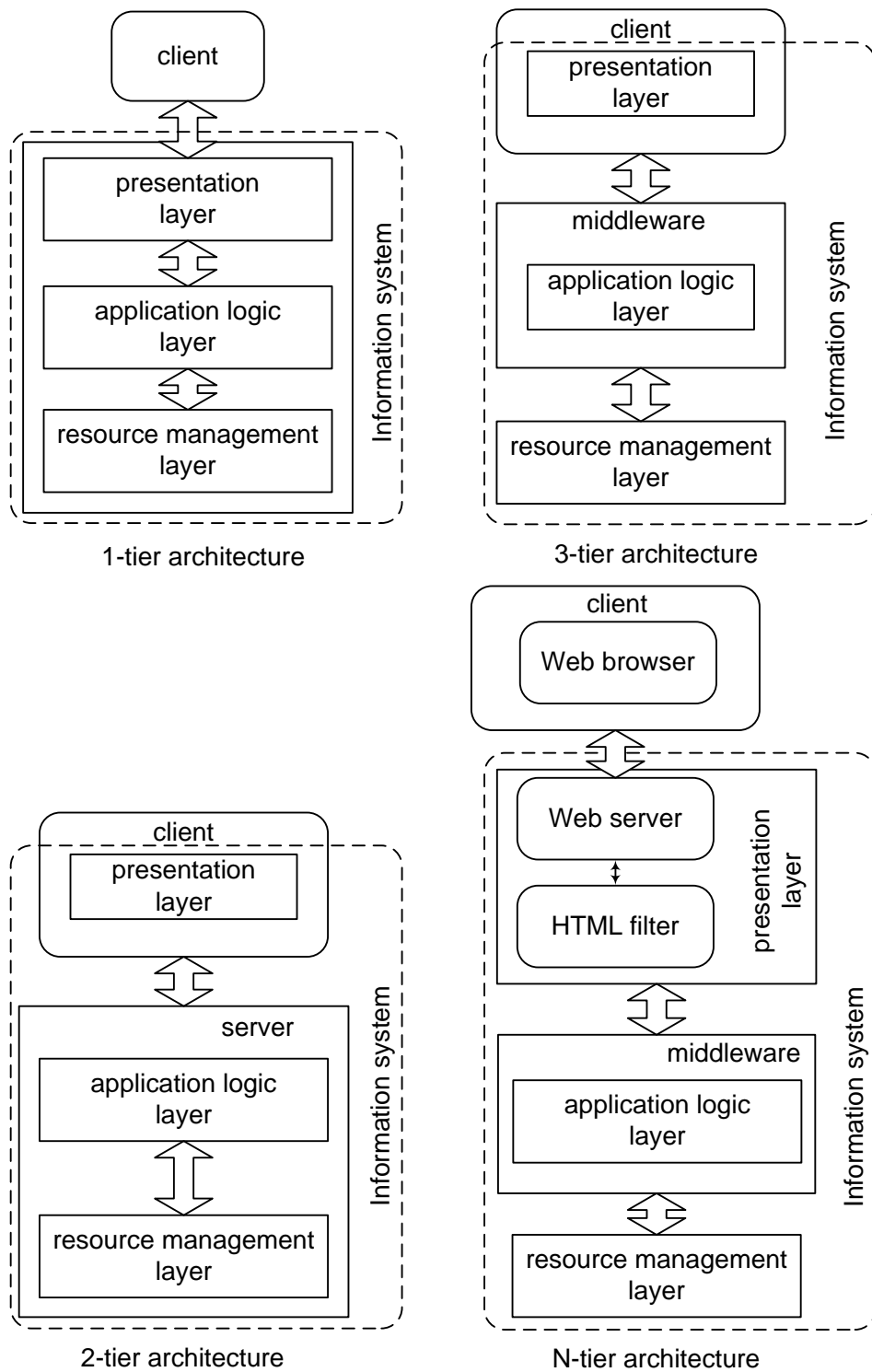


Figure 3.19: Client/Server architectures for information systems (based on [ACKM04])

The object-relational database model has emerged from the development of the relational model to support object-oriented concepts. Since the SQL:1999 version of the SQL standard the defined model is referred to as object-relational model. The current version of the SQL standard is SQL:2003. Profound descriptions of the standard can be found in [Tür03, Mel02]. In addition, for the support of multimedia data in databases, a multimedia extension of the SQL standard known as SQL/MM has been released. It can be regarded as a layer building on top of the basic object-oriented extension of the standard to support the development of multimedia database applications. It is described in a separate multipart package *SQL/MM: ISO/IEC 13249:2000 Information technology - Database languages - SQL Multimedia and Application packages*. It defines a number of packages of generic data types common to various kinds of data used in multimedia and application areas, to enable that data to be stored and manipulated in an SQL database. The SQL/MM standard consists of several parts, one of which is dedicated to still images. The still image part *ISO/IEC 13249-5:2003* defines user-defined functions and types to address the need to store, manage and retrieve information based on aspects of inherent image characteristics such as height, width, and format, and based on image features such as average color, color histogram, positional color and texture. It also addresses the need to employ image manipulation functions, such as rotation, scaling as well as similarity assessment. The existing multimedia extensions of the major DBMSs are developed based on this standard.

Thus, the standard for object-relational DBMSs provides a formal representation of the platform concepts which can be used as a basis for the platform specific meta model, i.e. platform modeling language. There are already some preliminary works aiming at building a DBMS meta model, which can be used in model-driven development tools. These meta models do not claim to be exhaustive. They do not provide modeling concepts for all ORDBMSs features, but offer possibilities for extensions, because all these ORDBMSs meta models are developed as UML-Profiles.

There is still no standardized UML profile for SQL databases. The OMG (Object Management Group) has issued a Call for Proposals in December 2005 to define a standard meta model for Information Management, which should be used among others for the definition of a UML2 Profile for Relational Data Modeling, with a mapping to the IMM meta model and SQL DDL. This effort is, however, not completed.

**Rational's Data Modeling Profile** for relational databases [Rat03, Gor02] is one of the first DBMS meta model used in the Rational Rose Data Modeler design tool. It provides ways to model basic relational concepts such as: schemas, tables, columns, keys, relationships etc. Stored procedures can be modeled as operations of a class, representing a container of these procedures. This profile does not deal with object-relational concepts, such as user-defined types, user-defined functions etc. A full list of the UML extensions for modeling relational databases is given in Table 3.2.

**A Data Modeling Profile for agile databases** is proposed by Scott Ambler in [Amb03]. This approach refines the Rational UML Profile for Data as shown in Table 3.3, but also does not consider object-relational concepts.

**Marcos et al.** propose for the first time a model-driven design methodology for object-relational databases in [MVC04, VVCM07]. In addition they propose a set of transformation rules for mapping conceptual UML models onto ORDBMS UML models. The authors provide support for structured data types, typed tables, REF and collection types etc. They focus

Database Element	UML Element	Stereotype
Tablespace	Component	«Tablespace»
Database	Component	«Database»
Schema	Package	«Schema»
Table	Class	«Table»
View	Class	«View»
Derive	Association	«Derive»
Column	Attribute	«Column»
Primary Key	Attribute	«PK»
Foreign Key	Attribute	«PK»
Combine Key	Attribute	«PFK»
Index	Operation	«Index»
Unique Constraint	Operation	«Unique»
Trigger Constraint	Operation	«Trigger»
Primary Key Constraint	Operation	«PK»
Foreign Key Constraint	Operation	«FK»
CHECK Constraint	Operation	«Check»
Identifying Relationship	Association	«Identifying»
Non-Identifying Relationship	Association	«Non-Identifying»
Stored Procedures Container	Class	«SP Container»
Stored Procedure	Operation of «SP Container»	«SP»

Table 3.2: UML Extensions for modeling DBMSs by Rational (based on [Rat03, Gor02])

on the structural part of the model and do not treat the problem of modeling functionality in detail. The UML extensions, defined to model relational databases and object-relational databases, proposed in [Mul99] are listed in Table 3.4.

**Muller** [Mul99] describes a practical design approach for databases using UML. The author discusses the meaning of each object-oriented UML concept for building structural diagrams (class diagrams) and how this semantics can be interpreted or extended in order to represent concepts of ORDBMSs. The author thus defines UML as an advanced alternative to the Entity-Relationship Model (ERM) for modeling databases. The extensions of the UML model are not formalized in a UML-Profile, but each modeling topic is discussed verbosely and thus the book provides a good practical reference for applying UML for modeling ORDBMSs.

**UML modeling tools**, such as Visual Paradigm, Magic Draw, Enterprise Architect, also provide support for creating relational database models. They make use of predefined UML-Profiles, which currently also do not represent the state-of-the-art in database management systems. Modeling tools often allow the integration of predefined profiles and thus would allow the reuse of a graphical modeling environment for a more extensive UML-Profile for ORDBMSs.

Since none of the above models represents a complete ORDBMS Profile, a more exhaustive ORDBMS UML-Profile has to be defined. Therefore, a compilation of the existing Profiles can be used as a basis, by choosing the most adequate modeling policy from the different alternatives. In addition, new stereotypes have to be defined in order to support better the

Database Element	UML Element	Stereotype
Table	Class	«Table»
View	Class	«View»
Index	Class	«Index»
Associative Table	Class	«Associative Table»
Lookup Table	Class	«Lookup Table»
Stored Procedures	Class	«Stored Procedures»
Identifying Relationship	Association	«Identifying»
Non-Identifying Relationship	Association	«Non-Identifying»
Dependency	Association	«Dependency»
Primary Key	Attribute	«PK»
Foreign Key	Attribute	«FK»
Alternate Key	Attribute	«AK»
Auto Generated Value	Attribute	«Auto Generated»
Column	Attribute	«Column»
Not Null	Attribute	«Not Null»
Nullable	Attribute	«Nullable»
Surrogate Key	Attribute	«Surrogate»
Stored Procedure	Operation	«Stored Procedure»
Trigger	Operation	«Trigger»

Table 3.3: UML Extensions for modeling DBMSs by Scott Ambler (based on [Amb03])

modeling of user-defined functionality.

### 3.2.3 Model-to-Model Transformation

In database design model-to-model transformations are applied broadly for different tasks such as mapping of conceptual Entity-Relationship models (ERM) to relational models [Che75], schema integration [MIR93], schema matching [RB01], design optimization etc. Many problems of model transformations for these purposes have been treated in order to provide a good theoretical basis for the development of database applications.

In software development, model transformations techniques have been used even before model-driven development approaches came into discussion. Compiler engines also function on some kind of a model transformation basis. Only in this case, the models which have to be transformed, are programs. The model-driven development approach applies transformation on more abstract models instead. This development approach reminds a lot of the database design approach, which also uses different models at consecutive design steps.

However, database design approaches have been developed for relational databases in the very beginning and rely on using the Entity-Relationship model for conceptual modeling. Approaches for mapping Entity-Relationship models onto relational schema are also intensively studied and methodologies broadly used. The development of relational databases into object-relational databases offers new possibilities to represent information. Therefore, new conceptual modeling approaches had to be employed. Numerous extensions of the Entity-

	Database Element	UML Element	Stereotype
Relational	Database	Component	«Database»
	Schema	Package	«Schema»
	Tablespace	Component	«Tablespace»
	Index	Class	«Index»
	Table	Class	«Table»
	View	Class	«View»
	Column	Attribute	«Column»
	Primary Key	Attribute	«PK»
	Foreign Key	Attribute	«FK»
	NOT NULL Constraint	Attribute	«NOT NULL»
	Unique Constraint	Attribute	«Unique»
	Trigger	Constraint	«Trigger»
	CHECK Constraint	Constraint	«Check»
	Stored Procedure	Class	«Stored Procedure»
Object-relational	Structured Type	Class	«UDT»
	Typed Table	Class	«Object Type»
	Knows	Association	«Knows»
	REF Type	Attribute	«REF»
	ARRAY	Attribute	«Array»
	ROW Type	Attribute	«row»
	Redefined Method	Method	«redef»
	Deferred Method	Method	«def»

Table 3.4: UML Extensions for modeling ORDBMSs by Marcos et al. (based on [VVC07])

Relationship model have been proposed to capture more semantics of the real world, which partially reflect the object-oriented features of the databases. The extensions which have been widely accepted are for example specialization and generalization relationships. However, there are object-oriented concepts, such as dynamic aspects, which are not considered in these extensions. The object-oriented modeling concepts from the software design, such as UML, seem to be a better alternative for the conceptual design of object-relational databases. However, there is still an expressiveness gap between the representation possibilities of the object-oriented model UML and the ORDBMS model. This is a well known phenomenon and in the literature it is referred to as the “impedance mismatch” between object-oriented and relational technologies. For example, the mapping of a generalization relationship to the object-relational model requires the additional usage of triggers in order to implement the generalization constraints. In relational databases the generalization constraint “disjoint”, which requires that an object of the generalization cannot belong to different subclasses at the same time, has to be assured through triggers. Triggers have to check each time a new object is inserted into a table corresponding to a subclass if this object is already in another subtable. Such cases still have to be handled with the help of workarounds and compromises, but the longterm aim is to bring ORDBMSs one day to a state where they can support all object-oriented design concepts, and at the same time more clearness in the semantics of UML.



There is, however, a very fine but important difference between the database design approach and the model-driven design approach that has to be considered. In the first case, the model used at the conceptual level is especially designed for the modeling of relational or object-relational databases. The ER model represents relational concepts, such as key attributes and the UML conceptual models adopt extensions for class diagrams to represent object-relational concepts, such as properties for operations, showing if these are stored procedures or methods. In the model-driven design the starting point is a domain-specific model, which does not specify platform-specific concepts, such as primary keys for example. Since the domain-specific model in the current thesis is built using the object-oriented modeling language UML, the mapping onto a platform specific model in UML can be also regarded as augmenting the domain-specific model with platform-specific concepts. Therefore, the gap between the conceptual and the logical models in the first case is smaller than the gap between the abstract PIM and the PSM in the second. In the following section, existing approaches for object-relational database design using UML as a conceptual modeling language are summarized.

### 3.2.3.1 Approaches for Mapping UML Models to ORDBMS Schema

**Saake and Türker [CT06]** The authors of this textbook have summarized different works on the mapping of UML class diagram concepts onto an object-relational SQL schema. They have described the mapping of basic UML concepts, such as classes, attributes, associations, association classes and class methods and inheritance hierarchies in their methodology. Thereby, the most straightforward mapping possibilities are chosen. In addition, workarounds for preserving the consistency of the data in cases where the database management system does not offer enough support are discussed. For example, the associations between classes are represented by reference data types, which need an additional mechanism, e.g. a trigger, to assure that referenced instances are not deleted. To what extent normal form rules can be applied to object-relational design has been also discussed. The authors leave out the design of database functionality in form of operations for the user-defined types.

**Marcos et al. [VVCM07]** A methodology for the mapping of UML class diagrams onto object-relational models and specific database management system models has been proposed in [MVC04] and [VVCM07]. In this methodology, the starting point is a conceptual data model represented in UML and the target is an object-relational data model represented by means of an object-relational UML Profile. In the second paper cited above, the mappings have been further formalized with graph transformation rules in order to support the model-driven development process. The described mappings in both papers, however, are not exhaustive, for example, the mapping of operations is not discussed. The resulting database structure is an almost direct mapping of the UML structures. This technique does not prove the result of the transformation. Preservation of information capacity and integrity is not discussed. The mapping rules, specified in the works of Marcos et al. are shown in Table 3.5.

**Mok and Paper [MP01]** In [GCR06] the approach of Marcos et al. is compared with a more formal technique, presented in [MP01] for the transformation of UML class diagrams, based on the graph theory. In this technique, two algorithms are applied subsequently on the UML class diagram graph. The first algorithm removes semantically overloaded elements from the graph and the second algorithm converts the resulting diagram in nested normal form tables. The object-relational schema which is the result of the formal transformation

Data PIM	Standard Data PSM (SQL:2003)
Class	Structured Type + Typed Table
Class Extension	Typed Table
<b>Attributes</b>	
Multivalued	ARRAY/MULTISET
Composed	ROW/Structured Type (column)
Calculated	Trigger/Method
<b>Association</b>	
One-to-One	Ref/[Ref]
One-to-Many	[Ref]/[Multiset/Array]
Many-to-Many	Multiset/Multiset, Array/Array
Aggregation	Multiset/Array
Composition	Multiset/Array
Composition	Types/Typed Tables
<b>Operations</b>	
Signature of class operation	signature of method of a structured type

Table 3.5: Mapping of UML PIM onto OR PSM by Marcos et al. (based on [VVCM07])

technique consists mostly of nested tables. This has the advantage that when transforming large models a more compact representation with less tables is produced. However, some of the information of the original UML model might get lost.

**Dietrich and Urban [DU04]** This is another textbook in which basic rules for mapping UML classes, attributes, associations and inheritance onto object-relational features are presented. The authors verbosely describe in detail the mapping approaches for each of these conceptual elements. The result of the mapping is represented in SQL DDL syntax. Interpreting platform specific features from the pure UML class diagram is supposed to be done by the developer, e.g. choosing a key attribute. The authors also suggest that the mapping onto object-relational concepts can be combined with a mapping onto relational model, but in this thesis it is argued that only object-relational concepts should be used for mapping object-oriented concepts onto ORDBMSs. In addition to the rules of Marcos et al. a special attention is given to how associations with and without attributes, recursive associations, n-ary associations and constraints can be mapped. The mapping of class hierarchies and categories is also elaborated.

These works set a good basis for an ORDBMS design methodology using UML class diagrams for the conceptual design. However, they leave some unsolved problems, which play an important role for the task of automating the design process. First of all, the above approaches do not cover enough the design of database functionality. The modeling and mapping of user-defined functions is not treated at all. Not all concepts of UML class diagrams are considered and it is not meaningful to forbid the usage of any parts of the modeling language to the modeler. ORDBMSs are extended with a lot of features to support object-oriented concepts. However, these are still not enough to guarantee consistency and integrity of data stored in the database. Additional workarounds have to be implemented for that. Some basic object-oriented concepts are still not supported fully, such as different variations of

inheritance, overlapping, disjoint subclasses etc. This makes the mapping of some design elements difficult.

Some of the approaches suggest that also purely relational concepts can be used to implement some object-oriented design concepts. However, this would be a step back in the development of object-relational databases. The object-relational design should try to solve the implementation challenges by suggesting improvements for the support for object data instead of looking for workarounds.

And finally, most of these methods are proposed to support the SQL standard specification and only one or two real DBMSs. Real life DBMSs do not always comply completely with the standard. They sometimes offer less features, as in the case of IBM DB2 where no MULTISSET and ARRAY data types are supported. Others go beyond the suggestions of the standard, for example the PostgreSQL provides support for multiple inheritance. To have a functioning application, the final generation step should produce source code for a particular DBMS system. Therefore, adequate adaptations have to be carried out to the SQL schema.

The main conclusion from this discussion is that there are still a lot of gaps between object-oriented and object-relational concepts, which have to be taken into account when using ORDBMSs as an implementation platform for CBIRSs. The above considerations should be taken into account when developing a methodology for mapping the CBIR PIM onto an ORDBMS PSM.

### 3.2.3.2 Requirements for the Model-to-Model Transformation

Existing transformation approaches discussed in the previous section do not provide mapping rules for all elements of the PIM meta model. This leads us to the question “Must all PIM elements have a corresponding mapping in the PSM?”, “What consequences can missing mappings have for the quality of the transformation?”. In order to provide an answer to these questions some kind of quality criteria for the transformation has to be defined.

One of the aims of a transformation is to retain as much as possible information from the higher abstraction level model in the implementation model. The differences in the expressive power of both modeling languages should be analyzed. In this section, basic requirements towards the transformation are set, which should assure information preservation and correctness of the resulting model.

The task of the model-to-model transformation in the current case is to translate the information represented in a PIM modeling language into the terms of the PSM modeling language. The transformation also implies that a more abstract model has to be converted into a more concrete model of the application. These statements classify the needed transformation as inter-model (exogenous) and vertical, respectively, according to the taxonomy of Mens and Gorp [MG06].

As shown in Figure 3.20, in order to translate a PIM into a PSM, transformation rules have to be defined at the meta model level. The mapping rules translate each concept of the source meta model (PIM meta model) into concepts of the target meta model (PSM meta model). These rules are then used to induce the transformation of a concrete PIM (source model) to a concrete PSM (target model).

A model-to-model transformation can be regarded as a kind of function in the mathematical sense. This means that for each element of the source model there exists at most one element

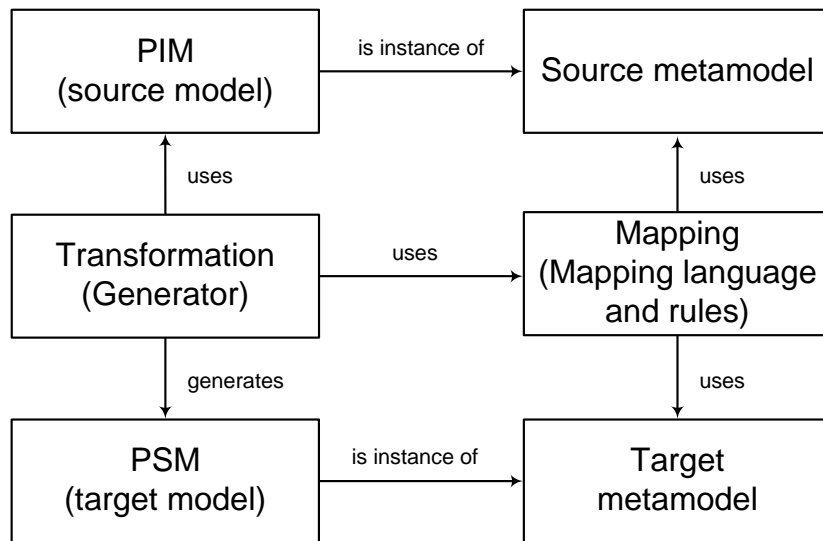


Figure 3.20: Meta model based transformation (from [PM06])

in the target model, which is the result of the transformation of the source model element. If a transformation is defined on each element of the source model then the function is also total. If the transformation is functional in the reverse direction then the transformation function is injective. If the reverse is total then the transformation function is bijective. These properties of a transformation are used in a well-known criterion for checking the correctness of the transformation in database design - the information capacity preservation [Hul86]. This criterion is applied for the design of relational database systems as shown in Figure 3.21. The transformation rules in this design approach are defined for mapping the concepts of an Entity-Relationship Model (ERM) onto concepts of the Relational Database Model (RDM). This level corresponds to the Metamodel Level of the model-driven development approach depicted in Figure 3.25. Each of these models can be used to define multiple schemas, corresponding to the model level in Figure 3.25. Schemas can have different states, which correspond to the different model instances in MDSD. In the database design approach, the transformation is considered as information capacity preserving if it is a bijection between the instances of the models. This is measured by checking whether the information represented in the ERM instance can be found again in the RDM instance through analyzing the results of queries against the RDM instance. This approach, however, requires the presence of instances of both models and corresponding query languages. In order to use the information capacity preservation criterion for the MDSD transformation some adaptations of the above approach are made.

First, the requirements towards the transformation function are defined for the Metamodel level, corresponding to the Model level in the previous case. The transformation should be functional in the direction from PIM-to-PSM, so that for a PIM concept exactly one corresponding PSM concept should exist. It should be also total to assure that all PIM concepts are mapped to the PSM. In the reverse direction the transformation should be also functional in order to assure that it is possible to differentiate between the PSM concepts in the same manner as in the PIM. The total property of the function in the reverse direction is not necessary, because at the implementation level there are concept which are not mappable

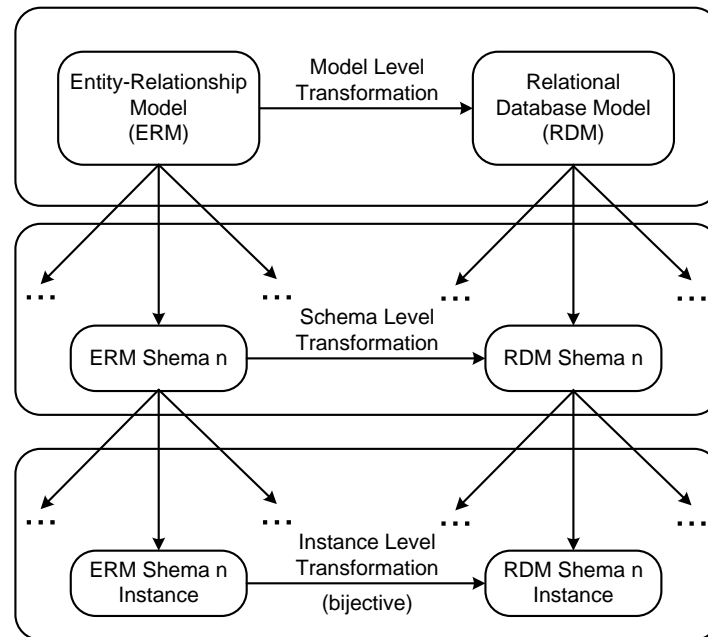


Figure 3.21: Transformation levels in the database design

to the conceptual model. It would be important to have this property if reverse engineering should be possible. However, the idea behind MDSD is to propagate the implementation changes through the conceptual model, rather than make changes in the implementation and send them to the conceptual model. The following requirements towards the transformation rules at the Metamodels Level, can thus be defined:

- In order to assure that each PIM concept can be mapped to a PSM concept, a mapping rule for each PIM meta model concepts has to be defined. This means that no PIM meta model concepts, which are not mappable to the PSM meta model as shown in Figure 3.22, should exist. In order to achieve this, sometimes workarounds in the PSM level can be expected.

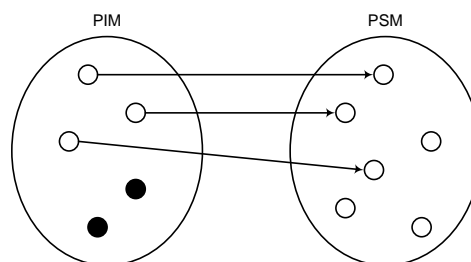


Figure 3.22: Transformation rule requirements: not-mappable concepts are not allowed in the PIM

- If there is more than one possibility to map a PIM meta model concept to the PSM meta model domain, one of these possibilities should be chosen for a given transformation. The choice can be made based on heuristics or cost-based rules.

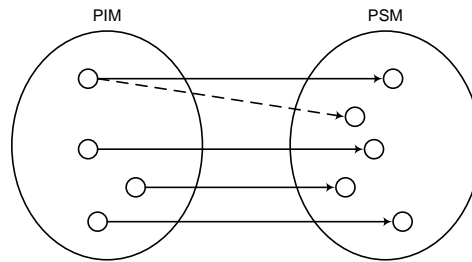


Figure 3.23: Transformation rule requirements: only one of multiple mapping possibilities should be applied in a transformation

- If two different concepts from the PIM meta model are mapped to the same concept in the PSM, differentiating information could be lost. Therefore, these kinds of mappings should be avoided. One of the problems which arises when such mappings are used is losing the possibility for reverse engineering, since there might be minimum two different sources of one and the same concept. An even more undesirable effect is that by mapping two different PIM concepts, which do not have an equivalent meaning in the PIM, onto the same PSM concept the differentiating information between the PIM concepts will get lost, which might be an important factor for the application.

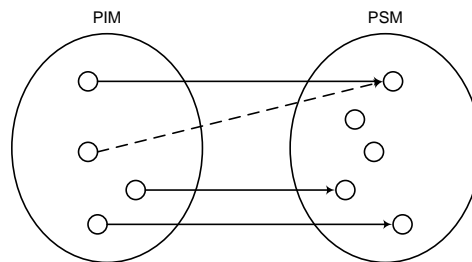


Figure 3.24: Transformation rule requirements: two different PIM concepts should not be mapped to the same PSM concept

Consequently, it can be summarized that the transformation at the Metmodel level of MDSD has to be a total injection, as illustrated in Figure 3.25. Since, the transformation rules at the instance level are derived from these at the Metamodel level, it can be assumed that they have similar properties.

The domain-specific PIM of a CBIRS proposed in this thesis is a UML-based model consisting mostly of structural diagram concepts. Therefore, the mapping rules are defined for this subset of UML concepts which are needed to model the CBIRS application. In Chapter 5 the transformation of this model onto an object-relational model, represented in terms of an UML Profile is described.

### 3.3 Summary

In this chapter, the components of a CBIR system, which have to be modeled in order to generate a CBIR system, were identified and their generic and variable features were determined.

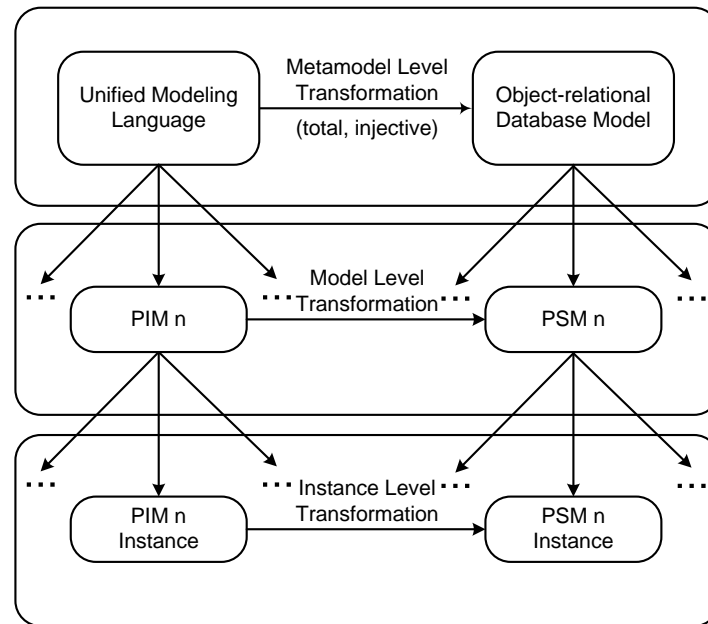


Figure 3.25: Transformation levels in MDSD

Thus, the requirements towards a modeling approach for CBIR systems were set. Existing modeling paradigms for image retrieval systems were surveyed based on these requirements and the eNoteHistory example application. This survey, justifies the development of a new generic and adaptable model with the recognition that current modeling techniques do not offer enough flexibility in order to support different application requirements, in particular with respect to modeling functionality.

Transformation techniques for generating a platform specific model of the CBIR application were discussed in the second half of this chapter. The choice of these techniques is influenced mainly by the choice of a software architecture and an implementation platform. The application logic and persistence layer of a 2-tier client/server were chosen for the target architectural components of the implementation. An ORDBMS was chosen as a target platform for both architecture layers. The requirements towards the transformation techniques are, therefore, dictated by the target software architecture and platform and the modeling approach. Existing UML Profiles for modeling ORDBMSs were analyzed, and it was found that none of them provides a full representation of the latest SQL standard, describing ORDBMS concepts. An exhaustive transformation methodology for generating an ORDBMS PSM from a given OO PIM is also not available. Finally, different correctness criteria for transformation rules, which have to describe the transformation methodology were formulated.

In the following two chapters, concrete CBIR development techniques for both modeling and transformation phases are elaborated in detail.

## Chapter 4

# GiACoMo–IRS – A Generic and Adaptable Conceptual Model for Image Retrieval Systems

The aim of the development approach presented in this thesis is to provide extensibility and adaptability at the conceptual level of the development of CBIR systems. Therefore, a general conceptual model is required which represents universal structures and functionality of CBIR systems and at the same time provides a possibility to adapt and/or extend these structures and functionality depending on the specific application. In this chapter, such a model is proposed.

Digital images and their content set certain challenges for conceptual modeling and their mapping onto database models. These data have characteristics with complex, composite values, which are often not directly interpretable. The definition of appropriate functions is required in order to compare, classify or manipulate these values. Furthermore, different semantic interpretations or views on the image data are possible, depending on the application context. Different levels of abstraction and semantically rich relationship hierarchies are further characteristics of digital image content. All these factors make the creation of a universal conceptual image model a challenging task and convey special requirements also to the implementation level. Contemporary relational database systems as an implementation platform do not provide facilities for the storage of these kinds of data. Object-relational databases offer better techniques to support these applications. User-defined types, user-defined functions and indexing mechanisms, and binary large objects make it possible to accommodate complex data structures.

General conceptual models for images have been continuously proposed since the very beginning of image retrieval systems. In the field of multimedia information systems conceptual models have been designed which also cover images as one type of media. However, the main drawback of existing models is that they seldom provide a concrete modeling methodology with well-defined extensibility and adaptability interfaces. They rather focus on describing an overall architecture of a model, but not on how it can be used for deriving domain-specific models. A detailed survey and evaluation of existing models with respect to the predefined requirements is represented in the previous chapter.

Abstracting from the common characteristics of an image data model to provide a general



image model is not enough to support the development of domain-specific CBIR applications. In addition to such a generic image model, appropriate mechanisms and structures should be defined, which will allow the extension and adaptation of the model for the concrete application. In this chapter, an extensible conceptual image model is represented as a UML framework, which can be adapted for domain-specific applications.

## 4.1 The Modeling Approach

For the representation of the Generic and Adaptable Conceptual Model for Image Retrieval Systems (GiACoMo-IRS), at first an adequate modeling approach had to be selected according to the requirements and quality goals defined in the previous chapter. In this section, the chosen modeling approach is described.

### 4.1.1 Framework Model

GiACoMo-IRS is designed as a framework model, which can be extended and customized for the particular requirements of domain-specific applications. A framework in the context of object-oriented programming languages [GVJH98] and UML [BRJ99] refers to a customizable, extendable skeleton of a software architecture, which can be used for subclassing domain-specific applications. Reusing a framework for building specific applications is referred to as *framework specialization*, *framework adaptation*, or *framework instantiation* as pointed out in [Vil06]. In [IB05] the concept of a framework model was introduced as a generic model for the design of multimedia databases, where still images are only one type of a multimedia component. Based on this work the framework model GiACoMo-IRS for the design of image retrieval systems has emerged. In Figure 4.1 the different abstraction levels for the application of this modeling approach are shown. The framework level represents the framework model, which instantiates the concepts of the UML meta model. At the application level a concrete application-specific model for the eNoteHistory Image Retrieval System (IRS), which is used as an example for formulating the requirements towards the modeling techniques in Chapter 3, is represented. The eNoteHistory-IRS model is derived by adapting and/or extending the abstract and adaptable classes and interfaces from the framework model. The concrete application model should be used as the basis for generating the implementation. The Meta Model Level corresponds to the M2 level in the meta level of MDA, depicted in Figure 2.4. Both Framework and Application Level belong to the M1 meta level of MDA.

The UML modeling paradigm is chosen for the design of GiACoMo-IRS. The main reasons for making this choice are: the extensibility of the meta model, integration possibilities with other models of components of the application, the support for functionality design, the fact that UML is used as the basis of the model-driven architecture and thus MDA-tools for modeling and for generating implementation models from UML are available, and last but not least its broad usage. Although the basic concepts of the UML model do not have an extensive support and notations for adaptability and extensibility interfaces, extensions of the UML model, which aim at providing adaptability and extensibility patterns for the design of frameworks are proposed in [FPR00] and in [OAF<sup>+</sup>04].

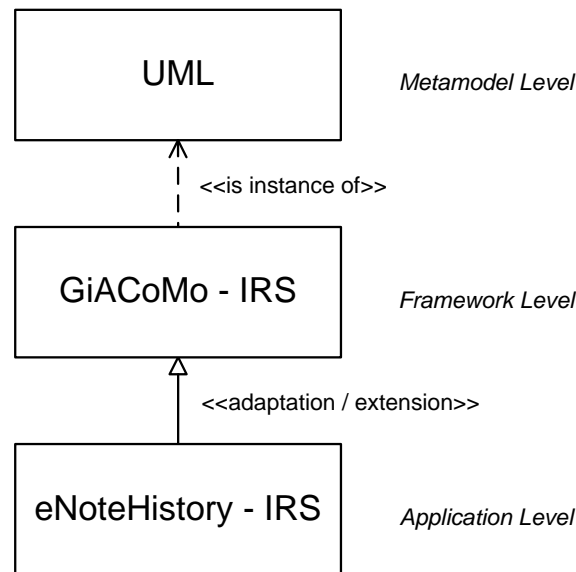


Figure 4.1: Modeling approach

#### 4.1.2 UML for Frameworks

The UML-F profile from [FPR00] is used to represent adaptability and extensibility interfaces in GiACoMo-IRS. It can represent framework design concepts, such as *hot spots* and *frozen spots*. In [Pre95] the parts of the framework which represent adaptability and extensibility interfaces are named *hot spots*. The parts which should remain unchanged in the derived application are called *frozen spots*. The *hot spots* of a framework can be implemented by means of *white boxes* and *black boxes*. White boxes provide the possibility to adapt the framework by making use of inheritance and dynamic binding, where black boxes employ predefined classes from the framework which can be used through the composition and delegation concepts. Gray box frameworks combine both possibilities, by letting the developer decide whether to use the predefined classes or implement own ones. Furthermore, GiACoMo-IRS makes use of the so called *cookbook* approach [Pre95] by providing recipes (examples) for making use of particular adaptability/extensibility interfaces during the framework instantiation.

In the next two sections, the framework model with regard to representing the components of a CBIRS in terms of data structures and functionality is described. UML class diagrams are used for the design of the structure of the GiACoMo-IRS. Functionality is designed using use-case and activity diagrams to outline the main functions of the system and to represent the details of the single use cases, respectively. Each activity has then been mapped onto operations and classes in the UML class diagram.

In the first section the part of the model which covers mainly the “Image Store” and “Feature Store” storage components is represented. These components require the definition of a data structure for representing content-dependent and content-independent image data. Similar structural concepts can be used to represent data structures for the “Retrieval” and “Feature Extraction” components. However, these components are more functional, i.e. they do not have a generic data structure, but rather a generic functionality. Thus, in the GiACoMo-IRS framework model they play an important role in the the design of the functionality of the

system, described in the second section. On the other hand, the functionality of the storage components is relatively generic, but it is not always necessary to be modeled because many target platforms, e.g. DBMSs, already implement a persistence storage mechanism.

## 4.2 Modeling the Data Structure of CBIRS Components

In Figure 4.2, a UML class diagram of the generic image model that was defined in [IB05] to represent the still image component of a multimedia document is shown. This part of the multimedia database framework model is used to design the basic data structure of GiACoMo-IRS, shown in Figure 4.3. A redesign of the model defined in [IB05] was necessary on one side in the structural part in order, for example, to allow omitting certain concepts of the generic model, if they are not needed in the concrete application model. This is achieved by redefining class attributes as associated classes, and assigning the optional multiplicities to the associations. On the other side, the functional part in the model represented in [IB05] is not systematically derived from the functional requirements of the system. Thus, it is difficult for the developer to understand the meaning of the different class operations and how to use them to derive own applications. Moreover, the predecessor model did not consider modeling other retrieval mechanisms except the metric approach. The extended image model GiACoMo-IRS is published in [IH08]. In the following section, the main parts of the structure of GiACoMo-IRS are described as extensions of the multimedia model in Figure 4.2. Section 4.3 explains the functional part of GiACoMo-IRS which is the main extension of the predecessor model.

### 4.2.1 StillImage

The class **Image Component**, derived from the abstract class *Component* (component of a multimedia document) allows the representation of the raw image data. In GiACoMo-IRS this class is renamed to *StillImage* and is defined as an abstract class, which should be used to derive concrete classes for each application-specific model. The **Image Component** class has attributes, which represent the raw image as a binary sequence and/or a reference to a file through a path description. Other representations of the raw image such as a matrix of pixel values or a fourier transformation could also be added to the class by deriving an application-dependent specialization of this abstract class. Additionally, an optional thumbnail representation of the image can be used in the class. The problem which has to be solved in GiACoMo-IRS is to make the proposed attributes of the class optional and adaptable, so that they can be freely included or omitted and changed in the domain-specific specialization of the abstract class. Normally the implementation is realized by directly inheriting from the abstract class, which does not allow any changes on the inherited structure of the derived class. Therefore, since the representation of the raw image of some type is an obligatory attribute of *StillImage* an abstract class *RawImageRep* is defined and associated with the *StillImage* class through a mandatory association. Different types of image representations can be defined for a particular application. The optional attribute Thumbnail can also be regarded as a type of representation of the image. An image can be composed of multiple images, which are interrelated through the “consists of” aggregation association. This association is optional, which is represented by the multiplicities at its both ends. At this stage no explicit methods will be included in GiACoMo-IRS. The representation of object behavior is described in the following section.

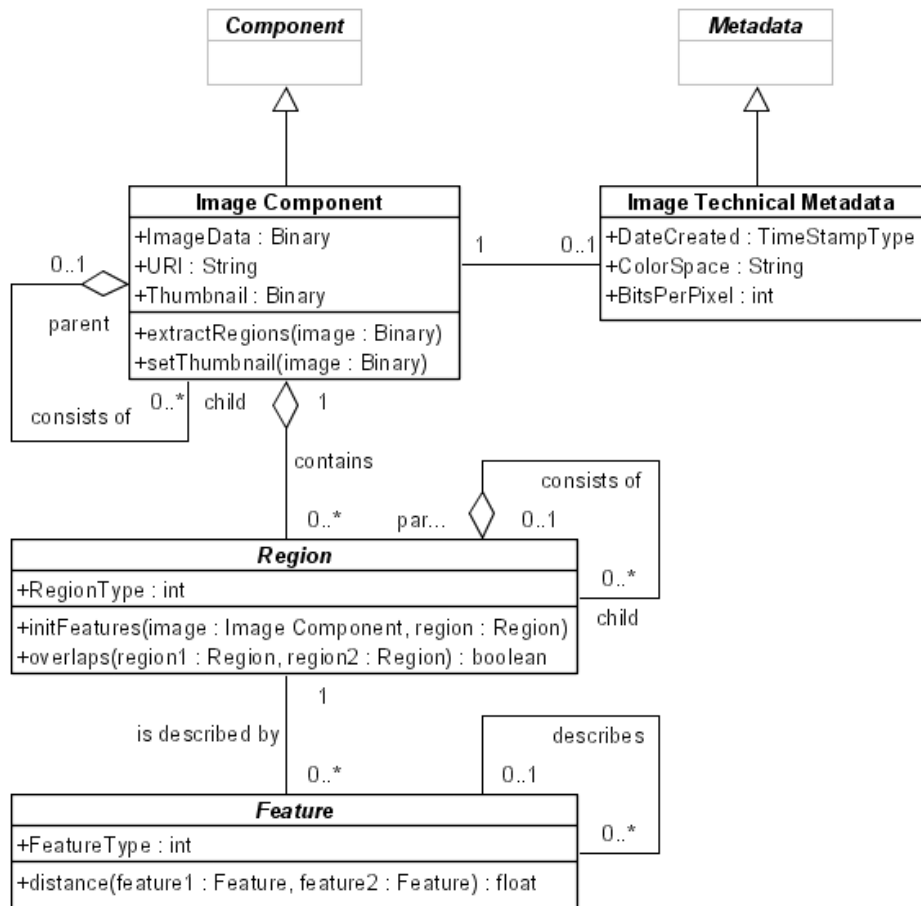


Figure 4.2: Generic Image Database Model (from [IB05])

### 4.2.2 Metadata

Content-independent information, such as creator, creation data etc., is represented by the class **Image Technical Metadata** which is derived from the multimedia document class *Metadata*. In GiACoMo-IRS, the abstract class *Metadata* is associated directly to the *StillImage* class to provide means to represent different types of content-independent data. **Image Technical Metadata** can be regarded as an specialization of this class, which is application dependent.

### 4.2.3 Region

The representation of the content of an image is based on spatial abstractions which are derived from the segmentation of the image. Thereby, the content of an image is interpreted as a set of regions. These regions are represented by the abstract class *Region*. Each image can contain regions or segments of an image. These containment relationships are modeled as an aggregation association. The relationship allows building a hierarchy of regions of an image. Each region can be characterized by its type (circle, ellipse etc.). In GiACoMo-IRS

again a decision has to be made which attributes are mandatory and which are optional. The type of a region can be regarded as a kind of feature associated with the region. However, for most applications it is necessary to assign some kind of localization information to a region, such as centroid coordinates, bounding box etc. This data is represented as an abstract class *RegionLocalization* which is associated with the class *Region* by an optional association. Application specific regions can be defined by the developer by deriving concrete classes from the abstract class *Region*. In GiACoMo-IRS, the association between regions is not restricted to an aggregation, so that other than hierarchical organization of regions are possible as well. An association class **Relationship** is defined, which can be used to assign the appropriate type of relationship. Application-specific specializations for different kinds of spatial relationships is given below.

#### 4.2.4 Feature

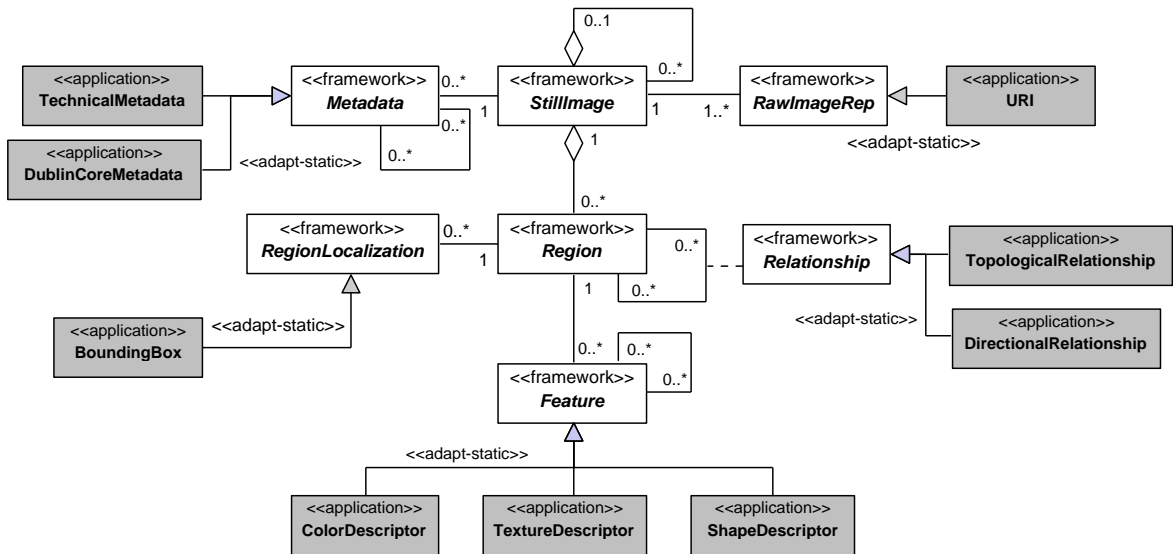
A feature can be assigned to each region of an image, whereby the whole image can also be described as a region. Various features can be defined to describe the content of images, by inheriting from the abstract class *Feature*. These can be low-level features such as height, width, dominant color or color histogram, shape, as well as high-level features, such as names of objects or concepts. Therefore, relationships between features have been defined with an association. These relationships can be used to link low-level features with high-level features for example, when the latter are derived from the first.

#### 4.2.5 Key Attributes and OIDs

In relational databases the concept of keys is used in order to provide a way of identifying data objects. Object-oriented databases bring with them the concept of an object identity, represented by an object identifier (OID) independent of the values of its attributes. OIDs are not explicitly modeled as attributes of the classes. In ODMG 2.0, it is stated that the OID is separate from the state and invisible to the user. As mentioned in [ST93], OIDs are useful for implementing sharing, mutability of values and representing cyclic structures, but they do not carry any meaning for the user and should, therefore, remain hidden. OIDs are implicit (existence-based). The identification of an object can also be explicit (value-based). In the latter, the object is identified with the values of one or more of its attributes. Muller [Mul99], however, uses OID to identify any kind of key that uniquely identifies an object. The explicit approach has the benefit of keeping all your attributes clear and open in your design. This makes the connection between the conceptual model and database schema more direct. In ORDBMSs the instances of structured types do not have the notion of existence on their own. The OIDs are generated in a self-referencing column of the typed table (structured types stored in columns do not have any OID). In the conceptual model GiACoMo-IRS OIDs are implicit, since the model is purely object-oriented and independent of an implementation platform.

#### 4.2.6 Application-specific Classes

The predecessor of GiACoMo-IRS shown in Figure 4.2 determines only the overall structure of the data, but does not give any concrete suggestions for the possible specializations of the abstract classes. Since the role of the framework model is to support the developer

Figure 4.3: Main framework classes and application specific *black box* classes of GiACoMo-IRS

of domain-specific applications not only by providing a reusable architecture, but also by predefining reusable building blocks, the usage of concrete derived classes of the abstract classes, which can be needed in a broad range of CBIR applications should be considered. These derived classes are defined as *black boxes* of the framework, in terms of the definition found in [FPR00], which can be used by the developer later on. These black boxes represent examples of application-specific specializations of the abstract classes, which can be used in an application-specific model. The stereotype `<<adapt-static>>` of the realization association according to the UML-F Profile shows that the abstract classes can be furthermore adapted through subclassing at design-time. In Figure 4.3 the basic framework classes and predefined types of features, image meta data and region relationships of the framework model are shown. In order to keep the resulting model as compact as possible it should be possible to freely omit or exchange the black boxes. Therefore, in the current framework model the `<<adapt-static>>` stereotype means that the examples for the realization of the abstract class are optional for the application-specific model.

#### 4.2.7 CBIRS Data Types

UML does not predefine basic data types, which can be used in application specific models. Only four primitive types have been defined in the standard, which are used only in the meta model itself: Integer, Boolean, String and UnlimitedNatural. Depending on the platform which has to be modeled different basic data types or type systems can be used. Therefore, the standard suggests that for each modeled platform the needed types have to be predefined by specializing and/or instantiating the UML element `dataType`. Since UML is used in this thesis to model a platform independent model, it is not possible to use a platform specific type system. A possible domain specific type system which can be used to model CBIRSs has been defined in the diploma thesis [Czy05]. In [Czy05] the following CBIRS specific data types were suggested: simple data types: *CharacterType*, *IntegerType*, *FloatType*, *BinaryType*, *BooleanType*, *EnumType* and complex data types: *BagType*, *SetType*, *MatrixType*,

*VectorType*, *StructType*. The simple data types can be used, for example, for the representation of image format data with the *EnumType* or raw image data with the *BinaryType*. Some of the complex types can be represented through the multiplicity characteristic of a Property in UML. *BagType*, *SetType* and *VectorType* correspond to a multiplicity greater than 1, and the different combinations of `isUnique:false`, `isOrdered:false`; `isUnique:true`, `isOrdered:false` and `isUnique:false`, `isOrdered:true`, respectively. The complex data types *StructType* and *MatrixType*, where the latter represents a multidimensional *VectorType* have no corresponding presentations in UML. It is possible to represent each *StructType* as Class, but this will not fully correspond to the semantics of a structured type, since data types are not identified through their OIDs, but through their values. Therefore, for modeling GiACoMo-IRS these complex types are defined as stereotyped dataType elements «CBIRSComplexType» «CBIRSStructType», «CBIRSArrayType» and «CBIRSEnumType». The simple types are defined analogously as «CBIRSSimpleType». These stereotypes are added as extensions to UML. In GiACoMo-IRS «CBIRSCharacterType», «CBIRSIntegerType» etc. are defined as instances of «CBIRSSimpleType».

For modeling CBIRS in [Czy05] also some concrete complex data types were instantiated: *Date* as *StructType*, *MonthType* as *EnumType*, *Time* as *StructType*, *Timestamp* as *StructType*, *Point* and *RGBColorType* as *StructType*. These would be represented as instances of «CBIRSStructType» and «CBIRSArrayType» in the GiACoMo-IRS model. The question arises again if it is not better to represent these data types as classes in the model with corresponding methods. From a conceptual model point of view it would make more sense to represent these data types which we need in our model as classes, because we cannot be sure whether the target implementation platform would support the needed functionality for managing these data types. If we take a look at the platforms available for implementing the system it is most likely that these have support for *DateType*, but we cannot be sure that they would support *PointType* or *RGBColorType*. Therefore, in GiACoMo-IRS only the most common complex data types should be represented as data types. The rest have to be added for a specific application.

A complete list of the domain specific data types included in the GiACoMo-IRS model is shown in Appendix A.

#### 4.2.8 Instantiating the Framework

The specialization of the framework requires the following steps to be undertaken after the requirements to the application model are determined:

- Define a specialization for the *StillImage* class and one or more specializations for the *RawImageRep* class
- Redefine the association between *StillImage* and *RawImageRep* class for their specializations
- Optionally, redefine the self-association of the *StillImage* class in its specialization class
- Optionally, define one or more specializations of the *Metadata* class and redefine the association between *Metadata* and *StillImage* for their specializations

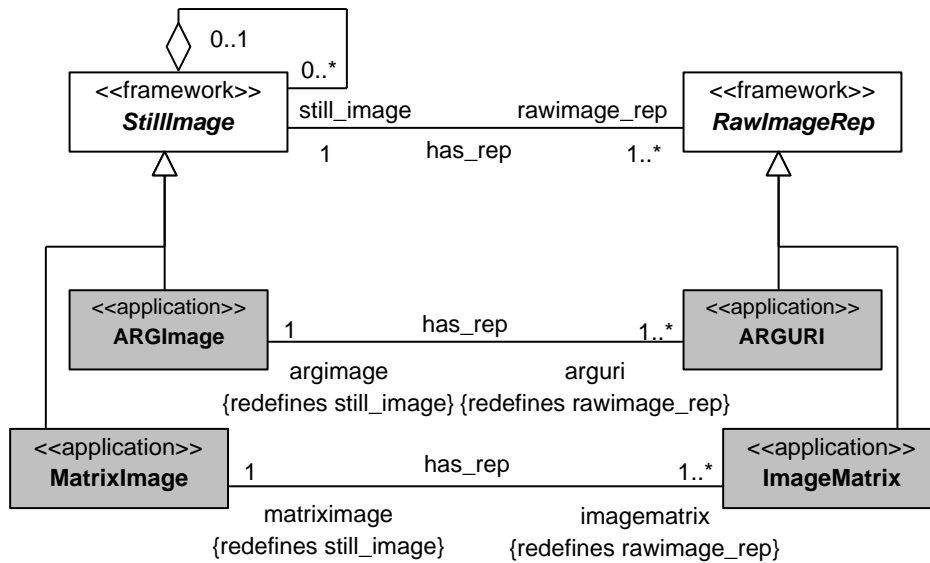


Figure 4.4: Association redefinition example

- If required one or more specializations for the abstract class *Region* should be defined. In this case, the association between the specializations of the classes *Region* and *StillImage* must be redefined.
- Optionally, specializations for the associated classes *RegionLocalization* and *Relationship* can be defined and their associations with the specializations of the class *Region* must be redefined.
- Finally, specialization classes for the abstract class *Feature* can be defined and the self-association can be redefined where necessary.

The associations between the abstract classes have to be redefined in their specializations using the association redefinition capabilities of UML. Association redefinition is a relatively new concept in UML. A detailed discussion on association redefinition is given in [CG06]. Redefinition is more similar to method overriding than to specialization. It is necessary to use this concept because an abstract class is generally a class which is not instantiated and thus no objects of this class and its associations can be created, which can be further specialized. Moreover, in the case of complex abstract class hierarchy it is not straightforward to derive the associations between their subclasses automatically. It depends on the semantics of the subclasses whether and how they can be associated to each other. An example for using association redefinition is given in Figure 4.4.

Two examples for applying the structural framework model for deriving application specific models are represented. The spatial structure of image is represented by Attributed Relational Graphs and 2D-Strings, respectively, which are two of the most common spatial structure representations for images. In Figure 4.5 the representation of Attributed Relational Graphs (ARGs) is shown. In Figure 4.6 the representation of 2D-Strings is illustrated. In these diagrams the UML notations for association redefinitions are left out to avoid overloading the diagram.



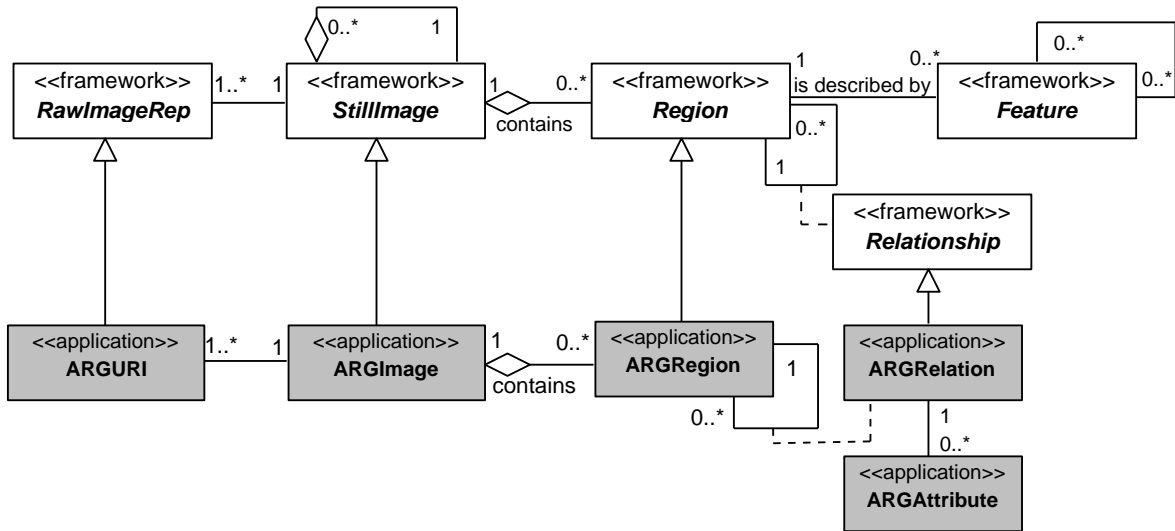


Figure 4.5: Modeling Attributed Relational Graphs Image Representations

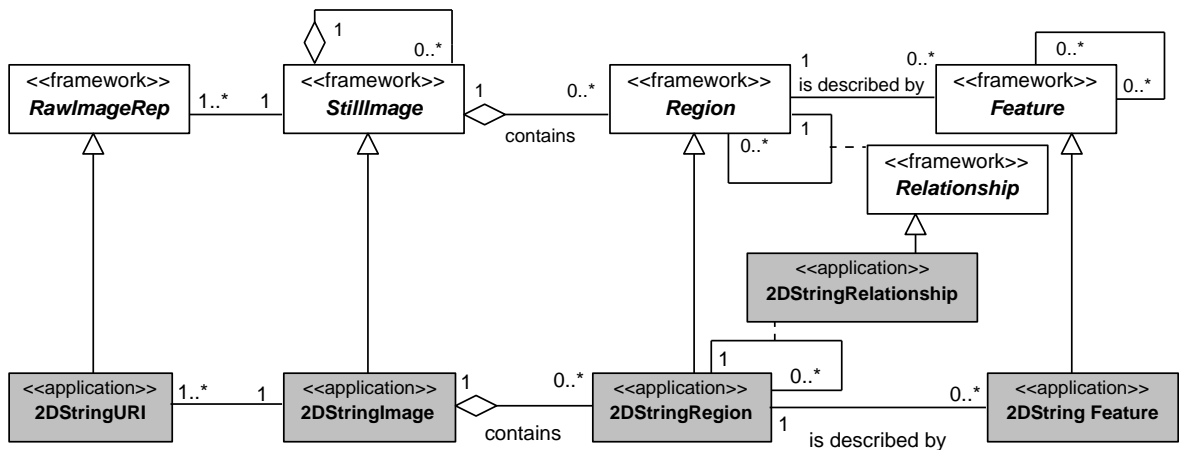


Figure 4.6: Modeling 2D-Strings Image Representation

### 4.3 Modeling Functionality of CBIRS Components

Integrating retrieval functionality is the second group of requirements of the conceptual image retrieval model. In this section, the functionality groups from a level-of-design point of view are defined, and a general design approach for integrating extensible and adaptable functionality in the model is described. In the design of a CBIR system two kinds of functionality can be distinguished:

- from the view point of a system user there is the application (system) functionality which is represented by the user interfaces;
- from the view point of the system developer there is the object functionality, which has to implement or provide the system functionality.

The application functionality is modeled at first by means of UML use cases and activity diagrams in order to define the functionality of the application as a whole, which is required by the users. The two groups of operations, which a CBIR system has to support from the view point of a user, are Updates (Insert, Delete, Update) and Queries on images and their content. From the view point of a system developer each of these functions has to be integrated into the building blocks of the application. This is achieved by mapping the activities from the detailed activity diagrams onto classes and methods in the class diagram.

### 4.3.1 Updates

The operations from the first group have very similar behavior, that is why only the Insert operation is considered as an example. In Figure 4.7 the use case diagram of the Insert operation is shown. Analogously, the use cases of the other update operations are defined. For each class in the current class diagram of the framework, which has to be made persistent in the system, there is a corresponding use case in the Insert use case diagram. Each of these use cases are complete by themselves and so can also be performed separately from the others. The integrity constraints for inserting dependent objects, such as Metadata, which have to be assigned to a particular Image and cannot exist alone, are not represented through the use case diagram, but through the mandatory association with an image object shown in the class diagram. Some use cases may extend others if certain conditions are met. This means that the extending use cases are inserted in a specific point of the extended use case if the condition is true. For example, the regions of an image can be inserted during the insertion of an image object if a variable for segmenting the image  $\{Segment\ Image\}$  is set to true. An image must have at least one image representation, therefore, the Insert RawImageRep use case is mandatory included in the Insert Image use case through the  $\llinclude\gg$  dependency. The  $\llinclude\gg$  stereotype of the dependency means that the included use case is always included at a certain point of the including use case. An extending or included use case can be invoked more than one time from an extended or including use case, respectively.

In order to provide a more detailed description for the use cases shown in the use case diagram, an activity diagram for each use case of an operation is designed. In Figure 4.8 the activity diagrams for the Insert Image use cases is shown. This activity diagram includes anchors to the activity diagrams representing the included or extending use cases - Insert RawImageRep, Insert Metadata and Insert Region. From the defined concrete activities (not anchors to other activity diagrams) it is now possible to identify the single methods which have to be supported by the classes in the class diagram in order to integrate the Insert Image functionality in the model. These methods determine the object behavior. In some cases new classes have to be added to the class diagram of the framework, such as the ImageStorageMechanism, the SegmentationAlgorithm classes to represent an interface for certain functionality which can have different interchangeable implementations in an application. These classes are used to provide the possibility to redefine and dynamically bind different implementations for a particular functionality.

In order to enable the integration of different specializations for the segmentation of images depending on the application requirements, the concept of template and hook methods is adopted in the class diagram. The same kind of template-hook combination can be used for the extraction of features in order to provide adaptation possibilities for these methods as well. The usage of template and hook methods in framework architectures is described in [FPR00].

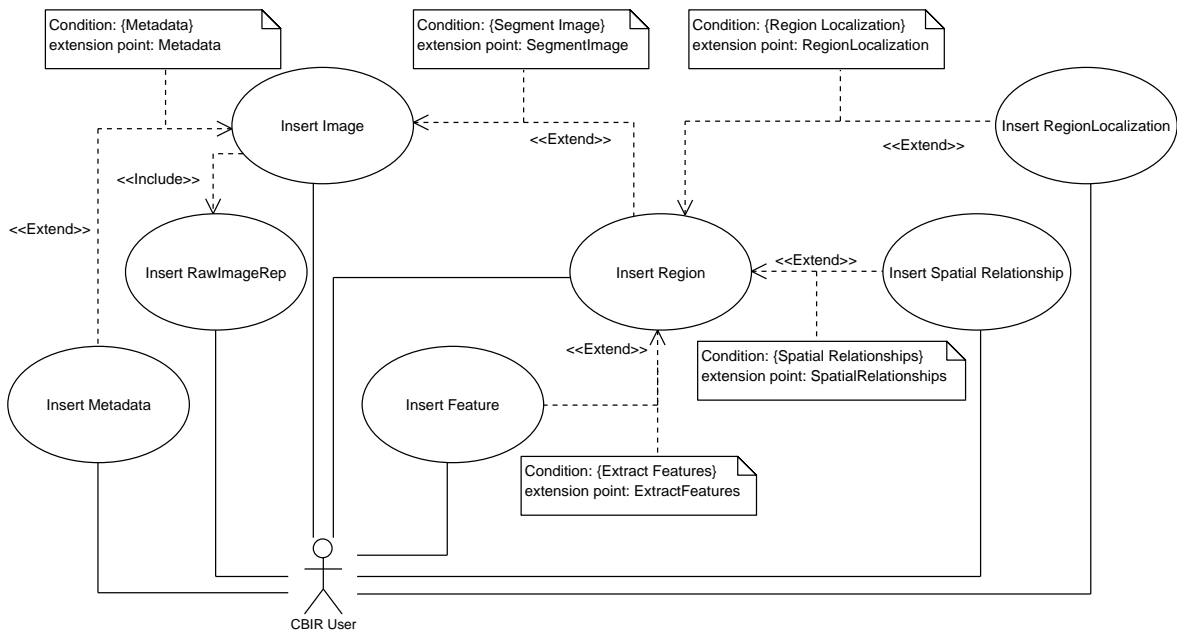


Figure 4.7: Use Cases for the Insert operation

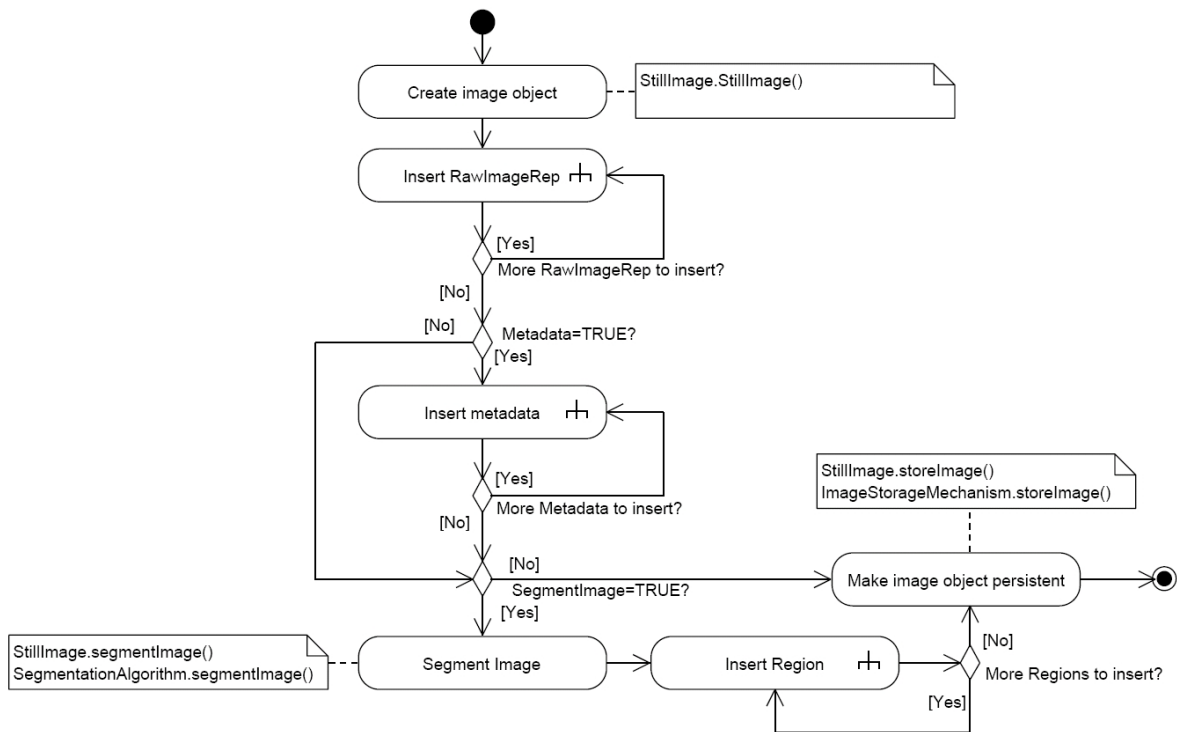


Figure 4.8: Activity diagram for the Insert operation

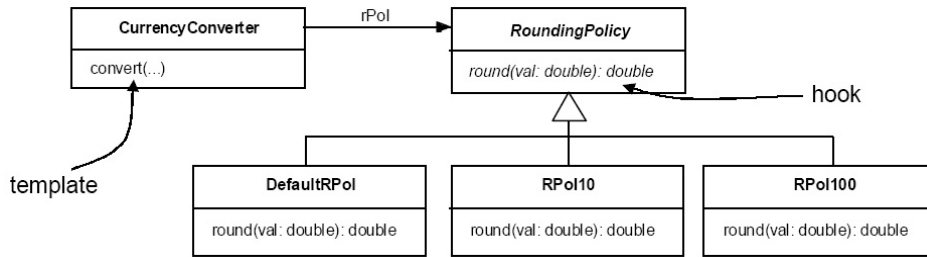


Figure 4.9: Example of a separation pattern for the template and hook methods (from [FPR00])

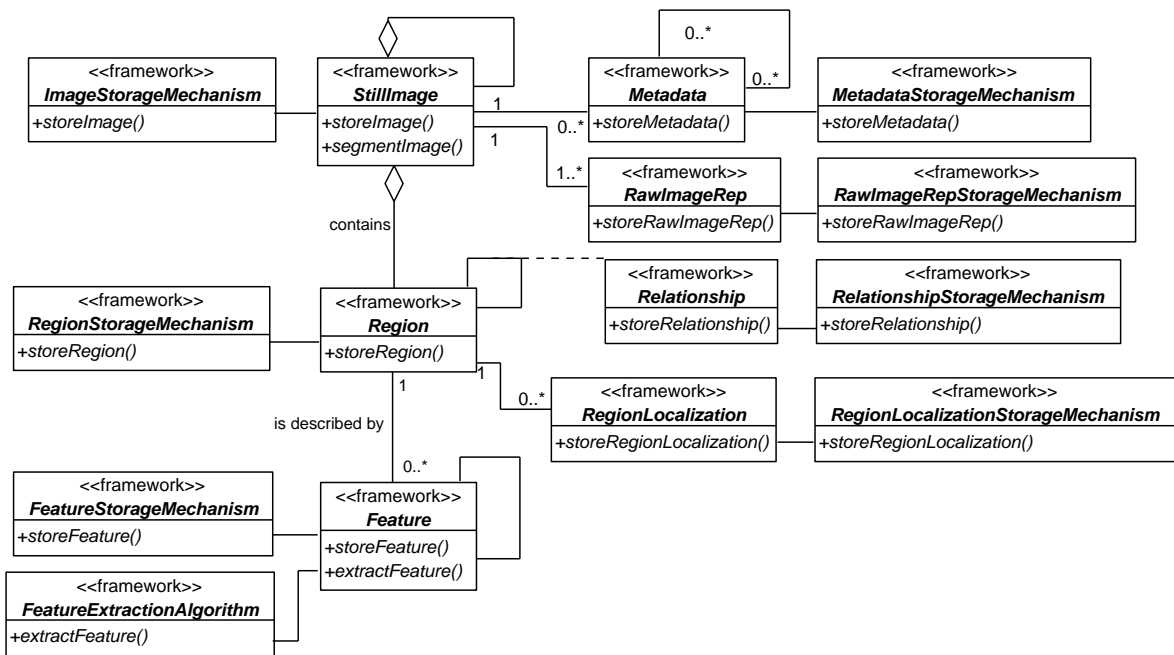


Figure 4.10: Integration of the insert functionality in the framework model

There are two possible construction principles, which can be applied for the template-hook concept. The first is called the principle of Unification. In this case, the template and hook methods reside in the same class and the adaptation of this construct is performed by subclassing the template-hook class and overriding the hook method. In GiACoMo-IRS, however, it is also possible to have different specializations (concrete subclasses) of a segmentation algorithm for the same types of images. Therefore, the separation construction principle is also adopted in the model, where the hook method, which needs to be adapted, resides in a different class. The class containing the template method is associated to the class containing the hook method. Through this directed association the template method invokes the hook method. In Figure 4.9 an example of a separation pattern for template and hook methods is shown.

The framework model classes with the integrated functionality for the Insert operation are represented in Figure 4.10.

The behavior of the objects in the current context refers to the implementation of the func-

tionality of the system. It is expressed through the operations of classes. However, there are no means to insure the consistence of the two behavior models automatically, because there are (too many) possible ways to realize the behavior of the system. So this mapping has to be done manually if new system functionality has to be added to the model. In the framework model the workflow of the system operations as represented by the activity diagram is not included. The system behavior shall be introduced in the platform specific model, because the system functionality depends strongly on the implementation platform. Database systems, for example, already support generic update and query functionality which does not have to be implemented separately.

### **4.3.2 Queries**

The Query operations are the part of the framework model where most variation points exist. This is due to the fact that there are a lot of possibilities to model the retrieval functionality of a system. The query model, also referred to as retrieval model, is created depending on the retrieval task, which has to be realized. Different retrieval models can be defined for the same data model in the same CBIR System. The retrieval models palette is quite rich, therefore, some restrictions for the framework model had to be met in this thesis. First of all, the kinds of query tasks supported by the framework model are determined as shown in Figure 4.11. The model supports similarity queries, based on global features, local features, meta data and structure of the images and the combination of these, as well as exact match queries on the latter. Furthermore, the metric and data mining approaches for image similarity retrieval are chosen to be supported by the model. The metric retrieval approach is one of the most used in the practice. The data mining approach, such as the classification of images, is especially useful to derive high-level information from the low-level features. It should be mentioned that these two approaches are represented in the model independently. This means that combined queries using both retrieval approaches are not yet considered. The problem which has to be solved before modeling combined queries is how to combine the different results. In the case of information retrieval the result of the query is a ranked list of similar images and in the case of a classification query, the result contains one or more names of categories, to which the query image might adhere.

The representation of the information retrieval and data mining query operations in GiACoMo-IRS is discussed below in more detail.

### **4.3.3 Modeling Retrieval Functionality**

The main purpose of a content-based image retrieval application is to provide mechanisms for accessing images based on a similarity evaluation. Therefore, part of the design process of image retrieval systems is the design of the retrieval model, where the possible queries are specified and the algorithms and data structures for processing these queries are determined. These access mechanisms depend to a great extent on the application domain and the retrieval tasks and thus they are part of the system with most variation points. In other words, there are numerous ways to provide access possibilities to images, depending on the types of queries, which have to be answered, the information retrieval approaches and the image data abstractions (features) which have to be used. How these factors influence the design of the retrieval model is shown in Figure 4.12. Existing similarity models for the visual information

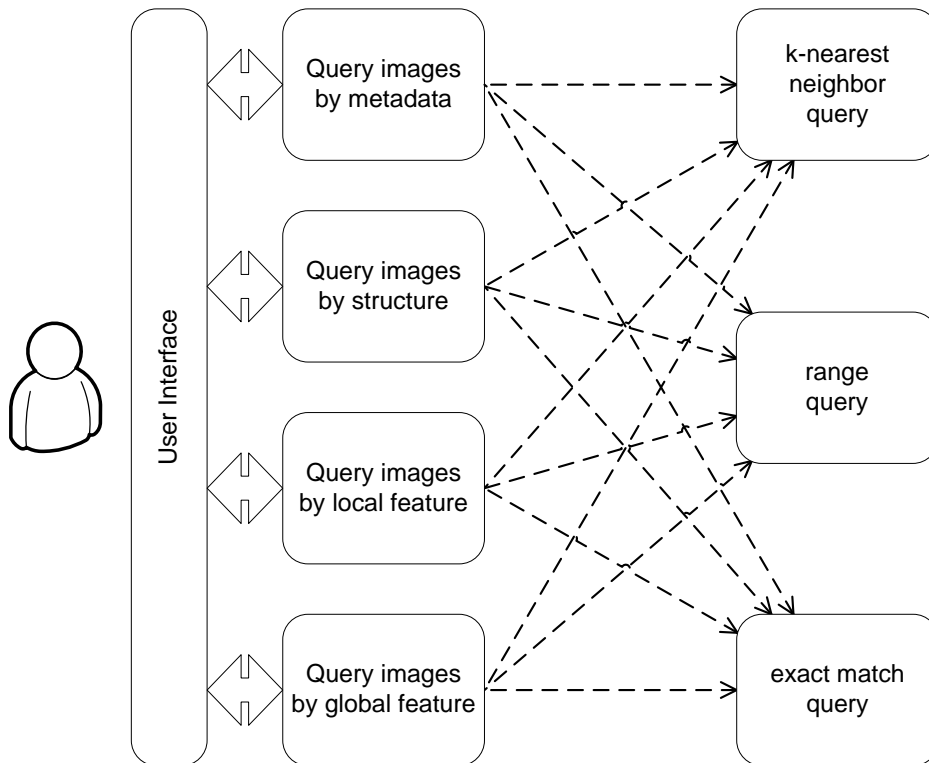


Figure 4.11: Query tasks supported by the framework model

retrieval have been summarized in [Bim99, VC02].

The image data abstractions and the retrieval tasks for the GiACoMo-IRS model are determined in the previous section. At this point, another restriction has to be made for the retrieval model design by choosing two concrete information retrieval approaches for the retrieval: the metric information retrieval approach and classification as a representative of the data mining retrieval approaches.

In this section, extensions for the GiACoMo-IRS are defined which can support the modeling of retrieval functionality, based on these two retrieval approaches.

#### 4.3.3.1 Information Retrieval Approach - The Metric Model

The metric approach is based on comparing the feature representations of images in the database with the ones of a query image, using a distance function. As a result the distances representing the degrees of similarity for all the images in the database are returned. The result can be assessed based on a k-nearest neighbor or range metric in order to return only relevant images.

The modeling of Query operations is tackled analogously to the Update operations. The query tasks from Figure 4.11 are represented as use case and activity diagrams in order to determine the needed methods which have to be supported, and then to map them to corresponding classes. A query is given as input for an image similarity query and is analyzed to extract its structure in form of regions and relationships between regions, and its content in terms of features. Additional meta data could be used to support the query processing.

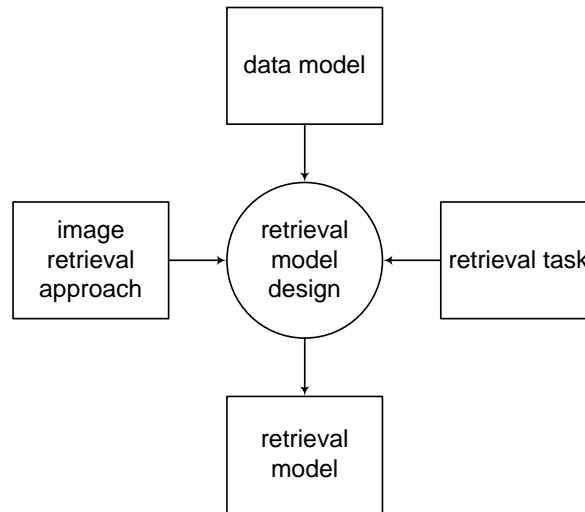


Figure 4.12: Retrieval Model Design

In the framework model there are already functions defined for the extraction of the feature representation, which have to be used during the Update/Insert of images. These can be used to analyze the query image when formulating the similarity query.

The queries based only on global features or meta data, or only based on structure can be processed in a relatively straightforward way using a suitable distance function for their comparison. Thus, the framework model requires a distance calculation method in the corresponding classes - Feature, Metadata, Region respectively. Combined queries and local feature queries, which require the combination of more than one feature and more than one region in the similarity measure, need additional aggregation functions in order to combine the distances from one or more features, or one or more regions. If we consider an image database with a set of images, where each image  $I$  has a set of regions:

$$R_I = \{r_i : i = 1..l\}$$

Regions can be salient objects, image regions with homogeneous texture or color or simply blobs of any shape. Each region is represented by a set of features:

$$F_{r_i} = \{f_k^{r_i} : k = 1..n\}$$

Different types of features can be associated with an image region, such as color histogram, bounding box of the region, associated names of concepts etc. If we have a query image  $Q$ , then the query would also be represented by the set of regions of the query image and their corresponding features:

$$R_Q = \{r_j : j = 1..m\}, F_{r_j} = \{f_l^{r_j} : l = 1..n\}$$

In order to find all similar images of the query image in the database we have to compare the query image to each database image and derive the similarity between the query image to the database image. The similarity is calculated through its opposite - the distance between the images. The distance  $D$  between the two images can be represented by the distance of the region(s) of the images as follows:

$$D(I, Q) = D(R_I, R_Q)$$

Where the distance between the regions can be represented by an aggregate function  $f$  on the distances between the feature sets, representing the regions:

$$D(R_I, R_Q) = f_{i=1..l, j=1..m}(d(F_{r_i}, F_{r_j}))$$

The function  $f$  has to combine the distances between multiple regions of the query and the target image. Depending on the application this function can have different semantics, e.g., - it can simply check if for each region of the query image a region in the target image exists where  $d(F_{r_i}, F_{r_j}) \leq \epsilon$  or - calculate the needed transformations, which need to be performed on the query regions to receive the target regions. In this function, additional factors can be considered, such as the weights of the regions, the number of regions in the query and in the target image. Spatial relationships between regions, reflecting the structure of the image can also be integrated in the similarity measure.

The distance between a region  $i$  of the query image and a region  $j$  of the target image is an accumulating function  $g$  on the basic distances between the single features of a region:

$$d(F_{r_i}, F_{r_j}) = g_{k=1..n}(\delta_{f_k}(f_k^{r_i}, f_k^{r_j}))$$

The function  $g$  has to accumulate the distances between different types of features. If the different types of features can be represented in the same feature space, then the function  $g$  can be the weighted sum of all single feature distances. The function  $g$  represents the distance between two regions in terms of their features. The basic function  $\delta$  represents a distance function for a feature space. It is defined only for feature values of the same feature type.

In Figure 4.13 the integration of the metric approach methods and classes for the “Query images by local features” use case in the framework model is demonstrated. The class diagram also depicts the classes and methods responsible for the image segmentation and feature extraction.

The feature specific distance functions are defined as methods of the Feature classes, on which they are applied. The aggregate distance functions for combining the distance measures of different features or regions are defined in the parent class in the image data structure hierarchy. Since different kinds of distance functions exist, e.g., Manhattan, Euclidean, Hamming distance etc., in this case, we also choose the template-hook method combination approach to define the distance measure functions. This approach requires overriding the hook methods or deriving their classes to provide adaptation possibilities. The combination of features or other characteristics of the images to perform a query requires the introduction of weights for the participating partial queries. In many applications these weights are empirically predetermined, but they can also be acquired from the user during the query formulation process. Therefore, we leave these outside the framework model.

The same adaptation mechanism as the one for the adaptation of the insert functionality can be applied.

#### 4.3.3.2 Data Mining Approach - Classification

Another approach for retrieving image data offer data mining methods. There are numerous definitions for Data Mining in the literature, which describe it as the process of finding



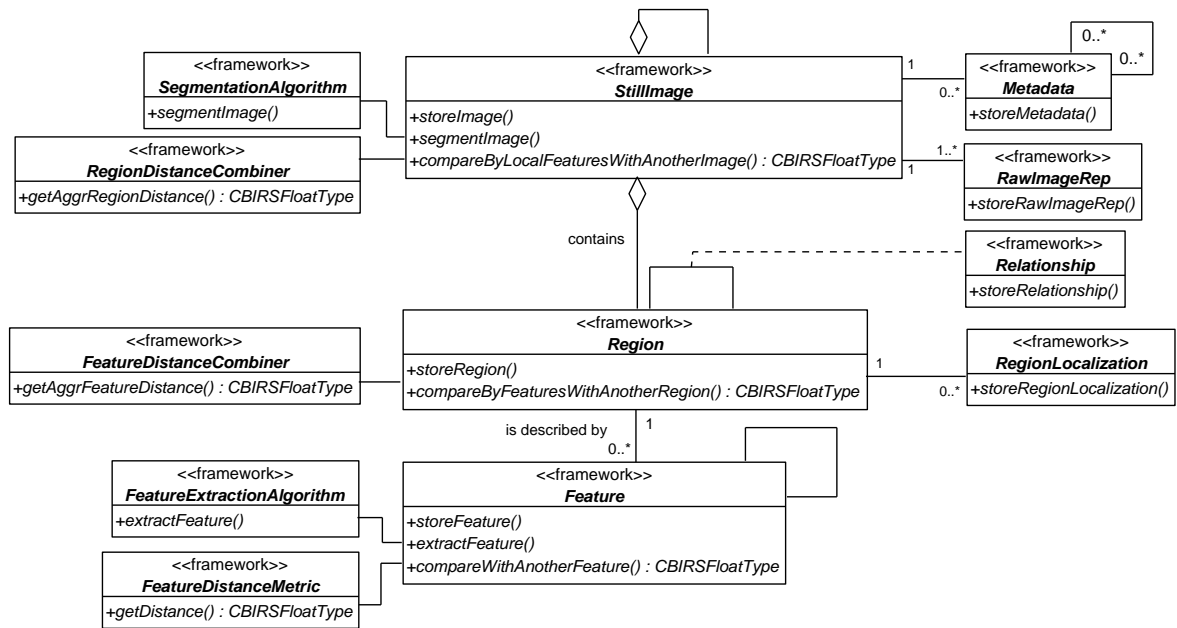


Figure 4.13: Class diagram of the “image insertion” and “query by local features” classes and methods

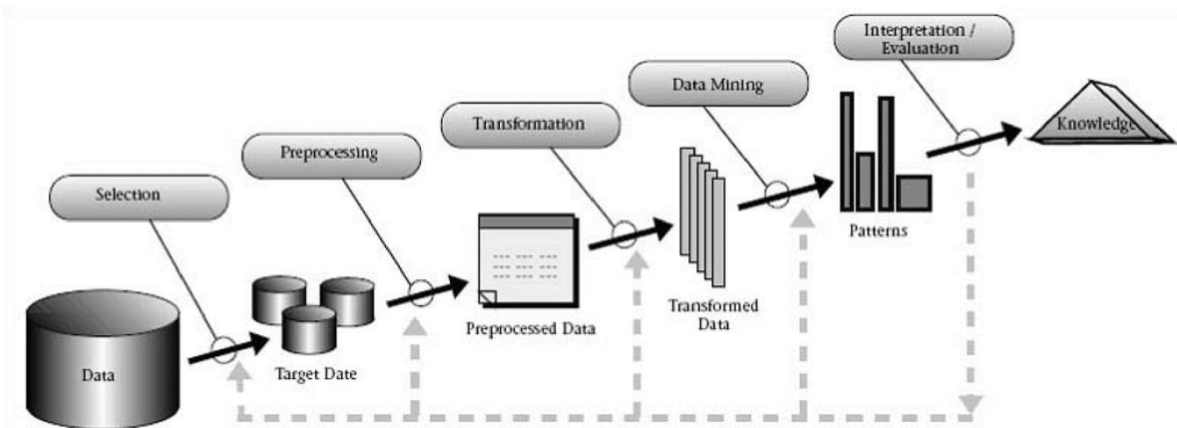


Figure 4.14: Phases of the KDD process (from [FPSS96])

patterns in data sets and dependencies between these data. In [FPSS96], Data Mining is defined as one of the phases of the KDD (Knowledge Discovery in Databases) process. The KDD process is the discovery of new knowledge based on the stored information. The phases of this process comprise of the choice and cleaning of data, the application of Data Mining algorithms and finally the interpretation and application of the knowledge (see Figure 4.14).

The KDD process can be applied for the classification of images. The data in this case is represented by the extracted content abstractions from the images (in form of feature vectors). The aim of the KDD process is to find some kind of feature vector patterns, which can be used to assign the feature vectors and respectively the images to related classes.

The input parameters of the data mining algorithms are the data to be analyzed and the out-

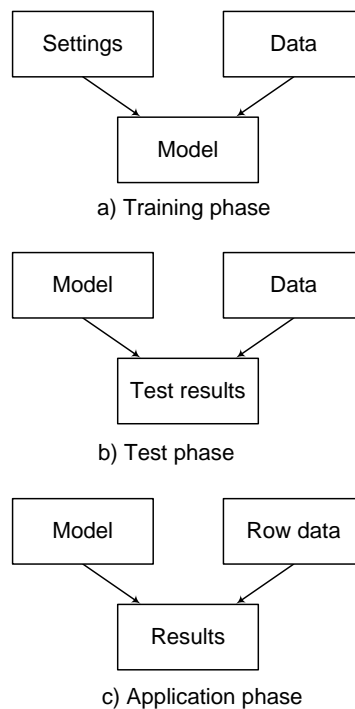


Figure 4.15: Phases of the classification process (based on [SQL00])

put is a model for representing the found patterns in the data. The input data is represented as a set of instances. Each instance is described by the values of its set of attributes. Each attribute can have a different type, e.g. we can distinguish between continuous and discrete types of attributes. Furthermore, the continuous attributes can be divided into interval and ratio types, where ratio types have a fixed null point and interval attributes do not have any. Discrete attributes can have only one value from a specified value set. Nominal and ordinal attributes can be distinguished, where nominal attributes do not have an order set of values (e.g., red, brown, yellow) and ordinal attributes do (e.g.,  $1 < 2 < 3 < 4$ ).

There are basically two types of models for the representation of patterns, predicative and descriptive. Descriptive models describe common characteristics of the data, whereas predicative models allow conclusions and predict future developments. Examples of descriptive Data Mining techniques are association rules and clustering, and for predicative techniques examples are classification and regression.

Classification enables the assignment of objects (e.g. images), respectively instances (features) to different predefined classes. Thereby the input instances have to be already assigned to specific classes, which is represented by a special nominal attribute of the instances, called the class attribute. The aim of the Data Mining algorithm in this case is the development of a model, which can be used to assign a value to the class attribute of a new instance. According to [HK00] the classification process can be split in two phases: training and testing. The SQL/MM Standard part for data mining also introduces an application phase as shown in Figure 4.15.

A generic classifier, which can be adapted for different classification algorithms, is provided in GiACoMo-IRS using the concepts suggested in the work of Henning Masuch [Mas05].

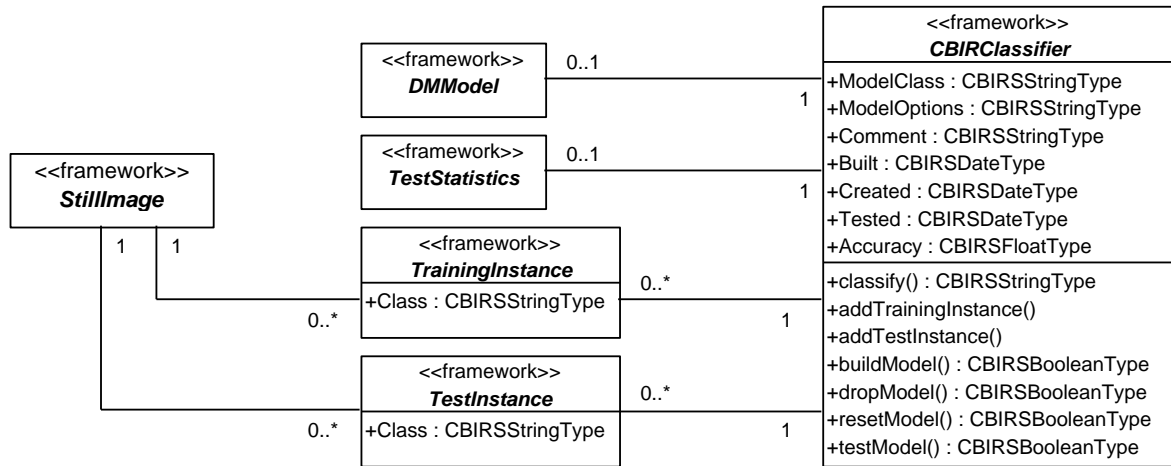


Figure 4.16: Generic classes for the classification of images

The aim of the classifier in [Mas05] is to integrate a classification application for identifying scribes of music note images implemented in Java into the ORDBMS DB2. This classifier uses the SQL/MM standard as a basis. The generic classifier provides operations corresponding to each of the phases of the classification process as shown in Figure 4.16. Thus, these operations have to be applied in a specific order in the application.

The attribute **ModelClass** of the framework class *CBIRClassifier* provides the possibility to define the type of the classifier, for example decision tree, artificial neuronal network etc. Parameters for the classification process can be provided through the **ModelOptions** attribute. The **Comment** attribute can be used to store any additional information useful for the user of the classifier. The three timestamp attributes can be used to check if the classifier has been trained and tested. The instances or the objects used for training and testing the classifier are represented through associated classes, which are themselves associated to corresponding *StillImage* objects. The developer should provide one additional attribute for each instance, which shows the belonging of the instance to a specific class. After the test phase the accuracy of the model and test statistics are determined. They can be stored in the corresponding attributes of the class. The classification model itself can have different structure depending on the classifier type used and thus has to be subclassed from the developer. Analogously the test statistics can have varying data structure.

The operations of the generic classifier include getter and setter methods for the attributes as well as methods corresponding to the different classification phases as follows.

- initialization: The implicit constructor of the *CBIRClassifier* class has to provide values for all the attributes needed to build a classification model, such as the **TrainingInstances**, **ClassType** etc. The `addTrainingInstance()` and `addTestInstance()` methods can be used to construct the set of instances from images in the collection.
- training: The operation `buildModel()` is called once for a certain set of training data. It has to be repeated if the data has changed. It initializes the **DMMModel** attribute of the generic classifier.
- testing: The operation `testModel()` is used to determine the accuracy of the model. It

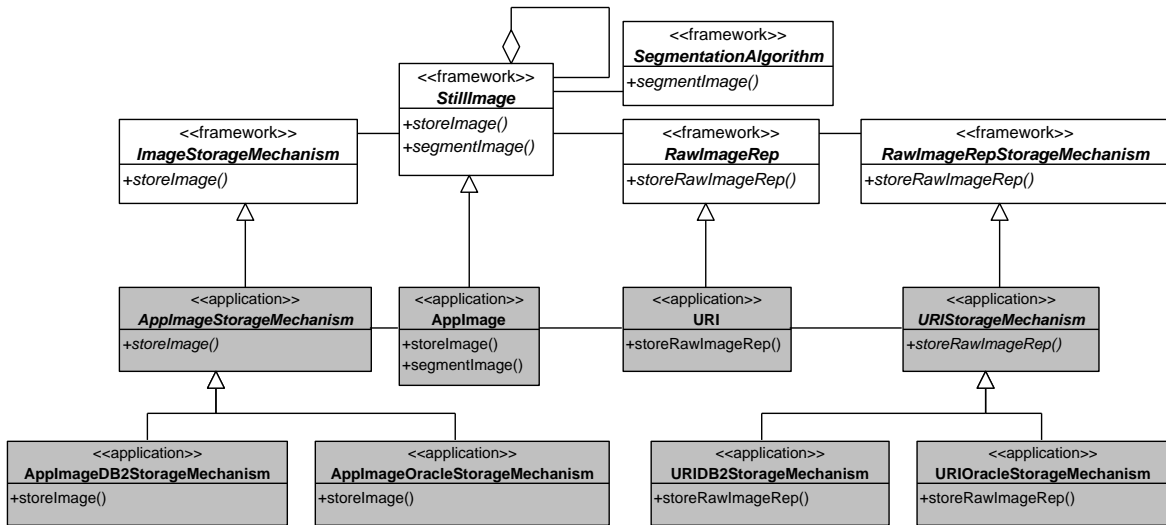


Figure 4.17: Modeling functionality with the GiACoMo-IRS framework

makes use of the **TestInstance** attributes of the generic classifier.

- application: To classify new image instances the `classify()` operation has to be implemented. The operation has to take care of the conversion of the *StillImage* object into an instance for the classifier. Therefore, the structure of the *TestInstance* class can be used.

It is assumed that each classifier requires a combination of attributes and operations which strongly depend from each other. Therefore, all methods of the classifier have to be adapted using the Unification pattern. If a new classification algorithm has to be modeled it should be created in a new type of classifier, i.e. a new classifier class.

#### 4.3.4 Implicit Object Behavior

Apart from application specific functionality, objects need to implement some implicit behavior, which determines their life-cycle. Generally these methods comprise of (see also [Heu97] Chapter 6): constructor/destructor, identity, equality(shallow, deep), assignment, copy (shallow, deep), equals. For each class in GiACoMo-IRS these operations are defined implicitly.

#### 4.3.5 Instantiating the Framework

There are two mechanisms to adapt the functionality of the framework. First, the methods of the abstract classes can be redefined in their subclasses and, second, the hook methods can be redefined (implemented) by subclassing the hook classes. The first method is carried out on the result of the data structure adaptation - the derived concrete classes can override the abstract methods, and the second method requires deriving new classes from the hook classes and redefining the required methods. Which one of the methods should be used depends on whether the application should support different algorithms for the same functionality which should be interchangeable in the application. In Figure 4.17 the adaptation of the framework

to support a concrete Insert Image functionality is shown. Only one segmentation algorithm should be defined in the application. Therefore, the adaptation of the `segmentImage()` function is done by overriding the method `segmentImage()` in the subclass of `StillImage`. The Unification pattern is applied in this case for adapting the functionality. The adaptation of the `storeImage()` and `storeRawImageRep()` is done based on the Separation pattern. For each hook class a specialization in the application domain is defined and the association between the hook-class and the template class is redefined in the application domain. Different specializations of the hook-method for storing are provided by subclassing the hook class.

## 4.4 Summary

In this chapter, a modeling approach for creating application-specific CBIR models was proposed. The chosen modeling strategy uses the UML framework model GiACoMo-IRS as a starting point for deriving application-specific models. Both the structural and the functional aspects of a CBIR system were reflected in the framework model. Guidelines and examples for the instantiation of the different parts of the framework model were described. In comparison to existing CBIR system models, in this thesis not only an abstraction of generic CBIR features was represented, but also a methodology for using these abstractions to create user-defined models was described. A generation mechanism for the implementation of the instantiated GiACoMo-IRS models is discussed in the following chapter.

## Chapter 5

# Mapping Rules for Generating CBIRSs on Top of ORDBMSs

The implementation of the application on a specific platform is the next step for the developer after the conceptual modeling step. The automation of this step is the main topic of this chapter. More specifically, the mapping of the PIM onto an implementation model is discussed. The mapping of the model follows the meta model-based approach described in Chapter 3. The mapping rules are defined based on the meta models and have to be applied on the concrete domain specific models.

In this chapter, first, the meta models of the PIM and PSM are revisited. Both meta models use extensions of UML to represent domain-specific or platform-specific concepts. These concepts have to be defined clearly before the transformation rules can be elaborated. The transformation rules for each of the PIM concepts are then represented. To what extent these mapping rules can assure a correct transformation of the concrete models, i.e. preserve the information represented in the PIM in the transformed PSM, is also examined. Therefore, the mapping rules defined in this chapter are grouped according to their characteristics, such as if the rule can be defined as a direct mapping or if there is more than one possibility for mapping a PIM concept onto the PSM.

### 5.1 Modeling Deployment

The GiACoMo-IRS model employs only domain-specific concepts. No specific software architecture is considered. In model-driven development at this point, after the domain-specific model has been defined and before it can be transformed into an application, a decision about this architecture has to be made. This step resembles a kind of deployment step at which the elements of the model have to be transformed or grouped into architectural components (tiers). Since the choice of supporting one specific architecture consisting of two tiers is made in Chapter 3, two new stereotypes have to be introduced in the model at this stage. The stereotypes «persistent» and «application logic» have to be applied to classes in the model in order to associate them to the specific tier. A class can be at the same time a «persistent» and an «application logic» class. This would lead to the multiple participation of the class in the transformation. An example of the deployment annotation of the model is shown in Figure 5.1.

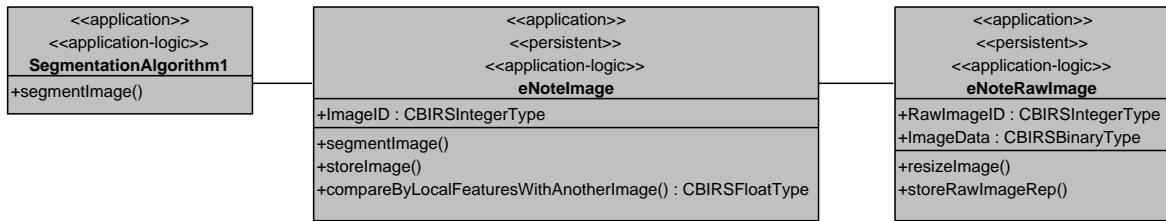


Figure 5.1: Example for the deployment annotation

These extensions do not convey domain-specific information to the PIM. Their role is to support the transformation process. Therefore, the deployment model can be considered as partly platform specific. In this way, the first step of the transformation process is undertaken. In the annotated model it is now possible to identify which parts of the model have to be mapped to the platform for the persistence and which to the application logic platform. Before we can define how these mappings have to look like, in the following section, the meta model elements of the PIM which have to be mapped and the needed concepts of the PSM meta model are reviewed.

## 5.2 Meta Models

### 5.2.1 PIM Meta Model

GiACoMo-IRS and its derivatives (application-specific models) are represented as UML class diagrams, using some additional domain-specific stereotypes and data types. Therefore, the meta model of the PIM can be regarded as an extended subset of UML. In this section, the PIM meta model is summarized. A full list of the PIM meta model can be found in Appendix A.

#### 5.2.1.1 UML2.0 meta model

The UML meta model is defined in the UML2.0 Standard [UML07]. The standard consists of two parts:

- **UML2.0 Infrastructure:** describes the architectural foundation of the UML2.0 Superstructure. It consists of two basic packages Core and Profiles. Core consists of Basic, Abstractions, PrimitiveTypes and Constructs packages and defines a library of meta-classes which can be imported or specialized to define other meta models. The InfrastructureLibrary is used to define meta meta models such as MOF as well as meta models such as UML. In fact UML is an instance of MOF, so each UML metaclass is an instance of an element in the InfrastructureLibrary. The package Profiles provides means to adapt the meta models to particular platforms, such as EJB for example, or domains, such as software modeling.
- **UML2.0 Superstructure:** defines the user-level constructs of UML, based on the foundation language constructs, defined in UML2.0 Infrastructure.

Additionally, two auxiliary parts have been defined:

- **Interchange:** defines a format and methodology for transforming, storing, and exchanging between applications of UML models. It defines the XML Schema for XMI.
- **OCL:** can be used to represent class invariants, pre- and post- conditions of operations etc. It cannot be used to represent flow control or program logic.

UML2.0 concepts used for the modeling of user applications are defined in the part Superstructure. For modeling an image database application, of main importance are the concepts used for building structural diagrams, such as class and package diagrams. Some of these concepts are:

- **Class:** A class describes a set of objects that share the same specifications of features, constraints, and semantics. The features of a class are its attributes and operations. Classes are used to represent the main elements of the digital image object types, their attributes and their behavior. The behavior of the class also defines operations for the creation, storing and deletion of objects. A class can also implement a global behavior, available to the whole system.
- **Associations:** An association specifies a semantic relationship between instances of classifiers. The instances of an association are called links. It has at least two ends represented by properties, each of which is connected to the classifier of the end. An association may represent a composite aggregation (i.e., a whole/part relationship). In GiACoMo-IRS associations are used to represent different kinds of dependencies between the classes composing a digital image type, such as part-of relationships, hierarchical organizations or derivation.
- **Generalization:** A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier. Generalization plays an important role in the GiACoMo-IRS model especially if it is used to derive the application model, because the derivation process uses mainly Generalization for creating application specific classes and associations.
- **Interface:** An interface is a kind of classifier that specifies a set of operations and constraints, which have to be implemented and fulfilled, respectively by each class implementing the interface. Interfaces themselves are not instantiable. They are instantiated indirectly by an instantiable classifier, which implements the interface. Although interfaces are not used in the GiACoMo-IRS model, they should be considered in the mapping rules because application specific models might consider using this concept, for example in order to model multiple inheritance.
- **Package:** A package is used to group elements, and provides a namespace for the grouped elements. GiACoMo-IRS is defined as a package in UML and thus can be imported or merged with other UML packages. For example when deriving an application specific model the GiACoMo-IRS package is imported into the new application specific package.



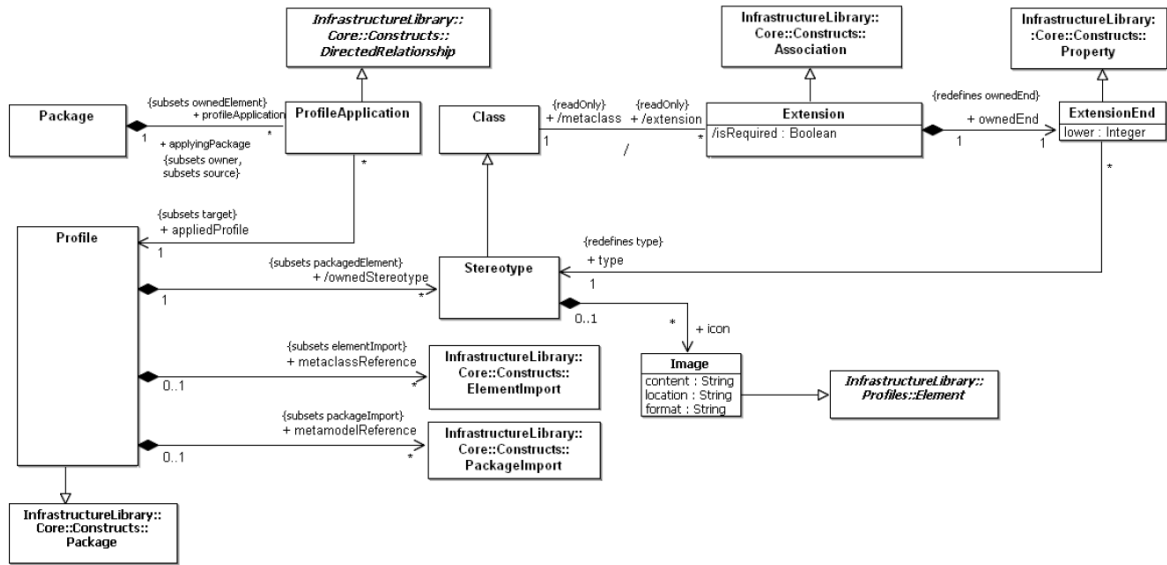


Figure 5.2: UML meta model part for Profiles (from [UML07])

### 5.2.1.2 UML Profiles

Tagged values, stereotypes and profiles represent an extension mechanism to the UML standard. They can be used to define specific domain dependent meta models. Stereotypes are specific meta classes, tagged values are standard meta attributes, and profiles are specific kinds of packages. In Figure 5.2 the UML meta model parts for UML profiles are depicted.

### 5.2.1.3 UML-F Profile

Since the GiACoMo-IRS model is a framework model it has to provide information about how it can be used to derive application specific models. The UML-F Profile is used for that. The UML-F Profile introduces the notion of tags, where no differentiation between stereotypes and tagged values is made. In Figure 5.3 the different layers of tags are shown. In GiACoMo-IRS and its derivatives the following tags are used:

- **«framework»**: applies to Class, Package, Interface and means that the element belongs to the framework model
- **«application»**: applies to Class, Package, Interface and means that the element belongs to the application-specific model
- **«adapt-static»**: applies to Interface, Class, Method, Generalization and shows that the element can be adapted during design-time through subclassing.
- **«template»**, **«hook»**: apply to Class and Method and show which functionality has to be adapted by subclassing the classes and redefining the methods. For more specificity higher abstract-level tags such as **«Unification-template»** and **«Unification-hook»** or **«Separation-template»** and **«Separation-hook»** can be used.

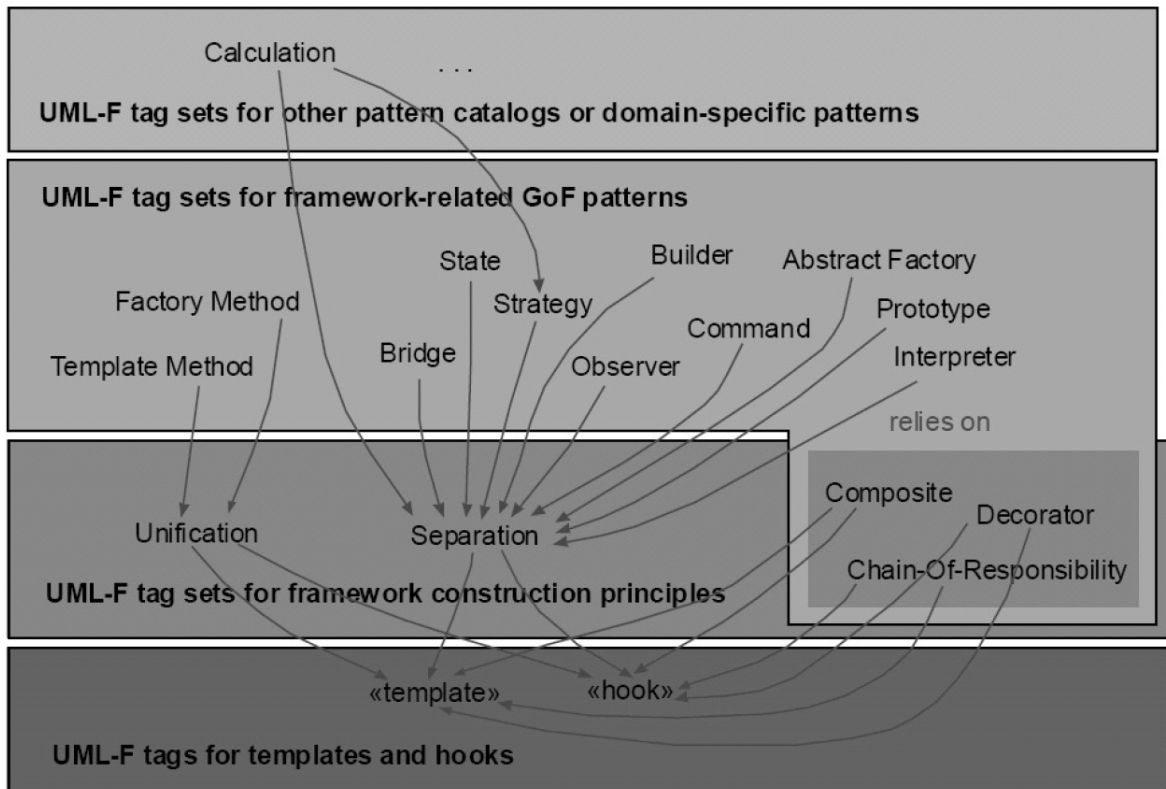


Figure 5.3: Layers of UML-F tag sets (from [PFR02])

These stereotypes, however, have no influence on the mapping process. They are used to support the adaptation of the framework model to the needs of a specific application. Other stereotypes of the UML-F Profile are not included in the model. The user could choose to generate only the application classes without using the framework classes. This would require that the mapping omits the classes with the stereotype `«framework»`.

#### 5.2.1.4 Deployment stereotypes

Two new stereotypes `«persistent»` and `«application-logic»`, introduced in the beginning of this chapter, tell the transformation algorithm to which kind of platform each element of the PIM should be mapped. Thus, the algorithm can decide if a class has to be mapped to a user-defined type and a typed table in the database or to a Java class, used for providing application logic.

#### 5.2.1.5 Domain Specific Stereotypes

UML extensions for a specific application domain can be defined in order to represent domain concepts which have a fixed structure and can be parametrized in the concrete model. In fact it can be argued that it makes more sense to define the GiACoMo-IRS model as UML extensions. However, the structure of GiACoMo-IRS concepts, such as a digital image can vary a lot depending on the application. Therefore, it is difficult to define a parameterizable

meta modeling concept for a digital image.

There are data structures in the GiACoMo-IRS model, which have a relatively clear and simple structure and can be included as domain-specific extensions in the meta model. These are basic data types, such as integer, float, boolean and complex data types, such as arrays. For the definition of the UML Superstructure only four basic types have been defined: Boolean, Integer, UnlimitedNatural, and String. New data types can be defined for a particular domain based on the `dataType` classifier. The `dataType` classifier allows defining data types, which are identified only by their value. For the image domain a set of domain specific data types is defined, which can be used to represent the value ranges of attributes used to describe the elements of a digital image object. The question arises if it is not better to model these data structures as classifiers. However, the data types discussed in this chapter are uniquely identified by their values and do not need to be represented as objects of a class, identified by an OID.

The following stereotypes are used to make these data types part of the UML meta model. `«CBIRSDataType»` is a stereotyped UML `dataType`. `«CBIRSSimpleType»` and `«CBIRSComplexType»` are stereotyped `«CBIRSDataType»` elements. As specializations of the `«CBIRSComplexType»`, `«CBIRSStructType»`, `«CBIRSArrayType»` and `«CBIRSEnumType»` are defined and a corresponding notation (syntax) is given in Appendix A.2. Some of the derived simple data types from `«CBIRSSimpleType»` are `«CBIRSIntegerType»`, `«CBIRSFloatType»`, `«CBIRSBinaryType»`. Derivates of the GiACoMo-IRS model can define additional data types, by deriving from `«CBIRSSimpleType»` or `«CBIRSComplexType»`. However, also new rules for the mapping of these data types have to be defined. A better way is to derive user-defined types from `«CBIRSStructType»` if possible, because there is a corresponding mapping rule predefined for these types.

The mapping rules have to be defined only for these UML elements and extensions used to model the CBIRS, which are part of the GiACoMo-IRS model instances.

### 5.2.2 PSM Meta Model

The chosen architecture for the generated application consists of two tiers (persistence and application logic), which can be implemented on different platforms. ORDBMSs as implementation platform offer the possibility to integrate both of these architecture tiers. This platform can accommodate the persistence tier directly. In order to support the application logic different than persistence management and exact-match queries, ORDBMSs rely on host programming languages, such as Java and C. Therefore, the application logic elements of the model should be mapped to the platform specific model for these languages. Mapping UML concepts to Java is a relatively straightforward process, supported by most UML modeling and code generation tools. This transformation is, therefore, not the focus of the current discussion. In an ORDBMS environment the functionality implemented in a host language is invoked from the database query language, and therefore, the interfaces of the application logic have to be registered as stored procedures or user-defined functions. The latter represent the bridge between the persistence tier and the application logic tier PSMs.

The platform specific meta model is in the current case predetermined by the SQL standard. In order to allow the adaptation of the implementation on a modeling level this meta model has to be represented in a form, which can be used by the developer for modeling. Therefore,

a UML-Profile for SQL is needed. As mentioned in Chapter 3 there is still no standard UML profile for SQL databases. Therefore, a compilation of the Rational's Data Modeling Profile and Scott Ambler's UML extensions is proposed in this thesis for the purely relational parts of the SQL:2003 standard. For the required object-relational SQL:2003 concepts UML extensions have been defined using partially the work of Marcos et al. [MVC04, VVCM07]. These UML extensions are based on a subset of the class diagram UML concepts, such as Class, Package, Association etc. The aim of the current thesis, however, is not to define an exhaustive UML-Profile for SQL, which can quickly become a very time-consuming task, but to represent the concepts needed to implement the GiACoMo-IRS instances in a UML-Profile for SQL. A full specification of the extensions is given in Appendix B.

The SQL:2003 standard remains mainly relational, although also object-oriented concepts have been integrated to support more complex database applications, such as multimedia databases. It is possible to combine both relational and object-oriented concepts in a database schema. Therefore, both groups of concepts are included in the UML extensions for SQL, proposed here.

#### 5.2.2.1 SQL:2003 Relational Concepts

The following SQL concepts fall into this category: Schema, Table, Column, View, Table Constraint, Relationship, Stored Procedure, Trigger. The relational concepts are almost exhaustively covered by Rational's and Scott Ambler's UML Profiles for SQL. In Appendix B these two Profiles are easily merged, because of their very slight differences. Deficiencies of both profiles, such as the exact notations for Column, Table Constraint, Stored Procedure and Trigger are compensated in the compilation represented in Appendix B.

#### 5.2.2.2 SQL:2003 Object-Relational Concepts

Although the latest SQL standard has developed in the direction of object-orientation, which should allow a direct representation of the SQL concepts in terms of the object-oriented modeling language UML, there are still a lot of discrepancies between the two models. Therefore, additional properties have to be added to the UML elements in order to represent the SQL object-relational concepts. For example, the SQL standard differentiates between different types of routines. A user-defined type has methods, which can be implemented in SQL or a host language. There are system routines, such as stored procedures and user-defined functions etc. The following object-relational concepts are included in the UML extensions of the PSM:

- **Structured Types, Typed Table:** The concepts of **Structured Types** and **Typed Tables** form the basis of the object-relational model. **Structured Types** have different usage in the model and thus can be regarded as different concepts depending on the role they play.

On the one side **Structured Types** can be used to define value types, which are identified only by their value. They are used as data types for tuple table attributes or for attributes of other **Structured Types**. Although using tuple tables in object-relational design is not appropriate it is often used for workarounds. Marcos et al. suggest to represent value-identified **Structured Types** as a stereotyped class «UDT».

It can only be used to define value types (not classes). Value types are identified by the values of their attributes. In this thesis, it is argued that for representing value identified types it is more appropriate to use the `dataType` UML meta class (see SQL:2003 Data Types below).

On the other side **Structured Types** can be used to define classes of objects identified by their object IDs (OIDs). In this role, **Structured Types** can be used to define **Typed Tables**. Marcos et. al suggest that both these concepts can be regarded as one and use the `«ObjectType»` stereotyped UML class to represent the concept in the profile. One problem of this generalization is that there also exist abstract **Structured Types**, referred to as not instantiable, which cannot be used to build a **Typed Table**. In order to distinguish between these two kinds of **Structured Types**, two specializations of the stereotyped class `«ObjectType»`, namely `«NonInstantiableObjectType»` and `«InstantiableObjectType»` are introduced. The last two are the stereotypes which can be used for representing an abstract **Structured Type** or a **Structured Type** in combination with a **Typed Table** as one concept, respectively.

If an attribute of a **Structured Type** is also a **Structured Type** (analogously to an “inline class”), which is not value-identified there are two possibilities for representation. One possibility is to represent the attribute as a reference to an object of the **Structured Type**, stored in another **Typed Table**. And the other possibility is to create the **Structured Type** as an inline object of the parent **Structured Type**. The second possibility can lead to redundancies, therefore, it is more appropriate to use references.

- **Structured Type Attributes:** OID-identified **Structured Types** correspond to the concept of **Classes** in UML. Therefore, the **Attributes** of a **Structured Type** can be represented as **Properties** of the corresponding stereotyped **Class** `«ObjectType»`. As data types, all SQL specific stereotyped `dataType` **Classifiers** can be used (see SQL:2003 Data Types below). A stereotyped **Property** `«Attribute»` can be used to represent the **Attributes**, whereby the explicit notation in the graphical representation of this stereotype can be omitted.

Marcos et. al suggest to represent the REF, ARRAY and ROW type constructors (complex data types) as stereotyped **Properties**. In this thesis, it is argued that these should be represented as data types, so that it can be possible to nest them in one another, for example, to define an ARRAY of REFS.

If the **Structured Type** is instantiable (used in combination with a **Typed Table**) additional options and integrity constraints for **Attributes** can be defined. These correspond to the **Column Constraints** defined in the relational concept **Column**.

Instantiable **Structured Types** can also assign constraints for the whole **Typed Table**. These correspond to the relational concept **Table Constraint**.

- **Structured Type Methods:** A **Method** of a **Structured Type** is an SQL-invoked routine. There are three kinds of **Methods**: SQL-invoked constructor **Methods**, instance SQL-invoked **Methods** and static SQL-invoked **Methods**. All SQL-invoked **Methods** are associated with a **Structured Type**, also known as the type of the **Method**. Therefore, they are represented as operations of an `«ObjectType»` class of a stereotype `«Method»` and have an attribute of type : enum {constructor, instance, static}.

- **User-Defined Functions:** A **User-Defined Function** is an SQL-invoked routine whose invocation returns a value. Every parameter of a **User-Defined Function** is an input SQL parameter, one of which may be designated as the result SQL parameter.

An SQL-invoked routine can be an SQL routine or an external routine. An SQL routine is an SQL-invoked routine whose language clause specifies SQL. An external routine is one whose language clause does not specify SQL. The routine body of an external routine is an external body reference whose external routine name identifies a program written in some standard programming language other than SQL.

Different SQL-invoked routines can have equivalent routine names. Two SQL-invoked functions in the same schema are not allowed to have the same signature. Two SQL-invoked procedures in the same schema are not allowed to have the same name and the same number of parameters.

None of the existing UML Profiles for SQL consider the modeling of **User-Defined Functions**. They can be represented similarly as **Stored Procedures** as a stereotyped class «User-Defined Functions» containing only operations, corresponding to the **User-Defined Functions**.

The type (SQL or external) of the routine is represented by an additional stereotyped attribute of the operation.

- **Structured Type, Typed View:** A **Typed View** is based as a **Typed Table** on an instantiable **Structured Type**. Therefore, it is represented as a stereotyped class «TypedView».
- **Inheritance:** **Structured Types** can be used to derive other more specific **Structured Types** in SQL:2003. Such derivation relationships can be represented as «Inheritance» associations.
- **References:** Relationships between different **Structured Types** are represented as **Attributes** of type reference («SQLRefType») in SQL:2003. In the UML-based SQL meta model a stereotyped **Dependency** «References» is used in addition to these reference **Attributes** for better readability of the model.

#### 5.2.2.3 SQL:2003 Data Types

In addition to the main modeling concepts of SQL, the SQL basic and complex data types are defined as specializations of the UML dataType. These are listed in Appendix B.3. User-defined types identified by their value can be derived from «SQLStructType».

### 5.3 Mapping PIM onto PSM

In section 2.2.3 different approaches for representing the mappings between the PIM and PSM are described. In this thesis, a textual description of the mappings is used in order to provide a more exhaustive description of the transformation possibilities. This textual description can be later formalized to be implemented as a transformation algorithm using the relational, graph-based or structure based approaches, for example.

The UML class diagram of a CBIR system reflects the persistence and application logic levels of the system. The persistence level classes are marked with the stereotype «persistent» and the application logic classes and interfaces with the stereotype «application-logic». Both persistence and application levels of the system have to be integrated into an ORDBMS environment. Therefore, persistent classes have to be mapped onto database schema concepts, whereas application logic classes onto host language (e.g., Java) concepts which encapsulate user-defined functions or stored procedures. These user-defined functions and stored procedures have to be at the same time registered in the database schema in order to be used in SQL statements. Therefore, the operations of the application-logic classes directly associated with «persistent» classes have to be mapped to the ORDBMS schema, as well. The mapping rules defined in this chapter are only for the persistence classes. The application logic classes shall be mapped to the corresponding programming language classes.

The classes having the stereotype «framework» in the CBIR PIM should not be mapped, since they are only abstract classes helping the derivation of application specific classes. Only «application» classes have to be implemented.

Database management systems based on the SQL:2003 standard support pure relational as well as object-relational concepts which can be used separately, but also mixed to provide a database implementation. In the mapping rules defined below the usage of object-relational concepts for the implementation of the object-oriented conceptual model is pursued in order to provide a more direct representation of the object-oriented concepts. There are, however, cases where purely relational concepts have to be used in order to provide a workaround for mapping object-oriented concepts, which are not directly representable in object-relational databases, e.g., visibility of attributes, and overlapping generalization set classes.

The starting point of the mappings is the PIM meta model. Therefore, the mapping rules are organized according to the extended UML meta model for CBIRs.

### 5.3.1 Class

Two types of classes in the CBIR model, which have to be mapped to the SQL model can be distinguished, persistent and application logic classes.

**Persistent classes** represent classes, the instances of which have to be stored persistently in the database. Such classes correspond to the **Structured Types** in SQL:2003. In order to be able to create objects of a class and make them persistent in the database we need to define an extension of the **Structured Type** in form of a **Typed Table**. Therefore, the default mapping of a UML class in SQL:2003 is an «InstantiableObjectType». Abstract classes cannot be used to create instances of these classes, so they cannot be associated to a table or be used to create a typed table. Their purpose is to enable the specialization of other classes through inheritance. These classes also have a counterpart in the SQL:2003 model. They can be represented as «NonInstantiableObjectType» classes in the SQL:2003 meta model. The problem of non-instantiable **Structured Types** in SQL, mentioned in [CT06], is that they cannot be used to construct **Typed Tables** or **Typed Views**. Therefore, it is also not possible to represent the concrete subclasses of an abstract class through the view of an abstract type. In [CT06] it is suggested that an abstract class is mapped onto an instantiable type with an associated type view. If a **Typed Table** is used to represent the abstract type a special integrity constraint must assure that the objects of the view are only objects of the subclasses. However, it should be considered that the purpose of abstract classes is

not to be used instead of their subclasses, but to facilitate the definition of special classes. Therefore, the mapping of the abstract classes onto a non-instantiable **Structured Type** «NonInstantiableObjectType» is more appropriate.

**Application logic classes**, which have a direct association with a «persistent» class have to be represented in terms of routine containers, i.e. as **User-defined Functions** or **Stored Procedures** in the SQL:2003 Profile. Thus, each class should be mapped onto a routine container class: «Stored Procedures» or «User-Defined Functions». The attributes of the class are left out, since they do not play any role for the registration of the routines in the database. Only the operations which correspond to the single routines are mapped. By default, the «application logic» class is mapped onto a «User-Defined Functions» class because it is closer to the semantics of an operation than stored procedures, so almost no restructuring of the operation is required. Each of the operations is represented by default as an «external routine». The developer can change the mapping to a «Stored Procedures» class and assign «SQL routine» for the type of the routines, instead of «external routine».

#### 5.3.1.1 Property (Attribute)

A **persistence class property** is mapped to an «Attribute» of an «InstantiableObjectType» or a «NonInstantiableObjectType» class. The modifiers of a property have also to be mapped to the modifiers of «Attribute». Not all of these can be mapped directly to corresponding concepts in the SQL:2003 model. Below some suggestions for workarounds are discussed.

- **Visibility (public, private, protected):** In the object-relational model all attributes of a **Structured Type** are public, therefore, an additional concept is needed in order to preserve this information coming from the PIM. The workaround suggested in [CT06] is to use different **Typed Views** for the **Typed Table** corresponding to the different visibility types of attributes. Corresponding **Structured Types** for each **Typed View** have to be defined. Integrity constraint have to be defined to assure that the objects in the views are only objects from the **Typed Table** and that the changes of the objects in the view are propagated to the **Typed Table**.
- **Derived:** The presentation of derived attributes in SQL:2003 is possible through **Methods**, which can be invoked in SQL for on-the-fly calculation of the derived attributes or through **Triggers** in case that the attributes need to be stored in the database and changed automatically upon changing their source attributes. Thus, there are two options for mapping derived attributes. The default option is mapping the attribute to a method of the class representing the **Structured Type**. If the developer decides to store the attributes, these have to be mapped to attributes of the **Structured Type**, in combination with a method to calculate the attribute and an update **Trigger** for the table containing the instances of the given type. If the derivation function is more complex and requires to be implemented in a host language additionally a **Stored Procedure** or a **User-Defined Function** may need to be defined which can be invoked from the **Trigger**.
- **Name:** The name of the property is simply mapped to a name of the «Attribute».



- **Property type:** In case that the property type is a «CBIRSDatatype» a property type is mapped to a corresponding «SQLDataType». If the property type corresponds to a class then it is mapped to a «SQLRefType» of an existing «InstantiableObjectType».
- **Multiplicity:** If the multiplicity value is bigger than 1 the set property modifiers (ordered, unique) have to be interpreted. Combining these characteristics different collection types can be defined: Set (isOrdered=false, isUnique=true), OrderedSet (isOrdered=true, isUnique=true), Bag (isOrdered=false, isUnique=false) and Sequence (isOrdered=true, isUnique=false). In SQL:2003 there are only two collection types to map the above: ARRAY and MULTISSET. Set and OrderedSet do not have a direct correspondent collection type in SQL since both collection types do not allow duplicate values. Therefore, the relational concept **Table** has to be used. Set and OrderedSet can be mapped to **Tables** with unique values of the **Columns** corresponding to the attributes. An OrderedSet would require an additional index attribute in the table, to store the order of the values. These tables have to contain an additional column with a reference to the **Structured Type** instances stored in the **Typed Table** representing the class. Bag can be mapped to a «SQLMultisetType» and Sequence to an «SQLArrayType».
- **Default:** An attribute of a **Structured Type** can have a default value, which is assigned to the attribute if no other value is assigned to it during the instantiation of the class. Therefore, this mapping is a direct mapping.
- **readOnly:** In order to represent readOnly attributes Read only views can be used, analogously to the mapping of the Visibility modifier.
- **union:** Attributes whose values are built from the union of other attributes can be represented as a derived attribute, i.e. through a **Method** or/and a **Trigger**.
- **subsets property\_name:** Attributes whose values are built as a subset of other attributes can be represented as a derived attribute, i.e. through a **Method** or/and a **Trigger**.
- **redefines property\_name:** Redefined attributes have to be defined as new attributes, the inherited attributes corresponding to these have to be updated, when these attributes are updated by a **Trigger**.
- **constraint:** Constraints are mapped to «ObjectType» Class Operations, used to represent **Table Constraints** or **Column Constraints**. If the constraints are none of the above, they can be mapped to a combination of a trigger and function.

In the case of database design it is important to be able to define attributes used for the identification of the instances of a class (Keys, OIDs etc.). In the standard UML conceptual model we have no direct means of achieving this. OIDs are implicit in the conceptual model. Therefore, the default mapping is also implicit, i.e the OIDs are system generated.

**application logic class properties:** The properties of «application logic» classes are mapped onto Java class attributes, retaining the visibility modifier and mapping to the host language (Java) data type system. Application logic classes, which are mapped to routine containers, e.g. **Stored Procedures** do not have any attributes.

### 5.3.1.2 Operation

**Persistence class operations** are mapped to stereotyped operations of «InstantiableObjectType» or «NonInstantiableObjectType», i.e. «Method». Class operations also have modifiers, similarly to properties, which have to be mapped correspondingly in the SQL:2003 model.

- **Visibility** (public, private, protected): In SQL:2003 all methods of a **Structured Type** are public. Therefore, a solution with the help of views, similar to the same modifier for properties has to be used.
- **Name**: The name of the operation is mapped directly to the name of a «Method».
- **Return type**: The return type of the operation is mapped to a return type of the «Method». It can be an existing «SQLDataType», or an «ObjectType». The return type can also have different multiplicity. In this case, the modifiers (ordered, unique) have to be evaluated and the returned type has to be mapped to the corresponding complex type or table.
- **Redefines oper\_name**: The operation name which is redefined by this operation can be given directly as a modifier of the «Method»
- **Query**: This operation modifier can be mapped directly to the ‘*reads sql data*’ modifier of a **Method**.
- **Operation constraint**: Operation constraints are pre- and postconditions for the operation. Postconditions can be mapped to CHECK Constraints of the «ObjectType», i.e. to «Check» operations of «ObjectType» - perhaps this can be used as a criteria to choose between the two mapping possibilities. Preconditions, however, do not have a direct counterpart in SQL:2003, and therefore, have to be implemented as part of the **Method**.
- **Parameter list**: These are mapped directly to parameters of the operations of the «Method» of an «ObjectType» class.

**Application logic class operations**, which are associated directly with «persistent» classes are mapped to operations of «Stored Procedures» or «User-Defined Operations» classes, respectively.

### 5.3.2 Associations

Different types of associations between «persistent» classes have to be considered in the mapping. In [DU04], [CT06] and [MVC04] the possibilities for mapping associations to object-relational models have been described extensively. These mappings are summarized in the following paragraphs.

The REF (Reference) data type in SQL:2003 is the main concept used for representing associations between objects in an ORDBMS. However, some additional concepts have to be used in order to represent different kinds of associations. A reference type can have properties specifying its scope (to which table it refers), and referential integrity constraints: references

are [not] checked, on delete (no action, set null, cascade). Currently we have defined the possibility to define only one table per structured-typed, so the scope value does not have to be set explicitly.

### 5.3.2.1 Unidirectional, Bidirectional Associations

In the CBIR meta model used in this thesis, associations between classes can be represented as uni- or bidirectional. By default, bidirectional associations are used.

- bidirectional associations: Reference types in the classes on both ends of the association are used to represent the bidirectional association in ORDBMSs. In order to preserve the consistency of the reverse relation, each time one of the relations is changed its reverse has to be updated. Therefore, **Triggers** and optionally **Stored Procedures** can be used. **Triggers** and **Stored Procedures** can be used as integrity preserving mechanisms as well. Referential Integrity has to assure that an instance which is referenced by another one is not deleted or that an instance cannot reference an instance which does not exist.
- unidirectional associations: In order to represent unidirectional relationships it is enough to include a reference in the class which has to be used as a starting point to traverse the relation. Referential integrity can be enforced with the same mechanisms.

### 5.3.2.2 Different Cardinalities of Associations

- $Max = 1$  - A single reference type is sufficient.
- $Max > 1$  - A collection type, such as an `«SQLArrayType»` of a `«SQLRefType»` can be used.
- $Min = 0$  - In this case additional mechanisms for preserving the cardinality constraint are not required.
- $Min > 0$  - A not null constraint for the reference type in the typed table has to be introduced if the reference is not a collection. Respectively, in the UML Profile an attribute `Nullable:boolean` has to be set to false for the reference type property. In case that the reference is a collection, a **Trigger** has to be used to assure that the elements of the collection of references is not NULL and a CHECK constraint, `«Check»`, can be used to prove that the collection is not empty (as suggested in [CT06]).

### 5.3.2.3 Recursive, N-ary Associations

Recursive associations are mapped as the above, only in this case the references are represented in the same class which is referenced.

An N-ary association is represented similarly as an association class as suggested by [DU04]. An association table is used, which includes the references to all the structured types participating in the N-ary relationship. And in each of the participating structured types a reference, or a collection of references to the association table have to be defined.

#### 5.3.2.4 Aggregation, Composition

In [MVC04] these association kinds are represented by an array of references in the **Structured Type**, representing the whole, which contains different parts. To assure that the components of the whole are also deleted upon deletion of the whole, in the case of a composition **Triggers** can be used.

#### 5.3.2.5 Association Class

An Association Class can be seen as an association that also has class properties, or as a class that also has association properties. It not only connects a set of classes but also defines a set of features that belong to the relationship itself and not to any of the classes. The specialization and refinement rules defined for class and association are also applicable to an association class. It should be noted that in an instance of an association class, there is only one instance of the associated classifiers at each end, i.e. from the instance point of view, the multiplicity of the associations ends are 1.

There are different possibilities to represent association classes.

- ROW type: The reference and the association class attributes are represented as a `«SQLRowType»` attribute of the `«ObjectType»` class.
- Structured type: The references and the association class attributes and methods are represented as a **Structured Type** and a **Typed Table**, which serves as a reference table.

Attributes of an association are represented through an association typed table according to [DU04]. The **Structured Type** of the table consists of two attributes for each referenced structured type and the attributes of the association. In this way, also operations of the association class can be represented.

#### 5.3.2.6 Association Qualifier

An attribute related to an association represents an end of the association. The type of attribute is the type of the end of the association. When an attribute is an association end, its value or values are related to the instance or instances at the other end(s) of the association. The association type is represented as a typed table. This **Typed Table** includes then the association attributes and two reference types corresponding to references to the typed tables objects participating in the association.

Qualified associations represent a concept similar to weak entities in the Entity-Relationship model. In [DU04] it is suggested to represent weak entities by including the identifying attributes from the parent class in the attributes of the child class as primary keys.

### 5.3.3 Dependency

In the object-relational model there is no counterpart concept for this kind of relationship. And since it does not have any runtime impact and does not concern the instances of the model elements it does not have to be mapped to the implementation-specific model. The

same holds for the two subtypes of Dependency, **Usage** and **Realization**. These are concepts which have impact only on the model, but not on the application. They can be used for better understanding of the implementation, but do not influence its execution.

### 5.3.4 Generalization, GeneralizationSet

SQL:2003 support inheritance of user-defined types and of typed tables. So these kind of associations can be mapped directly as inheritance associations between «ObjectType» classes. Whereas an inheritance association between «InstantiableObjectType» implies an inheritance association between the typed tables if the user-defined types are instantiable.

The different constraints of a GeneralizationSet can be represented as follows:

- disjoint: The standard inheritance in SQL results in disjoint sets.
- overlapping: The object-relational model does not provide a direct representation for overlapping specialization of classes. A specialization in an object-relational schema is always disjoint. In order to represent these classes, a general solution to the problem is suggested in [CT06]. The intensions of the classes of the class hierarchy can be directly mapped to **ObjectTypes**. The extensional specialization has to be resolved by using a help **Table**, in which the OIDs of objects from the overlapping classes, representing the same object in the real world are stored correspondingly.
- complete: This condition has to be realized with workarounds for a hierarchy of typed tables. For example, as suggested in [CT06] the insertion of objects in the superclass can be forbidden by a **View** or access rights. Another possibility is to use a **Trigger** to assure the consistency between the super and subclasses.
- incomplete: In this case no additional workarounds are needed.

The concepts difficult to map are shown in Figure 5.4 in *italic*. The example shows an inheritance hierarchy for the eNoteHistory application. Two types of images are used in the application, the *IncipitImage* which represents the opening notes of a music piece in a standard contemporary notation, and the *ManuscriptImage* which represents a scanned page of a music manuscript. Each of these different image types has different meta data assigned to it. In order to map the overlapping and complete generalization set the workarounds mentioned above have to be applied.

Multiple inheritance is also a concept which is not supported in ORDBMS. In [CT06], the authors propose an approach to transform multiple inheritance, which can be used in UML models, into simple inheritance. The approach is based on redefining one of the superclasses of the multiple inheritance as the complement of the subclass with respect to the superclass. The subclass thus becomes independent of the redefined superclass. If the redefined superclass has other subclasses, these have to be redefined also. The complements of these subclasses have to be respectively defined as subclasses of the detached subclass of the multiple inheritance. The simple inheritance hierarchy can be mapped to **ObjectTypes**. The original classes can be represented by **Typed Views**.

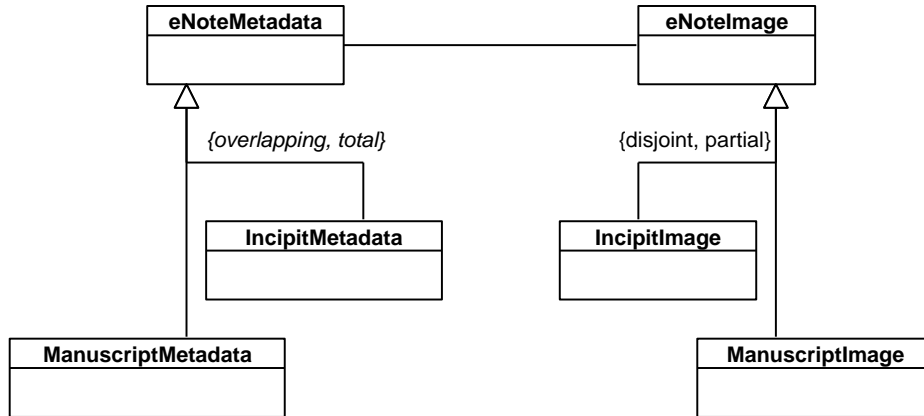


Figure 5.4: Mapping problems with inheritance hierarchies

### 5.3.5 Interface

An interface is a kind of class that represents a declaration of a set of coherent public features and obligations. The obligations that may be associated with an interface are in form of various kinds of constraints (such as pre- and post-conditions) or protocol specifications, which may impose ordering restrictions on interactions through the interface. Since interfaces are declarations, they are not instantiable. Instead, an interface specification is implemented by an instance of an instantiable class. Note that a given class may implement more than one interface and that an interface may be implemented by a number of different classes. Interfaces can be used to specify behavior global to the system. They need to be implemented by classes in order to become instantiable. An important feature is that a class may implement more than one interface, so multiple inheritance is allowed.

Interfaces are not part of the current model. If they are used in any instances of GiACoMo-IRS they can be mapped to «NonInstantiableObjectType» classes.

#### 5.3.5.1 InterfaceRealization

An InterfaceRealization is a specialized Realization relationship between a Class and an Interface. This relationship signifies that the realizing class conforms to the contract specified by the Interface.

If used in an instance of the GiACoMo-IRS model it can be mapped as an inheritance association in the SQL:2003 model.

### 5.3.6 Package

A package groups other elements logically, therefore, it corresponds to the SQL notion of a database schema. It is, thereafter, mapped to a stereotyped package «Schema».

However, Relationships between schemata cannot be represented directly in SQL. Schemas in SQL can always access other Schemas when the corresponding right are set. Therefore, the **Package Merge** and **Package Import** relationship types can be mapped to access privileges for schemas.

### 5.3.7 DataType

There are two types of CBIRS data types which have to be considered in the mapping. The «CBIRSSimpleType» can be mapped almost directly to corresponding «SQLBasicDataType» derivatives as shown in Table 5.1. The derivatives of «CBIRSComplexType» which at the moment are only two can be also mapped relatively straightforward. The «CBIRSArrayType» can be mapped to an «SQLArrayType», and any derivative of «CBIRSStructType» defined in an instance of the GiACoMo-IRS model can be mapped to a derivative of a «SQLStructType».

CBIRS Simple Data Type	SQL:2003 Data Type
«CBIRSIntegerType»	«SQLSmallIntType» or «SQLBigIntType» or «SQLIntegerType»
«CBIRSBooleanType»	«SQLBooleanType»
«CBIRSCharacterType»	«SQLCharType» or «SQLVarCharType»
«CBIRSStringType»	«SQLVarCharType» or «SQLClobType»
«CBIRSFloatType»	«SQLFloatType» or «SQLNumericType» or «SQLDecimalType»
«CBIRSBinaryType»	«SQLBitType» or «SQLVarBitType» or «SQLBlobType»
«CBIRSEnumType»	«SQLDomainType»

Table 5.1: Mapping of «CBIRSSimpleData» derivatives to SQL:2003

### 5.3.8 Applying the Mapping Rules

The mapping rules have to be applied in a specific order in an algorithm to assure the existence of elements needed for the creation of others. The sequence of applying these rules is as follows.

1. At first all Data Types used in the conceptual model should be mapped to their counterparts in the SQL:2003 model. Thereby, derivatives of «CBIRSStructType» would require the creation of new «SQLStructType» subclasses.
2. Subsequently, all «persistent» classes, which are not subclasses of other classes should be mapped to «InstantiableObjectType» and «NonInstantiableObjectType», respectively. The properties and operations of these classes should be mapped simultaneously.
3. The different kinds of associations between these first level classes should be mapped to reference attributes of the existing «InstantiableObjectType» and «NonInstantiableObjectType» classes.
4. For each class which has subclasses, the whole subclass hierarchy should be mapped one after the other.

5. Next, the new associations created by the mapping of the subclasses should be mapped to additional attributes of the «InstantiableObjectType» and «NonInstantiableObjectType» classes.
6. The «application-logic» classes directly connected to «persistent» classes should be mapped to «User-Defined Functions» classes and their operations to «User-Defined Function» operations of these classes.
7. Finally, all «application-logic» classes can be transformed in the host language meta model.

This sequence for applying the transformation rules reflects the specifics of the SQL data definition language.

## 5.4 Quality of the Transformation

The requirements towards the transformation rules are defined in section 3.2.3.2. It is induced that in order to achieve a good quality of the transformation for each concept of the conceptual model a suitable mapping onto the platform specific model should be found. Mappings, which lead to the presentation of different conceptual model elements as one and the same target model element should be avoided. In order to evaluate the quality of a transformation, based on the mapping rules defined in the previous section the rules are classified in the following groups, which are used to qualify the result of the transformation.

### 5.4.1 Direct mappings

These mappings, illustrated with the dotted lines in Figure 5.5, represent trivial mapping rules, which have been also discussed by existing mapping approaches [CT06, DU04].

A «persistent» UML **Class** has to be transformed into a **Structured Type** and a corresponding **Typped Table** in the terms of the SQL:2003 standard. Although the result of the mapping comprises of two object-relational concepts, they can be regarded as one complex concept, therefore, the mapping is considered as a direct mapping.

The mapping of **Classes** does not conclude with the transformation of the **Class** concept. A **Class** is represented through its **Properties** and **Operations**. These can be mapped directly to **Attributes** and **Methods** of the **Structured Type**, respectively. These features in both PIM and PSM models have additional characteristics which have to be specified. These details can also be partially directly mapped, such as mapping a **Property** of a «CBIRS-BooleanType» to an **Attribute** of a «SQLBooleanType». Some of these characteristics in the PIM, however, do not have direct counterparts in the PSM, and therefore, workarounds have to be used in order to provide a suitable mapping.

### 5.4.2 Not-directly-mappable concepts

As mentioned above not all of the concepts defined in the conceptual model can be mapped directly onto the object-relational model (e.g. property modifiers, such as private and public). These elements cannot be omitted during the mapping if it has to be made sure that each



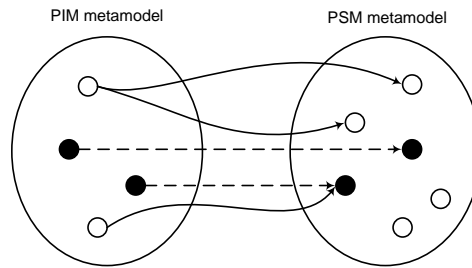


Figure 5.5: Metamodel mappings: direct mappings

PIM concept can be represented in the PSM. This is one of the requirements for preserving the information capacity of the model discussed in section 3.2.3.2. A workaround based on using **Views** has been suggested in the mapping rules to make the mappings of private **Properties** of a **Class** possible. Applying workarounds, illustrated with the wavy lines in Figure 5.6, leads inevitably to two other classes of transformation rules listed below, multiple transformation possibilities and transformation rules mapping different PIM concepts to the same PSM concept.

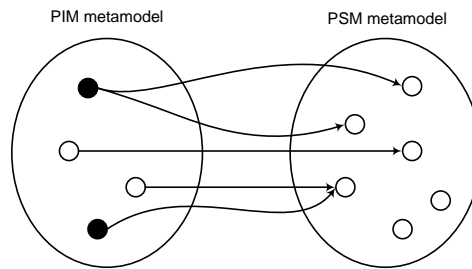


Figure 5.6: Metamodel mappings: not-directly-mappable PIM metamodel concepts

### 5.4.3 Multiple mapping possibilities

Sometimes there are multiple ways to represent conceptual elements in the logical model as illustrated in Figure 5.7. For example, an **Association Class** can be represented as an attribute of `«SQLRowType»` or as an `«InstantiableObjectType»`. This requires the developer's decision and/or the usage of default values during the automatic mapping process. A decision can also be made by the mapping algorithm if optimal mapping rules, based on heuristics or costs, can be identified. Multiple mapping possibilities would not lead to loss of information during the transformation. During one transformation process only one of these possibilities is used. The challenging question is which criteria can be used to decide which one of the transformations will lead to a more efficient application.

### 5.4.4 Mappings resulting in the same PSM Concept

This type of transformation rules, illustrated with the dotted lines in Figure 5.8, results mostly from the usage of workarounds for mapping PIM concepts which do not have direct counterparts in the PSM. In workarounds, PSM concepts which are already used to map a certain

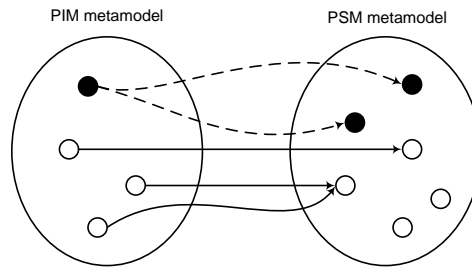


Figure 5.7: Metamodel mappings: multiple mapping possibilities

PIM concept in a direct mapping are used again to represent other PIM concepts, which are not directly mappable. One example of such a mapping is the mapping of **Abstract Classes** and **Interfaces**. Both of these concepts are mapped to a `«NonInstantiableObjectType»`. Another example is the creation of `«Methods»` for representing **Class Operations** and for **Derived Attributes**.

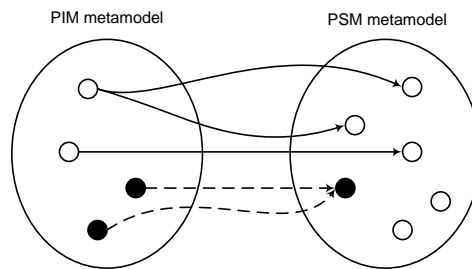


Figure 5.8: Metamodel mappings: mappings resulting in the same PSM metamodel concept

The problem which arises from such kind of mappings is that it is most likely that the differentiation between two different PIM concepts will be lost if they are represented by the same PSM concept. This makes the reverse design step, i.e. the derivation of the PIM from the PSM model, difficult. Reverse engineering is, however, not required by the MDSM paradigm. The main principle in this development approach is that the changes in the implementation should be made in the model and propagated through model-to-model and model-to-code transformations to the implementation. A more important undesirable consequence is that there is a possibility that some of the semantics of the mapped concepts might be lost. This will be the case if different PIM concepts cannot be used interchangeably, i.e. do not have an equivalent semantics, in the conceptual model but at the same time are represented as the same concept in the PSM. In order to prove if the suggested mapping rules which transform different PIM concepts to the same PSM concepts lead to such problems, the semantic equivalence of the transformed PIM concepts is analyzed below. The PIM concepts are grouped according to the PSM concepts to which they are mapped.

#### 5.4.4.1 Instantiable Structured Type + Typed Table

**Structured Types** are the direct counterpart of **Classes** in SQL. **Association Classes** and **Association Qualifiers** can be represented through the same PSM concept as well. **Association Classes** are used to assign properties and operations to an **Association**, which

cannot be assigned to any of the **Classes** participating in the **Association**. In [GR98] the authors also give a semantically equivalent presentation of an **Association Class** using a **Class** and a ternary **Association**, as shown in Figure 5.9 with the following constraint:

```
C->forall(c |
  c.ra->size=1 and c.rb->size=1 and
  C->forall(c | (c.ra=c.ra and c.rb=c.rb) implies c=c))
```

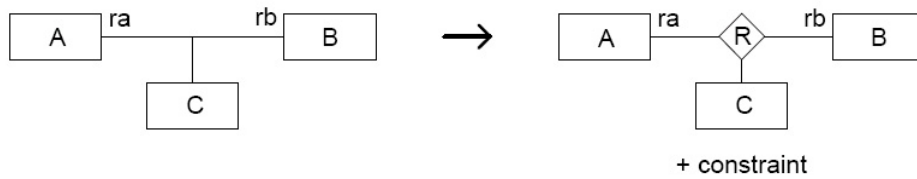


Figure 5.9: Equivalence rule for association class (from [GR98])

This constraint can also be represented as cardinality of 1 on the side of *C*. The cardinality of 1 means that each object of *C* is assigned to a unique pair of *A* and *B* objects. An equivalent representation of an **Association Qualifier** is created using an **Association Class** as shown in Figure 5.10.

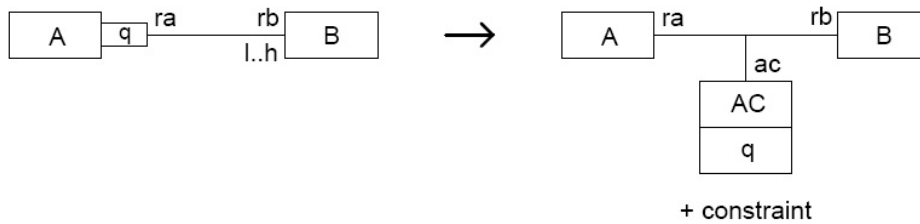


Figure 5.10: Equivalence rule for association qualifier (from [GR98])

The corresponding constraint is:

```
A->forall(a |
  a.ac->forALL(ac |
    a.rb->select(b |
      b.ac->exists(ac | ac.q=ac.q and ac.ra=a))->size>=1
    and
    a.rb->select(b |
      b.ac->exists(ac | ac.q=ac.q and ac.ra=a))->size<=h))
```

This constraint can also be replaced by cardinalities at the side of the dependent Class, in this case B.

Thus, **Classes**, **Association Qualifiers** and **Association Classes** can be used interchangeably in the conceptual model, and therefore, their mapping onto the same PSM concept will not lead to loss of semantical information as long as the respective constraints are also implemented.

#### 5.4.4.2 Non-instantiable Structured Type

This PSM concept is used to represent an **Abstract Class** and an **Interface** from the PIM model. An **Interface**, however, can also be represented by an **Abstract Class** containing only abstract operations in the conceptual model.

#### 5.4.4.3 Typed View

**Typed Views** are used to encapsulate private, protected **Attributes** and **Methods** of **Classes** and to represent **readOnly Attributes** of **Classes**. Additionally, **Typed Views** are used to represent overlapping and complete **GeneralizationSets**. In the last case the **Typed Views** are defined over more than one **Structured Type**, whereas in the first case these are defined over only one **Structured Type**. Therefore, although both PIM concepts have completely different semantics, it is possible to differentiate between the mappings of these two different concepts in the implementation model at an instance level, in an instance of the PSM. The properties of the **Typed View** instances, such as the number of **Structured Types** used for constructing the **Typed View** allow to unambiguously distinguish the **Typed Views** representing public and **readOnly Attributes** and **Methods** from the **Typed Views** representing a **GeneralizationSet**.

#### 5.4.4.4 Method

The counterpart of **Methods** in the PIM are **Operations** of **Classes**. However, they are one of the ways to represent also **Derived Attributes**, **Union Attributes** and **Subset Attributes**. In UML, **Operations** can also be used to model such kind of calculated **Attributes**.

#### 5.4.4.5 Trigger

**Triggers** in combination with **Attributes** and optionally with a **User-Defined Function** or **Stored Procedure** are an alternative way to represent **Derived Attributes**, **Union Attributes** and **Subset Attributes**. This combination is also used to represent **Redefined Attributes**. Additionally, **Triggers** have to be defined for representing **Property Constraints**, which are not Check, Primary Key, Foreign Key or Unique constraints. Finally, **Triggers** are also used to express consistency and referential integrity rules when mapping different kinds of **Associations**. Thus, there are four groups of PIM concepts which are semantically completely different, and therefore, a way to distinguish them in the PSM should be sought. Here, the properties and structure of the **Triggers** are used to differentiate between them. Examples for concrete **Triggers**, i.e. patterns of **Triggers**, for each of the PIM concepts are given below.

#### Derived Attributes, Union Attributes and Subset Attributes:

These **Triggers** are invoked when an Update operation on an attribute of a table takes place and their action changes the value of another attribute of this table using the updated attribute in the calculation.

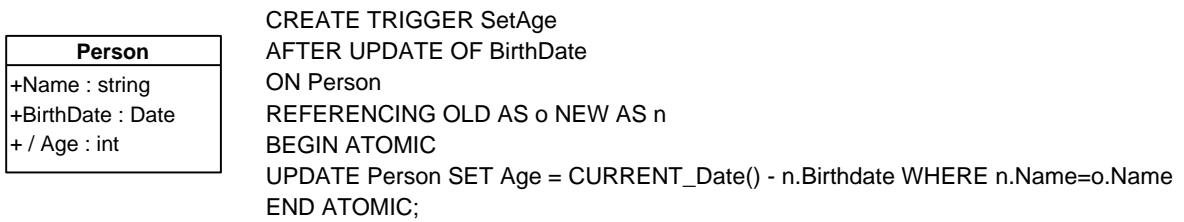


Figure 5.11: Example of a Trigger for a Derived Attribute

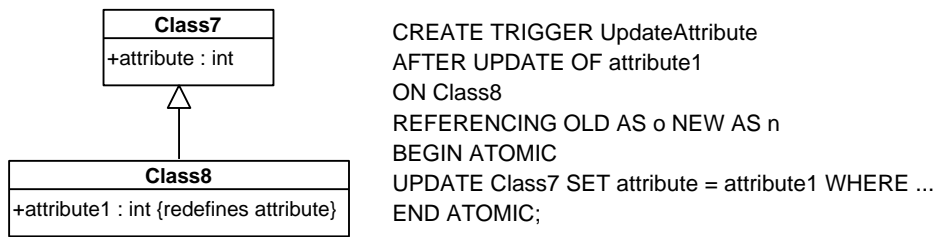


Figure 5.12: Example of a Trigger for Redefined Attributes.

**Redefined Attributes:**

This **Trigger** also shows a regular pattern. It is always issued when updates to an attribute of a super- or sub table take place and changes an attribute in the corresponding sub- or super table respectively with the same value as the updated value.

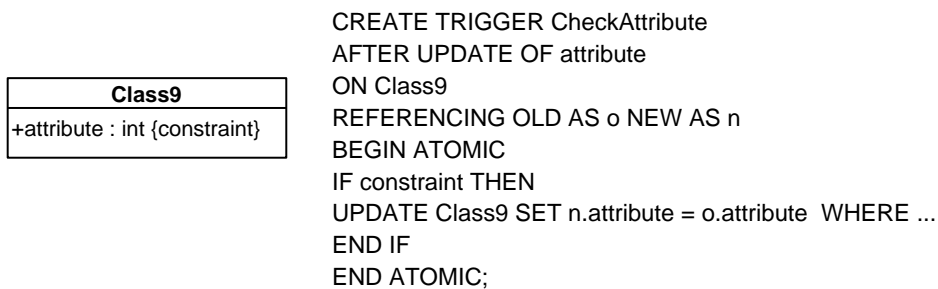


Figure 5.13: Example of a Trigger for a Property Constraint

**Property Constraints:**

These **Triggers** are issued upon update of an attribute and after evaluating a predicate they either revoke the update or do nothing.

**Associations:**

These two examples of **Triggers** show that in the case of association mapping, **Triggers** are used on reference types.

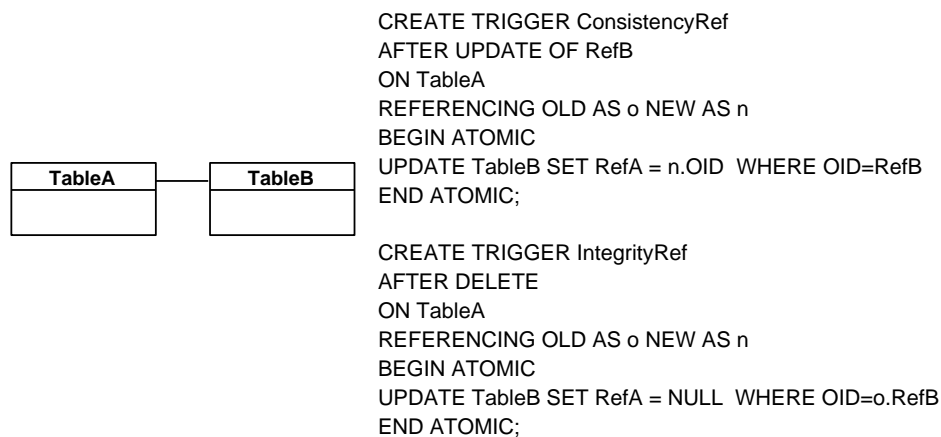


Figure 5.14: Examples of Triggers for consistency and integrity constraints in mapping of Associations

In order to identify the corresponding PIM concepts in the PSM based on their structure and properties, these **Trigger** patterns can be used.

#### 5.4.4.6 Table

The relational concept **Table** is used for representing a help table for overlapping GeneralizationSet **Classes** and for representing the Set and the OrderedSet collection types. In the first case the **Table** always contains exactly two **Attributes** corresponding to the two OIDs of the matching objects. In the second case an **Attribute** and a reference to the object containing this **Attribute** are the **Columns** of the **Table**. In the case of an OrderedSet additionally an index **Attribute** is defined. Therefore, analogously to **Typed Views**, the two PIM concepts are distinguishable at an instance level of the PSM.

#### 5.4.5 Implementation specific concepts

Some concepts from the logical model cannot be represented in the conceptual model, as illustrated with the filled elements of the PSM metamodel in Figure 5.15. This requires further adaptation of the logical model by the developer. Modifiers of methods and user-defined function, such as LANGUAGE, PARAMETER STYLE have to be added if not set with default values. This additional information would not lead to loss of conceptual model information. It does not also influence the reverse engineering step, because the additional concept can be simply omitted in the transformation.

To conclude, it can be summarized that not-mappable concepts do not exist in the PIM which guarantees the preservation of the information capacity in the PSM to some extent, i.e. the information representable in the PIM should be representable in the PSM. Through the workarounds used in order to provide mapping rules for all PIM concepts other problems have raised. Especially, mapping different PIM concepts to one and the same PSM concepts leads to the fact that the reverse engineering step will not be possible. As shown above, the differentiating semantics of the mapped concepts, however, remains preserved at the instance

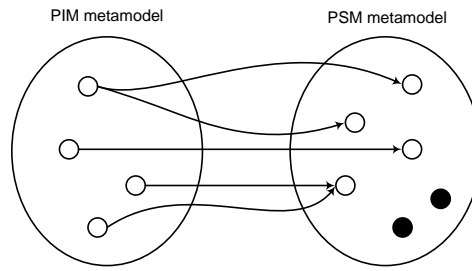


Figure 5.15: Metamodel mappings: implementation specific concepts

level of PSM. Hence, the mapping rules defined in the previous section should be able to produce an instance of a PSM, which can represent all the information that an instance of the PIM can, i.e. the transformation is a total and injective function.

## 5.5 Implementation of an Image Database Generator

The model-driven development techniques represented in this and the previous chapter are implemented as a prototype of an integrated modeling and code generation tool for CBIR-Systems. The task of the so called Image Database Generator (IDBG) is to support the modeling and generation of a three-tier CBIR application. As illustrated in Figure 5.16, the generated CBIR application consists of a persistence management layer, an application layer, and a user interface layer. The persistence and application layer are implemented on an ORDBMS platform. The user interface layer is implemented as a web application. The prototypical implementation of IDBG is the result of different student projects [Sch07, KSW07]. The tool is built as a plug-in for the Eclipse integrated development environment. It makes use of the Eclipse Modeling Framework libraries. It consists of two main plug-ins, a Modeling plug-in and a Generator plug-in.

The Modeling plug-in is responsible for the creation of an Image Database Generator (IDBG) project. It supports the derivation of a conceptual model of the CBIRS from the integrated framework model and transforms the conceptual model into a PSM, based on a SQL:2003 UML Profile. The plug-in provides a graphical modeling user interface for both PIM and PSM. The implemented transformation algorithm currently supports only basic mapping rules. The plug-in provides a dialog for the developer to refine the mapping in cases where multiple mappings are possible, or where more platform specific information has to be provided. It can be further developed to support all mapping rules defined in this chapter. The employment of a transformation engine, such as the ATLAS Transformation Language [ATL07], should be considered for the implementation of transformation rules.

The Generator plug-in is responsible for converting the PSM into DDL scripts and executable source code, for each target platform, respectively. Different generators can be created for different ORDBMSs. At this point, it should be mentioned that for the final implementation of the SQL:2003 concepts a direct conversion to a concrete ORDBMS SQL dialect cannot always take place. Existing ORDBMSs do not fully support the standard. Therefore, for each generator a mapping from SQL:2003 to a concrete database model, for example for IBM DB2, has to be implemented. In the IDBG, IBM DB2 is used as a target ORDBMS. The main differences between the SQL:2003 standard and the IBM DB2 ORDBMS are the

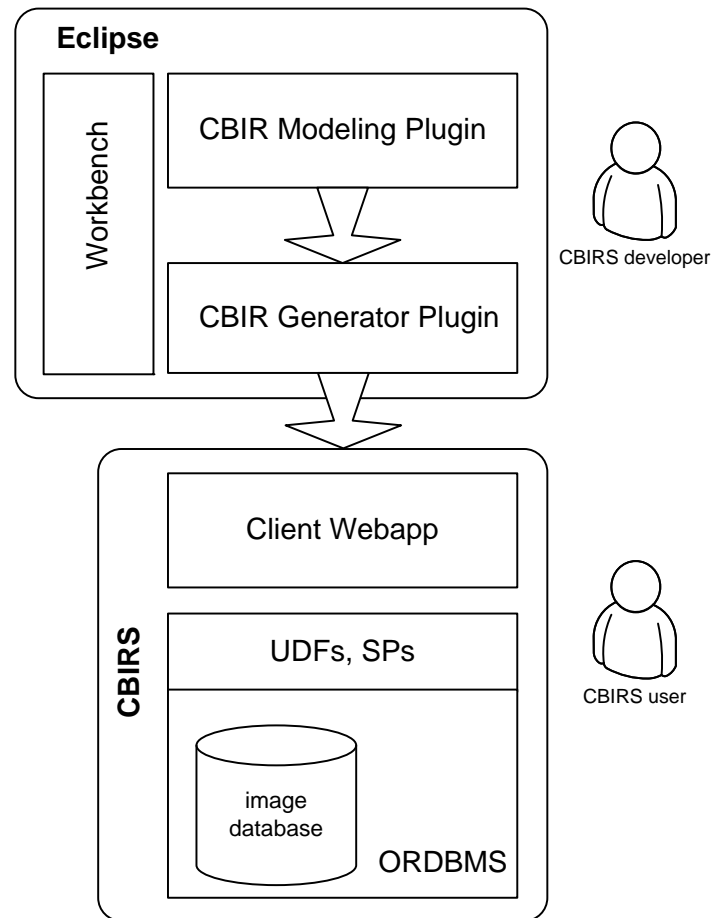


Figure 5.16: Architectures of the Image Database Generator and the resulting CBIRS

missing concepts for ARRAYS, MULTiset and ROW in DB2. For each of these discrepancies a workaround has to be found for the Generator plug-in.

The user interface of the system is not part of the conceptual model and thus it cannot be generated from the model. In Chapter 3 it is mentioned that there are special modeling techniques, such as task models [WFDR05], for designing graphical user interfaces, which can be considered for this task. In order to facilitate the creation of a user interface the IDBG generates the basic structure of a JSP application. This structure is determined by predefined building blocks, such as feature class or a search query class. These building blocks use information from the conceptual model in order to create specific classes during the generation process. For example, for a specific “average color” feature of the conceptual model a corresponding feature class in the web application will be generated. Further adaptations of the classes have to be made by the developer by hand. Thanks to the fact that the IDBG is implemented as a plug-in in the Eclipse IDE, a seamless transition from a design to implementation perspective is possible.

Screenshots of the Image database Generator plug-in can be found in Appendix D.



## 5.6 Summary

This chapter defines the transformation mechanisms which have to be applied on the PIM in order to derive a PSM. In Chapter 3 it is stated that, therefore, a model-to-model mapping has to be defined. Model-to-model mappings are defined for the meta-model elements of both models and applied on the instance of the meta-models. Therefore, at first a detailed summary of both meta-models was provided. Following, the mapping rules for each of the PIM meta-model concepts, more specifically for each of the UML structure diagram concepts, one or more translation possibilities to the PSM meta-model concepts were defined. In some cases, where direct counterparts in the PSM were missing, such as in the case of the visibility of class properties, workarounds had to be introduced. It was important to give a suitable mapping for each PIM concept in order to avoid not-mappable concepts. Not-mappable concepts can lead to loss of information in the PSM. The workaround brings, however, other mapping problems, as shown in the analysis of the quality of the transformation. Especially mappings which lead to the representation of different PIM concepts as the same PSM concepts could lead to lost of distinguishable information. However, the detailed analysis of the concrete cases showed that either the PIM concepts can be used interchangeably in the conceptual model, which makes them semantically equivalent, or the differentiation of the PSM concepts can be made on an instance level. Thus, these mappings still preserve the information capacity of the transformed model.

Finally, a prototypical implementation of the modeling and transformation mechanisms as an Eclipse plug-in was described.

## Chapter 6

# Evaluation of the Model-Driven Development Methodology for CBIRSs

There are different aspects which can be considered in order to evaluate a model-driven development approach. An important criterion for evaluating the quality of MDS is the quality of the models used to derive the applications. The factors influencing the quality of the models in model-driven software development have been studied in different works summarized in [MA07]. The main factors are the quality of the modeling language and the modeling process, the combination of which is referred to as the modeling approach in this chapter. The quality of the modeling approach has to be evaluated in order to prove if it is sufficient to represent all the information for the modeled domain, if it is compact enough in order to avoid ambiguity and complexity, if it is easy to understand and apply etc. According to [MA07] apart from the quality of the modeling approach other factors, such as modeling tools, required knowledge and expertise of the modeler, and the usage of quality assurance techniques influence the quality of the resulting models. At this stage of the current work it is possible to consider only the quality of the modeling approach for the evaluation. For the evaluation of the modeling approach the set of criteria introduced in Chapter 3 are used.

- **Syntactic correctness:** Since the meta model used to create the framework model as well as the derived application specific model is UML, all resulting models should comply with the UML 2.0 standard. The framework model is designed so as to comply with the standard and the derived models can achieve this if adequate modeling tools are used, which can perform syntactical checks during the modeling process.
- **Validity:** Validity ensures that only concepts relevant to the problem are included in the model. The framework model is designed to include only relevant concepts. Its derivations, however, may be freely extended as far as the meta model allows. Thus, the transformation rules cannot be influenced.
- **Completeness:** Completeness requires that the model contains all the statements about the domain. The framework model is a generic model and thus cannot contain all the variations of concepts needed for different applications. However, it should be possible to derive as much as possible application specific concepts from the generic

concepts. This criterion is measured below by deriving three different CBIR applications from the generic model.

- **Understandability:** The evaluation of this criterion requires the involvement of experienced developers in order to measure the time needed for applying the modeling approach objectively. This can be done after the approach is implemented completely in a tool. It is important to mention that the provided cookbooks in Chapter 4 and the examples in this chapter provide a helpful introduction in the modeling approach.
- **Implementability:** The transformation rules showed that all the concept of the PIM meta model can be transformed into the meta model of the implementation. And thus this criterion is considered as fulfilled.

Another aspect of the evaluation is to prove whether the result of the transformation approach does not deteriorate the conceptual design. The quality of the transformation approach is evaluated in the previous chapter. Therefore, in this chapter the focus of the evaluation lies on the first aspect. Some further discussion of metrics for the quality of the transformation is given in [MD07].

Finally, the gain from the usage of this development approach has to be estimated, based on the amount of implementation, which still has to be done manually in order to produce a full fledged CBIRS. Therefore, three test case applications are used in this chapter, the eNoteHistory application, a 2D-Gel application, and a Photo Annotation application. The first two of them have been already implemented as a CBIRS on top of an ORDBMS. The Photo Annotation application has been implemented based on the MetaXa framework, introduced in Chapter 1. This fact provides the possibility to verify which part of the application can or cannot be generated by the model-driven approach, or has to be implemented by hand. All criteria which require a cost or an effort to be estimated do not consider time, since this metric requires the implementation of the approach in a model-driven development tool and the participation of developers with good programming skills. Since the existing prototype does not implement fully the development approach it cannot be yet used for the evaluation. In this chapter, on first place the completeness of the framework model, i.e. how far do the derived models for the example applications fit in the generic model, is estimated. Second, the amount of implementation, which still has to be done by the developer after the transformation of the model is estimated.

## 6.1 Test Case eNoteHistory

The eNoteHistory scribe recognition system is introduced as a case study in the beginning of this thesis. In fact, the eNoteHistory project was the inspiration for the model-driven development approach for CBIRSs. The functionality and structural requirements for CBIRSs in Chapter 3 are explained with the help of examples from the scribe recognition application. In this section, the eNoteHistory projects and in particular the image retrieval functionality implemented for the recognition of music manuscript scribes are described in more detail in order to show how the modeling framework can be applied for this application.

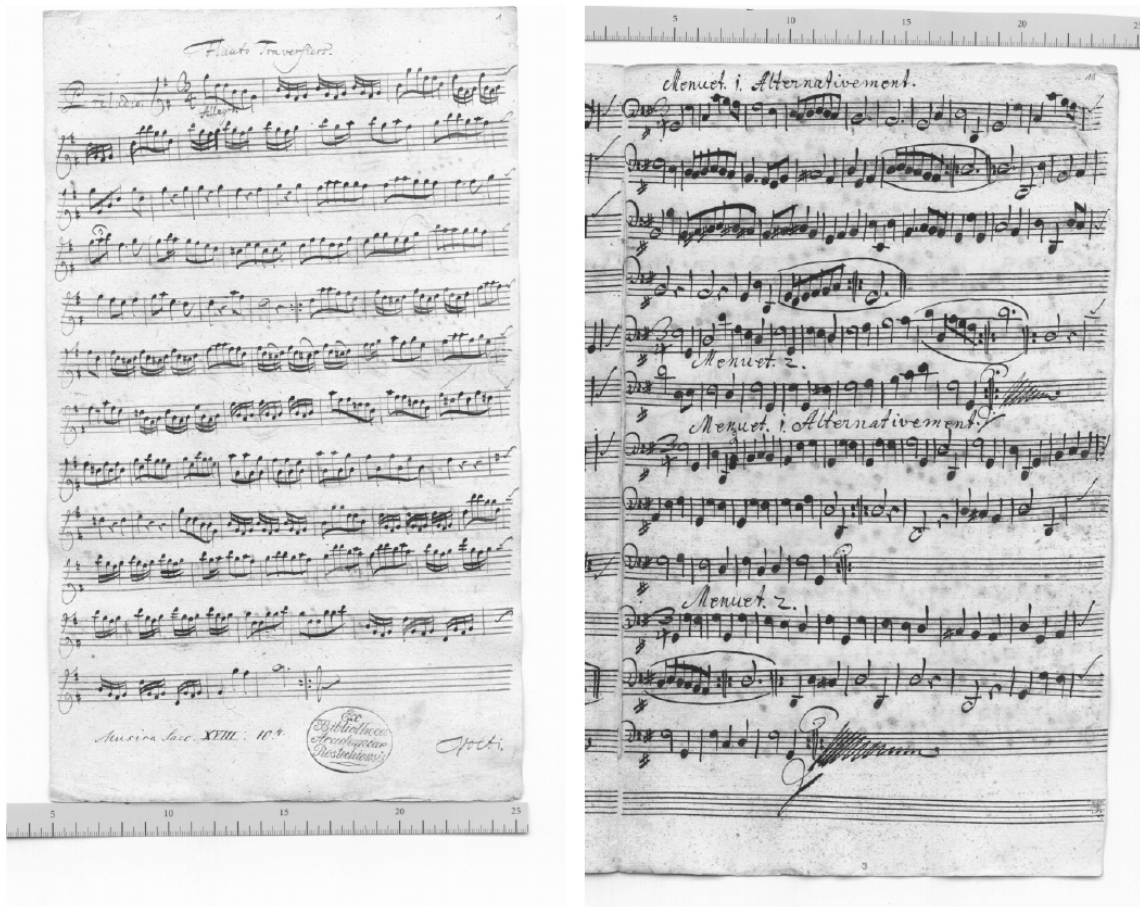


Figure 6.1: Examples of music manuscripts written by different scribes

### 6.1.1 Requirements Analysis

The project eNoteHistory<sup>1</sup> is a cross disciplinary pilot project in which a digital archive of historical music manuscripts was developed. The participating parties comprised database experts and music scientist from the University of Rostock and image processing experts from the Fraunhofer Center in Rostock. The project had a duration of 3 years and was completed in February 2006. In addition to the common digital catalog search functionality, the eNoteHistory digital archive provides a special similarity search functionality for music experts to identify unknown scribes of music manuscripts. The common usage scenarios supported by the digital archive system are full text search and browsing through the catalog metadata, as well as visualization of the scanned pages of music manuscripts. In order to support this functionality, a database model was defined to store the catalog data and the digitized music manuscripts. The database search functionality provided by the IBM DB2 UDB and the IBM Net Search Extender are used in a relatively straightforward way to query and browse these data.

However, the main challenge of the application was to support the scribe identification sce-

<sup>1</sup>[www.enotehistory.de](http://www.enotehistory.de)

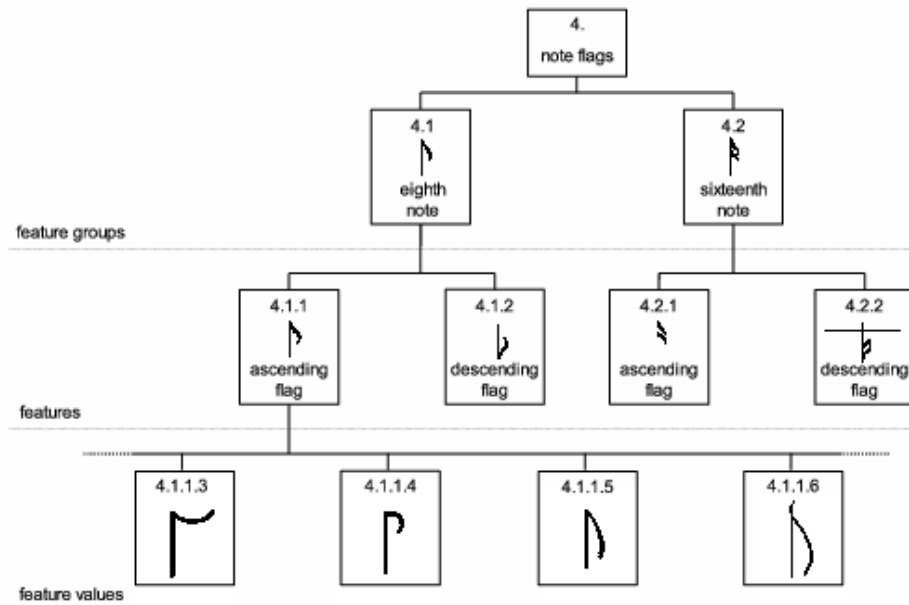


Figure 6.2: Excerpt from the feature dictionary

nario. Figure 6.1 shows two manuscripts written by different scribes. The identification of scribes is based on the comparison of the handwriting characteristics, represented in terms of features, of a manuscript whose scribe is not known with those of the manuscripts in the database with already identified scribes. Therefore, two different techniques for extracting and comparing the handwriting characteristics were developed in the project. These techniques will be further on referred to as the semi-automatic and the automatic techniques.

#### 6.1.1.1 The Semi-Automatic Scribe Identification Technique

The first technique, elaborated to a great extent in the diploma work of Lars Milewski [Mil04] and published in [BIM04], uses a so called *feature dictionary*, specified by the music scientists participating in the project, to represent the handwriting characteristics of music manuscripts.

**Data Structure:** The feature dictionary comprises 13 feature categories corresponding to objects in the music manuscript identifiable by music experts, e.g., clef, note head, note stem, note flag etc. Each of these categories is further specified by more detailed categories, e.g., G clef, white note head, half note stem, eight note stem etc., and/or concrete values for these categories. The categories and their values are partially represented as small images as shown in Figure 6.2 and/or textual descriptions.

A feature in this tree is the last node before a value node appears. All nodes before the feature node are considered as prefix for that feature, i.e., they describe the feature. The number of

features for all categories in the feature dictionary is about 80. Thus, a feature vector for a music manuscript can contain up to 80 features, whereas for a specific feature more than one value can be assigned. The description of a handwriting found in a manuscript in terms of these features is very detailed and extensive. However, it has to be done manually, because of the difficulties posed to image processing algorithms described in [Göc03]. Therefore, a supporting tool for browsing through the feature dictionary was implemented, in order to help assigning features to the database manuscripts with known scribes. It can also be used as help to formulate a query for identifying the scribe of manuscript outside the database.

**Retrieval Mechanism:** The identification process requires that for all manuscripts in the database with known scribes a representative feature vector is created by music experts. The first step towards identifying the scribe of a new manuscript is the creation of a query feature vector by browsing through the feature dictionary and choosing appropriate feature values for the manuscript at hand. The query feature is then send to the database and is compared with the feature vectors of the database manuscripts using the k-nearest neighbor algorithm, where the overall distance between two music manuscripts is calculated as the Hamming distance between the two feature vectors  $\gamma_a = (v_1^a, \dots, v_n^a)^T$  and  $\gamma_b = (v_1^b, \dots, v_n^b)^T$  as follows:

$$d_{\Gamma} = \frac{d_{f_1} + \dots + d_{f_n}}{n}$$

where  $d_{f_i}$  is the distance between the single features  $v_i^a$  and  $v_i^b$ . The distance function is not applied on the pictorial or textual representation of the feature values which can change to make the features more understandable to the users. It uses the point notation of the node in the feature dictionary. In the above example a value for the ascending flag would be 4.1.1.3. This notation corresponds to the path in the tree structure leading to this node. Thus, the distance between the different features can be partially calculated using the structure of the feature dictionary. However, for most features the distances have to be predefined in a so called similarity matrix, where similarity values for the feature values are assigned by music experts. The k-nearest neighbor algorithm is then applied to decide in which cluster of scribes does the query scribe fall.

**Application Model:** To give a better idea about the data needed to be stored to support this application scenario, the conceptual data model of the database is depicted in Figure 6.3. It should be noted that the functionality of the database application was not completely explicitly included in the database conceptual model during the project. The functionality was designed and implemented separately, thus, it has to be described additionally. The function responsible for retrieving the similar scribes from the database is implemented as a user-defined function which encapsulates the calculation or retrieval of distances between single features and the calculation of the overall similarity between the query vector and the database vectors. The query vector is given as an input parameter and the result is a list of scribes whose handwriting vectors have a distance smaller than a predefined threshold to the query vector.

### 6.1.1.2 The Automatic Scribe Identification Technique

The evaluation of this semi-automatic scribe identification approach provided important information for the second identification approach, developed in the eNoteHistory project, which

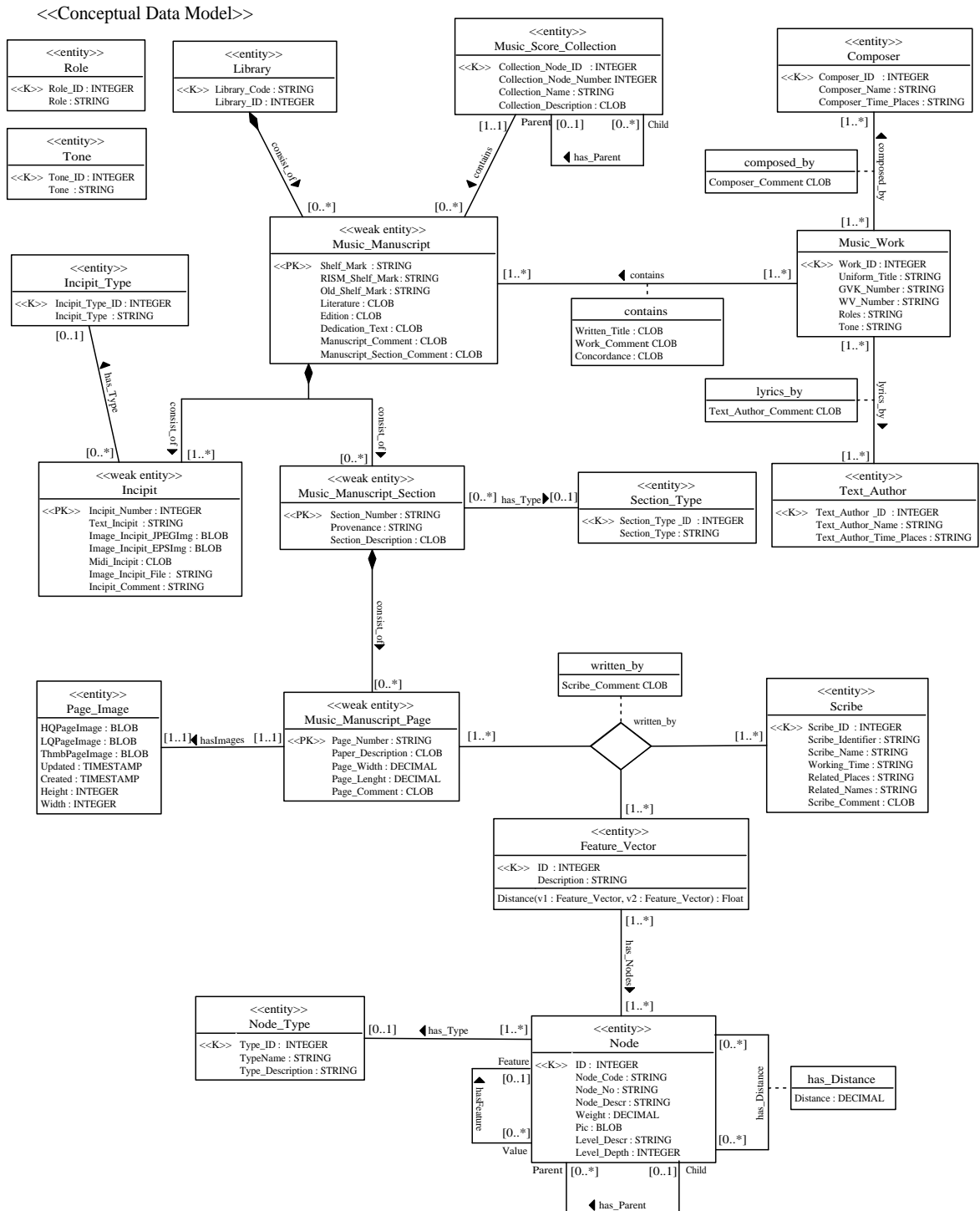


Figure 6.3: Conceptual data model of the digital archive for music manuscripts (from [Dol04])

relies on fully automatic feature extraction.

**Data Structure:** The evaluation experiments helped weighting the features used for the identification of scribes in the semi-automatic approach and thus minimizing the requirements towards the automatic extraction algorithms. Finally, only three feature categories, i.e. music objects, could be automatically recognized by the image processing algorithms, namely, note heads, note stems and bar lines. Each of these objects could be described with high precision by measuring the bounding box and the bounding ellipse parameters. Thus, the handwriting features could be represented by numeric parameters of all the note heads, note stems and bar lines found on the pages of a manuscript.

**Retrieval Mechanism:** The application steps involved in the automatic handwriting analysis and content-based retrieval are carried out in the following order. At first, for all digital scores in the database, for which information about the scribe (e.g., name of scribe) exists, image processing algorithms are applied to extract the visual features of the images, representing the handwriting characteristics. Figure 3.3 in Chapter 3 shows the recognized objects in the manuscript. For each recognized object: note heads, note stems, and bar lines, a set of geometrical features is extracted, such as: height and width of the bounding box, radius of the bounding ellipse, x, y coordinates of the centroids, orientation etc. In order to identify the unknown scribe of a manuscript two different retrieval techniques can be used. If the metric approach, also used for the semi-automatic analysis is applied, the handwriting characteristics of scores with unknown scribes can subsequently be compared with the set of extracted features in the database. Using distance metrics for calculating the similarity between features, a query result of the type: a list of k-most similar scores with associated scribes can be generated. A data mining retrieval approach can also be applied for the automatically extracted features. In this case, the classifier has to be trained and tested with the features of known scribes classes from the database and thus constructed tree can be consequently employed to classify unknown scribes. For the identification procedure in this case the data mining technique of decision trees was employed, in particular a Logistic Model Tree, since it returned better results than other techniques tested. The feature extraction and data mining functionality was developed by the image processing experts from the Fraunhofer Center as a stand alone application separately from the digital archive application until the algorithms were tested and refined enough. Consequently, this functionality was integrated in the digital archive system by the database experts by extending the data model to support the storage of automatically extracted features and to leverage the data mining functionality in the database.

**Application Model:** A simplified representation of the data types, proposed in the students work of Henning Masuch [Mas05] which defines these data structures and data mining functionality, are represented in Figures 6.4 and 6.5, respectively.

The automatic scribe identification is the one which is modeled for the evaluation of the GiACoMo model, since it does not require the modeling of a user interface for extracting the music features. Modeling user interfaces is out of the scope of the framework model for CBIRs. Both retrieval approaches, discussed above are integrated in the model in order to demonstrate that these can be used interchangeably or in combination in the application.



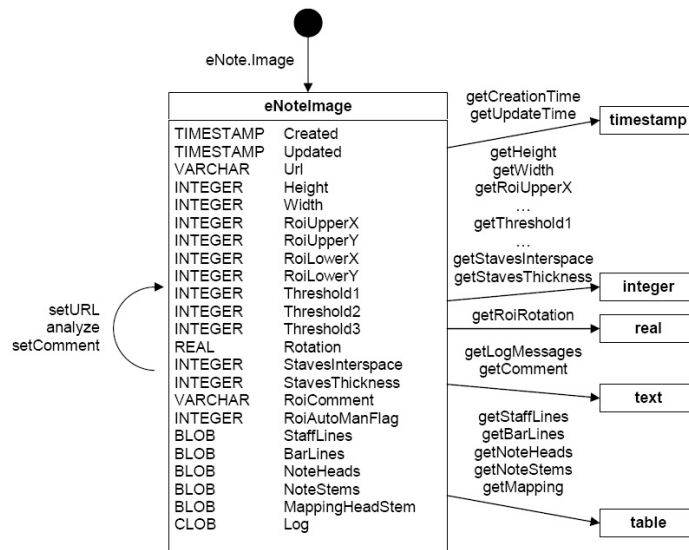


Figure 6.4: Simplified representation of the data type for storing the automatically extracted features (from [Mas05])

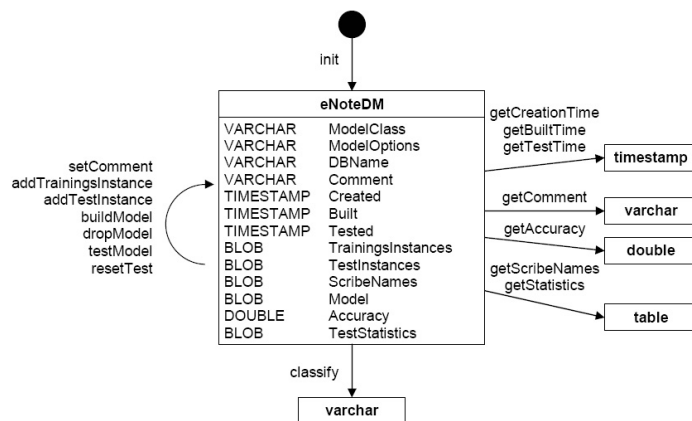


Figure 6.5: Simplified representation of the data type used for the classification of scribes (from [Mas05])

### 6.1.2 Modeling the eNoteHistory CBIRS

For this CBIRS the platform independent model shown in Figures 6.6 and 6.7 is derived from the framework model. The **eNoteImage** class represents the set of scanned page images of music manuscripts. In addition to **TechnicalMetadata**, which is predefined as a reusable application-specific class in the framework model, a class **LibraryMetadata** is defined, which is derived from the abstract class *Metadata*. The associations between the **eNoteImage** and **LibraryMetadata** and **TechnicalMetadata**, respectively, have been redefined. The names of the redefined associations, such as **has metadata** are left unchanged in order to recognize easier that they are redefined. Two classes of raw image representations are defined for the **eNoteImage**. The multiplicities of the associations **has rawimagerep** can be redefined to allow only one **eNoteRawImage** for an **eNoteImage**. Two types of regions are derived from

the class *Region*, **ROI** (Region of Interest) and **MusicObject**, which is further specialized in **NoteStem**, **NoteHead**, and **Clef**. The latter represent regions, which have been identified as the corresponding music elements. Unidentified regions can be stored as **MusicObjects**. A **ROI** is the region of the digital image which contains only the relevant information - the staff lines without the edge of the paper and notes at the corners of the page, such as page number. A **ROI** contains the music objects as shown by the redefined **related to** association. Furthermore, music elements can have directional relationships, which can be used for example to identify if a note head belongs to a note stem. The localization information for a **ROI** is represented in a separate class and the one for a music element inside the **MusicObject** class. The class **eNoteFeature** represents the set of features used to describe the regions of an **eNoteImage**. For the current application only a shape descriptor as a feature of the regions of music elements is applied to compare their similarity.

The operations, defined in the model, are used on one side to create the data which has to be derived from the image by segmentation or feature extraction. On the other side, they are used to support similarity queries on the images, by providing a distance function for the features representing the content of the images. The store operations should implement a storage mechanism for making the corresponding instance persistent if no such mechanism is provided by the platform. The operation **segmentImage()** can be used to create the instances of regions for a specific image. For extracting the features of a region the corresponding feature extraction function of a feature should be implemented. The retrieval functionality based on the metric approach is carried out based on the comparison of local features. Therefore, for each feature type a **compareWithAnotherFeature()** operation should be implemented. The accumulated distance function of the feature distances should be calculated by the **compareByFeaturesWithAnotherRegion()** operation. And finally, the aggregation operation for calculating the similarity between the whole images should be provided in the **compareByLocalFeaturesWithAnotherImage()** function. The data mining approach for retrieving the scribes of a manuscript is modeled with the help of the GiACoMo *CBIRClassifier*. The eNote specific classifier for a Logical Model Tree (LMT) **LMTClassifier** is derived from the generic classifier class. The implementation of the operations **addTrainingInstance()** and **addTestInstance()** creates the **eNoteTrainInstance** and **eNoteTestInstance** objects from the information of the **eNoteImage**. The decision tree can be built using the **build()** operation and the result is stored as an **eNoteDMMModel**. The query is processed by the **classify()** operation which requires either an image or a query instance as input.

In conclusion, it can be stated that the application specific model fits well in the generic framework model, since all required data structures and functionality could be derived from GiACoMo-IRS. This particular CBIR application is also a good example for the need of diverse multiple user interfaces of such systems. Music scientists are one type of user for the system. They require the scribe identification functionality in order to derive or prove theories about the origin of historical manuscripts. On the other hand there are the librarians, who require the possibility to view and edit the bibliographical or physical meta data of the digital manuscripts. A third group of users are musicians, who need the music manuscripts to adapt them for a performance. In order to satisfy these needs a web-based client application for the eNoteHistory CBIR system was implemented, which offers different functionality for different user groups. The model-driven development of these CBIR graphical user-interfaces would require additional models to be integrated into the GiACoMo framework model.

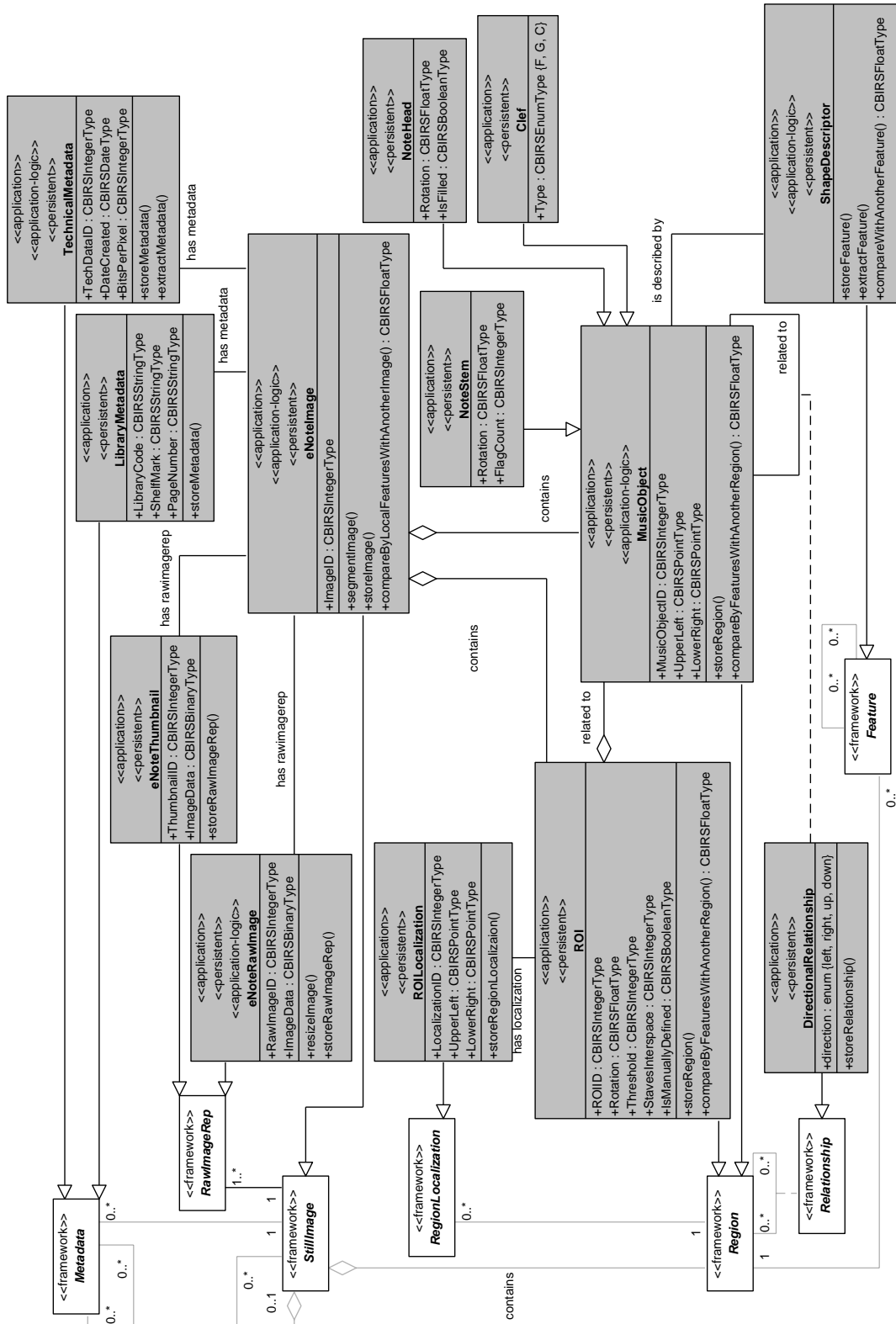


Figure 6.6: eNoteHistory CBIR PIM

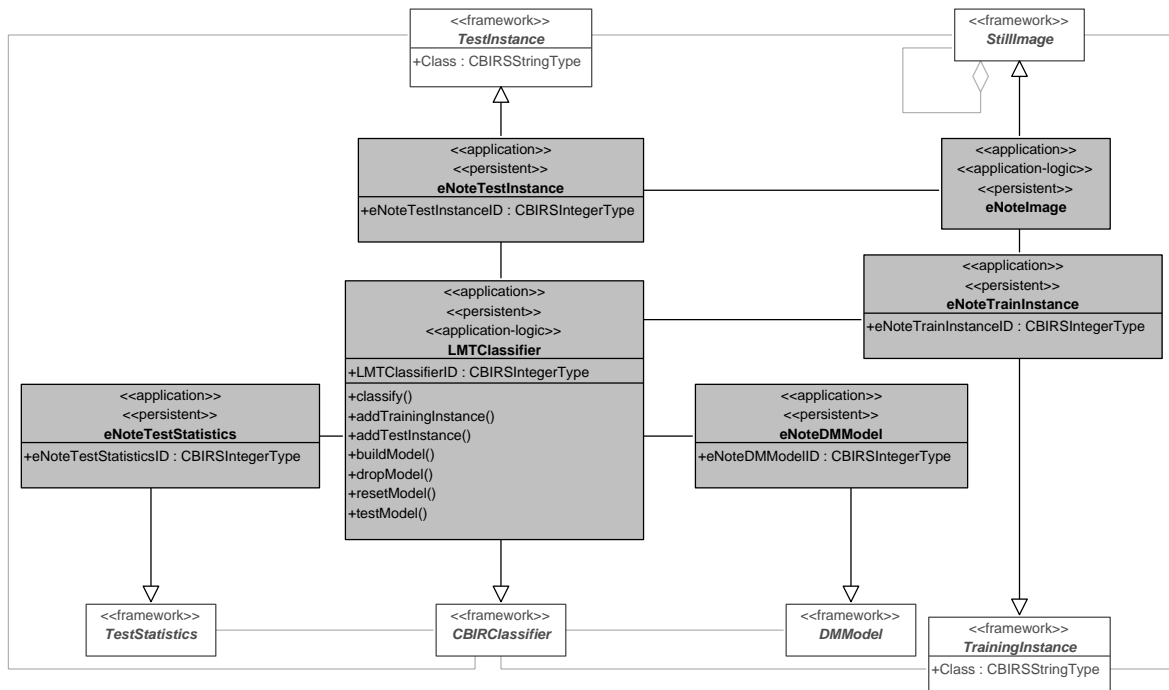


Figure 6.7: eNoteHistory CBIR PIM - data mining part

## 6.2 Test Case 2D-Gel Electrophoresis Images

In [Ign03] the implementation of a CBIRS on top of an ORDBMS for storing and retrieving 2D-Gel images along with other proteome laboratory data is described. Proteomics is a research field in molecular biology which is involved in the quantitative description of all proteins present in organic tissues and/or in body fluids. This study should help the process of developing new target compounds for new pharmaceuticals or new therapeutic approaches for treating autoimmune diseases. An essential part of a proteomics laboratory data are the 2D-Gel images, derived from the electrophoresis of body tissue and fluid samples. The CBIRS is designed to support the visual analysis of the 2D-Gel images. In order to identify the proteins in a new 2D-Gel image, this image is matched with already analyzed images, whose proteins have been identified. If an image is found which contains visually similar protein patterns it is presented to the scientist who decides if the match is correct and marks the matching proteins in the new image as identified. The final aim of the image analysis pursued by the scientists is to determine proteome differences between healthy and ill identical organisms.

### 6.2.1 Requirements Analysis

The input of a proteome experiment is a tissue or a body fluid from a living organism. This material is divided into small portions, which are prepared to be analyzed with different methods and in different conditions. Such a method is also the “2-dimensional gel electrophoresis”. A 2D-gel is the product of two separations performed sequentially in acrylamide gel media. Isoelectric focusing as the first dimension and a separation by a molecular size as the second



Figure 6.8: 2D-Gel electrophoresis image

dimension. A 2D pattern of spots, each representing a protein is the result of this process. Eventually spots are detected by staining or radiographic methods. After a scanning procedure of the gel media the so called 2D electrophoresis gel images are produced. An example of a 2D-gel image is given in Figure 6.8.

**Data Structure:** The produced gel images are most often stored as gray level, 8-bit digital images in TIFF files with a resolution of 200dpi. Their width is about 1000 pixels and the height is 2000 pixels. These characteristics lead to an average size of a gel image file without compression of about 2 MB - 3 MB. The images contain a certain number of dark gray spots with different shape, location and intensity on a light-gray background. Some of the spots represent proteins and others are just noise in the image. The  $x,y$  coordinates of each spot specify the Molecular Weight and Isoelectric Point of a protein spot in a calibrated image. The volume of the protein can be calculated by measuring the area, circularity and radius of the spot. The term “calibrated image” implies that the gel image is associated with a two-dimensional coordinate system, which provides a scale for calculating the Molecular Weight and Isoelectric Point, corresponding to the values of the  $x,y$  coordinates of a spot in this system.

The images are matched using the coordinates of the spots and their intensities as features. Thus, a content descriptor of a 2D-Gel image is a vector  $x,y,i$ , where  $x$  and  $y$  are non negative point coordinates of the centers of the spots in the Euclidean plane and  $i$  is a positive number, describing the intensity of the spots. The coordinates express the spatial characteristics of the objects on the image, which have to be compared. The additional information about the intensity of the spots can optimize the comparison function, since it sets more restrictions

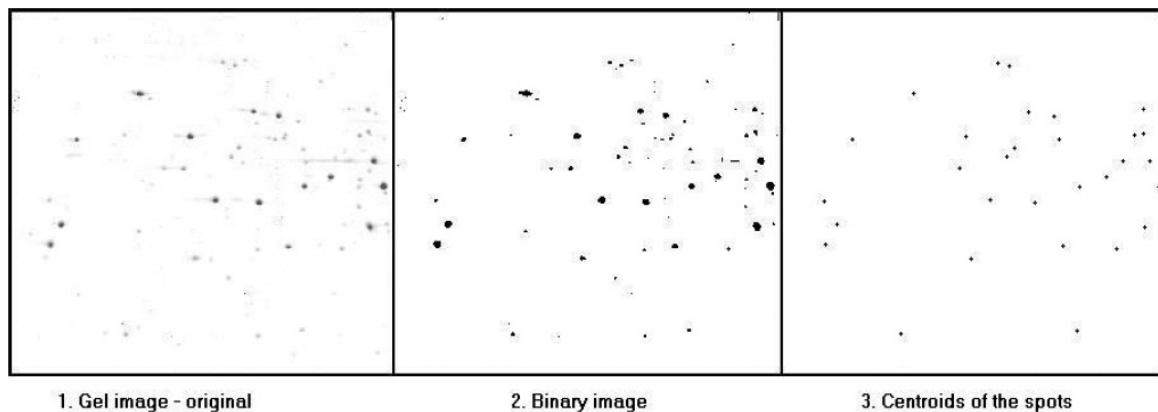


Figure 6.9: Results from the feature extraction algorithm

for the matching spots. In order to automatically extract this information from an image, several steps in an image-processing algorithm are performed.

- An automatic threshold of the image is performed in order to generate a binary image with segmented areas, which belong to the objects in the image. This operator is implemented using the local adaptive threshold algorithm of Otsu, which sets automatically the threshold value for local areas, defined by a sliding window, moving through the whole image.
- A morphological opening operator is applied on the binary image in order to separate the merged spots if there are any. It is preceded with one step skeletonization for better results and avoiding small noise in the image.
- The segmentation of the spots is performed by a labeling procedure with which the centroids of the detected spots are extracted and represented as a list of 2D coordinates of points and intensity values, taken from the original image. The intensity of each spot is calculated as the average intensity of the pixels of the original image in a given segment.

The results from executing these image processing steps are represented in Figure 6.9.

The considerable differences between images of the same cell, caused by inaccuracies in the staining procedure and scanning procedure etc. lead to different locations of the spots. This fact implies that a perfect match between two such images from an identical source is very improbable. The similarity between the images, however, still exists and it can be evaluated by applying metrics, which can represent the similarity between patterns (topologies) of points in the euclidean space, invariant to scaling, rotation, translation etc.

**Retrieval Mechanism:** The retrieval task which is supported by the CBIRS is to compare a 2D-gel image against a database of formerly analyzed images with identified protein spots, and return all images expressing the same patten of protein spots. If a match is found, the protein spots in the new image can be identified, otherwise they are marked as unidentified proteins and will wait for the expansion of the protein database to be identified. All images

from the experiments have to be stored and made available for later use for comparison with new images and identification of new proteins.

The spatial similarity quantification is done by comparing the locations of each of the spots in one of the images with the locations of all the spots in the other and the minimum of the maximum of all measured distances is considered to be the distance between the two images. For the implementation of this functionality a method for measuring the Hausdorff distance between point sets is used. The Hausdorff distance is a standard metric for determining the distance between two point sets by measuring the largest distance between a point in one set and its nearest neighbor in the other.

### 6.2.2 Modeling the 2D-Gel Electrophoresis CBIRS

In Figure 6.10 the derived PIM for the 2D-gel image CBIRS is represented. The 2D-gel images are represented by the **2DGelImage** class. The class implements the derived abstract operations for storing the images, for segmentation and for their comparison with other images based on local features. The raw image data is represented by the **2DGelRawImageRep** class. This class defined apart from the implemented inherited operations an operation for the manipulation of the image data **scale()**. Since this operation has to be implemented in the application-logic layer of the system, the class **2DGelRawImageRep** is marked as an «application-logic» class. Meta data, including technical meta data and data describing the experiment from which the images originate are represented through the reused black box class **TechnicalMetadata** and the derived class **LabMetadata**. The features of the image required for the content-based comparison are represented as a specialization of regions **ProteinSpot**, which resembles an area of the image identified as a protein. Each **ProteinSpot** is described on one side by its location **SpotCoord** and on the other by its **Intensity**. The calculation of the distance or similarity between single spots is assigned to the operation **compareWithAnotherRegion()**. This operation should use the operation provided by the intensity class **compareWithAnotherFeature()**.

Analogously to the first test case, new data types had to be defined, **CBIRSDateType** and **CBIRSPointType**, corresponding to a data type representing a date and two dimensional point coordinates, respectively.

All inherited associations between the derived classes have been redefined. The cardinalities of the redefined associations have been also adapted where necessary.

Hence, the conclusion can be made that this CBIR application can also be derived from the GiACoMo-IRS model almost completely.

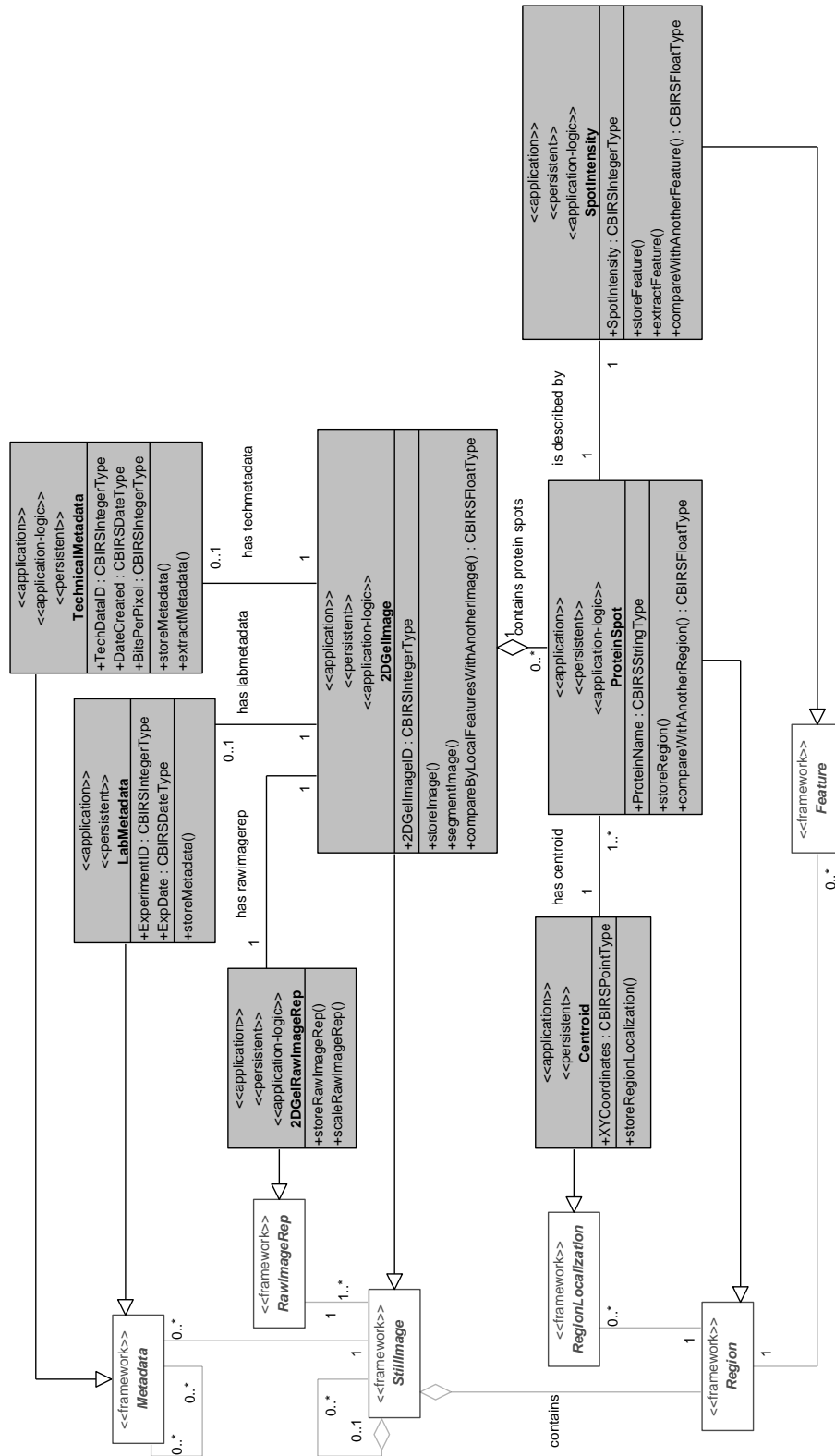


Figure 6.10: CBIRS PIM for 2D-gel electrophoresis images



## 6.3 Test Case Photo Album

In Chapter 1 an extendable framework MetaXa for the automatic annotation of photos using context and content based information was described. The extracted information can furtheron be used to create a photo album according to specific criteria. In this section the requirements of such a photo annotation system are described based on the example application from the MetaXa framework, represented in [BSST07], and a model for such an application is derived from the GiACoMo framework model.

### 6.3.1 Requirements Analysis

The main functionality of the photo annotation system is to extract content and context related data from the photos and, if possible, use these data to derive a higher level description for these. Therefore, the following content extraction algorithms have been implemented as components of the MetaXa framework:

- Histogram generation: This algorithm extracts four histograms from a photo. One histogram for each of the color channels RGB and a brightness histogram derived from the other three.
- Edge detection: Using the Sobel operator with two different kernels this edge detection algorithm produces two images with marked edges. The resulting information can be stored either as the images themselves or by some kind of an edge histogram descriptor, similar to the defined in the MPEG-7 standard.
- Face detection: This algorithm detects the number of faces in an image. It could be used to detect the bounding boxes of these faces also.

The context data extraction algorithms are defined with the aim to retrieve the general image information, which corresponds to content-independent image data in the sense described in Chapter 3, e.g. height, width, creation date etc. Furthermore, EXIF data is extracted by these algorithms, such as, exposure time, aperture, flash on/off, orientation of the camera, and even GPS coordinates if present.

From the data, which can be more or less directly extracted from the photos so called enhancement components derive higher level information, i.e. enhance the previously extracted data. The content-based data can be enhanced with algorithms for:

- Sharpness analysis: The good quality of a photo is considered to correlate to its sharpness. The sharpness of a photo, therefore, is proved by analyzing the brightness histogram in the center of the photo.
- Exposure analysis: Another criterion for a good quality of a photo is its good exposure. To prove that a photo is not underexposed or overexposed again the brightness histogram can be used.

The context-based data can be enhanced with the following algorithms:

- Light conditions determination: The exposure value which correlates to the brightness of the scene can be derived from the aperture and exposure time values found in the EXIF header.

- Calender event enhancement: By combining the photos date and time information with a personal calender application information about the event place and duration, and with less confidence names of participators can be associated to the photo.

Additionally, a component for In/Outdoor classification combining the content and context-based data in order to derive new information about the images has been implemented. It uses the light condition information extracted from the content of the photo and daytime information, flash on/off, and exposure from the context data in combination with rules to decide whether the photo was made in or outside.

Another component aims at harvesting the Web based on previously extracted data such as GPS coordinates and image similarity, to associate more relevant content from the Web. A component aiming at identifying similar photos of the same motif, taken one after the other has been also implemented using a kind of local color feature matching.

The data model which is used in MetaXa to represent the extracted metadata is a very generic model which allows building data types corresponding to the different metadata. However, it does not reflect the relationships between this data. These are defined as dependencies between the extraction and enhancement algorithms and are represented in a workflow component of the architecture. The architecture of the MetaXa application is built around the requirements for the functionality of the components. Less attention is paid to the data structures representing the extracted information, and how they can be organized in order to support the desired photo album creation scenario. Thus, when applying the GiACoMo model for modeling such an application, which is a data-centered rather than a functionality-centered approach, the features of the application have to be considered from a data-centric perspective. For example, for modeling the histogram extraction components not the extraction algorithm itself should be considered first, but the result of this algorithm, i.e. the histograms. Moreover, the dependencies between the different algorithms can be represented as dependencies between the resulting data structures.

### 6.3.2 Modeling the Image Annotation Application

In Figures 6.11, 6.12 the derived model for the photo annotation application is shown. The components of a CBIRS which have to be modeled in order to cover the requirements of the application are the “Image Store” and “Feature Store” storage components and the “Feature Extraction” component. The “Retrieval Component” can be designed to serve for benchmark purposes or for the photo album scenario. In this case, the benchmark scenario is considered. All extraction and enhancement components of the MetaXa architecture can be modeled as the “Feature Extraction” component of the CBIRS. The results of this functionality are reflected in the structural part of the derived CBIRS model. As already mentioned, since GiACoMo is a data-centered model the first parts of the model which are derived are the structural parts representing the images and their extracted content and context. On one side there are the content-based feature classes **PhotoHistograms**, **PhotoFaces** and **PhotoEdges**. On the other side, content-independent information are modeled as metadata classes **EXIFData**, **CalenderEvent** etc. The content extraction functionality is modeled as overridden methods of the respective feature classes `extractFeature()`. The context extraction functionality is represented analogously through the methods of the respective Metadata classes. The difference is that the GiACoMo model does not offer a generic method in the superclass Metadata for extracting these. Thus, the method had to be added additionally.

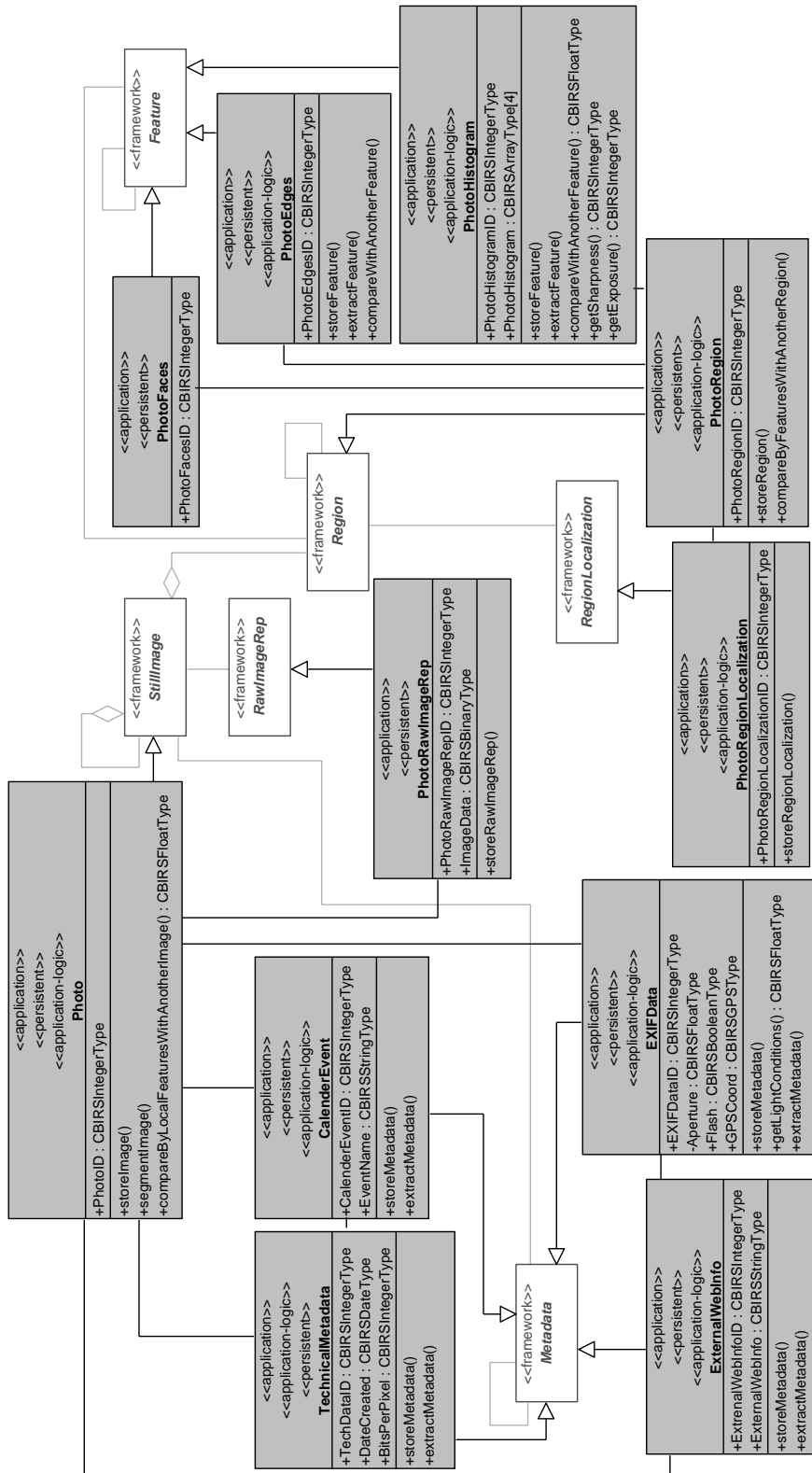


Figure 6.11: CBIRS PIM for a photo annotation application

For modeling the enhancement components for deriving higher-level image descriptions methods for calculating derived values as in the case of calculating the light conditions from the EXIF data, e.g. `getLightConditions()`, are used. The more complex derivation algorithm for In/Outdoor classification is modeled as a *CBIRClassifier*. The question which arises when modeling the enhancement components is if their results have to be treated as persistent features or metadata, or if they should be calculated on the fly when needed. The choice can be made in favor of the first approach if the required algorithms are very complex and time consuming, as well as in the case that more than one feature and/or metadata are used to derive the new data. Otherwise, the representation of the components as methods should be used.

The similarity functionality for finding images of the same motif is represented through the metric model similarity methods in the features, regions and image classes. If different combinations of features have to be used for selecting different image sets corresponding similarity methods have to be defined in the **Photo** class.

Consequently, the photo annotation application, which was not known before the creation of the GiACoMo-IRS, can be considered derivable from this model.

## 6.4 Estimating the Gain From the Result of the Transformation

As stated in the beginning of this chapter, in order to estimate the gain from the exploited development technique, an analysis of the effort, needed additionally to complete the application implementation is needed. Therefore, the results of a transformation of an arbitrary PIM are described and the required adaptations, i.e. enhancements to the implementation are identified.

### 6.4.1 Mapping Data Structure

Structural concepts, such as classes, data types, references etc. are mapped completely to corresponding concepts in the platform-specific model. The developer has to add code to the generated implementation only in cases where the PSM requires more specification data, which is not delivered by the conceptual model. In the case of mapping onto SQL:2003 these could be additional specifications of the **Structured Type**, such as FINAL or NOT FINAL.

### 6.4.2 Mapping Functionality

The mapping of functionality is divided into mapping of system functionality and mapping of object functionality. The object behavior is represented in SQL:2003 through the methods of user-defined types. The signature and body of these methods are separated in SQL:2003. In UML only the signature of an operation is given in a class diagram. The implementation of the methods must be provided directly in the programming language of the implementation, e.g. Java. In Figure 6.13 on the left side the UML diagram class and method are shown and on the right side the translation into SQL:2003. During the mapping process the declaration of the method in the relational model is based on the signature of the method in the UML class diagram. The implementation of the method can be added to the database user-defined

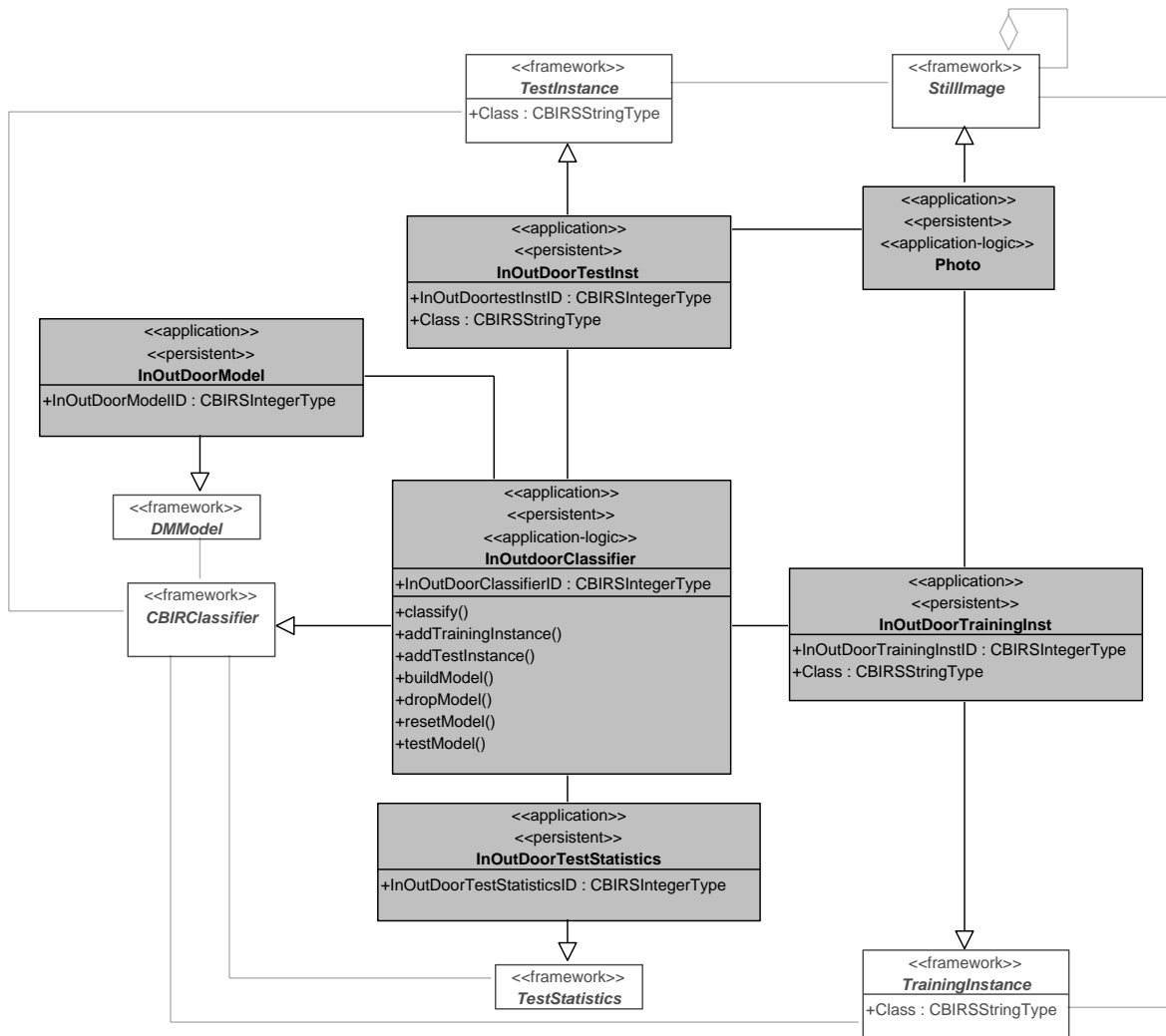


Figure 6.12: CBIRS PIM for a photo annotation application - data mining part

functions either from a predefined library of the framework model or from a customized implementation of the developer. One possible mechanism for facilitating the process of assigning implementations to the modeled methods is the MView plugin described in [Sch07]. This plugin is designed to help the developer to assign implementation packages to the model elements. The plugin offers a tree view of the model and the package structure and a drag and drop mechanism for associating the package structure elements with those of the model structure.

For mapping the system functionality the interface provided by the ORDBMS in terms of SQL data manipulation and data query language and the extensibility options in terms of user-defined functions and stored procedures are used. There is more than one possibility to realize the Insert operation for images between which the developer can choose. One possibility is to encapsulate the extraction of the regions and features and their insertion into the database in a user-defined function, representing the constructor of a StillImage object. Another possibility is to make use of the TRIGGER object in an ORDBMS to invoke the

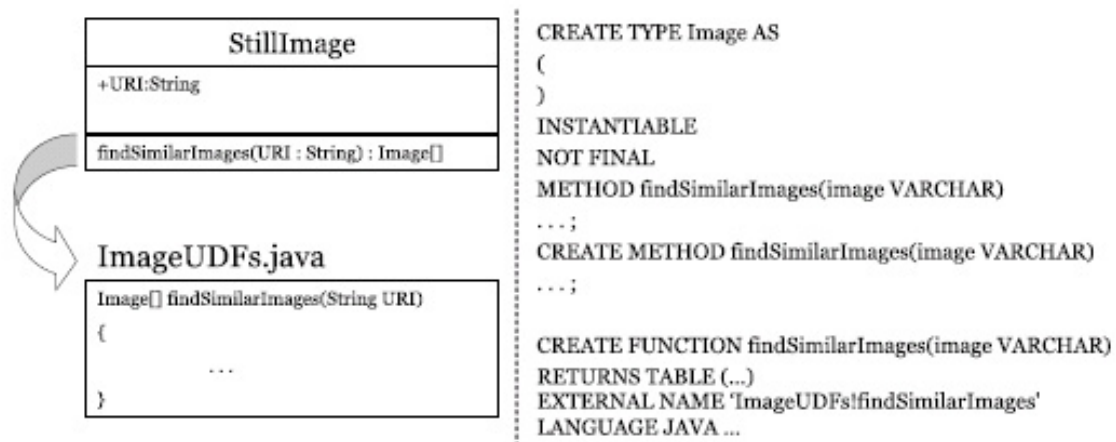


Figure 6.13: Mapping of methods onto user-defined functions

region and feature constructors for extracting the regions and features of an inserted image. In both cases the following SQL statement should be issued to insert an object of the type `StillImage` into the database:

[Oracle syntax:]

```
INSERT INTO SCHEMA.IMAGE
VALUES (StillImage(URL))
```

The efficient processing of queries is one of the main advantages of database management systems. The query processing operations supported by the DBMS are generic, and therefore, domain independent. They are global for the whole system. However, they have been defined to apply them on standard basic data types (alphanumeric data types). In order to support application specific queries such as the similarity queries on the `StillImage` data type we have to provide special operations for comparing object of this type. These operations realizing the query operation from the UML class diagram are mapped onto methods of user-defined types in the database. The possible usage of these methods in the SQL query can be as follows:

[SQL:2003 syntax:]

```
SELECT *
FROM getSimilarImages(URL, threshold)
```

With this implementation the whole query processing algorithm is encapsulated into the user-defined function `getSimilarImages(...)`. The main disadvantage of this approach is that it does not allow for any query optimization on behalf of the DBMS. Therefore, clustering methods for building predefined clusters of similar images by specific features or a combination of those should be used. Hence, the user-defined function for comparing the query image with the ones in the database would have to be applied on the first place only onto the cluster representatives of the images. In this way, we can improve the efficiency for the processing of the similarity queries.

The implementation of the functionality of the model requires more additional effort from the developer than the implementation of the structural part. Since the model does not provide a representation of the complex image processing or distance measure algorithms, these have to be implemented additionally. Different solutions can be suggested to facilitate this task. These algorithms can also be provided as part of the model (sequence diagrams, OCL etc.) or their implementations can be associated to the corresponding operation signatures in the class diagram before the transformation step takes place. The first approach could lead to very complex models, difficult to handle and to transform into an implementation. The second approach has the advantage that specific libraries or packages can be reused by the developer. It should be mentioned that in order to be able to use the system the developer should implement the needed user interface layer of the system, which is not part of the conceptual model. As pointed out in the previous chapter this task can be facilitated with a modular implementation of the user-interface, which can be parametrized using information from the conceptual model. This approach has been prototypically implemented in the Image Database Generator Eclipse Plug-In, described in Chapter 5.

## 6.5 Summary

In this chapter, the exploited development methodology for content-based image retrieval systems was evaluated, based on two main criteria. On one side the quality of the modeling approach was evaluated using three formerly implemented applications, by deriving their conceptual models from the GiACoMo-IRS. These tests showed that the applications are almost completely covered by the generic adaptable model, and therefore, it can be concluded that the GiACoMo-IRS model is complete for a certain class of CBIRSs. The possibilities to extend and adapt the model with further data types and special associations found also their application in the instantiated models.

The second part of the evaluation dealt with estimating the gain of the development approach by identifying the additional effort needed to finalize the implementation of the systems. The analysis showed that most of the additional implementation work which has to be done by the developer is for the implementation of the functionality of the system. As a way to facilitate this task, the usage of class libraries and/or packages with standard image-processing algorithms and distance metrics was suggested.

## Chapter 7

# Conclusions and Prospective Research Directions

Developing a universal CBIRS which can solve all past, present and future image retrieval tasks emerging from various domains is still a challenging task. Each different application field requires the adoption of often a few, but well chosen techniques for feature extraction, similarity matching etc. Although such specialized applications apply different sets of techniques, their overall architectures comprise of the same components. There is also a broad range of techniques, which can be reused to implement the components of new CBIRSs. Approaches for how to make use of this know-how, gathered through the almost 20 years developments in this field, in order to facilitate the development of tailor-made CBIRSs have been proposed recently. These approaches involve software engineering technologies, such as, software frameworks and code libraries, as well as DBMS implementation techniques, such as, database extensions. Software frameworks, however, are developed for a specific platform. They offer a reusable architecture but almost no reusable components. Thus each of the components has to be completely implemented by the developer for a specialized application. The resulting applications are not compact, but are more like an extension of the framework. Code libraries are also platform dependent and do not provide a reusable architecture. Their combination with the framework approach should be considered. Database extensions for image data define a reusable image data type in combination with some content-based retrieval functions. These data types and functionality are, however, very generic and cannot be adapted to the needs of a special application. This might be the reason why these extensions are not enough exploited in current DBMSs.

Applying the model-driven software development approach for the implementation of CBIRSs can considerably facilitate the development process and consequently improve the quality of the resulting software. This development approach shifts the development process on an abstract level, thus it guarantees platform independence of the resulting application. The reuse of a software architecture is provided at the modeling level by a generic framework model of the application. The reuse of particular techniques or algorithms is possible at both the modeling and the implementation level, either by pattern-like constructs or black box components in the framework model, or by code libraries, respectively. Another advantage of this approach is that the final implementation of the application in terms of a programming language is generated as far as possible automatically from the model, in order to reduce



the effort for typing source code for the developer. In order to apply this development approach for the implementation of CBIRSs, adequate domain-specific techniques for each of the processes in model-driven development, i.e. conceptual modeling, model transformation, were elaborated in this thesis. The techniques were elaborated based on a requirements analysis, which determined the parts of the CBIRS which have to be modeled and generated by the development approach, as well as the target software architecture of the application. The results of the analysis of these aspects and the corresponding model-driven techniques are summarized in the paragraphs below. For each of these techniques, open issues are pointed out.

### **A Generic Software Architecture of CBIRSs.**

CBIRSs have been described using a generic system architecture in numerous books and articles on the topic. These architectures claim to cover a broad range of existing systems. Thus, an integrated view of a generic architecture of a CBIRS can be constructed, which depicts all common components of such systems. To answer the first question posed in the beginning of this thesis “What should be modeled?” an analysis of the generic architecture and components of CBIR systems was carried out. Three groups of generic components were identified: user interface, compensator and core CBIRS components. The development of the core CBIRS components was chosen as the aim of the model-driven approach. User interfaces were left out of the requirements analysis because on the one hand there are already methods dealing with the model-driven development of user interfaces, e.g. task models, and on the other hand the development of user interfaces for different types of users, different types of terminal devices, can be done independently of the CBIR core components. Compensator components are needed in cases where the user interfaces have higher requirements towards the core CBIRS components, which cannot be implemented directly in the core. Thus, these components can be regarded as belonging more to the user interface and are left out of the requirements analysis. Consequently, the CBIRS components which can be designed with the help of the model-driven techniques are the components for extracting and storing image content and those for retrieving images by their content. The different implementation possibilities for these components were used to set the requirements towards the modeling technique.

The first group of components requires the possibility to model different representations of image content in terms of low- or high-level features, structure, objects etc., as well as content-independent image data, such as technical data. The possibility for modeling functionality is also required in order to represent the operations needed to extract the image content. Retrieval components set high requirements for the modeling approach. The large number of image retrieval mechanisms makes it difficult to generalize the requirements towards a model. Therefore, in this thesis two retrieval approaches: the metric the classification approach are included in the requirements for the modeling technique.

### **Implementation Paradigms for CBIRSs.**

A model-driven transformation technique can be defined only if a concrete software architecture and implementation platform are determined. An analysis of the currently applied implementation paradigms of CBIRSs was carried out in order to choose one of these for the

elaboration of the transformation technique. Most existing CBIRSs have been developed as centralized CBIRSs. Distributed CBIRSs are still in the research stage. In order to have more reference application the choice in this thesis was made in favor of the centralized CBIRSs.

Using DBMSs as a basis for developing CBIRSs is one possible approach for this task. DBMSs have been especially extended to support image data types and corresponding content-based image retrieval functionality. These extensions have not received the expected popularity in the practice. They have been developed, to be reused in different applications, building on top of the DBMS. However, the data structures and the functionality is very rudimentary so no real CBIR applications can use them. Furthermore, these extensions cannot be adapted for the particular requirements of a CBIR application. Therefore, developers of CBIR systems either implement their own database extensions from scratch or even shift the CBIR part of the application out of the database. This problem can be alleviated by facilitating the development process for own database extensions, which can profit from the useful features of the DBMS platform. Therefore, it is a challenging application for the model-driven development approach of CBIRSs. Thus, extendable DBMSs (ORDBMSs) were used as the target platform of the model-driven transformation techniques.

Meeting the above decisions leaves out a number of other possible implementation paradigms for CBIRSs out of the scope of this thesis. These can be the focus of further research. Especially, the development of distributed CBIRSs might become of bigger interest in the future.

### **A Conceptual Modeling Approach for CBIRSs.**

The conceptual modeling techniques are the first group of techniques which was elaborated for the model-driven development of CBIRSs. A framework model approach using UML as a modeling language was proposed. This modeling approach uses a so called framework model - GiACoMo-IRS, which represents the generic parts of a CBIRS model as abstract constructs, as a starting point for the development of each CBIRS. The developer has to model the concrete CBIRS by deriving the needed constructs from GiACoMo-IRS. Structural and functional aspects have been integrated into the framework model which is represented by a UML class diagram. For each of these aspects examples for the instantiation of application specific models and “cookbooks” were provided.

The structural part of the framework model allows the instantiation of images, their raw representations, their structural semantics, their content characteristics and content-independent data. The developer has the possibility to reuse not only the whole architecture of the framework model but also some concrete “black box” constructs. The functionality which is represented in the model reflects the storage and the retrieval functionality of the CBIRS. It is represented in terms of operations of the abstract classes. These operations have been induced by stepwise modeling the system functionality with the help of use case and activity diagramming. Each detailed activity was mapped to a class operation in the framework model. For modeling the retrieval functionality the metric and the classification retrieval approaches were considered. Developers of CBIRSs who need to implement other retrieval approaches can follow the same stepwise induction method for defining the needed operations in the framework model. However, it is assumed that a large number of CBIRSs can be implemented with the predefined retrieval methods.

For the purpose of model-driven development of CBIRSs this modeling approach has the

advantage that it uses UML as a modeling language, which is widely used in the practice for conceptual modeling of software applications. Thus, it is expected that experienced developers do not need to learn a new modeling language and can quickly grasp the basics of the model. The framework model, which is represented in terms of a class diagram, introduces some small extensions to the UML, such as domain-specific data types. This idea can be further followed in a future work to define a set of UML extensions for modeling CBIRSs in form of an UML Profile. The current framework model includes only domain-concepts which are common for most CBIR applications. Instantiation guidelines and application examples have been provided along with the framework model in order to make its usage clearer. The developer also has the possibility to adapt and extend the derived model using the UML constructs.

The evaluation of this modeling approach was carried out using three CBIR applications from different application domains, which had been formerly implemented on top of ORDBMSs. All applications required the modeling of a complex image data structure and retrieval functionality. Despite their unique requirements their conceptual models could be derived almost completely from the framework model GiACoMo-IRS which showed that special CBIR applications fit well in the generic model. Only some additional data types had to be defined, such as Point, which can be considered to be included in the framework model. In order to make more credible assertions from the evaluation of the modeling approach a broader range of application domains must be considered. A usability study with expert developers can also be performed in order to measure the gain from using this modeling approach.

### **Transformation of the Conceptual CBIRS Model onto an ORDBMS Model.**

The transformation techniques in the model-driven development process depend on the meta models used to represent the conceptual model and the implementation. Since the CBIRS may need to be implemented onto more than one platform, as in the current case, the constructs of the conceptual model were marked according to predefined stereotypes corresponding to the different tiers of an architecture. The ORDBMS meta model was represented in terms of extensions of UML in order to enable the developer to add some platform specific features to the implementation on a modeling level. For the transformation of the CBIRS model into an ORDBMS mapping rules for each of the platform independent meta model concept of the conceptual model were defined. The problems when defining such mappings arise from the discrepancies between the two models: object-oriented and object-relational. Therefore, direct mappings of the PIM to PSM concepts are not always possible and workarounds for missing PSM counterparts had to be defined. The defined transformation rules preserve the information capacity of the model in the implementation model by assuring that each conceptual concept has a corresponding mapping in the implementation model. Although some of the PIM concepts are mapped to the same PSM concept this does not lead to loss of information, because in these cases either the conceptual concepts have an equivalent meaning in the conceptual model, or the implementation concepts can be on an instance level. In some cases there is more than one possibility for transformation. In order to make the right choice between these, their efficiency has to be estimated by applying heuristic or cost-based methods. Such an optimization analysis can be the topic of a further research in the area.

Only the standard SQL:2003 concepts were considered in the mapping rules. A further research can exploit the mapping onto extended SQL concepts, such as SQL/MM concepts. An analysis of the gain of the final result of the transformation for the developer was estimated.

Additional effort is mostly required in order to implement concrete algorithms for feature extraction or similarity measures. This effort can be reduced by combining the model-driven development approach with the reuse of code libraries, in which frequently applied algorithms are implemented.

It can be further investigated if this development approach can be extended to support also the model-driven development of multimedia information systems. The generic multimedia model, represented in [IB05] gives some ideas about how this can be achieved. Further research on the problem is currently carried out by Ilvio Bruder in his dissertation thesis “Structural and Knowledge-based Modeling of Multimedia Information Systems”.

Both the modeling and the transformation techniques for CBIRSs have been implemented in a prototype of a development environment as a Plugin for Eclipse. This tool has to be further developed to meet all the requirements of the model-driven techniques elaborated in this thesis. Its employment in the practice can help developers of CBIRSs in creating efficient domain-specific CBIR application, by reusing current know-how in CBIR development and benefiting from the features of DBMSs.



# Bibliography

- [ABH97] Peter M. G. Apers, Henk M. Blanken, and Maurice A. W. Houtsma, editors. *Multimedia Databases in Perspective*. Springer, 1997.
- [ABK01] Solomon Atnafu, Lionel Brunie, and Harald Kosch. Similarity-Based Algebra for Multimedia Database Systems. In *Proc. of the 12th Australasian Database Conference (ADC)*, pages 115–122, 2001.
- [ABSS98] S. Adali, P. Bonatti, M. L. Sapino, and V. S. Subrahmanian. A Multi-Similarity Algebra. In *Proc. of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 402–413, 1998.
- [ACB02] Solomon Atnafu, Richard Chbeir, and Lionel Brunie. Content-Based and Metadata Retrieval in Medical Image Database. In *Proc. of the 15th IEEE Symposium on Computer-Based Medical Systems (CBMS)*, page 327, 2002.
- [ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, 2004.
- [AG01] Laurent Amsaleg and Patrick Gros. Content-based Retrieval Using Local Descriptors: Problems and Issues from a Database Perspective. *Pattern Anal. Appl.*, 4(2-3):108–124, 2001.
- [AHF<sup>+</sup>88] R. D. Appel, D. F. Hochstrasser, M. Funk, C. Roch, Thierry Pun, and Christian Pellegrini. Automatic Classification of Two-dimensional Gel Electrophoresis Pictures by Heuristic Clustering. In *Proc. of the Sixth Meeting of The International Electrophoresis Society Electrophoresis'88*, July 1988.
- [AJO04] Laurent Amsaleg, Björn Thór Jónsson, and Vincent Oria, editors. *Proceedings of the First International Workshop on Computer Vision meets Databases, CVDB 2004, June 13, 2004, Paris, France*. ACM, 2004.
- [Amb03] Scott Ambler. *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

- [ATL07] The ATLAS Transformation Language. Website, 2007. Available online at <http://www.eclipse.org/m2m/at1/>; visited on Februar 25th 2008.
- [BFG<sup>+</sup>96] Jeffrey R. Bach, Charles Fuller, Amarnath Gupta, Arun Hampapur, Bradley Horowitz, Rich Humphrey, Ramesh Jain, and Chiao-Fe Shu. Virage Image Search Engine: An Open Framework for Image Management. In *Proc. of the SPIE Conference on Storage and Retrieval for Image and Video Databases*, pages 76–87, 1996.
- [BFHI03] I. Bruder, A. Finger, A. Heuer, and T. Ignatova. Towards a Digital Document Archive for Historical Handwritten Music Scores. In *Proc. of the 6th International Conference of Asian Digital Libraries ICADL*, Kuala Lumpur, Malaysia, 2003.
- [Bim99] Alberto Del Bimbo. *Visual Information Retrieval*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
- [BIM04] I. Bruder, T. Ignatova, and L. Milewski. Knowledge-Based Scribe Recognition in Historical Music Archives. In *Proc. of the 8th European Conference of Digital Libraries ECDL 2004*, Bath, UK, 2004.
- [BK01] Susanne Boll and Wolfgang Klas. ZYX-A Multimedia Document Model for Reuse and Adaptation of Multimedia Content. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):361–382, 2001.
- [BKPS03] C. Bauckhage, T. Käster, M. Pfeiffer, and G. Sagerer. Content-Based Image Retrieval by Multimodal Interaction. In *Proc. of the 29th Annual Conference of the IEEE Industrial Electronics Society*, pages 1865–1870, 2003.
- [BLPR07] Bernhard Bauer, Florian Lautenbacher, Günther Palfinger, and Stephan Roser. “AgilPro”: Modellierung, Simulation und Ausführung agiler Prozesse. *OBJEK-Tspektrum - Die Zeitschrift für Software-Engineering und Management*, Januar/Februar 2007.
- [BMSW01] Klemens Böhm, Michael Mlivonic, Hans-Jörg Schek, and Roger Weber. Fast Evaluation Techniques for Complex Similarity Queries. In *Proc. of the 27th International Conference on Very Large Data Bases (VLDB)*, pages 211–220, 2001.
- [BPC<sup>+</sup>00] Ana B. Benitez, Seungyup Paek, Shih-Fu Chang, Qian Huang, Atul Puri, Chung-Sheng Li, John R. Smith, Lawrence D. Bergman, and Charles N. Judice. Object-Based Multimedia Content Description Schemes and Applications for MPEG-7. *Image Communication Journal (ICJ), Invited Paper on a Special Issue on MPEG-7*, 16(1):235–269, September 2000.

- [BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, Redwood City, USA, 1999.
- [BSST07] Susanne Boll, Philipp Sandhaus, Ansgar Scherp, and Sabine Thieme. MetaXa - Context- and Content-Driven Metadata Enhancement for Personal Photo Books. In *Proc. of the International conference on MultiMedia Modeling (MMM)*, pages 332–343, 2007.
- [CG06] Dolores Costal and Cristina Gómez. On the Use of Association Redefinition in UML Class Diagrams. In *Proc. of the 25th International Conference on Conceptual Modeling (ER)*, pages 513–527, 2006.
- [CH03] Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. In *Proc. of the Workshop on Generative Techniques in the context of Model Driven Architecture (OOPSLA)*, October 2003.
- [CH06] K. Czarnecki and S. Helsen. Feature-based Survey of Model Transformation Approaches. *IBM Syst. J.*, 45(3):621–645, 2006.
- [Che75] Peter P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. In *Proc. of the International Conference on Very Large Data Bases (VLDB)*, 1975.
- [CI03] Kent K. T. Cheung and Horace H. S. Ip. Developing an Object-oriented Framework for Content-based Image Retrieval. *Softw. Pract. Exper.*, 33(6):523–565, 2003.
- [CMF96] Y. Chiaramella, P. Mulhem, and F. Fourel. A Model for Multimedia Information Retrieval. Technical report, 1996. FERMI ESPRIT BRA 8134.
- [CSY87] S. K. Chang, Q. Y. Shi, and C. W. Yan. Iconic Indexing by 2-D Strings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(3):413–428, 1987.
- [CT06] Gunter Saake Can Türker. *Objektrelationale Datenbanken. Ein Lehrbuch*. dpunkt.verlag GmbH, Heidelberg, 2006.
- [Czy05] Sebastian Czymaj. Konzeptuelle Datenmodellierung für die inhaltsbasierte Suche in Kollektionen von digitalen Bildern. Diplomarbeit, 2005. Universität Rostock, Institut für Informatik.
- [DB07] Jürgen Dunkel and Ralf Bruns. Model-Driven Architecture for Mobile Applications. In *Proc. of the 10th International Conference on Business Information Systems (BIS)*, pages 464–477, 2007.



- [DDL<sup>+</sup>90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [Dol04] Sebastian Dolke. Umsetzung von Modellen und Methoden bei der Integration spezieller Dokumentenserver in eine IBM Content Manager Umgebung. Diplomarbeit, 2004. Universität Rostock, Institut für Informatik.
- [DU04] Suzanne W Dietrich and Susan D. Urban. *An Advanced Course in Database Systems: Beyond Relational Databases*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.
- [EB03] Horst Eidenberger and Christian Breiteneder. VizIR - A Framework for Visual Information Retrieval. *J. Vis. Lang. Comput.*, 14(5):443–469, 2003.
- [FBF<sup>+</sup>94] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and Effective Querying by Image Content. *J. Intell. Inf. Syst.*, 3(3-4):231–262, 1994.
- [FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
- [FPR00] Marcus Fontoura, Wolfgang Pree, and Bernhard Rumpe. *The UML Profile for Framework Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
- [FPSS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 17:37–54, 1996.
- [FSJ99] Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson. *Building Application Frameworks: Object-oriented Foundations of Framework Design*. John Wiley & Sons, Inc., New York, NY, USA, 1999.
- [GCR06] Emanuel S. Grant, Rajani Chennamaneni, and Hassan Reza. Towards analyzing UML class diagram models to object-relational database systems transformations. In *Proc. of the 24th IASTED international conference on Database and applications (DBA)*, pages 129–134, 2006.
- [GGKH03] T. Gardner, C. Griffin, J. Koehler, and R. Hauser. A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard. In *Proc. of the Metamodelling for MDA Workshop*, York, November 2003.
- [Gha95] Arif Ghafoor. Multimedia Database Management Systems. *ACM Comput. Surv.*, 27(4):593–598, 1995.

- [GM93a] J. Griffioen, R. Mehrotra, and R. Yavatkar. A Semantic Data Model for Embedded Image Information. In *Proc. of the Second International Conference on Information and Knowledge Management*, pages 393–402, 1993.
- [GM93b] Jim Griffioen, Rajiv Mehrotra, and Rajendra Yavatkar. An Object-Oriented Model for Image Information Representation. In *Proc. of the second international conference on Information and knowledge management (CIKM)*, pages 393–402, 1993.
- [Göc03] Roland Göcke. Building a System for Writer Identification on Handwritten Music Scores. In *Proc. of the International Conference on Signal Processing, Pattern Recognition and Applications (IASTED)*, 2003.
- [Gor02] Davor Gornik. UML Data Modeling Profile, White Paper, 2002.
- [GR98] Martin Gogolla and Mark Richters. Equivalence Rules for UML Class Diagrams. In *The Unified Modeling Language, UML'98 - Beyond the Notation. First International Workshop, Mulhouse, France, June 1998*, pages 87–96, 1998.
- [GRV96] Venkat N. Gudivada, Vijay V. Raghavan, and Kanonluk Vanapipat. A Unified Approach to Data Modeling and Retrieval for a Class of Image Database Applications. *Multimedia database systems: issues and research directions*, pages 37–78, 1996.
- [GS00a] William I. Grosky and Peter L. Stanchev. An Image Data Model. In *Proc. of the 4th International Conference on Advances in Visual Information Systems (VISUAL)*, pages 14–25, 2000.
- [GS00b] Amarnath Gupta and Simone Santini. Toward Feature Algebras in Visual Databases: The Case for a Histogram Algebra. In *Proc. of the Fifth Working Conference on Visual Database Systems (VDB)*, page 177, 2000.
- [Gud93] V. N. Gudivada. A Unified Framework for Retrieval in Image Databases. PhD thesis, 1993. University of Southwestern Louisiana, Lafayette, LA, USA.
- [GVJH98] Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm. *Design Patterns CD: Elements of Reusable Object-Oriented Software*. Addison Wesley, Boston, USA, 1998.
- [GWJ91] Amarnath Gupta, Terry E. Weymouth, and Ramesh Jain. Semantic Queries with Pictures: The VIMSYS Model. In *Proc. of the 17th International Conference on Very Large Data Bases (VLDB)*, 1991.

- [GY00] Sahudy Montenegro González and Akebo Yamakami. A general purpose architecture for image retrieval in databases. In *Proc. of the 11th International Workshop on Database and Expert Systems Applications (DEXA)*, page 686, 2000.
- [Heu97] Andreas Heuer. *Objektorientierte Datenbanken: Konzepte, Modelle, Standards und Systeme*. Addison-Wesley, 1997.
- [HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 1st edition, 2000.
- [HKRR93] D. P. Huttenlocher, G. A. Klanderman, W. A. Rucklidge, and W. A. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):850–863, 1993.
- [HNP95] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized Search Trees for Database Systems. In *Proc. of 21th International Conference on Very Large Data Bases (VLDB)*, pages 562–573, Zurich, Switzerland, 1995.
- [HR03] Andreas Henrich and Günter Robbert. RSV-Transfer: An Algorithm for Similarity Queries on Structured Documents. In *Multimedia Information Systems*, pages 65–74, 2003.
- [Hul86] Richard Hull. Relative Information Capacity of Simple Relational Database Schemata. *SIAM J. Comput.*, 15(3):856–886, 1986.
- [HZdVB07] Sándor Héman, Marcin Zukowski, Arjen P. de Vries, and Peter A. Boncz. Efficient and Flexible Information Retrieval using MonetDB/X100. In *Proc. of the Third Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 96–101, 2007.
- [IB03] Temenushka Ignatova and Ilvio Bruder. Utilizing Relations in Multimedia Document Models for Multimedia Information Retrieval. In *Proc. of the Int. Conf. - Information, Communication Technologies, and Programming*, Varna, Bulgaria, 2003.
- [IB05] Temenushka Ignatova and Ilvio Bruder. Utilizing a Multimedia UML Framework for an Image Database Application. In *Proc. of the ER2005 Workshops: 1st International Workshop on Best Practices of UML (BP-UML 2005)*, 2005.
- [Ign03] Temenushka Ignatova. 2D-Gel Electrophoresis Image Database. In *Rostocker Informatik-Berichte (RIB-03)*, Rostock, 2003.
- [Ign06] Temenushka Ignatova. Model-Driven Development of Content-Based Image Retrieval Systems. In *Proc. of the 1st International Conference on Digital Information Management (ICDIM)*, pages 137–144, 2006.

- [IH08] Temenushka Ignatova and Andreas Heuer. Model-Driven Development of Content-Based Image Retrieval Systems. *Special issue on Digital Information Management of the International Journal of Digital Information Management (JDIM)*, 6(1), 2008.
- [IMa06] Website: IMatch. Website, 2006. Available online at <http://www.photools.com/>; visited on August 23rd 2007.
- [img06] imgSeek. Website, 2006. Available online at <http://www.imgseek.net/>; visited on August 23rd 2007.
- [ISO91] ISO/IEC. Software Product Evaluation—Quality Characteristics and Guidelines for Their Use. Technical Report 9126, ISO/IEC, 1991.
- [JMM95] H. V. Jagadish, Alberto O. Mendelzon, and Tova Milo. Similarity-Based Queries. In *Proc. of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 36–45, 1995.
- [Jon02] Jon Siegel. OMG’s Model Driven Architecture. *EURESCOM mess@ge*, 2, 2002.
- [JV03] M. Jones and P. Viola. Face Recognition Using Boosted Local Features, 2003.
- [JWY<sup>+</sup>06] Feng Jing, Changhu Wang, Yuhuan Yao, Kefeng Deng, Lei Zhang, and Wei-Ying Ma. IGroup: Web Image Search Results Clustering. In *Proc. of the 14th annual ACM International Conference on Multimedia*, pages 377–384, 2006.
- [KB94] Setrag Khoshfian and A. Brad Baker. *Multimedia and Imaging Databases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.
- [KD05] Harald Kosch and Mario Döllner. Multimedia Database Systems: Where are we now? In *Proc. of the IASTED International Conference on Databases and Applications*, Innsbruck, Austria, 2005.
- [KKYL04] Duck Hoon Kim, Duck Hoon Kim, Il Dong Yun, and Sang Uk Lee. A New Attributed Relational Graph Matching Algorithm Using the Nested Structure of Earth Mover’s Distance. In *Proc. of the Pattern Recognition, 17th International Conference on (ICPR’04)*, pages 48–51, 2004.
- [KNS04] Irwin King, Cheuk H. Ng, and Ka C. Sia. Distributed Content-based Visual Information Retrieval System on Peer-to-Peer Networks. *ACM Trans. Inf. Syst.*, 22(3):477–501, 2004.
- [Kos02] Harald Kosch. MPEG-7 and Multimedia Database Systems. *SIGMOD Rec.*, 31(2):34–39, 2002.

- [KPPS03] Hans-Peter Kriegel, Martin Pfeifle, Marco Pötke, and Thomas Seidl. The Paradigm of Relational Indexing: a Survey. In *Proc. of the German Database Conference "Datenbanksysteme fr Business, Technologie und Web" (BTW)*, pages 285–304, 2003.
- [KSW07] Komplexe Software Systeme Projekt - Image Database generator II. KSWs Projektbericht, 2007. Universität Rostock, Institut für Informatik.
- [Kuz05] Mikhail Kuznetsov. Automated Model Transformation in MDA. In *Proceedings of the Spring Young Researcher's Colloquium on Database and Information Systems SYRCoDIS, St.-Petersburg, Russia, 2005*, 2005.
- [KZB04] M. L. Kherfi, D. Ziou, and A. Bernardi. Image Retrieval from the World Wide Web: Issues, Techniques, and Systems. *ACM Comput. Surv.*, 36(1):35–67, 2004.
- [LC03] Anthony J. T. Lee and Han-Pang Chiu. 2D Z-string: A New Spatial Knowledge Representation for Image Databases. *Pattern Recogn. Lett.*, 24(16):3015–3026, 2003.
- [Lew01] Michael S. Lew, editor. *Principles of Visual Information Retrieval*. Springer-Verlag, London, UK, 2001.
- [Li05] Xin Li. Negotiating the Semantic Gap in an MPEG-7 Aerial Image Database. Dissertation, 2005. Wayne State University, USA.
- [LKO02] J. Laaksonen, M. Koselka, and E. Oja. PicSOM - Self-Organising Image Retrieval with MPEG-7 Content Descriptions. *IEEE Transactions on Neural Networks, Special Issue on Intelligent Multimedia Processing 13*, 13(4):841–853, 2002.
- [LPS<sup>+</sup>04] Thomas M. Lehmann, Bartosz Plodowski, Klaus Spitzer, Berthold B. Wein, Hermann Ney, and Thomas Seidl. Extended Query Refinement for content-Based Access to Large Medical Image Databases. In *Proc. of SPIE Symposium on Medical Imaging: PACS and imaging informatics*, volume 5371, pages 90–98, 2004.
- [LSDJ06] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based Multimedia Information Retrieval: State of the Art and Challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, 2006.
- [LSS94] Odd Ivar Lindland, Guttorm Sindre, and Arne Sølvsberg. Understanding Quality in Conceptual Modeling. *IEEE Softw.*, 11(2):42–49, 1994.
- [LZLM07] Ying Liu, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. A Survey of Content-based Image Retrieval with High-level Semantics. *Pattern Recogn.*, 40(1):262–282, 2007.

- [MA07] Parastoo Mohagheghi and Jan Aagedal. Evaluating Quality in Model-Driven Engineering. In *Proc. of the International Workshop on Modeling in Software Engineering (MISE)*, page 6, 2007.
- [Mas05] Henning Masuch. Entwurf und Implementierung einer objektrelationalen Datenbankweiterung für die automatische Schreiberklassifikation in historischen Notenhandschriften. Projektarbeit, 2005. Universität Rostock, Institut für Informatik.
- [MBC95] M. Mechkour, C. Berrut, and Y. Chiaramella. Using Conceptual Graph Framework for Image Retrieval. In *Proc. of the International conference on MultiMedia Modeling (MMM)*, pages 127–142, 1995.
- [MC07] Jennifer Munnely and Siobhán Clarke. ALPH: A Domain-specific Language for Crosscutting Pervasive Healthcare Concerns. In *Proc. of the 2nd workshop on Domain specific aspect languages (DSAL)*, page 4, 2007.
- [MD07] P. Mohagheghi and V. Dehlen. An Overview of Quality Frameworks in Model-Driven Engineering and Observations on Transformation Quality. In *Proc. of the 2nd Workshop on Quality in Modeling*, 2007.
- [Mec95a] Mourad Mechkour. EMIR2: An Extended Model for Image Representation and Retrieval. In *Proc. of the 6th International Conference on Database and Expert Systems Applications (DEXA)*, pages 395–404, 1995.
- [Mec95b] Mourad Mechkour. EMIR2: An Extended Model for Image Representation and Retrieval. In *Proc. of the 6th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 395–404, 1995.
- [Mel02] Jim Melton. *Advanced SQL 1999: Understanding Object-Relational, and Other Advanced Features*. Elsevier Science Inc., New York, NY, USA, 2002.
- [MF02] Oge Marques and Borko Furht. Content-based Visual Information Retrieval. *Distributed multimedia databases: techniques & applications*, pages 37–57, 2002.
- [MG06] Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. In *Electronic Notes in Theoretical Computer Science, Volume 152, Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005)*, pages 125–142, 2006.
- [MGP03] F. Monay and D. Gatica-Perez. On Image Auto-Annotation with Latent Space Models. In *Proc. of the ACM International Conference on Multimedia (ACM MM)*, 2003.

- [Mil04] Lars Milewski. Integration von Clustering/Classification-Techniken in eine objektrelationale Datenbankumgebung. Diplomarbeit, 2004. Universität Rostock, Institut für Informatik.
- [MIR93] Renée J. Miller, Yannis E. Ioannidis, and Raghu Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *Proc. of the 19th International Conference on Very Large Data Bases (VLDB)*, pages 120–133, 1993.
- [Mon05] MonetDB. Website, 2005. Available online at <http://monetdb.cwi.nl/projects/monetdb//Home/>; visited on August 23rd 2007.
- [Moo98] Daniel L. Moody. Metrics for Evaluating the Quality of Entity Relationship Models. In *Proc. of the 17th International Conference on Conceptual Modeling (ER)*, pages 211–225, 1998.
- [MP01] W. Mok and D. Paper. On Transformations from UML Models to Object-Relational Databases. In *Proc. of the 34th Annual Hawaii International Conference on System Sciences (HICSS)*, page 3046, 2001.
- [MS93] S. Marcus and V. Subrahmanian. Multimedia Database Systems, 1993.
- [MS96] Sherry Marcus and V. S. Subrahmanian. Foundations of Multimedia Database Systems. *J. ACM*, 43(3):474–523, 1996.
- [MSBS03] Daniel L. Moody, Guttorm Sindre, Terje Brasethvik, and Arne Sølvberg. Evaluating the quality of information models: empirical testing of a conceptual model quality framework. In *Proc. of the 25th International Conference on Software Engineering (ICSE)*, pages 295–305, 2003.
- [MSMP99] Henning Müller, David McG. Squire, Wolfgang Müller, and Thierry Pun. Efficient Access Methods for Content-Based Image Retrieval With Inverted Files. In *Proc. of the Multimedia Storage and Archiving Systems IV (VV02)*, 1999.
- [MSS97] Carlo Meghini, Fabrizio Sebastiani, and Umberto Straccia. The Terminological Image Retrieval Model. In *Proc. of the 9th International Conference on Image Analysis and Processing (ICIAP)*, pages 156–163, 1997.
- [MSS01] Carlo Meghini, Fabrizio Sebastiani, and Umberto Straccia. A model of Multimedia Information Retrieval. *J. ACM*, 48(5):909–970, 2001.
- [Mul99] Robert J. Muller. *Database Design for Smarties: Using UML for Data Modeling*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

- [MVC04] Esperanza Marcos, Belén Vela, and José María Cavero. A Methodological Approach for Object-Relational Database Design Using UML. *Inform., Forsch. & Entwickl.*, 18(3-4):152–164, 2004.
- [MW03] Klaus Meyer-Wegener. *Multimediale Datenbanken : Einsatz von Datenbanktechnik in Multimedia-Systemen*. Leitfäden der Informatik. B.G. Teubner, Wiesbaden, 2. überarb. und erw. edition, 2003.
- [Nes00] Niels Nes. Image Database Management System Design Considerations, Algorithms and Architecture. PhD thesis, 2000.
- [NFG06] Martin Nussbaumer, Patrick Freudenstein, and Martin Gaedke. Web Application Development Employing Domain-Specific Languages. In *IASTED Conf. on Software Engineering*, pages 13–18, 2006.
- [NMH03] Munehiro Nakazato, Ljubomir Manola, and Thomas S. Huang. ImageGrouper: A Group-oriented User Interface for Content-based Image Retrieval and Digital Image Arrangement. *J. Vis. Lang. Comput.*, 14(4):363–386, 2003.
- [OAF<sup>+</sup>04] Toacy C. Oliveira, Paulo S.C. Alencar, Ivan M. Filho, Carlos J.P. de Lucena, and Donald D. Cowan. Software Process Representation and Analysis for Framework Instantiation. *IEEE Transactions on Software Engineering*, 30(3):145–159, 2004.
- [OMG03] OMG. MDA Guide Version 1.0.1, 2003.
- [OMG05] OMG. MOF QVT Final Adopted Specification, OMG Adopted Specification ptc/05-11-01, 2005.
- [OÖ03] Vincent Oria and M. Tamer Özsu. Views or Points of View on Images. *Int. J. Image Graphics*, 3(1):55–80, 2003.
- [OÖL<sup>+</sup>97] V. Oria, M. Özsu, X. Li, L. Liu, J. Li, Y. Niu, and P. Iglinski. Modeling Images for Content-based Queries: The DISIMA Approach. In *Proc. of 2nd DEFINING VIEWS IN AN IMAGE DATABASE SYSTEM 19 International Conference of Visual Information Systems*, pages 339–346, 1997.
- [PF95] Ulrich Pfeifer and Norbert Fuhr. Efficient Processing of Vague Queries Using a Data Stream Approach. In *Proc. of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 189–197, 1995.
- [PF97] Euripides G. M. Petrakis and Christos Faloutsos. Similarity Searching in Medical Image Databases. *IEEE Trans. on Knowl. and Data Eng.*, 9(3):435–447, 1997.



- [PFR02] Wolfgang Pree, Marcus Fontoura, and Bernhard Rumpe. Product Line Annotations with UML-F. In *Proc. of the Second International Conference on Software Product Lines (SPLC)*, pages 188–197, 2002.
- [PM06] Roland Petrasch and Oliver Meimberg. *Model Driven Architecture Eine praxisorientierte Einführung in die MDA*. dpunkt.verlag, 2006.
- [Pre95] Wolfgang Pree. *Design Patterns for Object-oriented Software Development*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.
- [Rat03] The UML and Data Modeling, White Paper, 2003.
- [RB01] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, 2001.
- [RBPR06] Oscar David Robles, José Luis Bosque, Luis Pastor, and Angel Rodríguez. CBIR on Grids. In *OTM Conferences (2)*, pages 1412–1421, 2006.
- [RHC99] Yong Rui, Thomas S. Huang, and Shih-Fu Chang. Image Retrieval: Current Techniques, Promising Directions, and Open Issues. *Journal of Visual Communication and Image Representation*, 10(1):39–62, March 1999.
- [SB00] John R. Smith and Ana B. Benitez. Conceptual Modeling of Audio-Visual Content. In *Proc. of the IEEE International Conference on Multimedia and Expo (ICME)*, New York, NY, July 2000.
- [Sch04] Ingo Schmitt. *Multimedia-Datenbanken: Retrieval, Suchalgorithmen und Anfragebearbeitung, Habilitationsschrift*. Fakultät für Informatik, Otto-von-Guericke-Universität, Magdeburg, 2004.
- [Sch07] Andre Schefe. Entwurf und Integration eines Plugins zur Anpassung und Erzeugen eines Bilddatenbank-Generatos in eine Rich-Client-Anwendung zur Modellgetriebene Entwicklung von Bilddatenbanken. Studienarbeit, 2007. Universität Rostock, Institut für Informatik.
- [SG01] Simone Santini and Amarnath Gupta. A Wavelet Data Model For Image Databases. In *Proc. of the IEEE International Conference on Multimedia and Expo (ICME)*, 2001.
- [SG02a] S. Santini and A. Gupta. An Extensible Feature Management Engine for Image Retrieval. In *Proc. of SPIE Storage and Retrieval for Media Databases*, volume 4676, San Jose, 2002.

- [SG02b] Simone Santini and Amarnath Gupta. An Extensible Feature Management Engine for Image Retrieval. In *Proc. of SPIE: Storage and Retrieval for Media Databases*, volume 4676, pages 86–97, 2002.
- [Söl07a] Gunnar Söllig. Integration von nutzerdefinierten hochdimensionalen Indexstrukturen in Oracle für das inhaltsbasierte Retrieval digitaler Bilder. Diplomarbeit, 2007. Universität Rostock, Institut für Informatik.
- [Söl07b] Gunnar Söllig. Multidimensionale indexierung in ordbms. In *BTW Studierendenprogramm*, pages 37–38, 2007.
- [Spi06] Ralph Spickermann. Konzeption eines Multimedia Dokumenten Repositories auf Basis von DB2, JSR-170 und einem MM Datenmodell. Diplomarbeit, 2006. Universität Rostock, Institut für Informatik.
- [SQL00] SQL Multimedia and Application Packages - Part 6: Data Mining. Working Draft, 2000. ISO/IEC.
- [SR96] U. Shaft and R. Ramakrishnan. Data Modeling and Feature Extraction for Image Databases. In *Proc. of SPIE Multimedia Storage and Archiving Systems*, volume 2916, pages 90–102, 1996.
- [SS02] Phillipe Salembier and Thomas Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
- [SSH05] Ingo Schmitt, Nadine Schulz, and Thomas Herstel. WS-QBE: A QBE-Like Query Language for Complex Multimedia Queries. In *Proc. of the International conference on MultiMedia Modeling (MMM)*, pages 222–229, 2005.
- [ST93] Klaus-Dieter Schewe and Bernhard Thalheim. Fundamental Concepts of Object Oriented Databases. *Acta Cybern.*, 11(1-2):49–84, 1993.
- [Sto01] Knut Stolze. SQL/MM Part 5: Still Image - The Standard and Implementation Aspects. In *BTW*, pages 345–363, 2001.
- [Sto02] Knut Stolze. Still Image Extensions in Database Systems - A Product Overview. *Datenbank-Spektrum*, 2(20):40–47, 2002.
- [Sto05] Knut Stolze. A DB2 UDB still image extender, IBM developerWorks. Website, 2005. Available online at <http://www.ibm.com/developerworks/db2/library/techarticle/dm-0504stolze/>; visited on August 23rd 2007.
- [Sub98] V. S. Subrahmanian. *Principles of Multimedia Database Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

- [SVEH07] Thomas Stahl, Markus Völter, Sven Efftige, and Arno Haase. *Modellgetriebene Softwareentwicklung*. Dpunkt Verlag, 2007.
- [SWS<sup>+</sup>00] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-Based Image Retrieval at the End of the Early Years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380, 2000.
- [TB04] Avinash Tiwari and Veena Bansal. PATSEEK: Content Based Image Retrieval System for Patent Database. In *Proc. of the International Conference on Electronic Business ICEB*, pages 1167–1171, 2004.
- [TCLP04] R. Troncy, J. Carrive, S. Lalande, and J-P. Poli. A Motivating Scenario for Designing an Extensible Audio-visual Description Language. In *Proc. of the International Workshop on Multidisciplinary Image, Video, and Audio Retrieval and Mining (CORIMEDIA)*, Sherbrooke, Canada, Octobre 2004.
- [Thi08] Torsten Thierbach. Anforderungsanalyse und Vergleich von Software-Frameworks für Content-Based Image Retrieval. Studienarbeit, 2008. Universität Rostock, Institut für Informatik.
- [TSMR03] Ricardo S. Torres, Celmar G. Silva, Claudia B. Medeiros, and Heloisa V. Rocha. Visual Structures for Image Browsing. In *Proc. of the twelfth International Conference on Information and Knowledge Management CIKM*, pages 49–55, 2003.
- [Tür03] Can Türker. *SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML*. dpunkt-Verlag, Heidelberg, 2003.
- [TXRN06] Kazem Taghva, Min Xu, Emma Regentova, and Tom Nartker. Utilizing XML Schema for Describing and Querying Still Image Databases. In *ITNG '06: Proceedings of the Third International Conference on Information Technology: New Generations*, pages 695–700, 2006.
- [UML07] Unified Modeling Language (UML), Version 2.1.2, OMG. Website, 2007. Available online at <http://www.omg.org/spec/UML/2.1.2/>; visited on Mai 6th 2008.
- [Urb87] Susan Darling Urban. Constraint Analysis for the Design of Semantic Database Update Operations. PhD thesis, 1987. University of Southwestern Louisiana, Lafayette, LA, USA.
- [VC02] Lawrence D. Bergman Vittorio Castelli. *Image Databases Search and Retrieval of Digital Imagery*. John Wiley & Sons, Inc., New York, 2002.
- [vdBKV04] Egon L. van den Broek, Peter M. F. Kisters, and Louis G. Vuurpijl. Design Guidelines for a Content-Based Image Retrieval Color-Selection Interface. In *Proc. of the Conference on Dutch Directions in HCI*, page 14, 2004.

- [Vil06] Antti Viljamaa. Specifying Reuse Interfaces for Task-Oriented Framework Specialization. PhD thesis, 2006. University of Helsinki, Finland.
- [Viz06] Website: VizIR project webserver. Website, 2006. Available online at <http://cbvr.ims.tuwien.ac.at>; visited on August 23rd 2007.
- [VS06] Markus Völter and Thomas Stahl. *Model-Driven Software Development*. Wiley, 2006.
- [VT02] R. Veltkamp and M. Tanase. Content-Based Image Retrieval Systems: A Survey, October 2002.
- [VVC07] Juan M. Vara, Belen Vela, Jose Maria Cavero, and Esperanza Marcos. Model Transformation for Object-relational Database Development. In *Proc. of the 2007 ACM Symposium on Applied Computing (SAC)*, pages 1012–1019, 2007.
- [Wes04] Gerd Utz Westermann. A Persistent Typed Document Object Model for the Management of MPEG-7 Media Descriptions. Dissertation, 2004. Techn. Universität Wien, Austria.
- [WFDR05] Andreas Wolff, Peter Forbrig, Anke Dittmar, and Daniel Reichart. Linking GUI Elements to Tasks: Supporting an Evolutionary Design Process. In *Proc. of the 4th international workshop on Task models and diagrams (TAMODIA)*, pages 27–34, 2005.
- [WHKL00] Jian-Kang Kang Wu, Dezhong Hong, Mohan S. Kankanhalli, and Joo-Hwee Lim. *Perspectives on Content-Based Multimedia Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.
- [WK03] Utz Westermann and Wolfgang Klas. An Analysis of XML Database Solutions for the Management of MPEG-7 Media Descriptions. *ACM Comput. Surv.*, 35(4):331–373, 2003.
- [YTD04] Tom Yeh, Konrad Tollmar, and Trevor Darrell. Searching the Web with Mobile Images for Location Recognition. *Computer Vision and Pattern Recognition (CVPR)*, 02:76–81, 2004.
- [ZG02] R. Zhao and W. Grosky. Narrowing the Semantic Gap - Improved Text-Based Web Document Retrieval Using Visual Features. *IEEE Trans. on Multim.*, 4(2):189–200, 2002.



# List of Figures

1.1	MDS techniques for the development of CBIRSs . . . . .	8
2.1	Levels of abstraction of visual content . . . . .	12
2.2	An integrated view of a CBIR system . . . . .	14
2.3	Basic structure of OMG's Model Driven Architecture (based on [Jon02]) . . . . .	23
2.4	Meta levels in MDA (based on [SVEH07]) . . . . .	24
2.5	UML PIM and PSM meta models (based on [SVEH07]) . . . . .	24
2.6	Transforming PIM to PSM (based on [PM06]) . . . . .	26
3.1	MDS techniques for the development of CBIRSs - Modeling . . . . .	32
3.2	Core CBIR components to be modeled by the CBIRS Generator . . . . .	33
3.3	A Region of Interest (ROI) from a music manuscript image and MBRs of the detected music objects . . . . .	36
3.4	Example queries for the six query types and possible results from the Corel image database (from [SWS <sup>+</sup> 00]) . . . . .	39
3.5	AIR Data Model (based on [GRV96]) . . . . .	44
3.6	Layers of the VIMSYS image model (based on [GWJ91]) . . . . .	46
3.7	Example of describing an image with the EMIR <sup>2</sup> model (from [Mec95a]) . . . . .	48
3.8	Image description example in EMIR <sup>2</sup> -CG (from [MBC95]) . . . . .	49
3.9	Example of a simple schema and an image (from [SR96]) . . . . .	52
3.10	A summary tree of the example image from Figure 3.9(from [SR96]) . . . . .	53
3.11	Example PIQ schema for music manuscripts . . . . .	53
3.12	The DISIMA image model (from [OÖL <sup>+</sup> 97]) . . . . .	55
3.13	Image and Salient Object type classification for music manuscripts . . . . .	55
3.14	MOODS image information processing system (based on [GMY93b]) . . . . .	57
3.15	MOODS semantic model for music manuscript recognition (from [GMY93a]) . . . . .	58
3.16	UML representation of the image description scheme (from [BPC <sup>+</sup> 00]) . . . . .	61

3.17	MDSO techniques for the development of CBIRs - Mapping . . . . .	68
3.18	Transforming a CBIR System PIM to a Database Schema (based on [PM06]) . . . . .	69
3.19	Client/Server architectures for information systems (based on [ACKM04]) . . . . .	71
3.20	Meta model based transformation (from [PM06]) . . . . .	79
3.21	Transformation levels in the database design . . . . .	80
3.22	Transformation rule requirements: not-mappable concepts are not allowed in the PIM . . . . .	80
3.23	Transformation rule requirements: only one of multiple mapping possibilities should be applied in a transformation . . . . .	81
3.24	Transformation rule requirements: two different PIM concepts should not be mapped to the same PSM concept . . . . .	81
3.25	Transformation levels in MDSO . . . . .	82
4.1	Modeling approach . . . . .	85
4.2	Generic Image Database Model (from [IB05]) . . . . .	87
4.3	Main framework classes and application specific <i>black box</i> classes of GiACoMo-IRS . . . . .	89
4.4	Association redefinition example . . . . .	91
4.5	Modeling Attributed Relational Graphs Image Representations . . . . .	92
4.6	Modeling 2D-Strings Image Representation . . . . .	92
4.7	Use Cases for the Insert operation . . . . .	94
4.8	Activity diagram for the Insert operation . . . . .	94
4.9	Example of a separation pattern for the template and hook methods (from [FPR00]) . . . . .	95
4.10	Integration of the insert functionality in the framework model . . . . .	95
4.11	Query tasks supported by the framework model . . . . .	97
4.12	Retrieval Model Design . . . . .	98
4.13	Class diagram of the “image insertion” and “query by local features” classes and methods . . . . .	100
4.14	Phases of the KDD process (from [FPSS96]) . . . . .	100
4.15	Phases of the classification process (based on [SQL00]) . . . . .	101
4.16	Generic classes for the classification of images . . . . .	102
4.17	Modeling functionality with the GiACoMo-IRS framework . . . . .	103
5.1	Example for the deployment annotation . . . . .	106
5.2	UML meta model part for Profiles (from [UML07]) . . . . .	108

---

5.3	Layers of UML-F tag sets (from [PFR02]) . . . . .	109
5.4	Mapping problems with inheritance hierarchies . . . . .	121
5.5	Metamodel mappings: direct mappings . . . . .	124
5.6	Metamodel mappings: not-directly-mappable PIM metamodel concepts . . .	124
5.7	Metamodel mappings: multiple mapping possibilities . . . . .	125
5.8	Metamodel mappings: mappings resulting in the same PSM metamodel concept	125
5.9	Equivalence rule for association class (from [GR98]) . . . . .	126
5.10	Equivalence rule for association qualifier (from [GR98]) . . . . .	126
5.11	Example of a Trigger for a Derived Attribute . . . . .	128
5.12	Example of a Trigger for Redefined Attributes. . . . .	128
5.13	Example of a Trigger for a Property Constraint . . . . .	128
5.14	Examples of Triggers for consistency and integrity constraints in mapping of Associations . . . . .	129
5.15	Metamodel mappings: implementation specific concepts . . . . .	130
5.16	Architectures of the Image Database Generator and the resulting CBIRS . .	131
6.1	Examples of music manuscripts written by different scribes . . . . .	135
6.2	Excerpt from the feature dictionary . . . . .	136
6.3	Conceptual data model of the digital archive for music manuscripts (from [Dol04])	138
6.4	Simplified representation of the data type for storing the automatically ex- tracted features (from [Mas05]) . . . . .	140
6.5	Simplified representation of the data type used for the classification of scribes (from [Mas05]) . . . . .	140
6.6	eNoteHistory CBIR PIM . . . . .	142
6.7	eNoteHistory CBIR PIM - data mining part . . . . .	143
6.8	2D-Gel electrophoresis image . . . . .	144
6.9	Results from the feature extraction algorithm . . . . .	145
6.10	CBIRS PIM for 2D-gel electrophoresis images . . . . .	147
6.11	CBIRS PIM for a photo annotation application . . . . .	150
6.12	CBIRS PIM for a photo annotation application - data mining part . . . . .	152
6.13	Mapping of methods onto user-defined functions . . . . .	153
C.1	MOODS image information processing system (based on [GMY93b]) . . . . .	236
C.2	Generic Image Database Model (from [IB05]) . . . . .	237



C.3	Main framework classes and application specific <i>black box</i> classes of GiACoMo-IRS . . . . .	238
C.4	Modeling Attribute Relational Graphs Image Representations . . . . .	239
C.5	Modeling 2D-Strings Image Representation . . . . .	240
C.6	Use Cases for the Insert operation . . . . .	241
C.7	Activity diagram for the Insert operation . . . . .	242
C.8	Integration of the insert functionality in the framework model . . . . .	243
C.9	Modeling functionality with the GiACoMo-IRS framework . . . . .	244
C.10	Class diagram of the “image insertion” and “query by local features” classes and methods . . . . .	245
C.11	Generic classes for the classification of images . . . . .	246
C.12	Example for the deployment annotation . . . . .	247
C.13	eNoteHistory CBIR PIM . . . . .	248
C.14	eNoteHistory CBIR PIM - data mining part . . . . .	249
C.15	CBIRS PIM for 2D-gel electrophoresis images . . . . .	250
C.16	CBIRS PIM for a photo annotation application . . . . .	251
C.17	CBIRS PIM for a photo annotation application - data mining part . . . . .	252
D.1	Project Wizard for creating an IDBG project . . . . .	253
D.2	Modeling environment and Generation Wizards . . . . .	254
D.3	Transformation options dialog . . . . .	255

# List of Tables

3.1	Evaluation of the Image Retrieval Models . . . . .	66
3.2	UML Extensions for modeling DBMSs by Rational (based on [Rat03, Gor02])	73
3.3	UML Extensions for modeling DBMSs by Scott Ambler (based on [Amb03]) .	74
3.4	UML Extensions for modeling ORDBMSs by Marcos et al. (based on [VVCM07])	75
3.5	Mapping of UML PIM onto OR PSM by Marcos et al. (based on [VVCM07])	77
5.1	Mapping of «CBIRSSimpleData» derivatives to SQL:2003 . . . . .	122
A.1	UML Metamodel for Structural Diagrams . . . . .	188
A.2	UML Metamodel Extensions for CBIRS . . . . .	205
B.1	Relational Concepts in UML . . . . .	208
B.2	Object-relational Concepts in UML . . . . .	221
B.3	SQL Data Types in UML . . . . .	229



# List of Abbreviations

API .....	Application Programming Interface
ARG .....	Attributed Relational Graph
CBIRS .....	Content-Based Image Retrieval System
CDBMS .....	Conventional Database Management System
DBMS .....	Database Management System
DDL .....	Data Description Language
EMOF .....	Essential Meta Object Facility
ERM .....	Entity-Relationship Model
EXIF .....	Exchangeable Image File Format
GiACoMo-IRS ...	Generic and Adaptable Conceptual Model for Image Retrieval Systems
GPS .....	Global Positioning System
HTML .....	HyperText Markup Language
HyTime .....	Hypermedia/Time-based Structuring Language
IDBG .....	Image Database Generator
IDE .....	Integrated Development Environment
IRS .....	Image Retrieval System
KDD .....	Knowledge Discovery in Databases
LHS .....	Left Hand Side
MBE .....	Minimum Bounding Ellipse
MBR .....	Minimum Bounding Rectangle
MDA .....	Model-Driven Architecture
MDSD .....	Model-Driven Software Development
MHEG .....	Multimedia and Hypermedia information coding Expert Group
MMDBMS .....	Multimedia Database Management System
MOF .....	Meta Object Facility
MPEG .....	Moving Picture Experts Group
OCL .....	Object Constraint Language
OID .....	Object Identifier
OMDL .....	Object Modeling Description Language
OMG .....	Object Management Group

OODBMS	.....	Object-Oriented Database Management System
ORDBMS	.....	Object-Relational Database Management System
PIM	.....	Platform Independent Model
PM	.....	Platform Model
PSM	.....	Platform Specific Model
QVT	.....	Query/Views/Transformations
RHS	.....	Right Hand Side
ROI	.....	Region Of Interest
SMIL	.....	Synchronized Multimedia Integration Language
SP	.....	Stored Procedure
SQL	.....	Structured Query Language
UDF	.....	User-Defined Function
UDT	.....	User-Defined Type
UML	.....	Unified Modeling Language
XMI	.....	MXL Metadata Exchange
XML	.....	Extensible Markup Language
XSLT	.....	Extensible Stylesheet Language Transformation

# Appendix



# Appendix A

## PIM Metamodel

In the first part of this appendix the UML concepts used for modeling structural diagrams are listed. The source used for this summary is the UML Standard Superstructure documentation version 2.1.1 formal/2007-02-03. The second part describes the UML extensions used for modeling the GiACoMo-IRS framework-model. And finally, the new data types, needed to model CBIR applications are listed.



Table A.1: UML Metamodel for Structural Diagrams

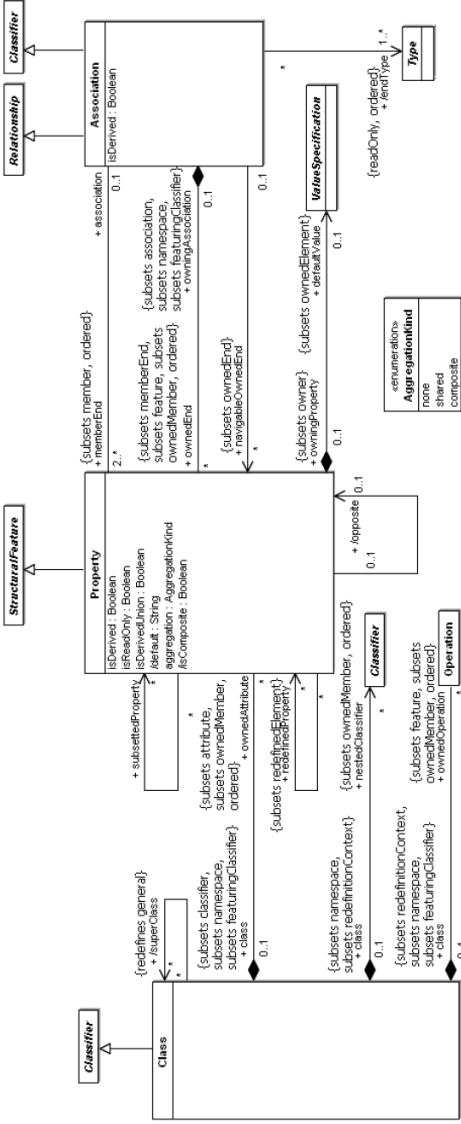
UML Concept	Description, Notation, Metamodel								
Class	<p><b>Description:</b>            A <b>Class</b> describes a set of objects that share the same specifications of features, constraints, and semantics. It is a kind of <b>Classifier</b> whose features are <b>Properties</b> and <b>Operations</b>. These features can be declared as private, protected or public. A <b>Class</b> cannot access private features of another <b>Class</b>, or protected features of another <b>Class</b> that is not its super type. A <b>Class</b> may be declared as abstract. In this case it cannot have any instances. The <b>Property</b> is Abstract: Boolean inherited from <b>Classifier</b> indicates that.</p> <p><b>Notation:</b></p> <table border="1" data-bbox="587 864 791 1077"> <thead> <tr> <th>Window</th> </tr> </thead> <tbody> <tr> <td>+ size: Area = (100, 100)</td> </tr> <tr> <td># visibility: Boolean = true</td> </tr> <tr> <td>+ defaultSize: Rectangle</td> </tr> <tr> <td>- xWin: XWindow</td> </tr> <tr> <td>display()</td> </tr> <tr> <td>hide()</td> </tr> <tr> <td>- attachX(xWin: XWindow)</td> </tr> </tbody> </table> <p><b>Metamodel:</b></p>  <pre> classDiagram     class Classifier     class Class     class Property     class Association     class Relationship     class ValueSpecification     class Type     class AggregationKind     class Operation      Classifier &lt; -- Class     Classifier &lt; -- Association     Classifier &lt; -- Relationship     Classifier &lt; -- Property     Classifier &lt; -- ValueSpecification     Classifier &lt; -- Type     Classifier &lt; -- Operation      Class "1" *-- "*" Association : {subjects namespace, subjects featuringClassifier, + class}     Class "1" *-- "*" Property : {subjects namespace, subjects definitionContext, + class}     Class "1" *-- "*" Operation : {subjects namespace, subjects featuringClassifier, + class}     Class "1" *-- "*" ValueSpecification : {subjects namespace, subjects definitionContext, + class}     Class "1" *-- "*" Type : {subjects namespace, subjects definitionContext, + class}     Class "1" *-- "*" Operation : {subjects namespace, subjects featuringClassifier, + class}      Association "2..*" -- "0..1" Property : + association     Association "0..1" -- "0..1" ValueSpecification : + memberEnd     Association "0..1" -- "0..1" Type : + memberEnd     Association "0..1" -- "0..1" Operation : + memberEnd      Property "0..1" -- "0..1" ValueSpecification : + owningProperty     Property "0..1" -- "0..1" Type : + owningProperty     Property "0..1" -- "0..1" Operation : + owningProperty      ValueSpecification "0..1" -- "0..1" Type : + defaultValue     ValueSpecification "0..1" -- "0..1" Operation : + defaultValue      Type "1..*" -- "1..*" Operation : + valueType      AggregationKind &lt; -- none     AggregationKind &lt; -- shared     AggregationKind &lt; -- composite     </pre>	Window	+ size: Area = (100, 100)	# visibility: Boolean = true	+ defaultSize: Rectangle	- xWin: XWindow	display()	hide()	- attachX(xWin: XWindow)
Window									
+ size: Area = (100, 100)									
# visibility: Boolean = true									
+ defaultSize: Rectangle									
- xWin: XWindow									
display()									
hide()									
- attachX(xWin: XWindow)									

Table A.1: UML Metamodel for Structural Diagrams

UML Concept	Description, Notation, Metamodel
Property	<p data-bbox="290 300 446 1648"><b>Description:</b> A <b>Property</b> is a structural feature of a <b>Class</b>. A <b>Property</b> related to a <b>Class</b> by the association ownedAttribute represents an attribute. It relates an instance of the <b>Class</b> to a value or collection of values of the type of the attribute. A <b>Property</b> related to an <b>Association</b> by the association memberEnd or its specializations represents an end of the association. The type of <b>Property</b> is the type of the end of the <b>Association</b>.</p> <p data-bbox="470 300 853 1648"><b>Notation:</b>  <code>&lt;property&gt; ::= [ &lt;visibility&gt; ] [ '/' ] &lt;name&gt; [ ':' &lt;prop-type&gt; ] [ '[' &lt;multiplicity&gt; ']' ] [ '=' &lt;default&gt; ] [ '{' &lt;prop-modifier&gt; [ '*' ] '}' ]</code>  where:  - &lt;visibility&gt; is the visibility of the <b>Property</b>. (VisibilityKind is an Enumeration with the following values: public, private, protected, package)  - &lt;visibility&gt; ::= '+'   '-'   '#'  - '/' signifies that the <b>Property</b> is derived.  - &lt;name&gt; is the name of the <b>Property</b>.  - &lt;prop-type&gt; is the name of the <b>Classifier</b> that is the type of the <b>Property</b>.  - &lt;multiplicity&gt; is the multiplicity of the <b>Property</b>. If this term is omitted, it implies a multiplicity of 1.  - &lt;default&gt; is an expression that evaluates to the default value or values of the <b>Property</b>.  - &lt;prop-modifier&gt; indicates a modifier that applies to the <b>Property</b>.  <code>&lt;prop-modifier&gt; ::= 'readOnly'   'union'   'subsets' &lt;property-name&gt;   'redefines' &lt;property-name&gt;   'ordered'   'unique'   'nonunique'   &lt;prop-constraint&gt;</code>  where:  * <code>readOnly</code> means that the <b>Property</b> is read only.  * <code>union</code> means that the <b>Property</b> is a derived union of its subsets.  * <code>subsets &lt;property-name&gt;</code> means that the <b>Property</b> is a proper subset of the <b>Property</b> identified by <code>&lt;property-name&gt;</code>.  * <code>redefines &lt;property-name&gt;</code> means that the <b>Property</b> redefines an inherited <b>Property</b> identified by <code>&lt;property-name&gt;</code>.  * <code>ordered</code> means that the <b>Property</b> is ordered.  * <code>unique</code> means that there are no duplicates in a multi-valued <b>Property</b>.  * <code>&lt;prop-constraint&gt;</code> is an expression that specifies a <b>Constraint</b> that applies to the <b>Property</b>.</p> <p data-bbox="877 300 930 1648"><b>Metamodel:</b> see <b>Class</b></p>

Table A.1: UML Metamodel for Structural Diagrams

UML Concept	Description, Notation, Metamodel
<p><b>Operation</b></p>	<p><b>Description:</b>  An <b>Operation</b> is a behavioral feature of a <b>Classifier</b> that specifies the name, type, parameters, and constraints for invoking an associated behavior.</p> <p><b>Notation:</b>  <code>&lt;operation&gt; ::= [&lt;visibility&gt;] &lt;name&gt; '(' [&lt;parameter-list&gt;] ':' [&lt;return-type&gt;] [{&lt;oper-property&gt;   '*' } ] ]</code>  where:  - <code>&lt;visibility&gt;</code> is the visibility of the <b>Operation</b>. <code>&lt;visibility&gt; ::= '+'   '-'   '#'</code>  - <code>&lt;name&gt;</code> is the name of the <b>Operation</b>.  - <code>&lt;return-type&gt;</code> is the type of the return result parameter if the <b>Operation</b> has one defined.  - <code>&lt;oper-property&gt;</code> indicates the properties of the <b>Operation</b>.  <code>&lt;oper-property&gt; ::= 'redefines' &lt;oper-name&gt;   'query'   'ordered'   'unique'   &lt;oper-constraint&gt;</code>  where:  * <code>redefines &lt;oper-name&gt;</code> means that the <b>Operation</b> redefines an inherited <b>Operation</b> identified by <code>&lt;oper-name&gt;</code>.  * <code>query</code> means that the <b>Operation</b> does not change the state of the system.  * <code>ordered</code> means that the values of the return parameter are ordered.  * <code>unique</code> means that the values returned by the parameter have no duplicates.  * <code>&lt;oper-constraint&gt;</code> is a constraint that applies to the <b>Operation</b>.  * <code>&lt;parameter-list&gt;</code> is a list of parameters of the <b>Operation</b> in the following format:  <code>&lt;parameter&gt; ::= [&lt;direction&gt;] [&lt;parameter&gt;] ':' [&lt;parameter-name&gt;] '*'</code>  <code>&lt;parameter&gt; ::= [&lt;direction&gt;] [&lt;parameter&gt;] ':' [&lt;multiplicity&gt;] !] ! '=' &lt;default&gt;] ! {&lt;parm-property&gt;   '*' } ] ]</code>  where:  - <code>&lt;direction&gt;</code> ::= 'in'   'out'   'inout' (defaults to 'in' if omitted).  - <code>&lt;parameter-name&gt;</code> is the name of the parameter.  - <code>&lt;type-expression&gt;</code> is an expression that specifies the type of the parameter.  - <code>&lt;multiplicity&gt;</code> is the multiplicity of the parameter.  - <code>&lt;default&gt;</code> is an expression that defines the value specification for the default value of the parameter.  - <code>&lt;parm-property&gt;</code> indicates additional property values that apply to the parameter.</p>

Table A.1: UML Metamodel for Structural Diagrams

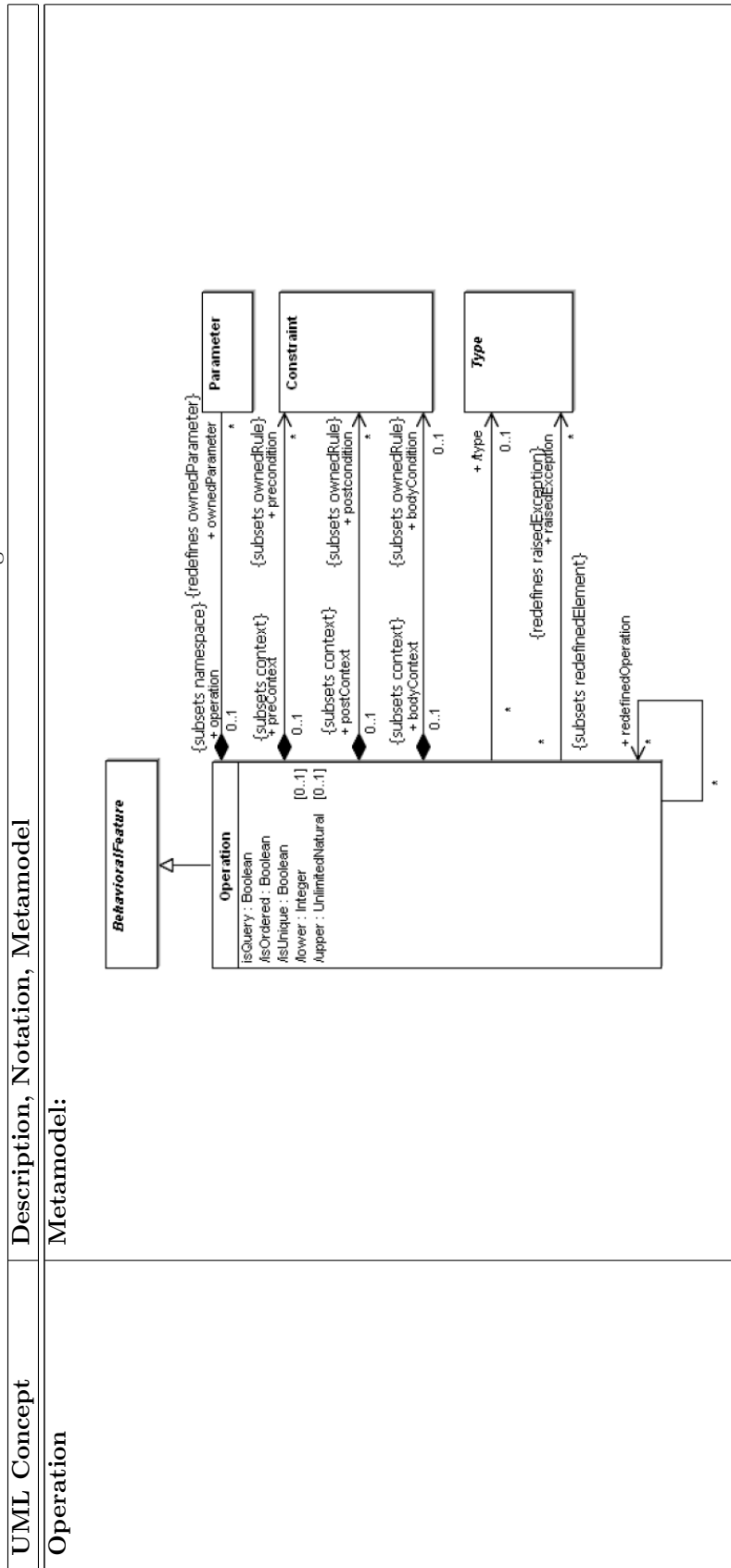


Table A.1: UML Metamodel for Structural Diagrams

UML Concept	Description, Notation, Metamodel
<p><b>Association, Aggregation, Composition</b></p>	<p><b>Description:</b></p> <p>An <b>Association</b> specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by <b>Properties</b>, each of which is connected to the type of the end. An <b>Association</b> is derived from <b>Classifier</b> and <b>Relationship</b> in the UML metamodel. The instances of an <b>Association</b> are called <b>Links</b>. The collection of links of an <b>Association</b> with N ends that refer to particular instances of any N-1 ends will identify a collection of instances at the Nth end. The multiplicity of the association end constraints the size of this collection. If the end is marked as ordered, this collection will be ordered. If the end is marked as unique, this collection is a set; otherwise, it allows duplicate elements.</p> <p>An <b>Association</b> may represent an <b>Aggregation</b> (i.e., a whole/part relationship). Only binary associations can be <b>Aggregations</b>. <b>Composition</b> is a strong form of <b>Aggregation</b> that requires that a part instance be included in at most one composite at a time. If a composite is deleted, all of its parts are normally deleted with it.</p> <p><b>Notation:</b></p> <p>An <b>Association</b> may be drawn as a diamond (larger than a terminator on a line) with a solid line for each association end connecting the diamond to the <b>Classifier</b> that is the end's type. An <b>Association</b> with more than two ends can only be drawn this way. A binary <b>Association</b> is drawn as a solid line connecting two <b>Classifiers</b>, or a solid line connecting a single <b>Classifier</b> to itself.</p> <div data-bbox="790 806 917 1153" data-label="Diagram"> <pre> classDiagram     class Team     class Year     class Player     Team "*" -- "*" Year : season     Year "*" -- "*" Year : year     Year --&gt; PlayedYear : PlayedYear     Year "*" -- "*" Player : goalie     Team "*" -- "*" Player : team     </pre> </div> <p>The solid arrow of the binary <b>Association</b> indicates that the <b>Association</b> is to be read as associating the end away from the direction of the arrow with the end to which the arrow is pointing. This notation is for documentation purposes only and has no general semantic interpretation. A slash appearing in front of the name of an <b>Association</b>, or in place of the name if no name is shown, marks the <b>Association</b> as being derived from another association or constraint. Generalizations between <b>Associations</b> can be shown using a generalization arrow between the <b>Association</b> symbols. Various other notations can be placed near the end of the <b>Association</b> line as follows:</p> <ul style="list-style-type: none"> <li>- A multiplicity</li> <li>- A property string enclosed in curly braces. The following property strings can be applied to an association end:             <ul style="list-style-type: none"> <li>* {subsets &lt;property-name&gt;} to show that the end is a subset of the property called &lt;property-name&gt;.</li> <li>* {redefines &lt;end-name&gt;} to show that the end redefines the one named &lt;end-name&gt;.</li> <li>* {union} to show that the end is derived by being the union of its subsets.</li> <li>* {ordered} to show that the end represents an ordered set.</li> <li>* {bag} to show that the end represents a collection that permits the same element to appear more than once.</li> <li>* {sequence} or {seq} to show that the end represents a sequence (an ordered bag).</li> </ul> </li> <li>* If the end is navigable, any property strings that apply to an attribute.</li> </ul>

Table A.1: UML Metamodel for Structural Diagrams

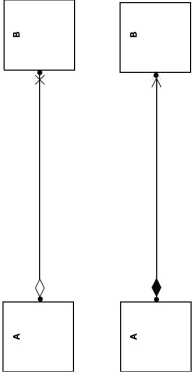
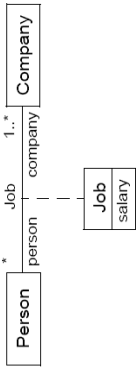
UML Concept	Description, Notation, Metamodel
<p><b>Association,</b> <b>Aggregation,</b> <b>Composition</b></p>	<p>By default an association end represents a set. An open arrowhead on the end of an <b>Association</b> indicates the end is navigable. A small x on the end of an <b>Association</b> indicates the end is not navigable.                      An <b>Association</b> with aggregationKind = shared differs in notation from binary <b>Associations</b> in adding a hollow diamond as a terminal adornment at the aggregate end of the association line. An <b>Association</b> with aggregationKind = composite likewise has a diamond at the aggregate end, but differs in having the diamond filled in. aggregationKind = none shows that the <b>Association</b> is not an <b>Aggregation</b> or a <b>Composition</b>.</p>  <p><b>Metamodel:</b> see <b>Class</b></p>
<p><b>AssociationClass</b></p>	<p><b>Description:</b> In the UML metamodel, an <b>AssociationClass</b> is a declaration of a semantic relationship between <b>Classifiers</b>, which has a set of features of its own. <b>AssociationClass</b> is both an <b>Association</b> and a <b>Class</b>.</p> <p><b>Notation:</b></p> 

Table A.1: UML Metamodel for Structural Diagrams

UML Concept	Description, Notation, Metamodel
Property - Association Qualifier	<p data-bbox="320 300 639 1648"> <b>Description:</b>  <b>Properties</b> can be assigned to <b>Associations</b> as <b>Association Qualifiers</b>. An <b>Association Qualifier</b> declares a partition of the set of associated instances with respect to an instance at the qualified end (the qualified instance is at the end to which the <b>Qualifier</b> is attached). A <b>Qualifier</b> instance comprises one value for each <b>Qualifier</b> attribute. Given a qualified object and a <b>Qualifier</b> instance, the number of objects at the other end of the association is constrained by the declared multiplicity. In the common case in which the multiplicity is 0..1, the <b>Qualifier</b> value is unique with respect to the qualified object, and designates at most one associated object. In the general case of multiplicity 0..*, the set of associated instances is partitioned into subsets, each selected by a given <b>Qualifier</b> instance. In the case of multiplicity 1 or 0..1, the <b>Qualifier</b> has both semantic and implementation consequences. In the case of multiplicity 0..*, it has no real semantic consequences but suggests an implementation that facilitates easy access of sets of associated instances linked by a given qualifier value.         </p> <p data-bbox="663 300 804 1648"> <b>Notation:</b>            A <b>Qualifier</b> is shown as a small rectangle attached to the end of an association path between the final path segment and the symbol of the classifier that it connects to as shown in the Figure below. The <b>Qualifier</b> rectangle is part of the association path, not part of the classifier. The <b>Qualifier</b> is attached to the source end of the association. The multiplicity attached to the target end denotes the possible cardinalities of the set of target instances selected by the pairing of a source instance and a <b>Qualifier</b> value. The <b>Qualifier</b> attributes are drawn within the <b>Qualifier</b> box. There may be one or more attributes shown one to a line. <b>Qualifier</b> attributes have the same notation as classifier attributes, except that initial value expressions are not meaningful. It is permissible (although somewhat rare), to have a <b>Qualifier</b> on each end of a single association. A <b>Qualifier</b> may not be suppressed.         </p> <div data-bbox="842 790 1002 1151" style="text-align: center;"> <pre> classDiagram     class Bank     class Person     class Chessboard     class Square      Bank "1" -- "*" Person : accountNo     Chessboard "1" -- "1" Square : rank: Rank, file: File           </pre> </div>

Table A.1: UML Metamodel for Structural Diagrams

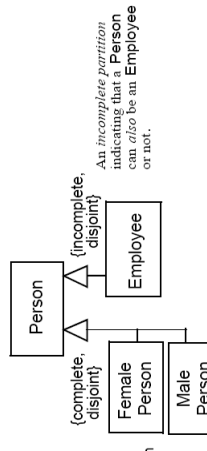
UML Concept	Description, Notation, Metamodel
<p>Generalization, GeneralizationSet</p>	<p><b>Description:</b></p> <p>A <b>Generalization</b> is a taxonomic <b>Relationship</b> between a more general <b>Classifier</b> and a more specific <b>Classifier</b>. Each instance of the specific <b>Classifier</b> is also an indirect instance of the general <b>Classifier</b>. Thus, the specific <b>Classifier</b> inherits the features of the more general <b>Classifier</b>. A <b>Generalization</b> has the attribute <b>isSubstitutable: Boolean</b>, which indicates whether the specific <b>Classifier</b> can be used wherever the general <b>Classifier</b> is used.</p> <p>A <b>GeneralizationSet</b> defines a particular set of <b>Generalization</b> relationships that describe the way in which a general <b>Classifier</b> (or superclass) may be divided using specific subtypes. For example, a <b>GeneralizationSet</b> could define a partitioning of the class <b>Person</b> into two subclasses: <b>Male Person</b> and <b>Female Person</b>. A <b>GeneralizationSet</b> has two attributes: <b>isCovering: Boolean</b>, which indicates whether or not the set of specific <b>Classifiers</b> are covering for a particular general classifier. When <b>isCovering</b> is true, every instance of a particular general <b>Classifier</b> is also an instance of at least one of its specific <b>Classifiers</b> for the <b>GeneralizationSet</b>. When <b>isCovering</b> is false, there are one or more instances of the particular general <b>Classifier</b> that are not instances of at least one of its specific <b>Classifiers</b> defined for the <b>GeneralizationSet</b>. The attribute <b>isDisjoint: Boolean</b> indicates whether or not the set of specific <b>Classifiers</b> in a <b>Generalization</b> relationship have instance in common. If <b>isDisjoint</b> is true, the specific <b>Classifiers</b> for a particular <b>GeneralizationSet</b> have no members in common; that is, their intersection is empty. If <b>isDisjoint</b> is false, the specific <b>Classifiers</b> in a particular <b>GeneralizationSet</b> have one or more members in common; that is, their intersection is not empty.</p> <p><b>Notation:</b></p> <p>The notations for these attributes are as follows:</p> <ul style="list-style-type: none"> <li>- complete, disjoint - Indicates the generalization set is covering and its specific <b>Classifiers</b> have no common instances.</li> <li>- incomplete, disjoint - Indicates the generalization set is not covering and its specific <b>Classifiers</b> have no common instances. (default)</li> <li>- complete, overlapping - Indicates the generalization set is covering and its specific <b>Classifiers</b> do share common instances.</li> <li>- incomplete, overlapping - Indicates the generalization set is not covering and its specific <b>Classifiers</b> do share common instances.</li> </ul>  <p>A <b>complete partition</b> indicating that a <b>Person</b> may be subtyped as either a <b>Female Person</b> or a <b>Male Person</b>.</p> <p>An <b>incomplete partition</b> indicating that a <b>Person</b> can also be an <b>Employee</b> or not.</p>



Table A.1: UML Metamodel for Structural Diagrams

UML Concept	Description, Notation, Metamodel
Dependency	<p><b>Description:</b></p> <p>A <b>Dependency</b> is a <b>Relationship</b> that signifies that a single or a set of model elements requires other model elements for their specification or implementation. This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s). Different kinds of <b>Dependency</b> have been specified by the UML metamodel.</p> <p><b>Usage</b> is a kind of <b>Dependency</b> in which one element requires another element (or a set of elements) for its full implementation or operation.</p> <p>An <b>Abstraction</b> is a <b>Relationship</b> that relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints. In the metamodel, an <b>Abstraction</b> is a <b>Dependency</b> in which there is a mapping between the supplier and the client.</p> <p><b>Realization</b> is a specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other represents an implementation of the latter (the client). <b>Realization</b> can be used to model stepwise refinement, optimizations, transformations, templates, model synthesis, framework composition, etc.</p> <p>A <b>Substitution</b> is a relationship between two <b>Classifiers</b> which signifies that the substituting <b>Classifier</b> complies with the contract specified by the contract classifier. This implies that instances of the substituting <b>Classifier</b> are runtime substitutable where instances of the contract <b>Classifier</b> are expected.</p> <p>The presence of <b>Dependency</b> relationships in a model does not have any runtime semantics implications, it is all given in terms of the model-elements that participate in the relationship, not in terms of their instances.</p> <p><b>Notation:</b></p> <p>A <b>Dependency</b> is shown as a dashed arrow between two model elements. The model element at the tail of the arrow (the client) depends on the model element at the arrowhead (the supplier). The arrow may be labeled with an optional stereotype corresponding to the type of the <b>Dependency</b>, e.g. «use&gt;», and an optional name.</p>

Table A.1: UML Metamodel for Structural Diagrams

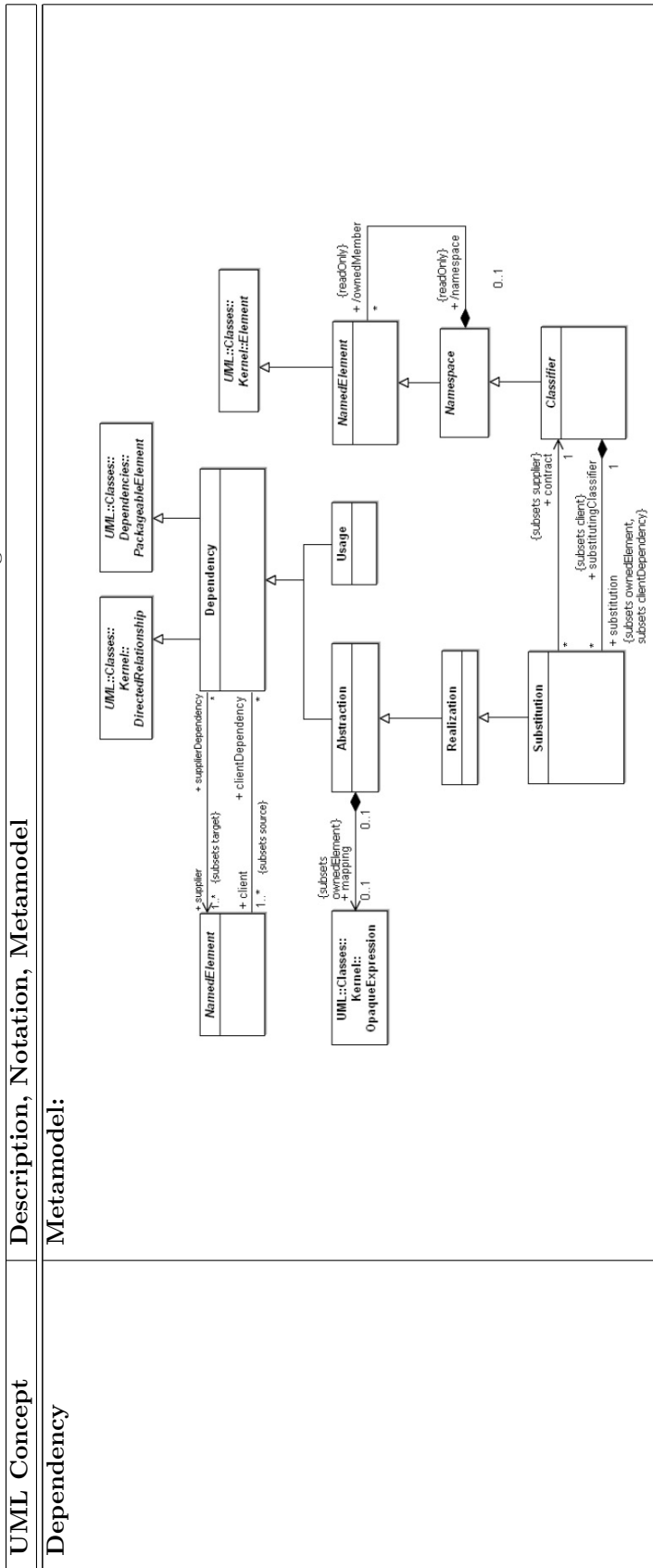


Table A.1: UML Metamodel for Structural Diagrams

UML Concept	Description, Notation, Metamodel
<b>Interface</b> <b>InterfaceRealization</b>	<p data-bbox="290 1469 316 1648"><b>Description:</b></p> <p data-bbox="322 300 571 1648">An <b>Interface</b> is a kind of <b>Classifier</b> that represents a declaration of a set of coherent public features and obligations. An <b>Interface</b> specifies a contract; any instance of a <b>Classifier</b> that realizes the <b>Interface</b> must fulfill that contract. The obligations that may be associated with an <b>Interface</b> are in the form of various kinds of constraints (such as pre- and postconditions) or protocol specifications, which may impose ordering restrictions on interactions through the <b>Interface</b>. Since <b>Interfaces</b> are declarations, they are not instantiable. Instead, an interface specification is implemented by an instance of an instantiable <b>Classifier</b>, which means that the instantiable <b>Classifier</b> presents a public facade that conforms to the interface specification. Note that a given <b>Classifier</b> may implement more than one <b>Interface</b> and that an interface may be implemented by a number of different <b>Classifiers</b>. An <b>InterfaceRealization</b> is a specialized <b>Realization</b> dependency between a <b>Classifier</b> and an <b>Interface</b>. This dependency signifies that the realizing <b>Classifier</b> conforms to the contract specified by the <b>Interface</b>.</p> <p data-bbox="663 1503 689 1648"><b>Notation:</b></p> <p data-bbox="695 300 746 1648">In cases where <b>Interfaces</b> are represented using the rectangle notation, interface realization and usage <b>Dependencies</b> are denoted with appropriate dependency arrows as shown in the following figure. The classifier at the tail of the arrow implements the interface at the head of the arrow or uses that interface, respectively.</p> <div data-bbox="788 680 887 1254" style="text-align: center;"> <pre> classDiagram     class TheftAlarm     class ISensor {         activate()         read()     }     class ProximitySensor     TheftAlarm .. &gt; ISensor     ProximitySensor .. &gt; ISensor </pre> </div> <p data-bbox="925 904 944 1648">Alternatively these <b>Dependencies</b> can be notated as shown in the following figures.</p> <div data-bbox="1002 797 1283 1151" style="text-align: center;"> <pre> classDiagram     class ISensor(( ))     class ProximitySensor     class TheftAlarm     ProximitySensor o-- ISensor     TheftAlarm o-- ISensor </pre> </div>

Table A.1: UML Metamodel for Structural Diagrams

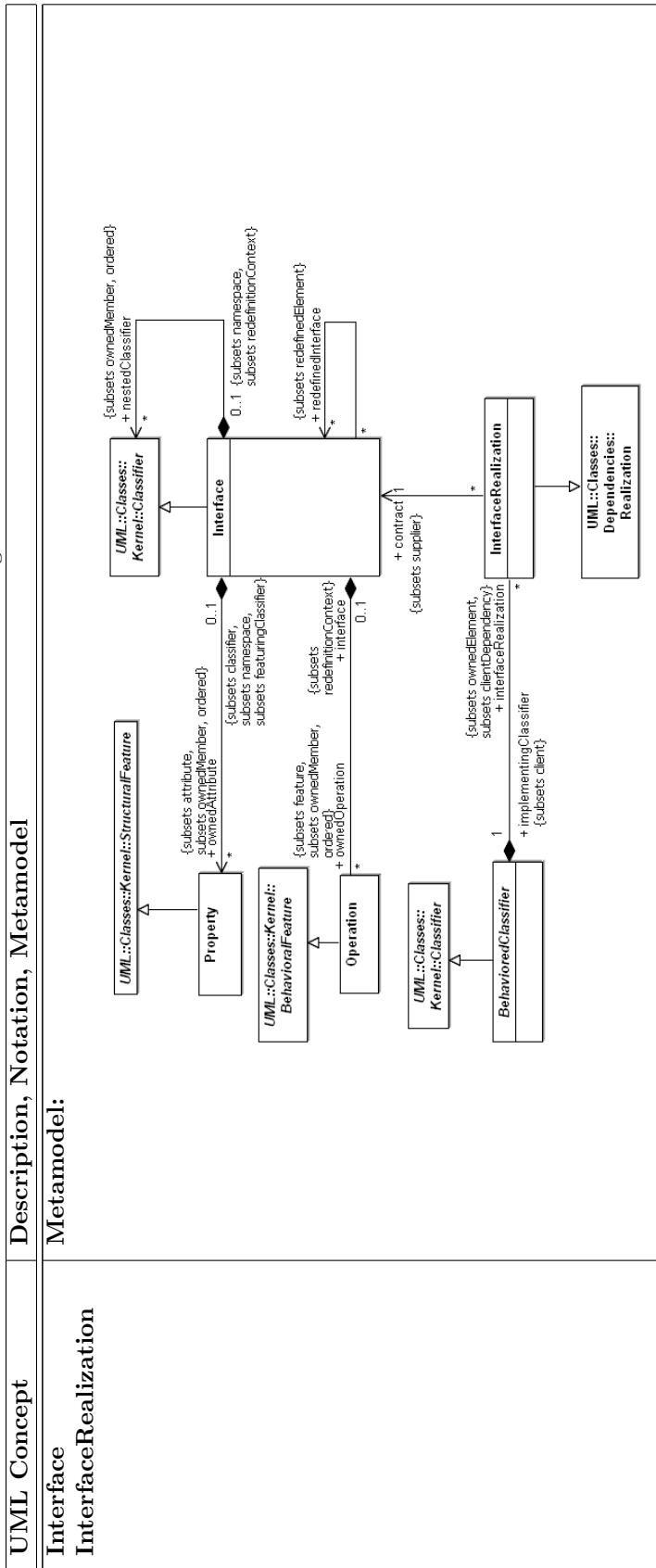


Table A.1: UML Metamodel for Structural Diagrams

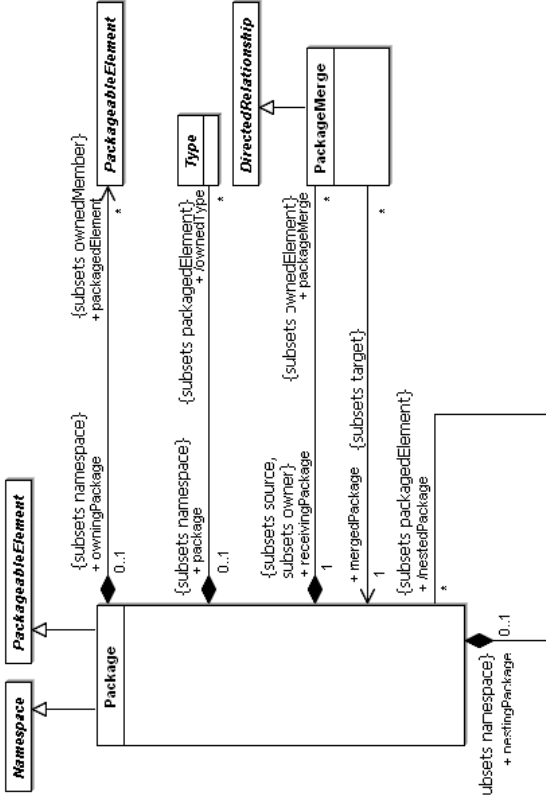
UML Concept	Description, Notation, Metamodel
<p><b>Package</b></p>	<p><b>Description:</b>                      A <b>Package</b> is used to group elements, and provides a namespace for the grouped elements. It can contain also other packages. A <b>Package</b> can import either individual members of other packages, or all the members of other packages. In addition a <b>Package</b> can be merged with other packages. A <b>Package</b> owns its owned members, with the implication that if a <b>Package</b> is removed from a model, so are the elements owned by the <b>Package</b>. The public contents of a <b>Package</b> is always accessible outside the <b>Package</b> through the use of qualified names. It can be used to define different name spaces in the model and group elements according to their level of abstraction.</p> <p><b>Metamodel:</b></p>  <pre> classDiagram     class Namespace     class Package     class PackageableElement     class Type     class DirectedRelationship     class PackageMerge      Namespace -- &gt; Package     PackageableElement -- &gt; Package     Package --&gt; PackageableElement "*" : {subsets namespace} + owningPackage     Package --&gt; Type "*" : {subsets namespace} + package     Package --&gt; PackageMerge "*" : {subsets source, subsets owner} + receivingPackage     PackageMerge -- &gt; DirectedRelationship     PackageMerge --&gt; Package "*" : {subsets target}     Package --&gt; Package "*" : {subsets namespace} + nestingPackage     </pre> <p>The diagram illustrates the UML metamodel for packages. It features several classes and their relationships:</p> <ul style="list-style-type: none"> <li><b>PackageableElement</b> is a base class, with <b>Package</b> as a specialization (indicated by a solid line with an open triangle arrowhead).</li> <li><b>Package</b> is a class that specializes <b>PackageableElement</b>. It has several associations:             <ul style="list-style-type: none"> <li>A self-association on <b>Package</b> with multiplicity 0..1 at the source and * at the target, labeled with <code>{subsets namespace} + owningPackage</code>.</li> <li>An association from <b>Package</b> to <b>Type</b> with multiplicity 0..1 at the source and * at the target, labeled with <code>{subsets namespace} + package</code>.</li> <li>An association from <b>Package</b> to <b>PackageMerge</b> with multiplicity 1 at the source and * at the target, labeled with <code>{subsets source, subsets owner} + receivingPackage</code>.</li> <li>An association from <b>PackageMerge</b> to <b>Package</b> with multiplicity 1 at the source and * at the target, labeled with <code>{subsets target}</code>.</li> <li>A self-association on <b>Package</b> with multiplicity 0..1 at the source and * at the target, labeled with <code>{subsets namespace} + nestingPackage</code>.</li> </ul> </li> <li><b>PackageMerge</b> is a class that specializes <b>DirectedRelationship</b> (indicated by a solid line with an open triangle arrowhead).</li> </ul>

Table A.1: UML Metamodel for Structural Diagrams

UML Concept	Description, Notation, Metamodel
Package Merge Package Import (private, public)	<p data-bbox="290 1469 316 1648"><b>Description:</b></p> <p data-bbox="322 300 513 1648">A <b>Package Merge</b> is a directed relationship between two packages, that indicates that the contents of the two packages are to be combined. It is very similar to <b>Generalization</b> in the sense that the source element conceptually adds the characteristics of the target element to its own characteristics resulting in an element that combines the characteristics of both. <b>Package Merge</b> is particularly useful in meta-modeling and is extensively used in the definition of the UML metamodel. In terms of model semantics, there is no difference between a model with explicit package merges, and a model in which all the merges have been performed.</p> <p data-bbox="536 300 691 1648">A <b>Package Import</b> is a relationship that allows the use of unqualified names to refer to package members from other namespaces. A <b>Package Import</b> is a relationship between an importing namespace and a package, indicating that the importing namespace adds the names of the members of the package to its own namespace. Conceptually, a <b>Package Import</b> is equivalent to having an element import to each individual member of the imported namespace, unless there is already a separately-defined element import.</p>

Table A.1: UML Metamodel for Structural Diagrams

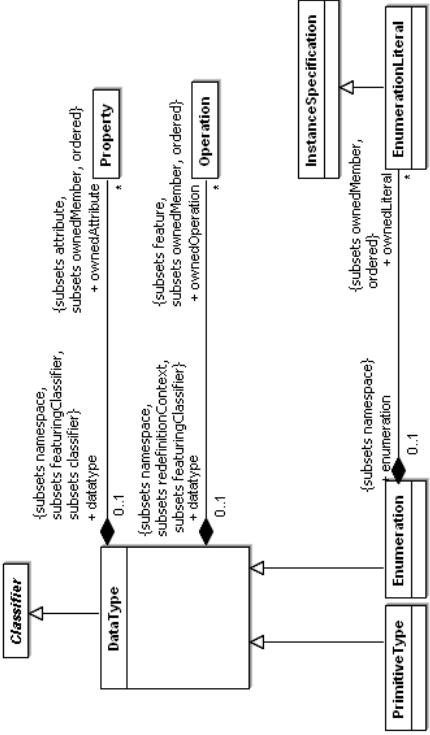
UML Concept	Description, Notation, Metamodel
<b>Data Type</b> <b>Enumeration</b> <b>EnumerationLiteral</b>	<p><b>Description:</b></p> <p>A <b>Data Type</b> is a type whose instances are identified only by their value. A <b>Data Type</b> may contain attributes to support the modeling of structured data types.</p> <p>The UML metamodel uses two <b>Data Type</b> specializations: <b>Primitive Type</b> and <b>Enumeration</b>. Four concrete <b>Primitive Types</b> are also defined: Integer, Boolean, String and UnlimitedNatural.</p> <p>A typical use of data types would be to represent programming language primitive types or CORBA basic types. For example, integer and string types are often treated as data types.</p> <p><b>Enumeration</b> is a kind of data type, whose instances may be any of a number of user-defined enumeration literals. It is possible to extend the set of applicable enumeration literals in other packages or profiles.</p> <p>An <b>EnumerationLiteral</b> is a user-defined data value for an enumeration.</p> <p><b>Notation:</b></p> <p>A <b>Data Type</b> is denoted using the rectangle symbol with keyword «data type» or, when it is referenced by (e.g., an attribute) denoted by a string containing the name of the <b>Data Type</b>.</p> <p>An <b>Enumeration</b> may be shown using the classifier notation (a rectangle) with the keyword «enumeration».</p> <p>An <b>EnumerationLiteral</b> is typically shown as a name, one to a line, in the compartment of the enumeration notation.</p> <p><b>Metamodel:</b></p>  <pre> classDiagram     class Classifier     class DataType     class PrimitiveType     class Enumeration     class EnumerationLiteral     class Property     class Operation     class InstanceSpecification      Classifier &lt; -- DataType     PrimitiveType &lt; -- Enumeration     Enumeration &lt; -- EnumerationLiteral     EnumerationLiteral &lt; -- InstanceSpecification      DataType "0..1" *-- "0..1" Property     DataType "0..1" *-- "0..1" Operation     Enumeration "0..1" *-- "0..1" EnumerationLiteral   </pre> <p>The diagram illustrates the UML metamodel for structural diagrams. It shows the following classes and their relationships:</p> <ul style="list-style-type: none"> <li><b>Classifier</b>: A base class for <b>Data Type</b>.</li> <li><b>Data Type</b>: A class that inherits from <b>Classifier</b>. It has two associations with <b>Property</b> and <b>Operation</b>, both with multiplicity 0..1 and a composition relationship (indicated by a filled diamond on the <b>Data Type</b> side).</li> <li><b>Primitive Type</b>: A class that inherits from <b>Enumeration</b>.</li> <li><b>Enumeration</b>: A class that inherits from <b>Primitive Type</b>. It has an association with <b>EnumerationLiteral</b> with multiplicity 0..1 and a composition relationship (indicated by a filled diamond on the <b>Enumeration</b> side).</li> <li><b>EnumerationLiteral</b>: A class that inherits from <b>Enumeration</b>.</li> <li><b>InstanceSpecification</b>: A class that inherits from <b>EnumerationLiteral</b>.</li> </ul> <p>Attributes and Stereotypes:</p> <ul style="list-style-type: none"> <li><b>Property</b>: {subsets namespace, subsets featuringClassifier, subsets classifier} + datatype</li> <li><b>Operation</b>: {subsets namespace, subsets redefinitionContext, subsets featuringClassifier} + datatype</li> <li><b>EnumerationLiteral</b>: {subsets namespace} + enumeration</li> <li><b>EnumerationLiteral</b> (in InstanceSpecification): {subsets ownedMember, ordered} + ownedLiteral</li> </ul>

Table A.1: UML Metamodel for Structural Diagrams

UML Concept	Description, Notation, Metamodel
<p><b>Comment</b></p>	<p><b>Description:</b>  A <b>Comment</b> is a textual annotation that can be attached to a set of elements. A <b>Comment</b> gives the ability to attach various remarks to elements. A <b>Comment</b> carries no semantic force, but may contain information that is useful to a modeler. A <b>Comment</b> can be owned by any element.</p> <p><b>Notation:</b>  A <b>Comment</b> is shown as a rectangle with the upper right corner bent (this is also known as a “note symbol”). The rectangle contains the body of the <b>Comment</b>. The connection to each annotated element is shown by a separate dashed line.</p>
<p><b>Constraint</b></p>	<p><b>Description:</b>  <b>Constraint</b> contains a <b>ValueSpecification</b> that specifies additional semantics for one or more elements. Certain kinds of <b>Constraints</b> (such as an association “xor” constraint) are predefined in UML, others may be user-defined. A user-defined <b>Constraint</b> is described using a specified language, whose syntax and interpretation is a tool responsibility. One predefined language for writing constraints is OCL. In some situations, a programming language such as Java may be appropriate for expressing a <b>Constraint</b>. In other situations natural language may be used. <b>Constraint</b> is a condition (a Boolean expression) that restricts the extension of the associated element beyond what is imposed by the other language constructs applied to that element. <b>Constraint</b> contains an optional name, although they are commonly unnamed.</p> <p><b>Notation:</b>  A <b>Constraint</b> is shown as a text string in braces () according to the following BNF: <code>&lt;constraint&gt; ::= '{' [ &lt;name&gt; ';' ] &lt;Boolean-expression&gt; '}'</code>. For an element whose notation is a text string (such as an attribute, etc.), the constraint string may follow the element text string in braces. For a <b>Constraint</b> that applies to a single element (such as a class or an association path), the <b>Constraint</b> string may be placed near the symbol for the element, preferably near the name, if any. A tool must make it possible to determine the constrained element. For a <b>Constraint</b> that applies to two elements (such as two classes or two associations), the <b>Constraint</b> may be shown as a dashed line between the elements labeled by the <b>Constraint</b> string (in braces).</p>





Table A.2: CBIRS Extensions to the UML Metamodel

Stereotype	Description
<b>Framework Stereotypes</b>	
« <b>framework</b> »	Applies to <b>Class</b> , <b>Package</b> , <b>Interface</b> and means that the element belongs to the framework model. These elements are used to derive application-specific elements. They are not mapped to the implementation platform.
« <b>application</b> »	Applies to <b>Class</b> , <b>Package</b> , <b>Interface</b> and means that the element belongs to the application-specific model. These elements are to be mapped to the implementation specific model.
« <b>adapt-static</b> »	Applies to <b>Interface</b> , <b>Class</b> , <b>Method</b> , <b>Generalization</b> and shows that the element can be adapted during design-time through subclassing.
« <b>template</b> » « <b>hook</b> »	Apply to <b>Class</b> and <b>Method</b> and show which functionality has to be adapted by subclassing the classes and redefining the methods. For more specificity higher abstract-level tags such as «Unification-template» and «Unification-hook» or «Separation-template» and «Separation-hook» can be used.
<b>Deployment Stereotypes</b>	
« <b>persistent</b> »	Has to be applied to <b>Classes</b> in the PIM, which should be mapped to the persistence layer of an application. These stereotype is used as a markup or guidelines for the mapping process. It does not carry domain specific information.
« <b>application-logic</b> »	Has to be applied to <b>Classes</b> or <b>Interfaces</b> in the PIM, which should be mapped to the application-logic layer of an application. These stereotype is used as a markup or guidelines for the mapping process. It does not carry domain specific information.
<b>Domain-Specific Stereotypes</b>	
« <b>CBIRSDataType</b> »	A specialization of <b>DataType</b> for deriving CBIRS specific data types.
« <b>CBIRSSimpleType</b> »	A specialization of <b>CBIRSDataType</b> for deriving CBIRS specific data types.
« <b>CBIRSComplexType</b> »	A specialization of <b>CBIRSDataType</b> for deriving CBIRS specific data types.
« <b>CBIRSStructType</b> »	A specialization of <b>CBIRSComplexType</b> for deriving CBIRS specific data types.

Table A.2: Extensions to the UML Metamodel

Stereotype	Description
«CBIRSArrayType»	<p>A specialization of <b>CBIRSComplexType</b> for representing arrays. The following properties are specified for this stereotype:</p> <ul style="list-style-type: none"> <li>- dimension_multiplicity: CBIRIntegerType[1..*]</li> <li>- datatype: CBIRSStringType</li> </ul> <p><b>Notation:</b>  <i>CBIRSArrayType</i> ::= '<i>CBIRSArrayType</i> [<i>&lt;number&gt;</i>]' [ '<i>[</i>' * '<i>&lt;number&gt;</i>' * '<i>]</i>' * '<i>;</i>' * '<i>&lt;datatype&gt;</i>' ]            where:            - <i>&lt;number&gt;</i> is a value of type CBIRIntegerType            - <i>&lt;datatype&gt;</i> is a name of a predefined dataType or a Class</p>
«CBIRSIntegerType»	A specialization of <b>CBIRSSimpleDataType</b> for representing integer values.
«CBIRSBooleanType»	A specialization of <b>CBIRSSimpleDataType</b> for representing boolean values.
«CBIRSCharacterType»	A specialization of <b>CBIRSSimpleDataType</b> for representing character values.
«CBIRSStringType»	A specialization of <b>CBIRSSimpleDataType</b> for representing string values.
«CBIRSFloaType»	A specialization of <b>CBIRSSimpleDataType</b> for representing float values.
«CBIRSBinaryType»	A specialization of <b>CBIRSSimpleDataType</b> for representing binary values.
«CBIRSEnumType»	A specialization of <b>CBIRSComplexDataType</b> for representing enumeration data types.

## Appendix B

# PSM Metamodel

In this appendix the UML extensions for representing SQL:2003 specific concepts are defined. The descriptions of the SQL:2003 concepts are taken from the Working Draft of the Standard: “Information technology Database languages SQL Part 2: Foundation (SQL/Foundation)”, ISO/IEC 9075-2:2003 (E) and [Tür03]. The first part of the appendix defines the UML extension for representing purely relational concepts. In the second part the object-relational metamodeling features are introduced and finally, in the third part, the needed basic and complex SQL data types are listed.

Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<p><b>Schema</b></p>	<p><b>Description:</b> All objects which form the database are grouped logically in so called <b>Schemas</b>. Thus a <b>Schema</b> corresponds to a kind of namespace.</p> <p><b>UML Standard Elements:</b> According to Rational's Data Modeling Profile a <b>Schema</b> is represented by a <b>Package</b> in UML, which is the organizational unit of UML. Thus, a <b>Schema</b> is a stereotyped <b>Package</b> «Schema» and a part of the UML Data Modeling Profile.</p> <p><b>Stereotypes:</b> The stereotype «Schema» is used.</p> <p><b>Constraints:</b> none</p> <p><b>Notation:</b></p> <div data-bbox="826 831 1031 1115" style="border: 1px solid black; padding: 5px; margin-top: 10px;"> </div>

Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<b>Table</b>	<p><b>Description:</b>  A <b>Table</b> is a collection of rows having one or more columns. A row is a value of the same <b>Table</b> has the same row type. The value of the i-th field of every row in a <b>Table</b> is the value of the i-th column of that row in the <b>Table</b>. The row is the smallest unit of data that can be inserted into a <b>Table</b> and deleted from a <b>Table</b>.</p> <p><b>UML Standard Elements:</b>  According to Rational's Data Modeling Profile a <b>Table</b> is a stereotyped <b>Class</b> «Table» and part of the UML Data Modeling Profile.</p> <p><b>Stereotypes:</b>  The stereotype «Table» is used.</p> <p><b>Constraints:</b>  - The operations of a «Table» <b>Class</b> can be only stereotyped operations, corresponding to the <b>Table Constraint</b> concept.  - A «Table» <b>Class</b> must have at least one «PK» <b>Attribute</b>.  - All <b>Attributes</b> and <b>Operations</b> of a «Table» <b>Class</b> are public.</p> <p><b>Notation:</b></p> <div data-bbox="922 757 1078 1182" style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre> «Table» <b>TableName</b> «PK»+Attribute1 : SQLIntegerType [1] +Attribute2 : SQLVarCharType(length: 20) [0..1] «PK»+PK_Attribute1( Attribute1 : SQLIntegerType ) </pre> </div>

Table B.1: Relational Concepts

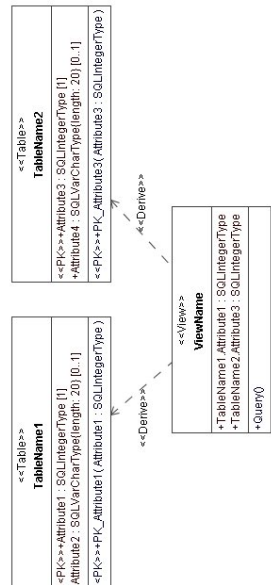
SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
View	<p><b>Description:</b>  A <b>View</b> is a named, derived <b>Table</b> from one or more other <b>Tables</b>, queried by a query returning a row as a result. The values of its attributes are gathered from the other <b>Tables</b> by evaluating the query.</p> <p><b>UML Standard Elements:</b>  A <b>View</b> is a stereotyped <b>Class</b> «View» and part of the UML Data Modeling Profile. The dependency to each <b>Table</b> involved in the definition of the <b>View</b> is indicated with a stereotyped dependency «Derive».</p> <p><b>Stereotypes:</b>  The stereotypes «View» and «Derive» are used.</p> <p><b>Constraints:</b>  - A «View» <b>Class</b> cannot have «PK» and «FK» or «PFK» <b>Attributes</b>.  - The <b>Attributes</b> of a «View» are derived from one of the «Table» <b>Classes</b> participating in the «View».  - The name of the «Table» in front of the <b>Attributes</b> name can specify the <b>Attribute</b>.  - A «View» may contain only one <b>Operation</b> representing the query for the <b>View</b>. The query itself can be specified as a <b>Constraint</b> of the <b>Operation</b>.</p> <p><b>Notation:</b></p>  <pre> classDiagram     class Table1 {         &lt;&lt;Table&gt;&gt;         +Attribute1 : SQLIntegerType [1]         +Attribute2 : SQLVarCharType(length: 20) [0..1]         &lt;&lt;PK&gt;&gt;+PK_Attribute1 : SQLIntegerType     }     class Table2 {         &lt;&lt;Table&gt;&gt;         +Attribute3 : SQLIntegerType [1]         +Attribute4 : SQLVarCharType(length: 20) [0..1]         &lt;&lt;PK&gt;&gt;+PK_Attribute3 : SQLIntegerType     }     class View {         &lt;&lt;View&gt;&gt;         +TableName1 : Attribute1 : SQLIntegerType         +TableName2 : Attribute2 : SQLIntegerType         +Query()     }     Table1 ..&gt; View : &lt;&lt;Derive&gt;&gt;     Table2 ..&gt; View : &lt;&lt;Derive&gt;&gt;   </pre>

Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Column	<p><b>Description:</b>  A <b>Table</b> consists of one or more <b>Columns</b>. Each <b>Column</b> has a name and a data type related to it. A <b>Column</b> can have a default value. The following integrity constraints can be assigned to a column:</p> <ul style="list-style-type: none"> <li>- Every <b>Column</b> has a nullability characteristic that indicates whether the value from that <b>Column</b> can be NULL.</li> <li>- A <b>Column</b> has a unique characteristic that indicates whether it is possible to have different rows in the same table with the same value for that column.</li> <li>- If the <b>Column</b> is defined as unique and not nullable then it can be chosen as the <b>Primary Key</b> for the <b>Table</b>.</li> <li>- A <b>Column</b> may play the role of a <b>Foreign Key</b> in a <b>Table</b>, defining a reference to another <b>Table</b>. Such a column may contain NULL values or values matching the values of a <b>Column</b> in the referenced <b>Table</b>. Each <b>Foreign Key</b> constraint is associated implicitly or explicitly with an action, which should resolve any referential integrity problems. Explicit actions are: ON [DELETE   UPDATE] [NO ACTION   CASCADE   SET NULL   SET DEFAULT   RESTRICT].</li> <li>- A CHECK constraint can be associated to a <b>Column</b>. It is used to limit the range of possible values for the <b>Column</b> through a boolean predicate.</li> </ul> <p><b>UML Standard Elements:</b>  All <b>Attributes</b> of a «Table» <b>Class</b> are <b>Columns</b> of the <b>Table</b>. The <b>Columns</b> are part of the UML Data Modeling Profile as stereotyped <b>Properties</b> «Column».</p> <p>A «Column» <b>Attribute</b> defines the data type to which it belongs, if it is a derived <b>Attribute</b>, if its value can be NULL and if the values of this <b>Column</b> have to be unique.</p> <p>In Rational's Data Model «Column» attribute can be a primary key «PK» or a foreign key «FK» attribute, or also at the same time a primary and a foreign key «PFK». A «Column» can have a Check constraint associated to it in order to limit the range of possible values.</p> <p><b>Stereotypes:</b>  The stereotype «Column» does not have to be shown in the diagram since all attributes of the «Table» class are of this stereotype.</p> <p>«PK», «FK» and «PFK» are specializations of the «Column» <b>Property</b> and represents a column, which is defined as a <b>Primary Key</b> or a <b>Foreign Key</b>, respectively.</p> <p><b>Constraints:</b>  - The data types which can be used for columns are only data types derived from «SQLDataType» or «ObjectType».</p> <ul style="list-style-type: none"> <li>- A «PK» or a «PFK» cannot be nullable and must be unique.</li> </ul>



Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Column	<p><b>Notation:</b></p> <p><code>&lt;property&gt;</code> ::= [<code>&lt;visibility&gt;</code>] [<code>'/'</code>] <code>&lt;name&gt;</code> [<code>':'</code>] <code>&lt;prop-type&gt;</code>] [<code>'/'</code>] <code>&lt;multiplicity&gt;</code> [<code>?'</code>] [<code>'='</code>] <code>&lt;default&gt;</code>] [<code>{</code>] <code>&lt;prop-modifier&gt;</code>] [<code>*</code>] [<code>}'</code>]</p> <p>where:</p> <ul style="list-style-type: none"> <li>- <code>&lt;visibility&gt;</code> is the visibility of the <b>Property</b>. (VisibilityKind is an <b>Enumeration</b> with the following values: public, private, protected, package)</li> <li>- <code>&lt;visibility&gt;</code> ::= <code>'+'</code>   <code>'-'</code>   <code>'#'</code>   <code>'~'</code>. Does not apply for <code>&lt;&lt;Column&gt;&gt;</code>.</li> <li>- <code>'/'</code> signifies that the property is derived. Does not apply for <code>&lt;&lt;Column&gt;&gt;</code>.</li> <li>- <code>&lt;name&gt;</code> is the name of the property.</li> <li>- <code>&lt;prop-type&gt;</code> is the name of the Data Type that is the type of the property.</li> <li>- <code>&lt;multiplicity&gt;</code> is the multiplicity of the <b>Property</b>. This term is not used for <code>&lt;&lt;Column&gt;&gt;</code> <b>Attributes</b>. In order to define sets or arrays of <code>&lt;&lt;Column&gt;&gt;</code> <b>Attributes</b> the corresponding complex SQL data types should be used (see Table SQL Data Types below).</li> <li>- <code>&lt;default&gt;</code> is an expression that evaluates to the default value or values of the property.</li> <li>- <code>&lt;prop-modifier&gt;</code> indicates a modifier that applies to the property.</li> <li>- <code>&lt;prop-modifier&gt;</code> ::= <code>'readOnly'</code>   <code>'union'</code>   <code>'subsets'</code>   <code>&lt;property-name&gt;</code>   <code>'redefines'</code>   <code>'unique'</code>   <code>'nonunique'</code>   <code>'notnull'</code>   <code>&lt;prop-constraint&gt;</code></li> </ul> <p>where:</p> <ul style="list-style-type: none"> <li>* <code>readOnly</code> means that the <b>Property</b> is read only. Does not apply for <code>&lt;&lt;Column&gt;&gt;</code>.</li> <li>* <code>union</code> means that the <b>Property</b> is a derived union of its subsets. Does not apply for <code>&lt;&lt;Column&gt;&gt;</code>.</li> <li>* <code>subsets &lt;property-name&gt;</code> means that the <b>Property</b> is a proper subset of the <b>Property</b> identified by <code>&lt;property-name&gt;</code>. Does not apply for <code>&lt;&lt;Column&gt;&gt;</code>.</li> <li>* <code>redefines &lt;property-name&gt;</code> means that the <b>Property</b> redefines an inherited <b>Property</b> identified by <code>&lt;property-name&gt;</code>. Does not apply for <code>&lt;&lt;Column&gt;&gt;</code>.</li> <li>* <code>ordered</code> means that the <b>Property</b> is ordered. Does not apply for <code>&lt;&lt;Column&gt;&gt;</code>.</li> <li>* <code>'unique'</code> the property cannot have duplicate values in the table. The meaning of this modifier is overridden for <code>&lt;&lt;Column&gt;&gt;</code> <b>Attributes</b>. It does not refer to the unique modifier for multiple properties.</li> <li>* <code>'notnull'</code> the property cannot have NULL values in the table. If this modifier is not used, the property can have null values. This modifier is introduced especially for <code>&lt;&lt;Column&gt;&gt;</code> <b>Attributes</b>.</li> <li>* <code>&lt;prop-constraint&gt;</code> is an expression that specifies a check constraint that applies to the property.</li> </ul> <p><code>&lt;&lt;FK&gt;&gt;</code> and <code>&lt;&lt;PFK&gt;&gt;</code> have additionally the following property modifier:  <code>references &lt;column name&gt;</code> <code>&lt;table name&gt;</code> determines the name of the referenced table and the corresponding column.</p>

Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<b>Table Constraint</b>	<p><b>Description:</b>  The following <b>Constraints</b> can be defined for a <b>Table</b>:</p> <ul style="list-style-type: none"> <li>- <b>Unique</b> (A1...An) constraint: assures that no two rows of the table can contain the same values for the given <b>Attributes</b>: A1...An.</li> <li>- <b>Primary Key</b> (A1...An) constraint: defines the combination of <b>Columns</b> as a <b>Primary Key</b> for the <b>Table</b>. The combination of these values must be unique.</li> <li>- <b>Foreign Key</b> (A1...An) REFERENCES Table (B1...Bn) constraint: defines the combination of <b>Columns</b> as a <b>Foreign Key</b> for the <b>Table</b>. In addition the <b>Foreign Key Constraint</b> may contain an explicit <b>MATCH</b> condition: <b>MATCH</b> [SIMPLE   PARTIAL   FULL] and a referential integrity preserving action (see constraints of <b>Columns</b>)</li> <li>- <b>CHECK</b> &lt;predicate&gt; constraint: see <b>Column</b>. The predicate may contain also subqueries.</li> </ul> <p><b>UML Standard Elements:</b>  Several types of <b>Table Constraints</b> are defined in the UML Data Modeling Profile. All of them are implemented as stereotyped <b>Operations</b> of «Table».</p> <p>Ambler suggest to model constraints with UML's Object Constraint Language (OCL). The advantage of his approach is that with OCL constraints can be assigned also to relationships whereas operations cannot be assigned to relationships. However, this is not needed in the logical design.</p> <p><b>Stereotypes:</b>  The <b>Unique Constraint</b> uses the stereotyped <b>Operation</b> «Unique» in the UML Data Modeling Profile.  The <b>Primary Key Constraint</b> uses the stereotyped <b>Operation</b> «PK» in the UML Data Modeling Profile.  The <b>Foreign Key Constraint</b> uses the stereotyped <b>Operation</b> «FK» in the UML Data Modeling Profile.  The <b>Check Constraint</b> uses the stereotyped <b>Operation</b> «Check» in the UML Data Modeling Profile.</p> <p><b>Constraint:</b>  - Only one <b>Primary Key Constraint</b> is allowed per «Table».</p>

Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<b>Table Constraint</b>	<p><b>Notation:</b>  <code>&lt;visibility&gt; &lt;name&gt; ' (' &lt;parameter-list&gt; ')' [ ':' &lt;return-type&gt; ] [ '{ ' &lt;oper-property&gt; [ '*' ] } ] ]</code>  where:  - <code>&lt;visibility&gt;</code> is the visibility of the <b>Operation</b>. <code>&lt;visibility&gt; ::= '+'   '-'   '#'</code>. It does not apply for <b>Table Constraints</b>.  - <code>&lt;name&gt;</code> is the name of the <b>Operation</b>.  - <code>&lt;return-type&gt;</code> is the type of the return result parameter if the <b>Operation</b> has one defined. It does not apply for <b>Table Constraints</b>.  - <code>&lt;oper-property&gt;</code> indicates the properties of the <b>Operation</b>. <code>&lt;oper-property&gt; ::= 'redefines' &lt;oper-name&gt;   'query'   'ordered'   'unique'   &lt;oper-constraint&gt;</code>  where:  * <code>redefines &lt;oper-name&gt;</code> means that the <b>Operation</b> redefines an inherited <b>Operation</b> identified by <code>&lt;oper-name&gt;</code>. It does not apply for <b>Table Constraints</b>.  * <code>query</code> means that the <b>Operation</b> does not change the state of the system. It does not apply for <b>Table Constraints</b>.  * <code>ordered</code> means that the values of the return parameter are ordered. It does not apply for <b>Table Constraints</b>.  * <code>unique</code> means that the values returned by the parameter have no duplicates. It does not apply for <b>Table Constraints</b>.  * <code>&lt;oper-constraint&gt;</code> is a constraint that applies to the <b>Operation</b>. It does not apply for <b>Table Constraints</b>.  * <code>&lt;parameter-list&gt;</code> is a list of parameters of the <b>Operation</b> in the following format:  <code>&lt;parameter&gt; [ '*' ] [ '&lt;direction&gt;' ] [ '&lt;type-expression&gt;' ] [ '&lt;multiplicity&gt;' ] [ '=' ] [ '&lt;default&gt;' ] [ '{ ' &lt;parm-property&gt; [ '*' ] } ] ]</code>  The parameters of a <code>&lt;&lt;Unique&gt;&gt;</code> and <code>&lt;&lt;PK&gt;&gt;</code> <b>Operations</b> are the names and data types of the attributes of the table on which this constraint is applied. The names of the parameters of a <code>&lt;&lt;FK&gt;&gt;</code> are a composition of the referenced table name and the name of the referenced attribute in this table.  The <code>&lt;&lt;Check&gt;&gt;</code> operation does not have any parameters. The predicate, which has to be evaluated is represented by an OCL expression.  where:  - <code>&lt;direction&gt;</code> ::= 'in'   'out'   'inout' (defaults to 'in' if omitted). It does not apply for <b>Table Constraints</b>.  - <code>&lt;parameter-name&gt;</code> is the name of the parameter.  - <code>&lt;type-expression&gt;</code> is an expression that specifies the type of the parameter.  - <code>&lt;multiplicity&gt;</code> is the multiplicity of the parameter. It does not apply for <b>Table Constraints</b>.  - <code>&lt;default&gt;</code> is an expression that defines the value specification for the default value of the parameter. It does not apply for <b>Table Constraints</b>.  - <code>&lt;parm-property&gt;</code> indicates additional property values that apply to the parameter. It does not apply for <b>Table Constraints</b>.</p>

Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Relationship	<p><b>Description:</b>  A dependency of any kind between <b>Tables</b> in a data model is called a <b>Relationship</b>. A <b>Relationship</b> in the relational model is represented as a set of primary and foreign key <b>Attributes</b> and/or <b>Table Constraints</b> of the related <b>Tables</b>.  Every relationship is between a parent and a child table, where a parent table must have a primary key defined. The child table creates a foreign key column and foreign key constraint to address the parent table. Thus, each <b>Relationship</b> is an uni-directional relationship in the relational model.  A non-identifying <b>Relationship</b> is a <b>Relationship</b> between two independent <b>Tables</b>. The foreign key of the child <b>Table</b> does not contain all of the primary key columns of the parent <b>Table</b>.  An identifying <b>Relationship</b> is a <b>Relationship</b> between two dependent <b>Tables</b>, where the child <b>Table</b> cannot exist without the parent <b>Table</b>. All of the primary keys of the parent <b>Table</b> become both primary and foreign key Columns in the child <b>Table</b>.</p> <p><b>UML Standard Elements:</b>  Relationships are represented in the UML Data Profile as a combination of stereotyped <b>Associations</b> and foreign and primary key <b>Attributes</b> and/or <b>Table Constraints</b>. In the relational model associations are not realized as instances of associations, but are represented by the PK and FK constraints in the related tables. Therefore, the <b>Associations</b> in the UML Data Profile are used only for better readability of the model.  According to Ambler the following stereotypes for associations can be used for the physical level: «Identifying» and «Non-Identifying».  «Aggregation» and «Composition» and «Uni-directional» are also proposed by Ambler as stereotypes for the physical model relationships. Since in the relational model a relationship is implemented always as a uni-directional relationship there is no need of the «Uni-directional» stereotyped <b>Association</b>.  «Subtype» is a further stereotype from Ambler which is however more suitable for the conceptual model. In the physical model we have to distinguish between subtypes and subtables.  All of these stereotyped <b>Associations</b> can be used. However, they do not carry any information for the implementation. They are used only for better readability of the model. The implementation specifics of each of these kinds of relations are represented as foreign key <b>Table Constraints</b>.</p>

Table B.1: Relational Concepts

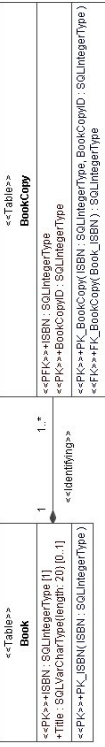
SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<p><b>Relationship</b></p>	<p><b>Stereotypes:</b>                      «Identifying», «Non-Identifying» <b>Associations</b>                      «Aggregation» and «Composition» <b>Associations</b></p> <p><b>Constraint:</b>                      none</p> <p><b>Notation:</b></p>  <pre> classDiagram     class Book {         ISBN : SQLIntegerType [1]         Title : SQLVarCharType(length: 20) [0..1]     }     class BookCopy {         ISBN : SQLIntegerType         BookCopyID : SQLIntegerType     }     Book "1" -- "1..*" BookCopy : «Identifying»     </pre>

Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Stored Procedure	<p><b>Description:</b></p> <p><b>Stored Procedures</b> are user-defined routines, which can be invoked to perform some predefined sequence of operations in the database. A <b>Stored Procedure</b> consists of a signature and a body. The signature determines the parameters of a <b>Stored Procedure</b>. A <b>Stored Procedure</b> does not have a return value. The parameters of a <b>Stored Procedure</b> can be OUT, IN or INOUT. The body contains the implementations of a <b>Stored Procedure</b>. The implementation can be given in SQL or in another programming language, called a host language. Therefore, we can distinguish between SQL and external <b>Stored Procedures</b>. The definition of a <b>Stored Procedure</b> requires the input of further characteristics, such as:</p> <ul style="list-style-type: none"> <li>- LANGUAGE (ADA   C   COBOL   SQL   JAVA   PASCAL   FORTRAN   MUMPS)</li> <li>- PARAMETER STYLE (SQL   GENERAL   JAVA)</li> <li>- SPECIFIC (Name of the Routine)</li> <li>- NOT DETERMINISTIC   DETERMINISTIC etc.</li> </ul> <p>see Standard for more detailed description.</p> <p><b>UML Standard Elements:</b></p> <p>Rational's Profile does not model stored procedures. Scott Ambler suggests the representation of Stored Procedures as operations of a stereotyped class «StoredProcedures». The signatures of all stored procedures of a database (or a schema) are represented in this class. Another variant also proposed by Ambler is to use the stereotype «Stored Procedure» for the definition of each stored procedure, defined in a class representing the database.</p> <p><b>Stereotypes:</b></p> <ul style="list-style-type: none"> <li>«StoredProcedures» <b>Class</b></li> <li>«StoredProcedure» <b>Operation</b></li> </ul> <p><b>Constraints:</b></p> <ul style="list-style-type: none"> <li>- The «StoredProcedures» <b>Class</b> does not have any attributes.</li> <li>- All operations of the «StoredProcedures» <b>Class</b> are «Stored Procedure» operations.</li> <li>- «StoredProcedures» cannot be redefined.</li> </ul>

Table B.1: Relational Concepts  
Description, UML Standard Elements, Stereotypes, Constraints, Notation

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Stored Procedure	<p><b>Notation:</b></p> <p>«Stored Procedures» <b>Class:</b> here comes an image</p> <p>«Stored Procedure» <b>Operations have the following notation:</b>  <math>\langle \langle \text{visibility} \rangle \langle \text{name} \rangle \langle \text{parameter-list} \rangle \langle \text{body} \rangle \langle \text{return-type} \rangle \langle \text{oper-property} \rangle \langle \text{oper-property} \rangle^* \{ \}</math></p> <p>where:</p> <ul style="list-style-type: none"> <li>- <math>\langle \text{visibility} \rangle</math> is the visibility of the <b>Operation</b>. <math>\langle \text{visibility} \rangle ::= '+' \mid '-' \mid \# \mid '\sim'</math>. For «StoredProcedures» it is always public.</li> <li>- <math>\langle \text{name} \rangle</math> is the name of the «StoredProcedure» <b>Operation</b>.</li> <li>- <math>\langle \text{return-type} \rangle</math> is the type of the return result parameter if the <b>Operation</b> has one defined. For «StoredProcedures» it is always void.</li> <li>- <math>\langle \text{oper-property} \rangle</math> indicates the properties of the <b>Operation</b>. <math>\langle \text{oper-property} \rangle ::= \text{'redefines' } \langle \text{oper-name} \rangle \mid \text{'query' } \mid \text{'ordered' } \mid \text{'unique' } \mid \langle \text{oper-constraint} \rangle</math></li> <li>  <math>\text{'language' } \langle \text{language name} \rangle \mid \text{'parameter style' } (\text{SQL} \mid \text{GENERAL} \mid \text{JAVA}) \mid \text{'specific' } \langle \text{routine name} \rangle \mid \text{'deterministic' } \mid \text{'not deterministic' } \mid \text{'no SQL' } \mid \text{'contains SQL' } \mid \text{'reads SQL data' } \mid \text{'modifies SQL' } \mid \text{'returns null on null input' } \mid \text{'called on null input' } \mid \text{'transform group' } (\langle \text{group name} \rangle \mid \langle \text{group specification list} \rangle) \mid \text{'static dispatch' } \mid \text{'external' } \langle \text{external procedure name} \rangle</math></li> </ul> <p>where:</p> <ul style="list-style-type: none"> <li>* redefines <math>\langle \text{oper-name} \rangle</math> means that the <b>Operation</b> redefines an inherited <b>Operation</b> identified by <math>\langle \text{oper-name} \rangle</math>. It is not used for «StoredProcedures».</li> <li>* query means that the <b>Operation</b> does not change the state of the system. Does not apply for «StoredProcedures».</li> <li>* ordered means that the values of the return parameter are ordered. Does not apply for «StoredProcedures».</li> <li>* unique means that the values returned by the parameter have no duplicates. Does not apply for «StoredProcedures».</li> <li>* <math>\langle \text{oper-constraint} \rangle</math> is a constraint that applies to the <b>Operation</b>. It is not used for «StoredProcedures».</li> <li>* language <math>\langle \text{language name} \rangle</math></li> <li>* parameter style (SQL   GENERAL   JAVA)</li> <li>* specific <math>\langle \text{routine name} \rangle</math></li> <li>* deterministic   not deterministic</li> <li>* no SQL   contains SQL   reads SQL data   modifies SQL</li> <li>* returns null on null input   called on null input</li> <li>* transform group (<math>\langle \text{group name} \rangle \mid \langle \text{group specification list} \rangle</math>)</li> <li>* static dispatch</li> <li>* external <math>\langle \text{external procedure name} \rangle</math></li> <li>* <math>\langle \text{parameter-list} \rangle</math> is a list of parameters of the <b>Operation</b> in the following format:  <math>\langle \text{parameter-list} \rangle ::= \langle \text{parameter} \rangle \langle \text{parameter} \rangle^* \{ \}</math></li> <li>* <math>\langle \text{parameter} \rangle ::= \langle \text{direction} \rangle \langle \text{parameter-name} \rangle \langle \text{multiplicity} \rangle \langle \text{type-expression} \rangle \langle \text{default} \rangle \langle \text{parameter-property} \rangle \langle \text{parameter-property} \rangle^* \{ \}</math></li> </ul>

Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Triggers	<p><b>Description:</b>  <b>Triggers</b> represent a combination of some kind of listener function and a response action. The listener awaits that a certain event in the database takes place (for example, that a table is manipulated). And the action performs some kind of database operations before or after the event has taken place. A <b>Trigger</b> comprises the following building blocks:</p> <ul style="list-style-type: none"> <li>- Activation time: BEFORE, AFTER; Shows if the trigger function has to be executed before or after the trigger event takes place.</li> <li>- Activation event: A trigger event which causes the trigger function to be executed can be an INSERT, UPDATE, DELETE on a basic table.</li> <li>- Granularity: A trigger function can be activated for each row in manipulation SQL statement or for each SQL statement. FOR EACH ROW, FOR EACH STATEMENT</li> <li>- Condition: An additional condition for the execution of a trigger function can be specified with a WHERE CLAUSE.</li> <li>- Action: This represents the body of the triggered function, which can consist of one or more SQL commands.</li> </ul> <p><b>UML Standard Elements:</b>  Rational's Profile deals with triggers as with <b>Table Constraints</b>. They are represented as stereotyped <b>Operations</b> of a «Table». Ambler suggests also the modeling of triggers as <b>Operations</b> with the stereotype «Trigger». The rest of the parameters are represented as modifiers.</p> <p><b>Stereotypes:</b>  «Trigger» <b>Operation</b></p> <p><b>Constraints:</b>  none</p>



Table B.1: Relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Triggers	<p><b>Notation:</b>  <code>&lt;visibility&gt; &lt;name&gt; ' (' &lt;parameter-list&gt;   ')' [ ':' &lt;return-type&gt; ] [ '{' &lt;oper-property&gt;   '*' &lt;oper-property&gt;   '*' &lt;oper-property&gt; } ] ]</code></p> <p>where:  - <code>&lt;visibility&gt;</code> is the visibility of the <b>Operation</b>. <code>&lt;visibility&gt;</code> ::= '+'   '-'   '#'   '~'. For <code>&lt;&lt;Trigger&gt;&gt;</code> it is always public.  - <code>&lt;name&gt;</code> is the name of the <code>&lt;&lt;Trigger&gt;&gt;</code> <b>Operation</b>.  - <code>&lt;return-type&gt;</code> does not apply for <code>&lt;&lt;Trigger&gt;&gt;</code>.  - <code>&lt;oper-property&gt;</code> indicates the properties of the <code>&lt;&lt;Trigger&gt;&gt;</code> <b>Operation</b>. <code>&lt;oper-property&gt;</code> ::= 'redefines'   <code>&lt;oper-name&gt;</code>   'query'   'ordered'   'unique'   <code>&lt;oper-constraint&gt;</code>   '(before'   'after')   '(on insert'   'on delete'   'on update')   '(for each row'   'for each table')   <code>&lt;action&gt;</code></p> <p>where:  * redefines <code>&lt;oper-name&gt;</code> means that the <b>Operation</b> redefines an inherited <b>Operation</b> identified by <code>&lt;oper-name&gt;</code>. It is not used for <code>&lt;&lt;Triggers&gt;&gt;</code>.  * query means that the <b>Operation</b> does not change the state of the system. Does not apply for <code>&lt;&lt;Triggers&gt;&gt;</code>.  * ordered means that the values of the return parameter are ordered. Does not apply for <code>&lt;&lt;Triggers&gt;&gt;</code>.  * unique means that the values returned by the parameter have no duplicates. Does not apply for <code>&lt;&lt;Triggers&gt;&gt;</code>.  * <code>&lt;oper-constraint&gt;</code> is a constraint that applies to the <b>Operation</b>. It can be used to formulate an additional condition for the trigger. The condition is represented in SQL.  * before   after  * on insert   on delete   on update  * for each row   for each table  * <code>&lt;action&gt;</code> is represented in SQL.  * <code>&lt;parameter-list&gt;</code> is a list of parameters of the <b>Operation</b>. Does not apply for <code>&lt;&lt;Triggers&gt;&gt;</code>.</p>

Table B.2: Object-relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<b>Structured Types, Typed Table</b>	<p data-bbox="317 1115 341 1648"><b>Description, UML Standard Elements:</b></p> <p data-bbox="349 1061 373 1648">The basic concepts of the SQL object model are:</p> <ul data-bbox="397 300 588 1648" style="list-style-type: none"> <li data-bbox="397 300 461 1648">• structured user-defined types: encapsulate the elements of a complex data type as well as its semantics in the form of behavior (methods of the data type)</li> <li data-bbox="477 300 541 1648">• typed tables: provide a mechanism to store values (instances) of user-defined types, and represent relationships between data types and their instances.</li> <li data-bbox="557 300 588 1648">• routines: enable the integration of methods for representing the behavior of the data types.</li> </ul> <p data-bbox="604 300 668 1648"><b>Structured Types</b> used to define value types are represented as specializations of the data type UML metaclass (see SQL Data Types below).</p> <p data-bbox="676 300 740 1648"><b>Structured Types</b> used to define classes of objects identified by their object IDs (OIDs) are represented in general as «ObjectType» stereotyped UML class. In order to differentiate between abstract (not instantiable) <b>Structured Types</b> and instantiable (combined with a <b>Typed Table</b>) <b>Structured Types</b> the specializations of «ObjectType», «NonInstantiableObjectType» and «InstantiableObjectType» are used, respectively.</p> <p data-bbox="748 300 812 1648">A structured type can be defined as final or not final. This characteristic can be represented as a property of the «ObjectType» class.</p> <p data-bbox="820 300 916 1648">A <b>Typed Table</b>, which is defined as a root table in an inheritance hierarchy can have all constraints defined for a «Table». Subtable are not allowed to have a Primary Key Constraint.</p>

Table B.2: Object-relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Structured Types, Typed Table	<p data-bbox="284 640 312 1648"><b>Stereotypes:</b></p> <p data-bbox="316 775 344 1648">«InstantiableObjectType»: Structured Type Instantiable + Typed Table</p> <p data-bbox="347 864 376 1648">«NonInstantiableObjectType»: Structured Type Not Instantiable</p> <p data-bbox="400 1469 429 1648"><b>Constraints:</b></p> <ul data-bbox="432 293 528 1648" style="list-style-type: none"> <li>- Subtable are not allowed to have a Primary Key Constraint.</li> <li>- The operations of an «ObjectType» <b>Class</b> can be only stereotyped operations, corresponding to the <b>Table Constraint, Method or Trigger</b> concept.</li> </ul> <p data-bbox="552 1503 580 1648"><b>Notation:</b></p> <div data-bbox="624 763 820 1178" style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre data-bbox="632 831 812 1167"> &lt;&lt;InstantiableObjectType&gt;&gt; <b>ObjectTypeName</b> +Attribute1 : SQLIntegerType +Attribute2 : SQLVarCharType(length: 20) +Method1() </pre> </div>

Table B.2: Object-relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Structured Type Attributes	<p data-bbox="284 1115 309 1648"><b>Description, UML Standard Elements:</b></p> <p data-bbox="316 300 437 1648">The <b>Attributes</b> of a <b>Structured Type</b> are represented as <b>Properties</b> of the corresponding stereotyped <b>Class</b> «ObjectType». As data types, all SQL specific stereotyped dataType <b>Classifiers</b> can be used (see dataType below). A stereotyped Property «Attribute» can be used to represent the attributes, whereby the explicit notation in the graphical representation of this stereotype can be omitted.</p> <p data-bbox="443 300 596 1648">If the <b>Structured Type</b> is instantiable (used in combination with a <b>Typed Table</b>) additional options and integrity constraints for attributes can be defined. These correspond to the Column Constraints defined in the relational concept <b>Column</b> above. If the structured type is a subtype then no Primary Key Attributes are allowed. Instantiable <b>Structured Type</b> can assign also constraints for the whole <b>Typed Table</b>. These correspond to the Table Constraints defined above.</p> <p data-bbox="628 1464 683 1648"><b>Stereotypes:</b> «Attribute»</p> <p data-bbox="715 1464 769 1648"><b>Constraints:</b> - If the Class of the Attributes is a Subclass the attribute cannot be a Primary key.</p> <p data-bbox="801 1352 855 1648"><b>Notation:</b> see <b>Structured Types</b></p>

Table B.2: Object-relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<b>Structured Type Method</b>	<p data-bbox="284 645 308 1653"><b>Description, UML Standard Elements, Stereotypes, Constraints, Notation</b></p> <p data-bbox="316 645 339 1653"><b>Description, UML Standard Elements:</b></p> <p data-bbox="347 293 499 1653">The attributes and operations of the <b>Class</b> correspond to attributes and methods of the <b>Structured Type</b>. A method of a <b>Structured Type</b> is an SQL-invoked routine. There are three kinds of methods: SQL-invoked constructor methods, instance SQL-invoked methods and static SQL-invoked methods. All SQL-invoked methods are associated with a <b>Structured Type</b>, also known as the type of the method. Therefore, they are represented as operations of an «ObjectType» class of a stereotype «Method» and have an attribute type : enum {constructor, instance, static}.</p> <p data-bbox="531 1473 555 1653"><b>Stereotypes:</b></p> <p data-bbox="563 1361 587 1653">«Method» <b>Operation</b>.</p> <p data-bbox="619 1473 643 1653"><b>Constraints:</b></p> <p data-bbox="675 1507 699 1653"><b>Notation:</b></p> <p data-bbox="707 1025 730 1653">«Method» <b>Operations</b> have the following notation.</p> <p data-bbox="738 645 762 1653">[&lt;visibility&gt; &lt;name&gt; ']' [&lt;parameter-list&gt; ]' [&lt;return-type&gt; ]' [&lt;oper-property&gt; ]' [&lt;oper-property&gt; * ]' ]]</p> <p data-bbox="770 1585 794 1653">where:</p> <ul data-bbox="802 293 1248 1653" style="list-style-type: none"> <li>- &lt;visibility&gt; is the visibility of the <b>Operation</b>. &lt;visibility&gt; ::= '+'   '-'   '#'. For «Method» it is always public.</li> <li>- &lt;name&gt; is the name of the «Method» <b>Operation</b>.</li> <li>- &lt;return-type&gt; is the type of the return result parameter if the <b>Operation</b> has one defined.</li> <li>- &lt;oper-property&gt; indicates the properties of the <b>Operation</b>. &lt;oper-property&gt; ::= 'redefines' &lt;oper-name&gt;   'query'   'ordered'   'unique'   'operator style' (SQL   GENERAL   JAVA)   'specific' &lt;routine name&gt;   'deterministic'   'not deterministic'   '(no SQL'   'contains SQL'   'reads SQL data'   'modifies SQL'   'returns null on null input'   'called on null input'   'transformation group' (&lt;group name&gt;   &lt;group specification list&gt;)   'static dispatch'   'external' &lt;external procedure name&gt; where:       <ul style="list-style-type: none"> <li>* redefines &lt;oper-name&gt; means that the <b>Operation</b> redefines an inherited <b>Operation</b> identified by &lt;oper-name&gt;.</li> <li>* query means that the <b>Operation</b> does not change the state of the system. Does not apply for «Method».</li> <li>* ordered means that the values of the return parameter are ordered. Does not apply for «Method».</li> <li>* unique means that the values returned by the parameter have no duplicates. Does not apply for «Method».</li> <li>* &lt;oper-constraint&gt; is a constraint that applies to the <b>Operation</b>. It is not used for «Method».</li> <li>* constructor   instance   static</li> <li>* language &lt;language name&gt;</li> <li>* parameter style (SQL   GENERAL   JAVA)</li> <li>* specific &lt;routine name&gt;</li> <li>* deterministic   not deterministic</li> <li>* no SQL   contains SQL   reads SQL data   modifies SQL</li> <li>* returns null on null input   called on null input</li> <li>* transform group (&lt;group name&gt;   &lt;group specification list&gt;)</li> <li>* static dispatch</li> <li>* external &lt;external procedure name&gt;</li> </ul> </li> <li>- &lt;parameter-list&gt; is a list of parameters of the <b>Operation</b> in the following format:       <ul style="list-style-type: none"> <li>* &lt;parameter-list&gt; ::= &lt;parameter&gt; [&lt;parameter&gt; * ]</li> <li>* &lt;parameter&gt; ::= [&lt;direction&gt;] &lt;parameter-name&gt; ':' &lt;type-expression&gt; [&lt;multiplicity&gt; ] [&lt;default&gt; ] [&lt;parm-property&gt; ]' &lt;parm-property&gt; * ]' ]]</li> </ul> </li> </ul>

Table B.2: Object-relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<b>User-Defined Function</b>	<p><b>Description:</b>  A <b>User-Defined Function</b> is an SQL-invoked routine whose invocation returns a value. Every parameter of a <b>User-Defined Function</b> is an input SQL parameter, one of which may be designated as the result SQL parameter.</p> <p>An SQL-invoked routine can be an SQL routine or an external routine. An SQL routine is an SQL-invoked routine whose language clause specifies SQL. An external routine is one whose language clause does not specify SQL. The routine body of an external routine is an external body reference whose external routine name identifies a program written in some standard programming language other than SQL.</p> <p>Different SQL-invoked routines can have equivalent routine names. No two SQL-invoked functions in the same schema are allowed to have the same signature. No two SQL-invoked procedures in the same schema are allowed to have the same name and the same number of parameters.</p> <p>The definition of a <b>User-defined Function</b> requires the input of further characteristics:</p> <ul style="list-style-type: none"> <li>- LANGUAGE (ADA   C   COBOL   SQL   JAVA   PASCAL   FORTRAN   MUMPS)</li> <li>- PARAMETER STYLE (SQL   GENERAL   JAVA)</li> <li>- SPECIFIC (Name of the Routine)</li> <li>- NOT DETERMINISTIC   DETERMINISTIC</li> <li>- NO SQL   CONTAINS SQL   READS SQL DATA   MODIFIES SQL DATA</li> <li>- RETURNS NULL ON NULL INPUT   CALLED ON NULL INPUT</li> <li>- TRANSFORM GROUP FOR TYPE</li> <li>- STATIC DISPATCH</li> </ul> <p><b>UML Standard Elements:</b>  None of the existing UML Profiles for SQL consider the modeling of <b>User-Defined Functions</b>. They can be represented similarly as <b>Stored Procedures</b> as a stereotyped class «User-Defined Functions» containing only operations, corresponding to the user-defined functions.  The type (SQL or external) of the routine is represented by an additional stereotyped attribute of the operation.</p> <p><b>Stereotypes:</b>  «User-Defined Functions» <b>Class</b>.  «User-Defined Function» <b>Operation</b>.</p> <p><b>Constraint:</b></p> <ul style="list-style-type: none"> <li>- The «User-Defined Functions» <b>Class</b> does not have any attributes.</li> <li>- All operations of the «User-Defined Functions» <b>Class</b> are «User-Defined Functions».</li> </ul>

Table B.2: Object-relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
User-Defined Function	<p><b>Notation:</b></p> <p>«User-Defined Function» <b>Operation</b> have the following notation.</p> <pre> &lt;visibility&gt; &lt;name&gt; '({&lt;parameter-list&gt;})' ['{':&lt;return-type&gt; '}' {&lt;oper-property&gt;}* }]] </pre> <p>where:</p> <ul style="list-style-type: none"> <li>- &lt;visibility&gt; is the visibility of the <b>Operation</b>. &lt;visibility&gt; ::= '+'   '-'   '#'   '~'. For «StoredProcedures» it is always public.</li> <li>- &lt;name&gt; is the name of the «User-Defined Function» <b>Operation</b>.</li> <li>- &lt;return-type&gt; is the type of the return result parameter if the <b>Operation</b> has one defined.</li> <li>- &lt;oper-property&gt; indicates the properties of the <b>Operation</b>. &lt;oper-property&gt; ::= 'redefines' &lt;oper-name&gt;   'query'   'ordered'   'unique'   &lt;oper-constraint&gt;   'language' &lt;language name&gt;   'parameter style' (SQL   GENERAL   JAVA)   'specific' &lt;routine name&gt;   (deterministic   not deterministic)   (no SQL   contains SQL   reads SQL data   modifies SQL)   (returns null on null input   transform group' (&lt;group name&gt;   &lt;group specification list&gt;))   'static dispatch'   'external' &lt;external procedure name&gt;</li> </ul> <p>where:</p> <ul style="list-style-type: none"> <li>* redefines &lt;oper-name&gt; means that the <b>Operation</b> redefines an inherited <b>Operation</b> identified by &lt;oper-name&gt;.</li> <li>* query means that the <b>Operation</b> does not change the state of the system. Does not apply for «User-Defined Function».</li> <li>* ordered means that the values of the return parameter are ordered. Does not apply for «User-Defined Function».</li> <li>* unique means that the values returned by the parameter have no duplicates. Does not apply for «User-Defined Function».</li> <li>* &lt;oper-constraint&gt; is a constraint that applies to the <b>Operation</b>. It is not used for «User-Defined Function».</li> <li>* language &lt;language name&gt;</li> <li>* parameter style (SQL   GENERAL   JAVA)</li> <li>* specific &lt;routine name&gt;</li> <li>* deterministic   not deterministic</li> <li>* no SQL   contains SQL   reads SQL data   modifies SQL</li> <li>* returns null on null input   called on null input</li> <li>* transform group (&lt;group name&gt;   &lt;group specification list&gt;)</li> <li>* static dispatch</li> <li>* external &lt;external procedure name&gt;</li> <li>* &lt;parameter-list&gt; is a list of parameters of the <b>Operation</b> in the following format: <pre> &lt;parameter-list&gt; ::= &lt;parameter&gt; [',' &lt;parameter&gt;]* &lt;parameter&gt; ::= [&lt;direction&gt;] &lt;parameter-name&gt; ':' &lt;type-expression&gt; [!&lt;multiplicity&gt;] [!&lt;default&gt;] [!&lt;parm-property&gt;] [',' &lt;parm-property&gt;]* } ] </pre> </li> </ul>

Table B.2: Object-relational Concepts

SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
Structured Typed View	<p data-bbox="284 300 467 1653"><b>Description:</b> In object-relational databases <b>Typed Views</b> can be defined which can be regarded as <b>Views</b> for objects of <b>Tabed Tybles</b> instead of tuples of <b>Tables</b>. A <b>Typed View</b> is based on a predefined <b>Structured Typed</b>. The <b>Structured Types</b> used to define <b>Typed Views</b> have to be instantiatible. In contrast to <b>Views Typed Views</b> can have methods, their objects are identified with an <b>OID</b> and a <b>Typed View</b> can be defined as a subview of another <b>Typed View</b>.</p> <p data-bbox="475 300 759 1653">Integrity constraints for the <b>Typed View</b> can be defined through the <b>CHECK</b> clause, the <b>CASCADE</b> clause and the <b>LOCAL</b> clause. The <b>CHECK</b> clause assures that all changes carried out on the <b>Typed View</b> have to include the <b>WHERE</b> clause of the query used for creating the view. <b>CASCADE</b> or <b>LOCAL</b> can be used in order to determine the scope of the query constructing the <b>Typed View</b> in case that the base table of the view has other subtables. The <b>OID</b> of a <b>Typed View</b> cannot be system generated. It is generated in the query constructing the view, thus it can be user generated or derived. The query responsible for constructing the <b>Typed View</b> must return a <b>Structured Type</b> corresponding to the <b>Structured Type</b> of the <b>Typed View</b>. The list of attributes in the <b>SELECT</b> clause of the query has to begin with a constructor of the <b>OID</b>. The <b>FROM</b> clause of the query can contain only one table or view, i.e. no joins and groupings are allowed.</p> <p data-bbox="783 300 908 1653"><b>UML Standard Elements:</b> The combination of a <b>Structured Type</b> and a <b>Typed View</b> shall be represented as a «Typed View» class in the SQL:2003 profile. Analogously to <b>View</b> a «Derive» dependency between the «Typed View» and the «InstantiatibleObjectType», «Table» or «Typed View» used to create the view.</p> <p data-bbox="932 300 1027 1653"><b>Stereotypes:</b> «Typed View» «Derive»</p> <p data-bbox="1051 300 1115 1653"><b>Constraints:</b> none</p> <p data-bbox="1139 300 1203 1653"><b>Notation:</b> analogous to <b>View</b></p>



Table B.2: Object-relational Concepts

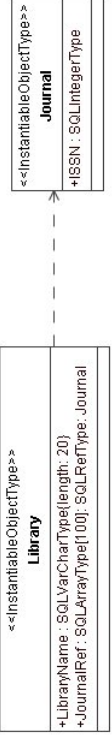
SQL Concept	Description, UML Standard Elements, Stereotypes, Constraints, Notation
<b>Inheritance</b>	<p><b>Description:</b> Inheritance can be defined in the object-relational model for user-defined types and for typed tables.</p> <p><b>UML Standard Elements:</b></p> <p><b>Stereotypes:</b> Generalization</p> <p><b>Constraint:</b> none</p> <p><b>Notation:</b> analogous to <b>Generalization</b> in PIM</p>
<b>References</b>	<p><b>Description:</b> Reference between objects of one <b>Structured Type</b> with objects of another are represented in the object-relational model as attributes of type REF of the <b>Structured Type</b>. Additionally in the UML profile associations can be used for better readability of the model, analogously to the <b>Relationship</b> concept in the relational model described above.</p> <p><b>UML Standard Elements:</b> Dependency</p> <p><b>Stereotypes:</b> «References»</p> <p><b>Constraints:</b> none</p> <p><b>Notation:</b></p>  <pre> classDiagram     class Library {         &lt;&lt;InstantiableObjectType&gt;&gt;         +LibraryName : SQLVarCharType(length: 20)         +JournalRef : SQLArrayType[100]     }     class Journal {         &lt;&lt;InstantiableObjectType&gt;&gt;         +ISSN : SQLIntegerType     }     Library ..&gt; Journal   </pre>

Table B.3: SQL Data Types in UML

Data Type	Description
SQLDataType	<b>Description:</b> «SQLDataType» is derived from «dataType».
SQLBasicDataType	<b>Description:</b> «SQLBasicDataType» is derived from «SQLDataType».
SQLComplexDataType	<b>Description:</b> «SQLComplexDataType» is derived from «dataType».
SQLBooleanType	<b>Description:</b> «SQLBooleanType» is derived from «SQLBasicDataType».
SQLSmallIntType	<b>Description:</b> «SQLSmallIntType» is derived from «SQLBasicDataType».
SQLIntegerType	<b>Description:</b> «SQLIntegerType» is derived from «SQLBasicDataType».
SQLBigIntType	<b>Description:</b> «SQLBigIntType» is derived from «SQLBasicDataType».
SQLDecimalType	<p><b>Description:</b></p> <p>«SQLDecimalType» is derived from «SQLBasicDataType». It has two attributes:</p> <ul style="list-style-type: none"> <li>- precision: SQLIntegerType</li> <li>- scale: SQLIntegerType</li> </ul> <p><b>Notation:</b></p> <p><i>SQLDecimalType</i> ::= '<i>SQLDecimalType</i> {precision: ' &lt;number&gt; ', scale: ' &lt;number&gt; }'</p> <p>where:</p> <ul style="list-style-type: none"> <li>- &lt;number&gt; is a value of type SQLIntegerType</li> </ul>
SQLNumericType	<p><b>Description:</b></p> <p>«SQLNumericType» is derived from «SQLBasicDataType». It has two attributes:</p> <ul style="list-style-type: none"> <li>- precision: SQLIntegerType</li> <li>- scale: SQLIntegerType</li> </ul> <p><b>Notation:</b></p> <p><i>SQLNumericType</i> ::= '<i>SQLNumericType</i> {precision: ' &lt;number&gt; ', scale: ' &lt;number&gt; }'</p> <p>where:</p> <ul style="list-style-type: none"> <li>- &lt;number&gt; is a value of type SQLIntegerType</li> </ul>
SQLFloatType	<p><b>Description:</b></p> <p>«SQLFloatType» is derived from «SQLBasicDataType». It has one attribute:</p> <ul style="list-style-type: none"> <li>- precision: SQLIntegerType</li> </ul> <p><b>Notation:</b></p> <p><i>SQLFloatType</i> ::= '<i>SQLFloatType</i> {precision: ' &lt;number&gt; }'</p> <p>where:</p> <ul style="list-style-type: none"> <li>- &lt;number&gt; is a value of type SQLIntegerType</li> </ul>

Table B.3: SQL Data Types

Data Type	Description
<b>SQLRealType</b>	<b>Description:</b> «SQLRealType» is derived from «SQLBasicDataType».
<b>SQLDoubleType</b>	<b>Description:</b> «SQLDoubleType» is derived from «SQLBasicDataType».
<b>SQLCharType</b>	<b>Description:</b> «SQLCharType» is derived from «SQLBasicDataType». It has one attribute: - length: SQLIntegerType <b>Notation:</b> <i>SQLCharType ::= 'SQLCharType {length: ' &lt;number&gt; }'</i> where: - <number> is a value of type SQLIntegerType
<b>SQLVarCharType</b>	<b>Description:</b> «SQLVarCharType» is derived from «SQLBasicDataType». It has one attribute: - length: SQLIntegerType <b>Notation:</b> <i>SQLVarCharType ::= 'SQLCharType {length: ' &lt;number&gt; }'</i> where: - <number> is a value of type SQLIntegerType
<b>SQLClobType</b>	<b>Description:</b> «SQLClobType» is derived from «SQLBasicDataType». It has one attribute: - size: SQLIntegerType <b>Notation:</b> <i>SQLClobType ::= 'SQLClobType {size: ' &lt;number&gt; }'</i> where: - <number> is a value of type SQLIntegerType
<b>SQLBitType</b>	<b>Description:</b> «SQLBitType» is derived from «SQLBasicDataType». It has one attribute: - length: SQLIntegerType <b>Notation:</b> <i>SQLBitType ::= 'SQLBitType {length: ' &lt;number&gt; }'</i> where: - <number> is a value of type SQLIntegerType

Table B.3: SQL Data Types

Data Type	Description
<b>SQLVarBitType</b>	<p><b>Description:</b> «SQLVarBitType» is derived from «SQLBasicDataType». It has one attribute: - length: SQLIntegerType</p> <p><b>Notation:</b> <i>SQLVarBitType</i> ::= '<i>SQLVarBitType</i> {length: &lt;number&gt; }' where: - &lt;number&gt; is a value of type SQLIntegerType</p>
<b>SQLBlobType</b>	<p><b>Description:</b> «SQLBlobType» is derived from «SQLBasicDataType». It has one attribute: - size: SQLIntegerType</p> <p><b>Notation:</b> <i>SQLBlobType</i> ::= '<i>SQLBlobType</i> {size: &lt;number&gt; }' where: - &lt;number&gt; is a value of type SQLIntegerType</p>
<b>SQLDateType</b>	<p><b>Description:</b> «SQLDateType» is derived from «SQLBasicDataType».</p>
<b>SQLTimeType</b>	<p><b>Description:</b> «SQLTimeType» is derived from «SQLBasicDataType».</p>
<b>SQLTimeStampType</b>	<p><b>Description:</b> «SQLTimeStampType» is derived from «SQLBasicDataType».</p>
<b>SQLIntervalType</b>	<p><b>Description:</b> «SQLIntervalType» is derived from «SQLBasicDataType».</p>
<b>SQLXMLType</b>	<p><b>Description:</b> «SQLXMLType» is derived from «SQLBasicDataType».</p>
<b>SQLRowType</b>	<p><b>Description:</b> A <b>ROW</b> type represents a composed attribute with a fixed number of elements, each of them can be of different data type. «SQLRowType» is derived from «SQLComplexDataType». The following properties are specified for this stereotype: - field: SQLFieldType[1..*] <b>Notation:</b> <i>SQLRowType</i> ::= '<i>SQLRowType</i> { &lt;field&gt; [ ; &lt;field&gt; ]* }' where: - &lt;field&gt; is a value of type SQLFieldType</p>

Table B.3: SQL Data Types

Data Type	Description
<b>SQLFieldType</b>	<p><b>Description:</b>  A <b>SQLFieldType</b> represents a field of an <b>SQLRowType</b>. <b>«SQLFieldType»</b> is derived from <b>«SQLComplexDataType»</b>.</p> <p>The following properties are specified for this stereotype:</p> <ul style="list-style-type: none"> <li>- name: <code>SQLVarCharType{length: 20}</code></li> <li>- type: <code>SQLVarCharType{length: 20}</code></li> </ul> <p><b>Notation:</b>  <code>SQLFieldType ::= 'SQLFieldType' { '&lt;name&gt;' : '&lt;datatype&gt;' }</code>  where:  - &lt;name&gt; is the name of the field, a value of type <code>SQLVarCharType</code>  - &lt;datatype&gt; is a name of a predefined <code>dataType</code></p>
<b>SQLArrayType</b>	<p><b>Description:</b>  An <b>Array</b> represents an indexed and bounded collection type (unbounded since SQL:2003). The elements of an <b>Array</b> can be of any data type except the <b>Array</b> type. It has a type of its elements and a number of elements. <b>«SQLArrayType»</b> is derived from <b>«SQLComplexDataType»</b>. Alternatively in this case also the multiplicity of the class attributes can be used, since <b>ARRAY</b> can have only one dimension in SQL.</p> <p>The following properties are specified for this stereotype:</p> <ul style="list-style-type: none"> <li>- dimension: <code>SQLIntegerType</code></li> <li>- datatype: <code>SQLVarCharType{length: 20}</code></li> </ul> <p><b>Notation:</b>  <code>SQLArrayType ::= 'SQLArrayType' [ '&lt;number&gt;' : '&lt;datatype&gt;' ]</code>  where:  - &lt;number&gt; is a value of type <code>SQLIntegerType</code>  - &lt;datatype&gt; is a name of a predefined <code>dataType</code> except <code>SQLArrayType</code></p>

Table B.3: SQL Data Types

Data Type	Description
SQLMultisetType	<p><b>Description:</b>            Since SQL:2003 it is possible to define also this type of collections in order to avoid the limits of arrays. «SQLMultisetType» is derived from «SQLComplexDataType». Alternatively in this case also the multiplicity of the class attributes can be used.</p> <p>The following properties are specified for this stereotype:</p> <ul style="list-style-type: none"> <li>- dimension: SQLIntegerType</li> <li>- datatype: SQLVarCharType{length: 20}</li> </ul> <p><b>Notation:</b>  <i>SQLMultisetType</i> ::= '<i>SQLMultisetType</i>:&lt;datatype&gt;            where:            &lt;datatype&gt; is a name of a predefined dataType</p>

Table B.3: SQL Data Types

Data Type	Description
<b>SQLRefType</b>	<p><b>Description:</b>  A <b>REF</b> represents a reference to an instance of a <b>Structured type</b> stored in a particular <b>Typed table</b>. An instance of a <b>REF</b> is thus an object identifier <b>OID</b>.  A reference can be defined as checked, which indicates that the existence of an object referred by this reference should be checked before inserting the reference in order to maintain referential integrity.  With the on delete set null clause for a reference the referential integrity for deleted referenced object is maintained.  The scope clause determines the typed table from which the objects can be referred by the reference. In the model defined here each object type defined as a «InstantiableObjectType» corresponds to exactly one <b>Typed Table</b>. Therefore the scope of the reference is uniquely identified through the name of the «InstantiableObjectType» class which it references.  The attributes isChecked:boolean, onDelete:boolean are therefore introduced for the stereotype «REF».  «SQLRefType» is derived from «SQLComplexDataType».  The following properties are specified for this stereotype:  - isChecked: SQLBooleanType  - onDelete: SQLBooleanType  - class: SQLVarCharType{length: 20}</p> <p><b>Notation:</b>  <i>SQLRefType</i> ::= '<i>SQLRefType</i>:&lt;class&gt; '{isChecked=' &lt;bool value&gt; ', onDelete=' &lt;bool value&gt; }'  where:  &lt;bool value&gt; ::= true'   false'  &lt;class&gt; is a name of a predefined <i>InstantiableObjectType</i></p>
<b>SQLStructType</b>	<p><b>Description:</b> «SQLStructType» is derived from «SQLComplexDataType». Can be used to define UDTs identified by value.</p>
<b>SQLDomainType</b>	<p><b>Description:</b> «SQLDomainType» is derived from «SQLComplexDataType». Can be used to define sets of allowed values, similarly to enumeration.</p>

## Appendix C

# Large Versions of Selected Figures



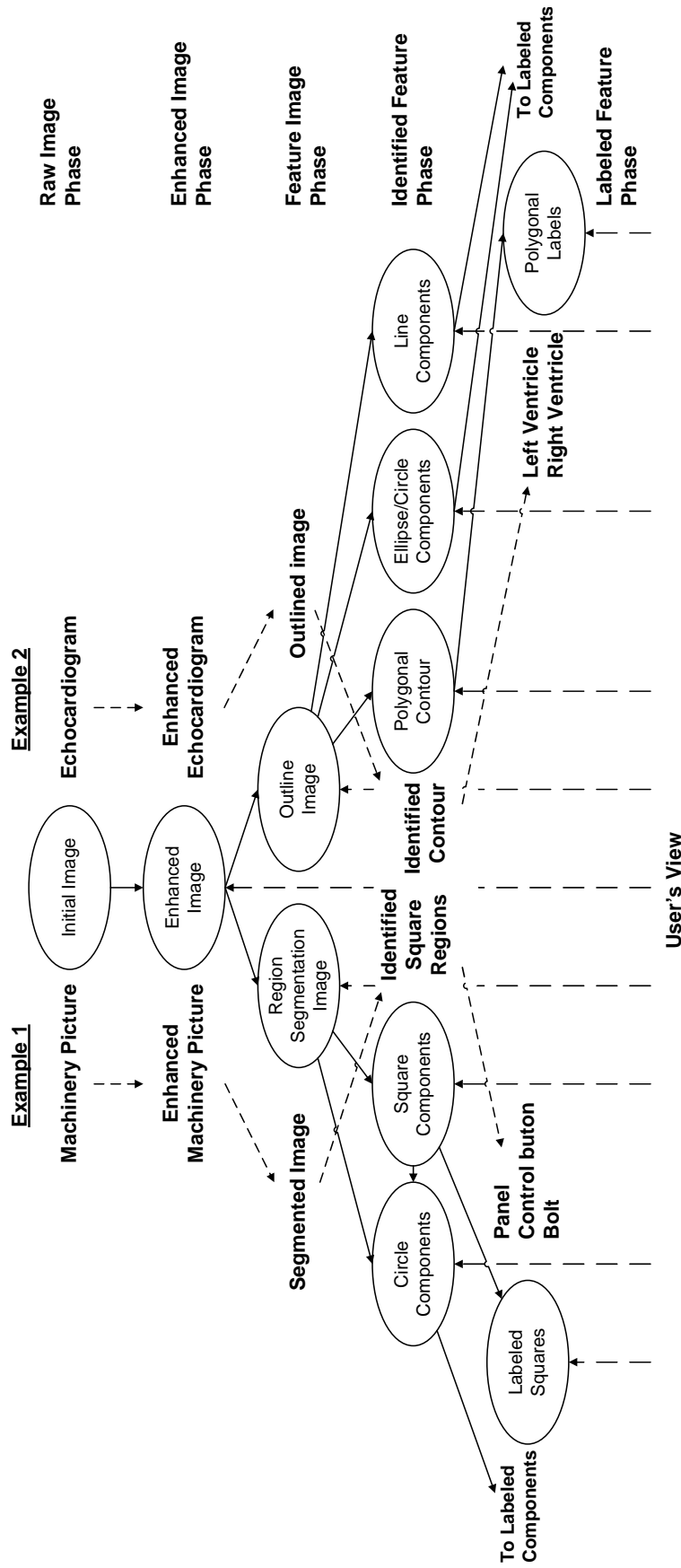


Figure C.1: MOODS image information processing system (based on [GMY93b])

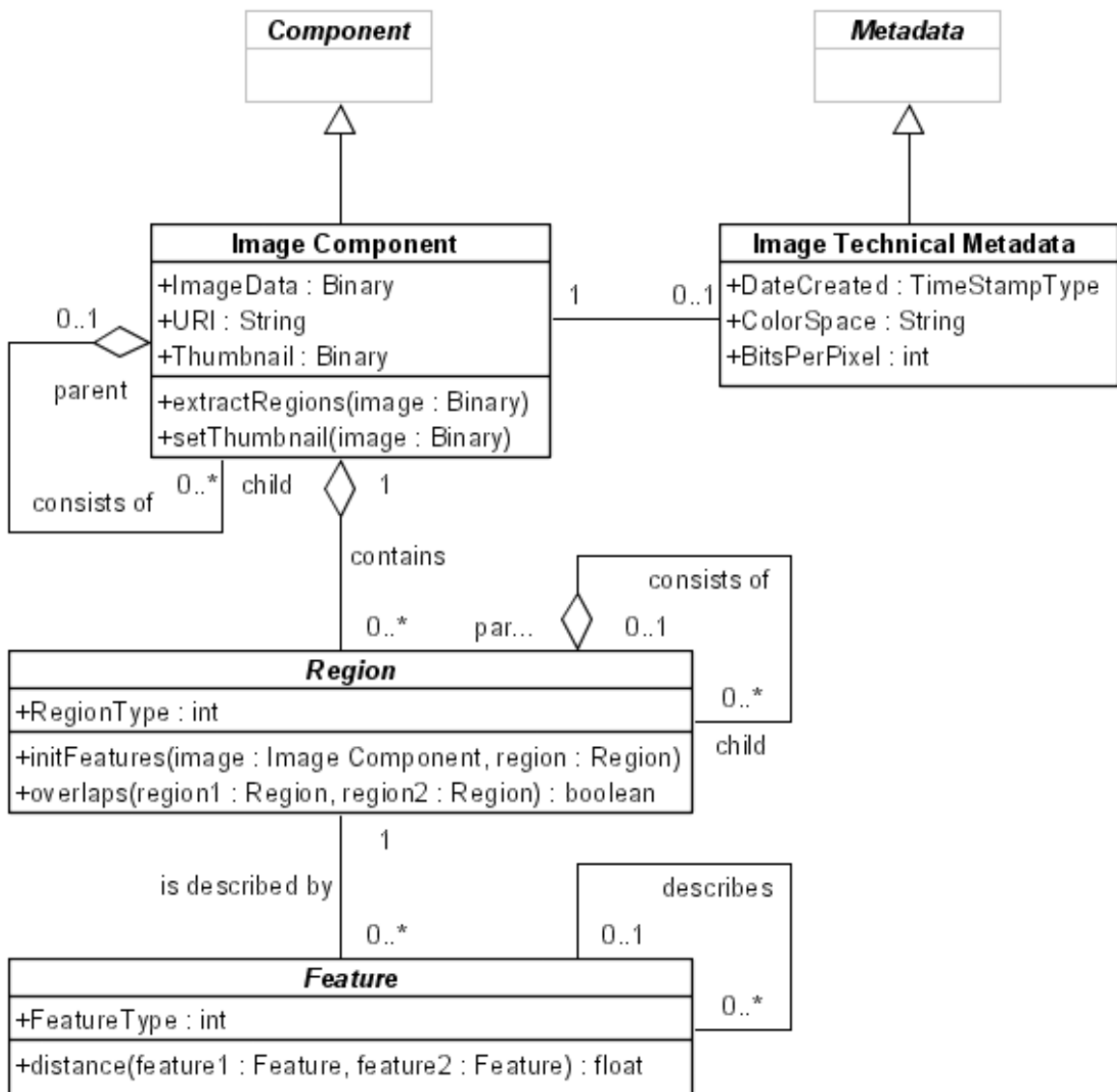


Figure C.2: Generic Image Database Model (from [IB05])

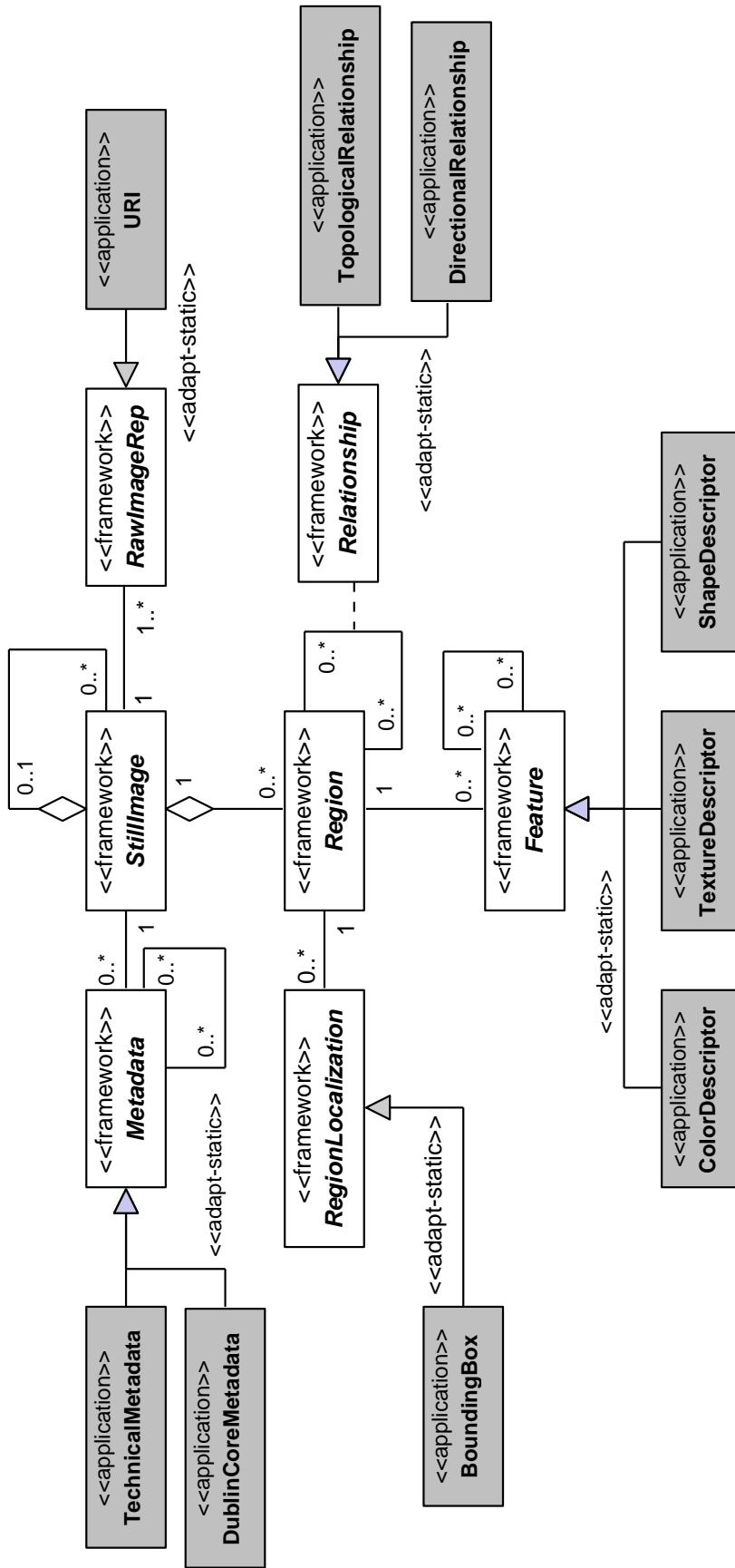


Figure C.3: Main framework classes and application specific *black box* classes of GiAComMo-IRS

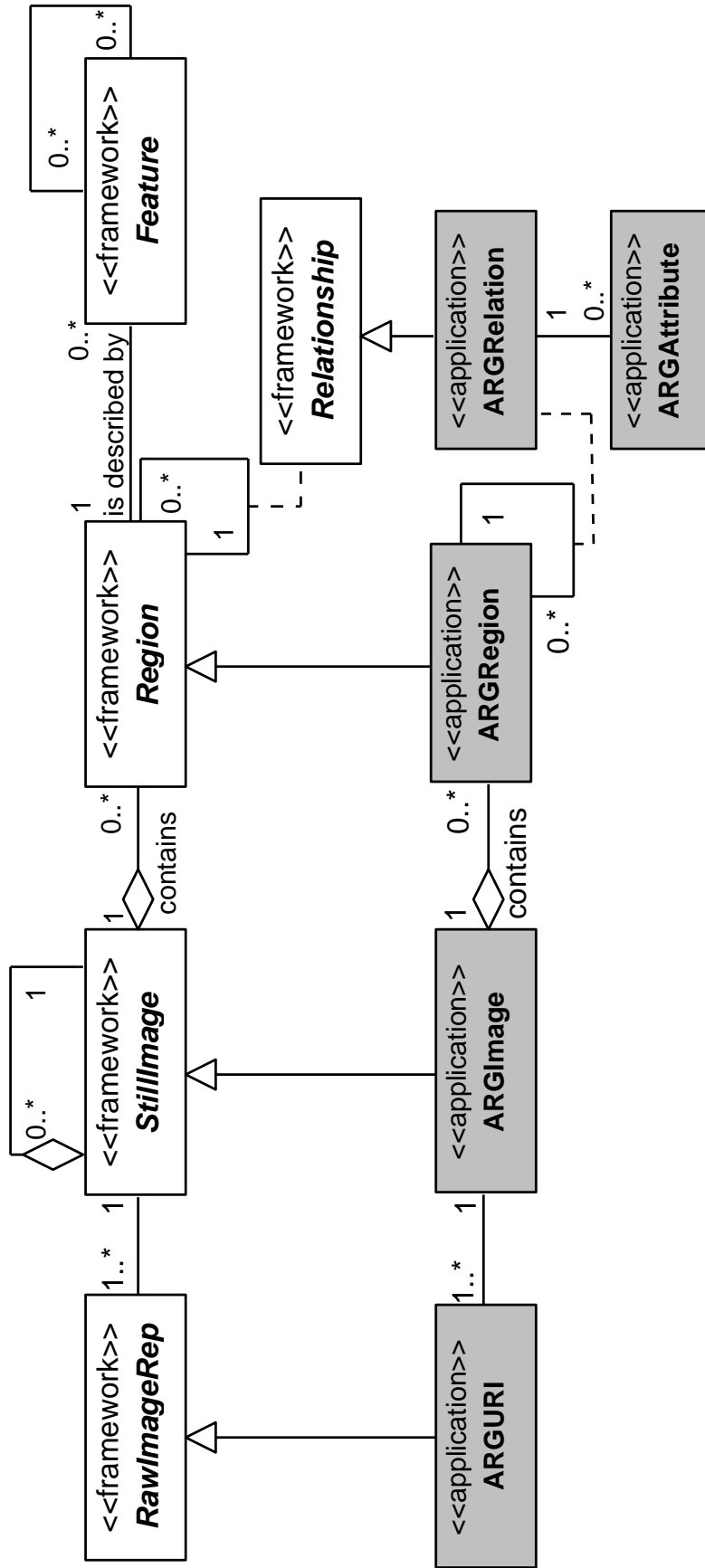


Figure C.4: Modeling Attribute Relational Graphs Image Representations

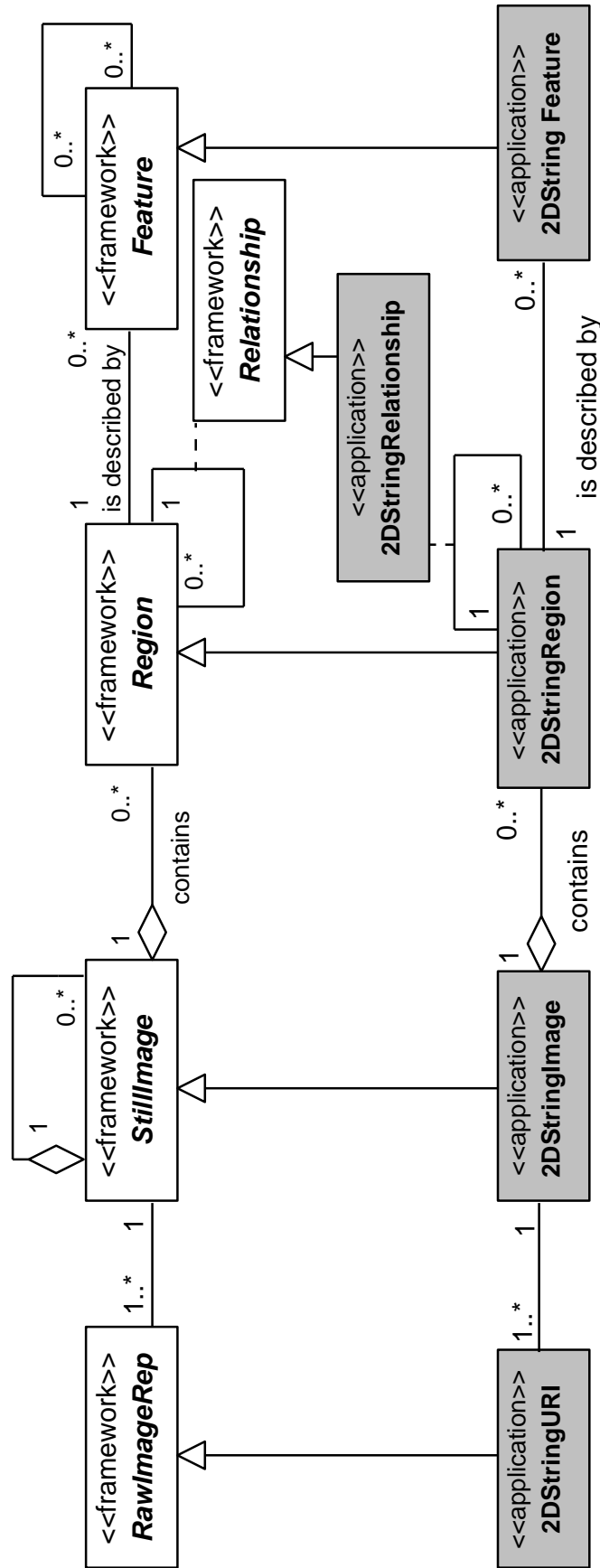


Figure C.5: Modeling 2D-Strings Image Representation

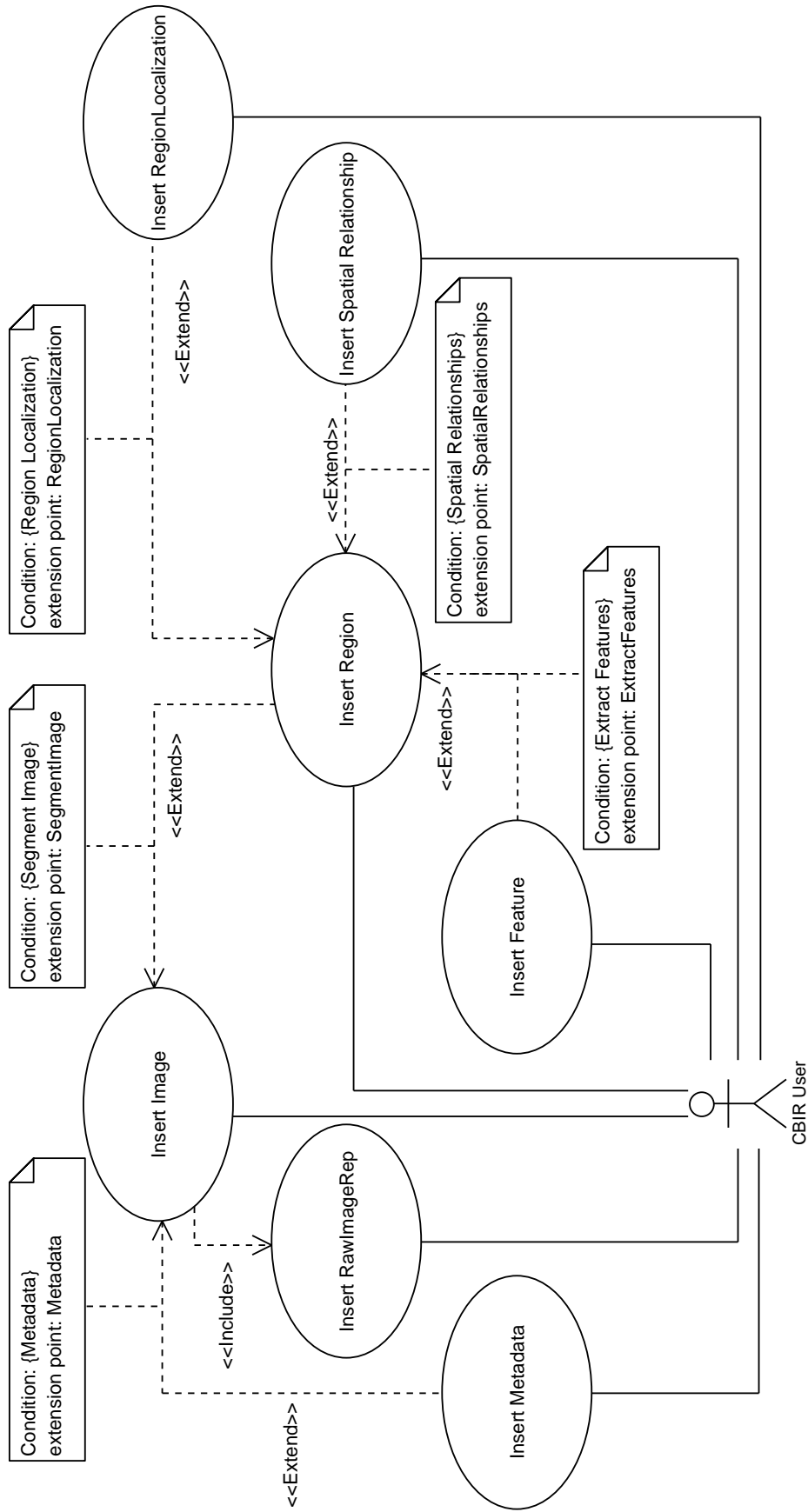


Figure C.6: Use Cases for the Insert operation

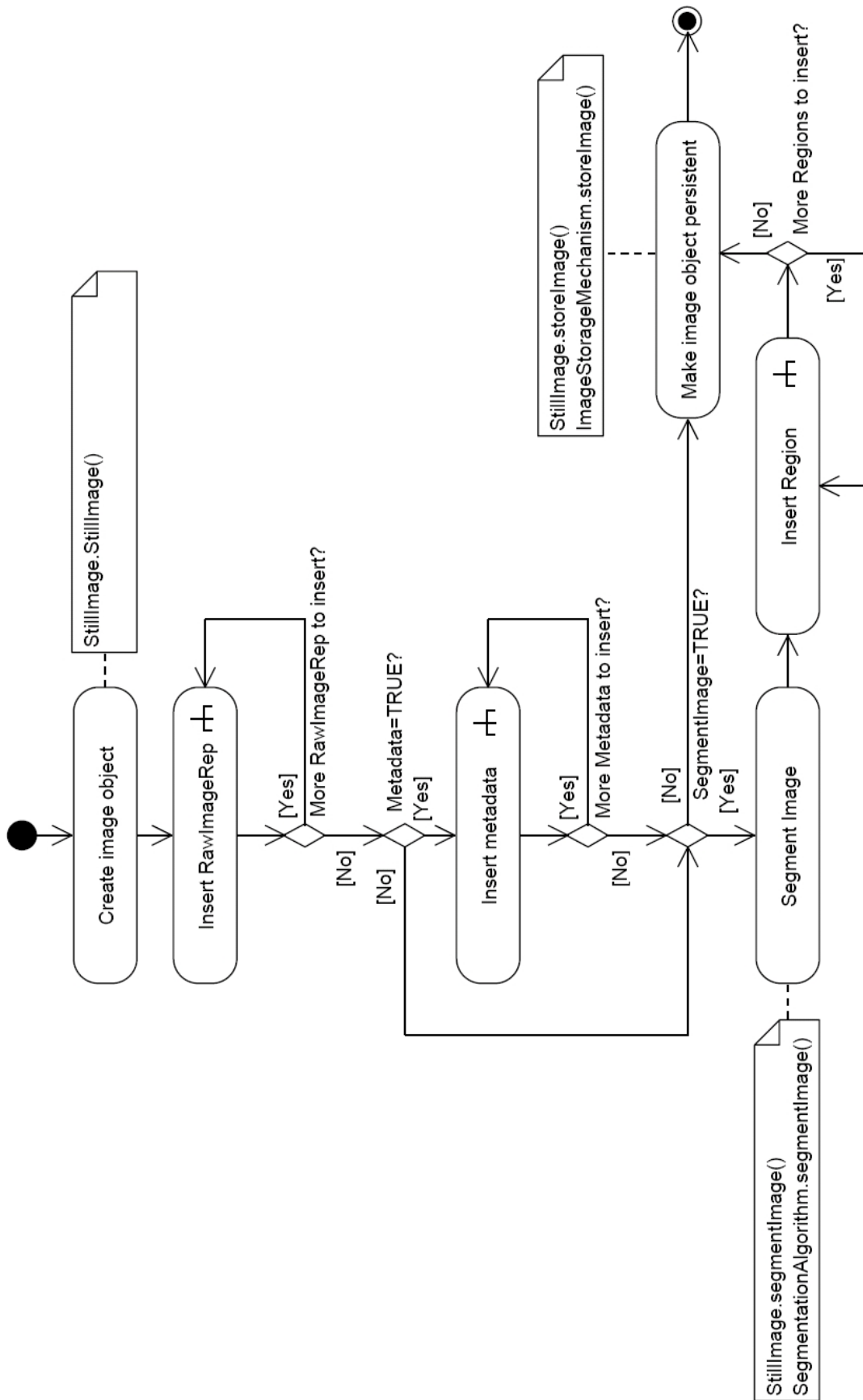


Figure C.7: Activity diagram for the Insert operation

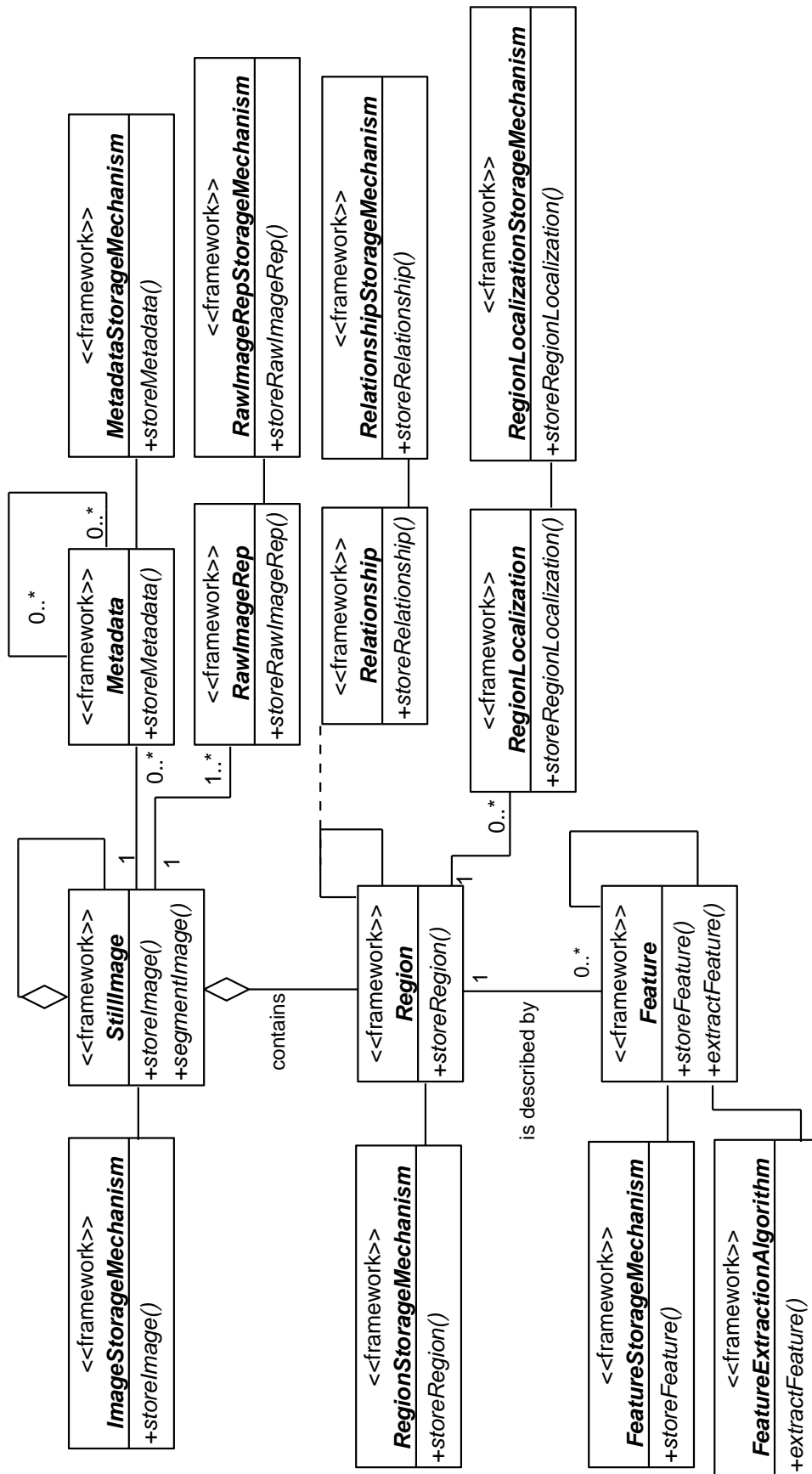


Figure C.8: Integration of the insert functionality in the framework model



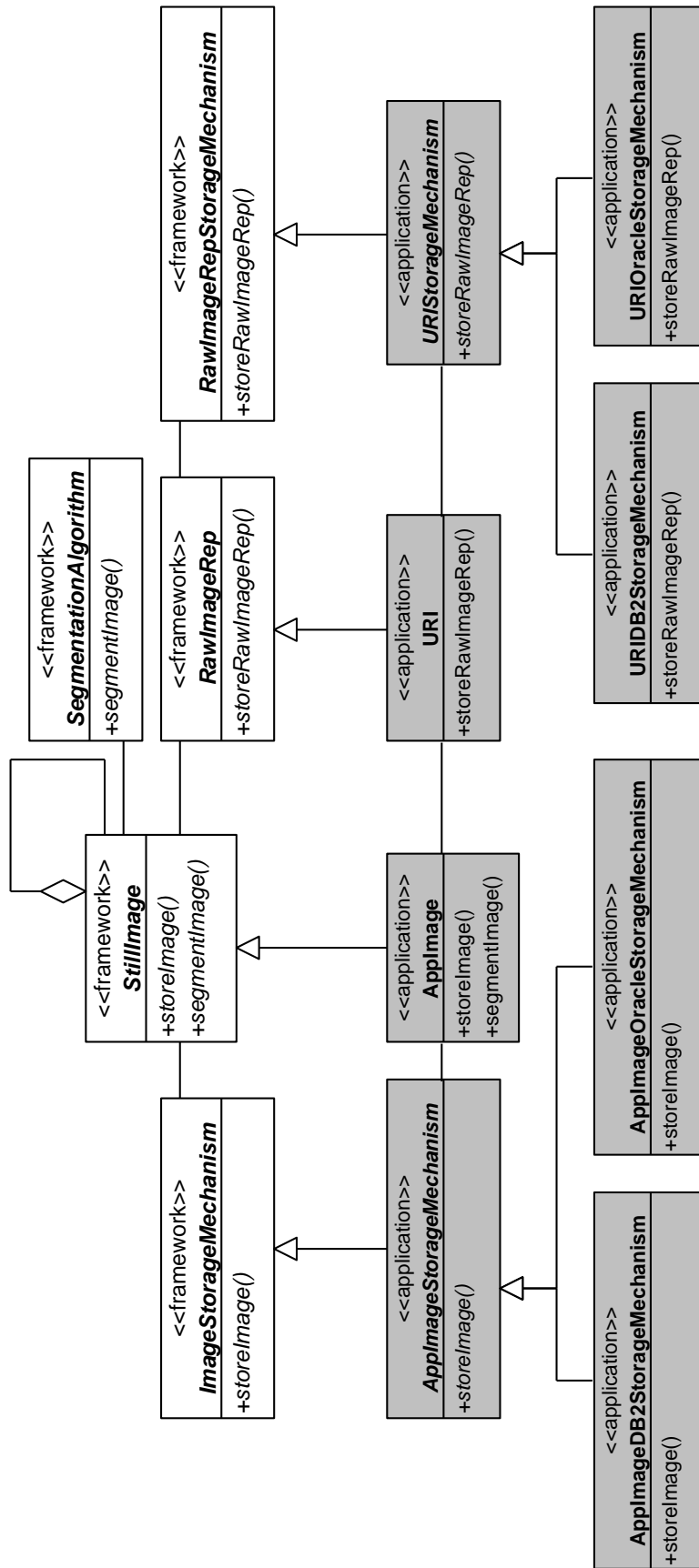


Figure C.9: Modeling functionality with the GiACoMo-IRS framework



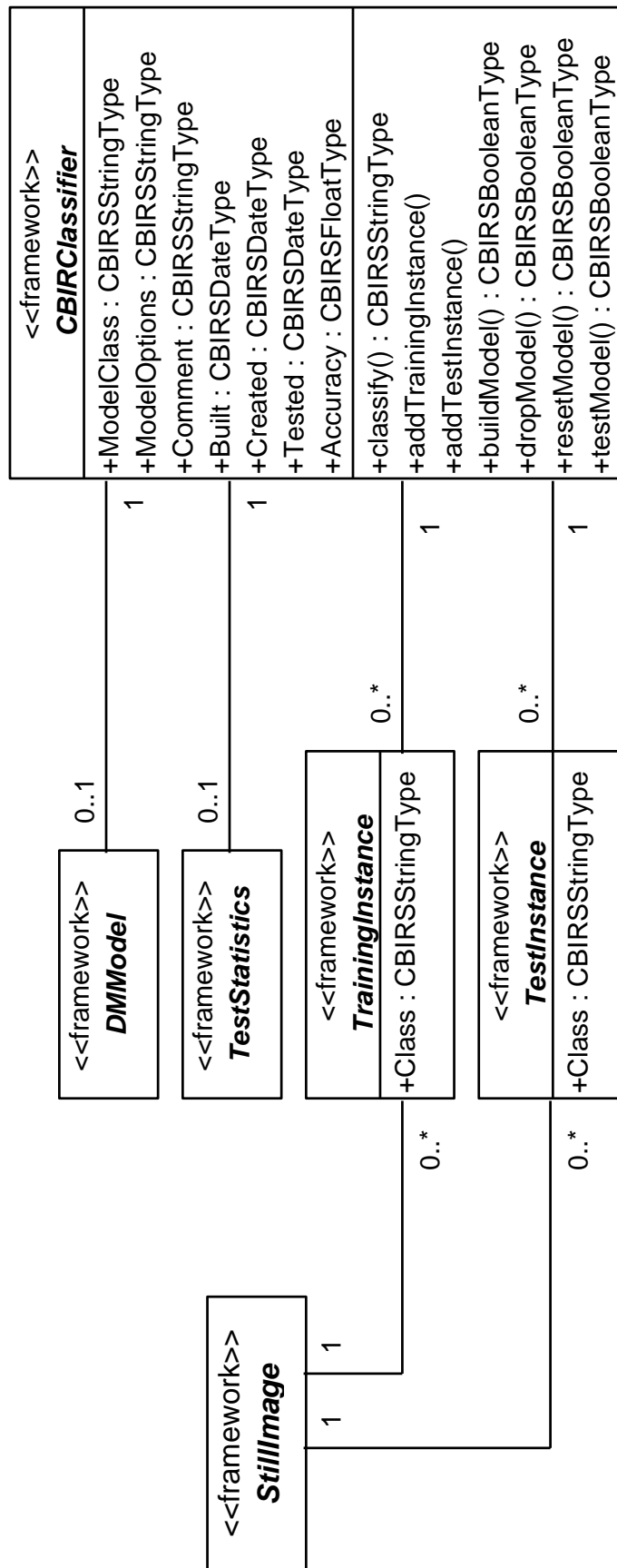


Figure C.11: Generic classes for the classification of images

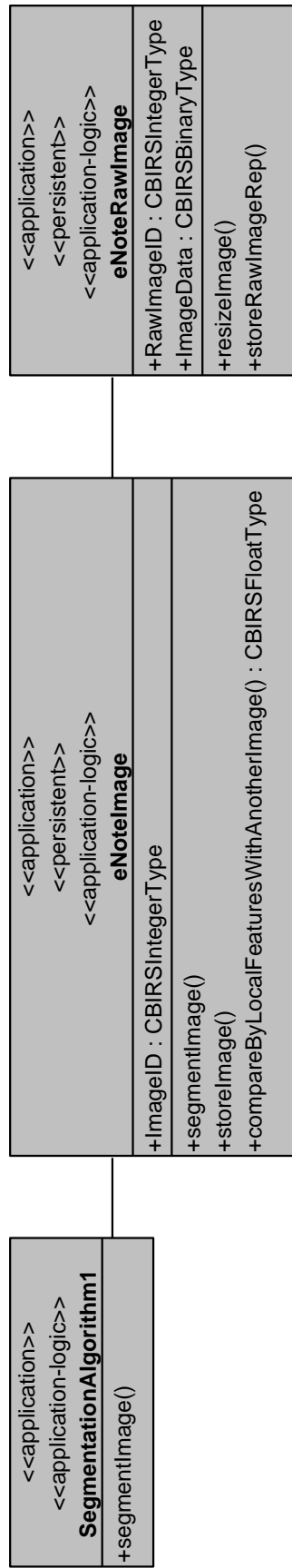


Figure C.12: Example for the deployment annotation

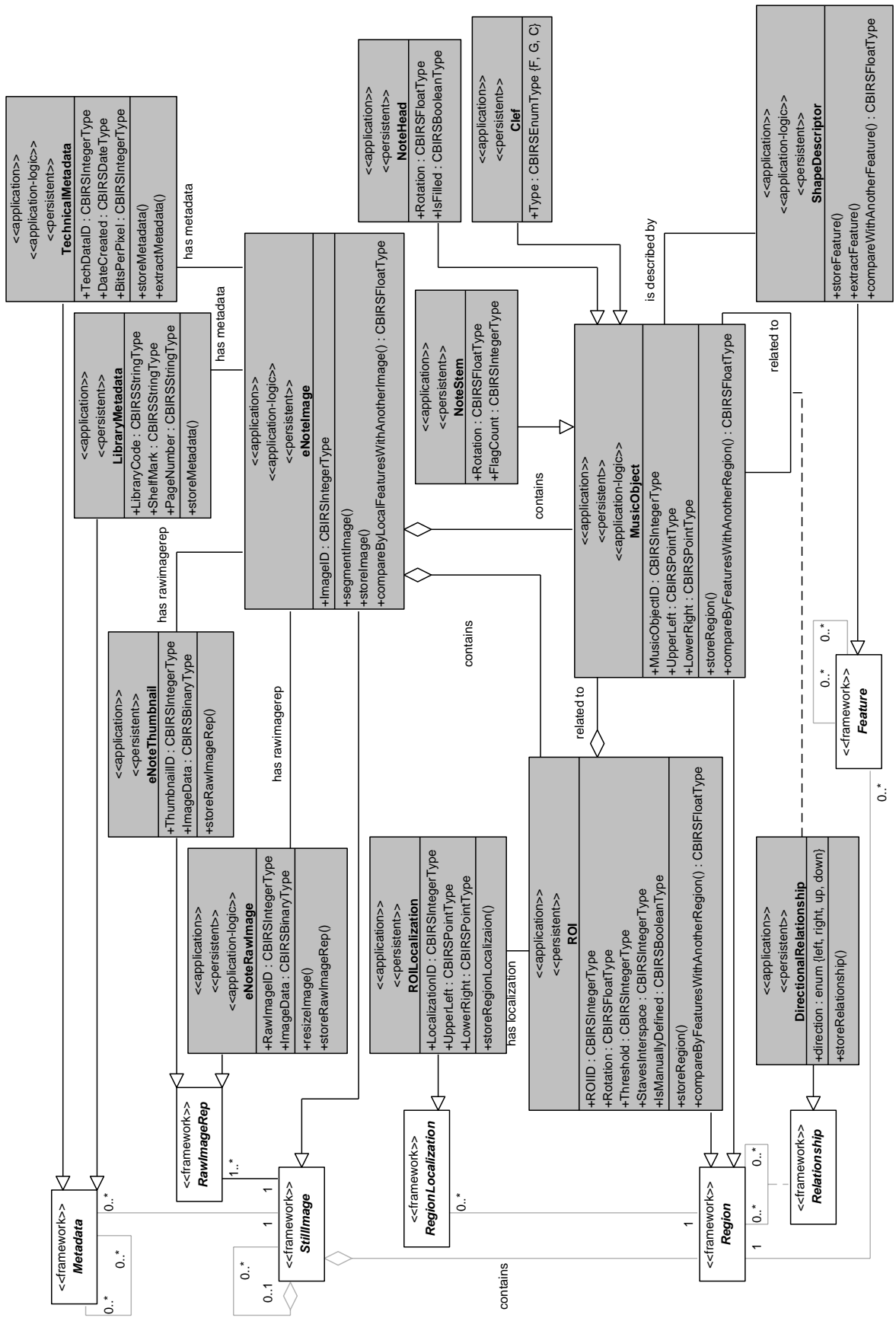


Figure C.13: eNoteHistory CBIR PIM

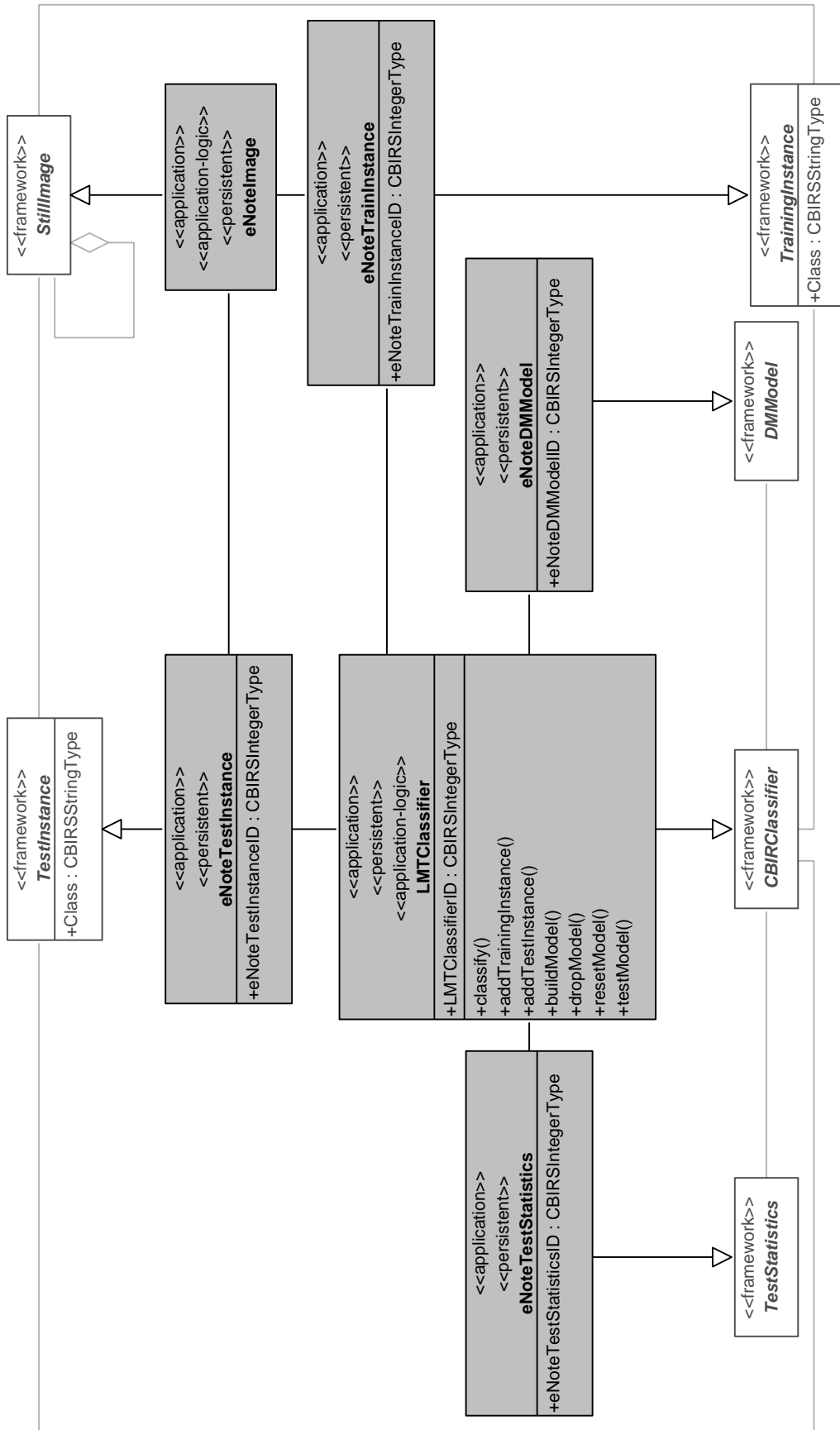


Figure C.14: eNoteHistory CBIR PIM - data mining part

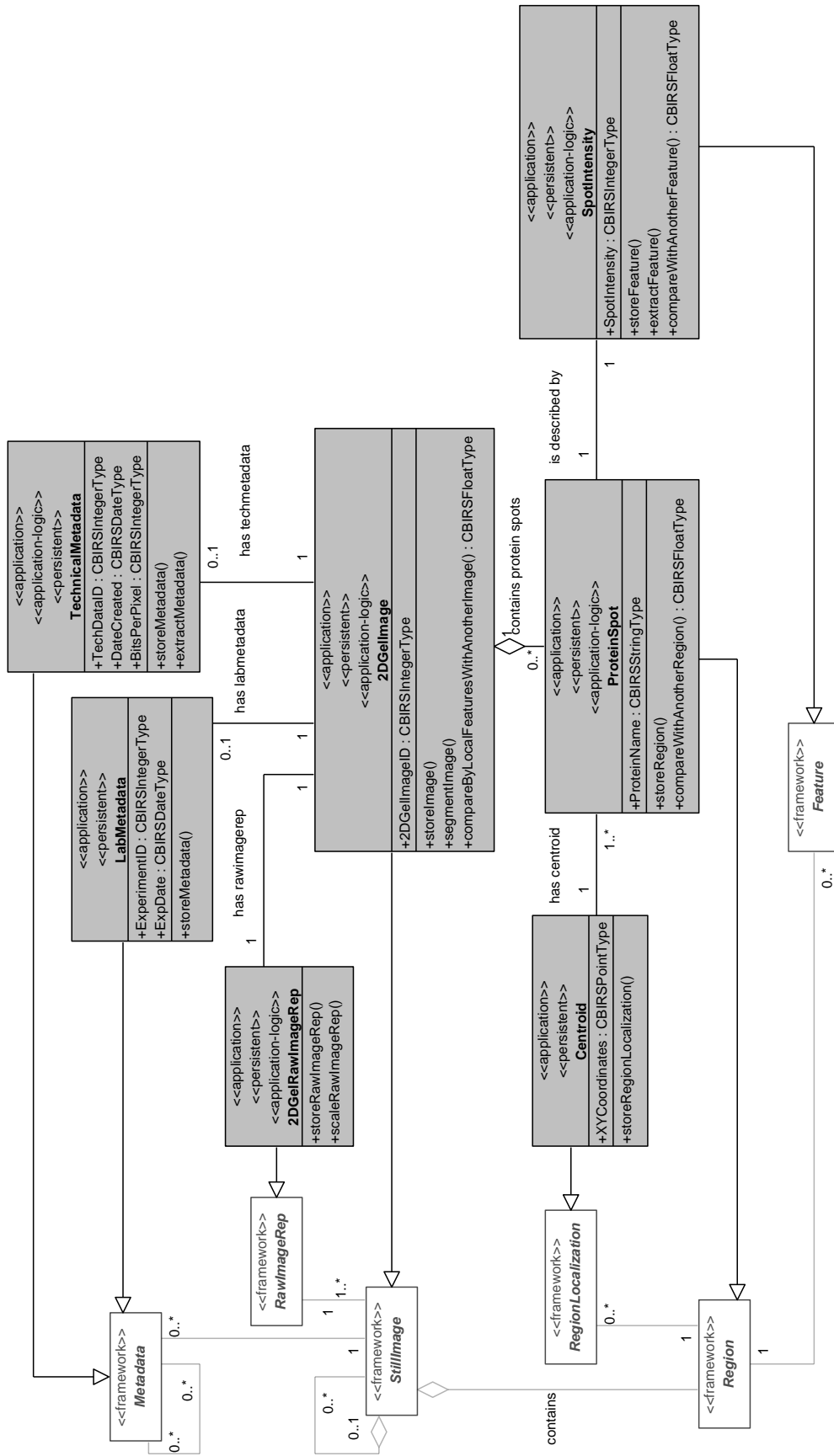


Figure C.15: CBIRS PIM for 2D-gel electrophoresis images

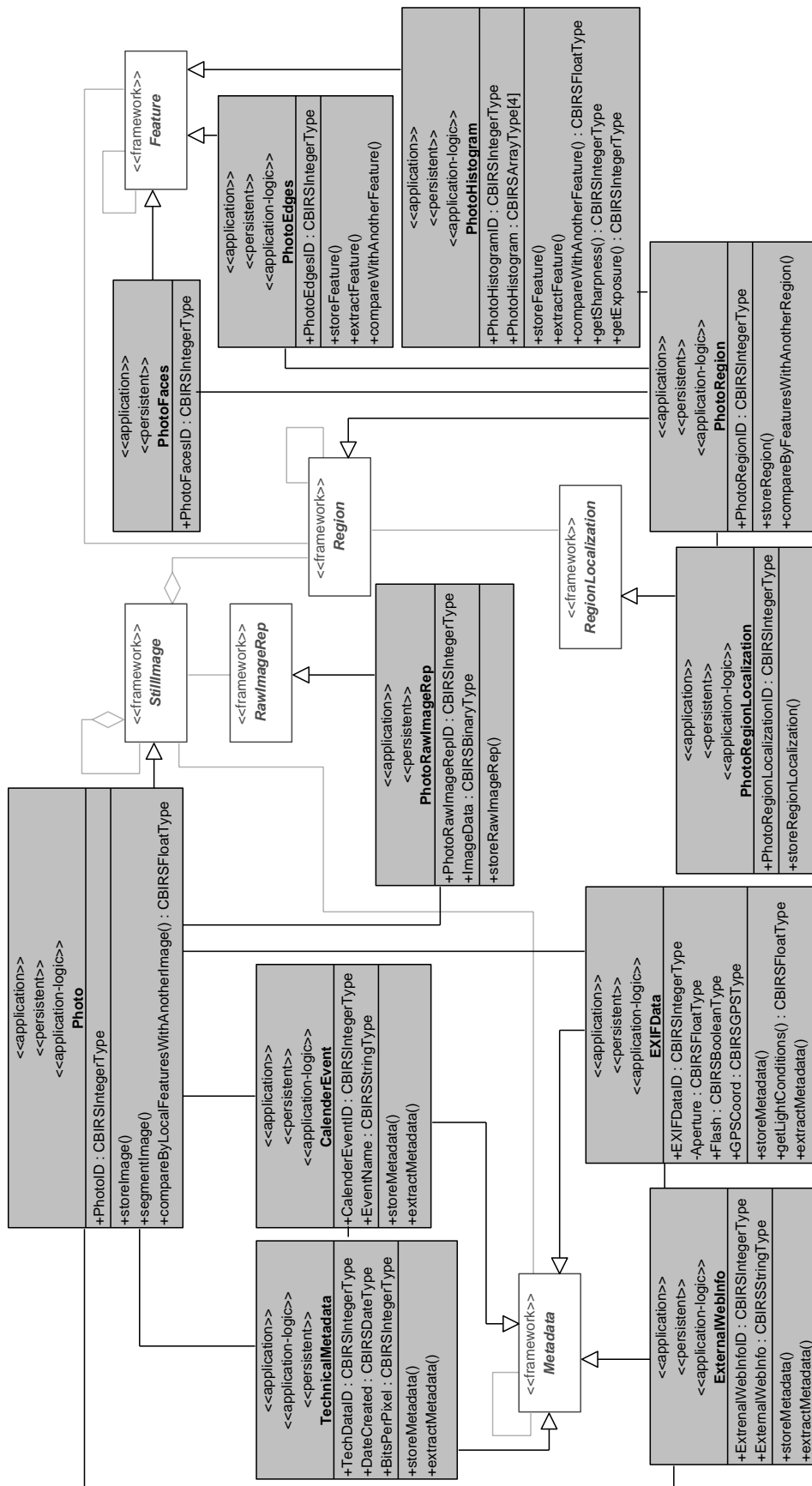


Figure C.16: CBIRS PIM for a photo annotation application



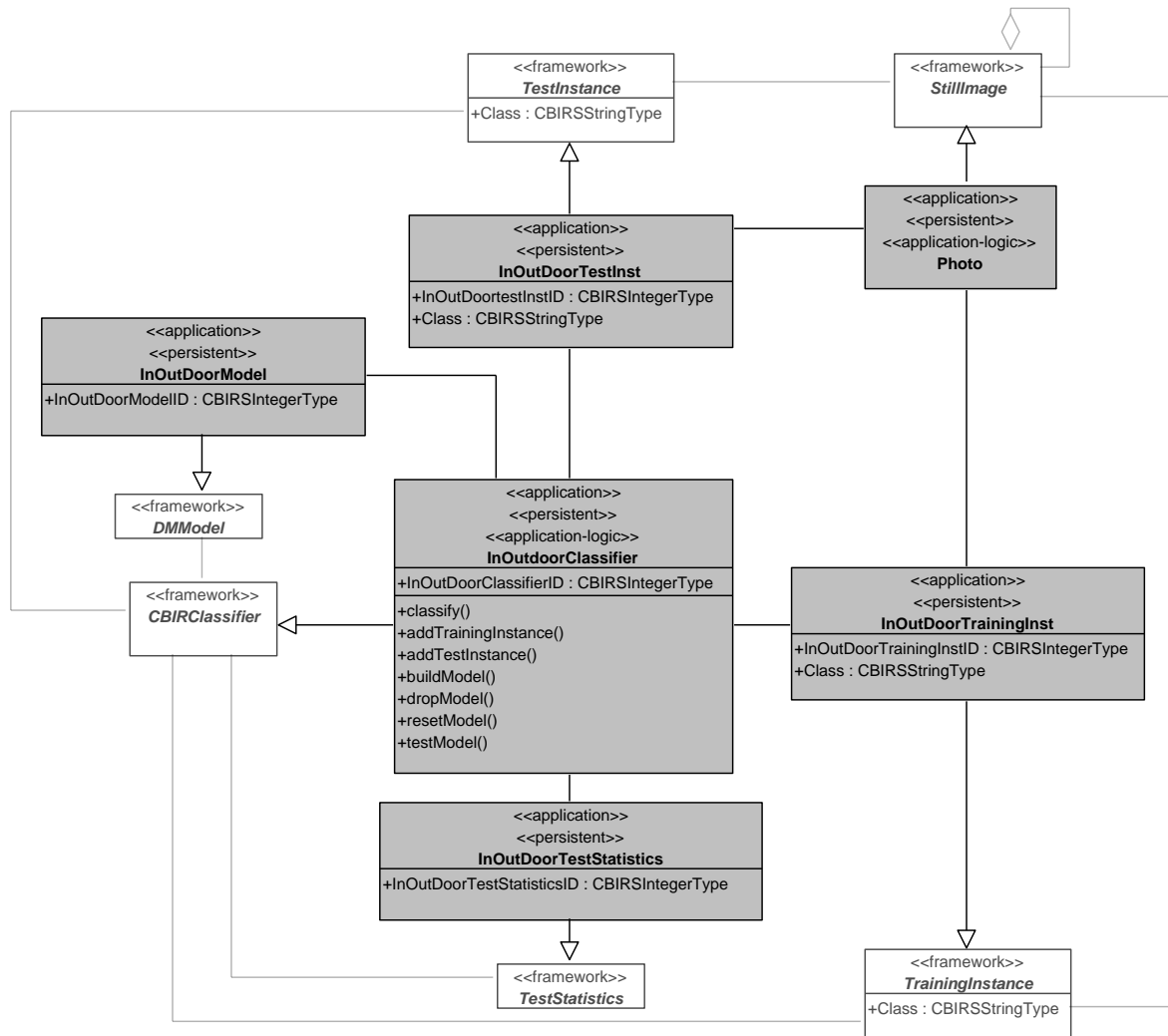


Figure C.17: CBIRS PIM for a photo annotation application - data mining part

## Appendix D

# Screenshots of the Image Database Generator Plug-In

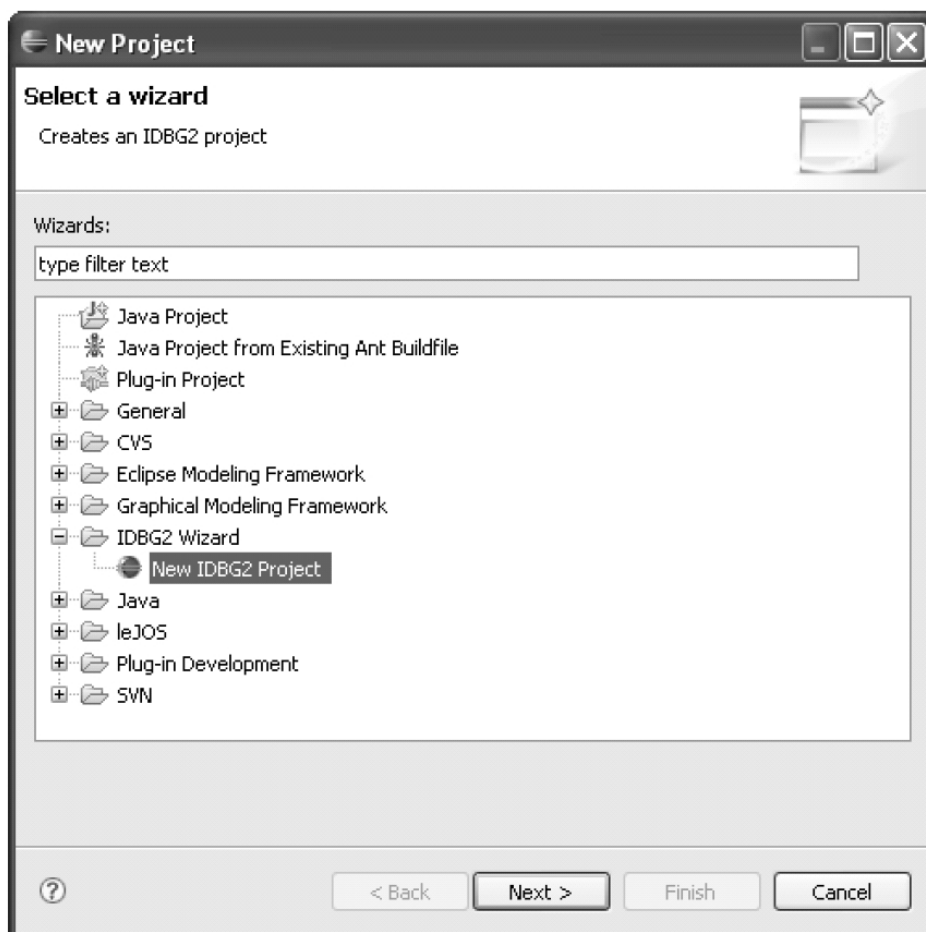


Figure D.1: Project Wizard for creating an IDBG project

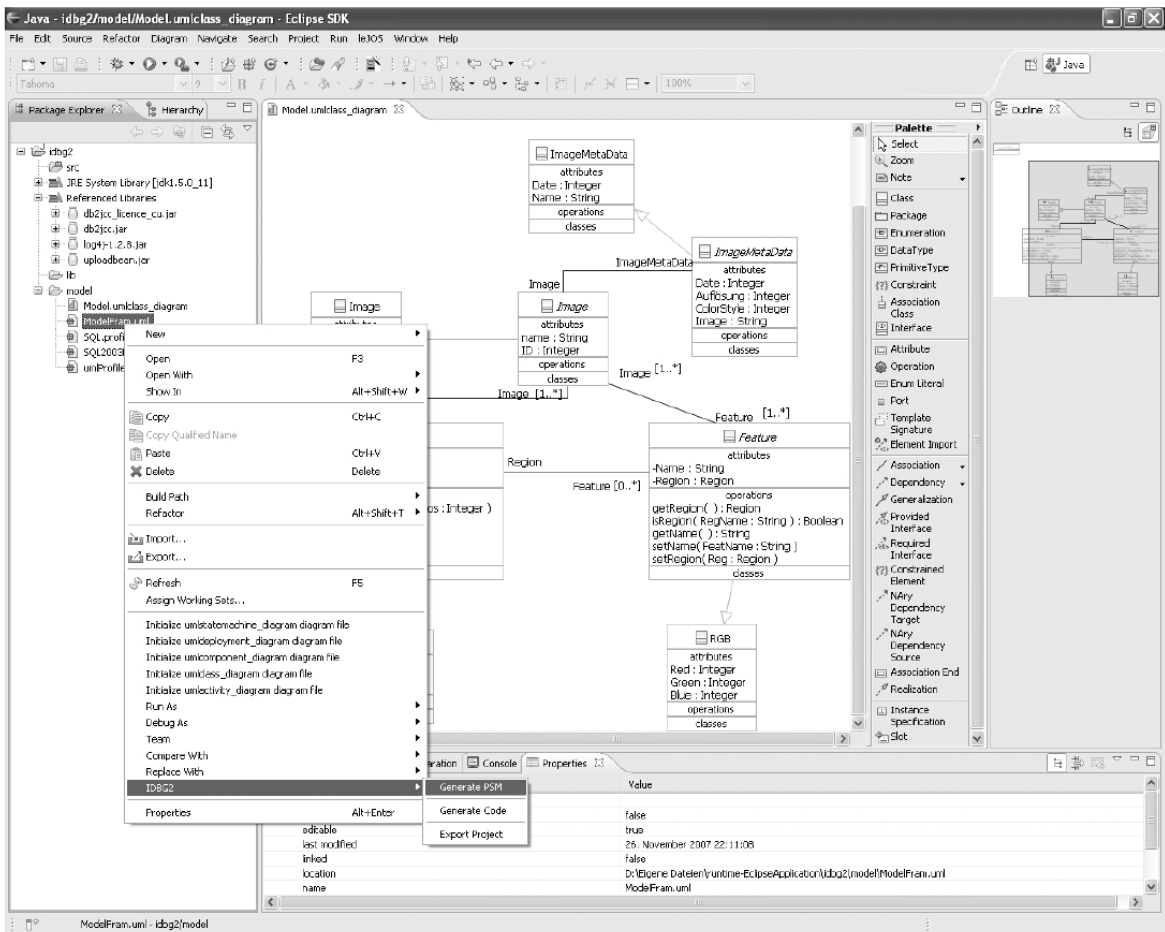


Figure D.2: Modeling environment and Generation Wizards

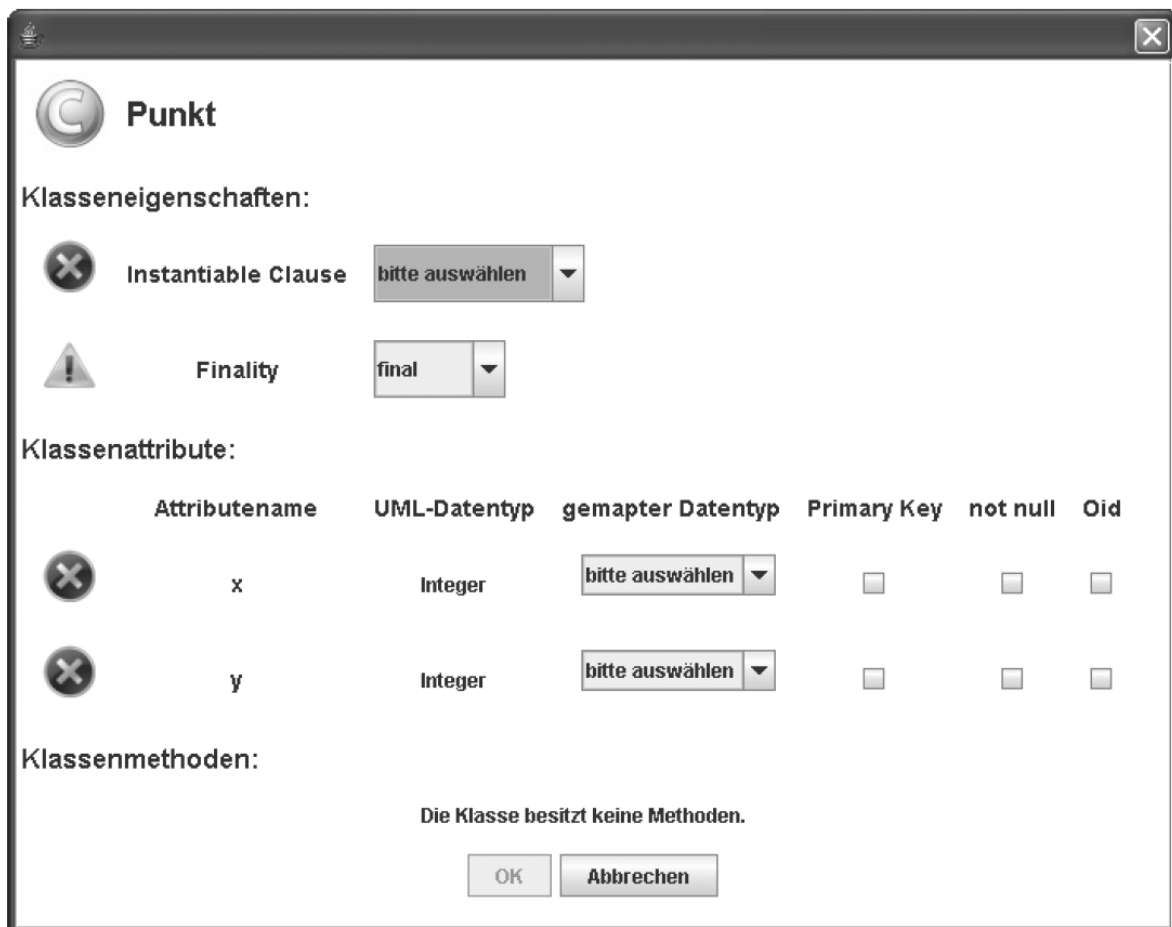


Figure D.3: Transformation options dialog