FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Model Driven Approach to Generate Ground Truth Data for Digital Preservation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Harald Mezensky
Matrikelnummer 0227250

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber, Univ. Doz.
Mitwirkung: Dipl.-Ing. Dr. Christoph Becker

Wien, 25.11.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Model Driven Approach to Generate Ground Truth Data for Digital Preservation

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Harald Mezensky
Registration Number 0227250

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber, Univ. Doz.
Assistance: Dipl.-Ing. Dr. Christoph Becker

Vienna, 25.11.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Harald Mezensky
Hernalser Hauptstraße 98/11/18, 1170 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

Nowadays a large percentage of the data that contains our cultural and scientific heritage is only available in digital form and the amount of digital information is growing rapidly every day. Since digital artifacts are often depending on specific hardware and software, but environments are subject to changes and might evolve or become obsolete, there is a stringent necessity to develop techniques that provide solutions for this problem. Digital preservation aims at providing these solutions by introducing various activities to enable long term accessibility and interoperability of digital objects over time and changing environments.
One of these strategies is the migration of data, which includes the transformation of a potentially obsolete file format into a more current, and more sustainable format. To ensure that the content of the new file is at least similar to the old version, digital preservation provides characterisation tools. These tools are able to analyze the properties of a file and therefore allow comparison of two different versions, in our case the object before and after the migration process.
To introduce quality assurance activities for characterisation tools it is mandatory to have access to meaningful benchmark data that can be used as input for actions such as black box tests. This benchmark data must rely on a known ground truth to be able to verify the result of executed test runs. Currently, however, no such data exists and therefore improvements of existing tools, or development of new ones, can only be performed in an unsystematic way.
This thesis presents a solution to that problem by providing a model driven concept that fills the gap between digital preservation and quality assurance. This approach is based on the idea to design a digital object by defining a model representation of it. More precisely, a platform independent source model is created that relies on a particular metamodel and contains all elements and attributes that are necessary to describe the digital artifact. This model is used as input for a model transformation step where out of one source model, multiple models are generated by diversifying the properties of the model. The motivation behind that step is to bring variety into the ground truth and create a collection of benchmark data that is similar to a real world collection regarding distribution of properties. This can be accomplished by using statistical data as input for the diversification step. For each diversified model again a model transformation is performed to transform the platform independent model into a platform specific model. In this way, benchmark data for multiple platforms can be generated within one run. The models are finally transformed into the actual output or into executable code that can be used to generate actual output by means of code generation. Additionally to each generated file a corresponding ground truth file is created, capturing the actual properties of the object.
The thesis contains an evaluation of the current status in these areas and provides a detailed de-

scription of the underlying concept the approach is based on. In order to prove the feasibility of the concept, a reference implementation has been developed. This prototype is also documented in this thesis, including the base for decisions regarding used technologies, as well as emerged problems. The implementation is used for various experiments, which are described including analysis of the results and the performance of its execution. Finally, an outlook for further work as well as details about applications and limitations of the current approach are given.

Kurzfassung

Heutzutage ist ein großer Teil unseres kulturellen und wissenschaftlichen Erbes nur noch in digitaler Form verfügbar und die Menge an digitaler Information wächst täglich mit rasanter Geschwindigkeit. Da digitale Artefakte meistens von bestimmter Hardware und Software abhängen, diese jedoch Änderungen unterworfen sind und sich entweder weiterentwickeln oder obsolet werden können, ist es von großer Bedeutung Techniken zu entwickeln, die dieses Problem lösen können. Digitale Langzeitarchivierung, auch bekannt als Digital Preservation, macht es sich zur Aufgabe Lösungsansätze für diese Problematik zur Verfügung zu stellen. Diese sollen durch verschiedene Aktivitäten die dauerhafte Nutzbarkeit digitaler Daten über sich ändernde Umgebungen hinweg gewährleistet.
Eine dieser Strategien ist die Migration von Daten, die die Transformation potentiell gefährdeter Objekte in aktuellere und langlebigere Formate beinhaltet. Um sicherzustellen dass der Inhalt von migrierten Dateien vor und nach der Migration zumindest annähernd gleich ist, setzt Digital Preservation sogenannte Characterisation Tools ein. Diese Programme analysieren die Eigenschaften von Dateien, um einen Vergleich von verschiedenen Versionen dieser Datei zu ermöglichen. In unserem Fall wäre das ein Vergleich der Datei vor und nach der Migration.
Für qualitätssichernde Maßnahmen an Characterisation Tools sind aussagekräftige Benchmark Daten notwendig, welche als Input-Daten für Black Box Tests verwendet werden können. Diese Benchmark Daten müssen auf einer bekannten Grundwahrheit, oder auch Ground Truth, basieren, um das Ergebnis von Testläufen verifizieren zu können. Zum jetzigen Zeitpunkt fehlt es jedoch an solchen Daten, weswegen Änderungen an bestehenden Programmen oder die Entwicklung neuer Programme nicht systematisch durchgeführt werden können.
Diese Arbeit stellt einen Lösungsansatz zu dem besagten Problem vor, indem ein modellgetriebener Ansatz eingesetzt wird, der die Kluft zwischen Digital Preservation und Qualitätssicherung überbrückt. Dieser Ansatz basiert auf der Idee, dass digitale Objekte in Form eines Modells repräsentiert werden können. Genauer gesagt wird ein plattformunabhängiges Quellmodell erzeugt, welches auf einem bestimmten Metamodell basiert und welches die Elemente und Attribute beinhaltet, die notwendig sind um ein digitales Objekt zu beschreiben. Dieses Modell wird als Input für eine Modelltransformation verwendet, bei der aus einem Quellmodell mehrere Modelle generiert werden, indem die Eigenschaften des Modells diversifiziert werden. Der Hintergrund dafür ist die Möglichkeit eine größere Varianz in die Grundwahrheit zu bringen um eine Sammlung von Benchmark Daten zu erzeugen, die bezüglich Verteilung der Eigenschaften Ähnlichkeiten zu realen Datensammlungen aufweist. Das kann gewährleistet werden indem statistische Daten als Input für den Diversifizierungsschritt herangezogen werden. Für jedes diversifizierte Modell wird wiederum eine Modeltransformation angewendet, um das plattformunab-

hängige Modell in ein plattformspezifisches Modell umzuwandeln. Dadurch können Benchmark Daten für unterschiedliche Plattformen in nur einem Lauf hergestellt werden. Abschließend wird Codegenerierung eingesetzt um aus den Modellen richtige Output-Dateien zu erzeugen, oder um Code zu erstellen dessen Ausführung oder Kompilierung die tatsächlichen Output-Dateien erzeugt. Zusätzlich zu jedem erstellten Objekt wird auch eine dazugehörige Ground Truth Datei generiert, die die Eigenschaften des eigentlichen Objektes beinhaltet.
Die Arbeit beinhaltet eine Analyse des Stands der Technik in diesen Bereichen und liefert eine detaillierte Beschreibung des zugrundeliegenden Konzepts. Um die Durchführbarkeit des Konzepts verifizieren zu können, wurde eine Referenzimplementierung entwickelt. Die Dokumentation dieses Prototyps, inklusive Details zu den Entscheidungen bezüglich eingesetzter Technologien und aufgetretener Probleme, ist ebenfalls in der Arbeit enthalten. Die Implementierung wird anschließend für diverse Experimente herangezogen, welche ebenso in der Arbeit beschrieben werden wie auch die Analyse der Resultate und der Performance der Tests. Abschließend wird ein Ausblick auf mögliche weitere Arbeitsschritte gegeben und auf die Anwendungsgebiete und Einschränkungen des aktuellen Ansatzes eingegangen.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

With the growing amount of information that is only available in digital form, while the number of formats is increasing, combined with the transitory nature of environments that are required to access this data, we are faced with a serious risk of losing cultural and scientific heritage. To prevent digital data from being unusable sometime in future, various strategies, technologies and policies are introduced to allow long term accessibility, also known as digital preservation.
When trying to preserve an object for a long time period, Hedstrom claims that especially the logical preservation is a challenging task, due to its complexity [24]. Moreover, hardware and software is not stable and might become obsolete quite fast nowadays. To make data available over changing environments, various strategies can be applied. One of these strategies is the migration of data, which is the transformation into a different and more current format. Suggestions regarding selection of appropriate file formats can be found in [5, 63]. According to Hunter and Choudhury, migration is *„the most commonly applied approach for digital preservation“* [26].

An important issue when migrating data into another format is to keep the content authentic. That means the significant properties, i.e. content and metadata of included information should still be the same, at least to a certain degree. To verify if the result of a migration process meets the predefined requirements regarding changes of content and properties, digital preservation introduces characterisation tools such as DROID[1], JHOVE [1] or XCL [61]. According to nestor, these tools are able to analyze the content of files, providing an abstract description which allows a comparison of the file before and after the migration, in order to detect differences [44].
To verify the correctness and reliability of characterisation tools in a systematic way, the availability of benchmark data is essential. By comparing output that is well-known beforehand with the actual output, it is possible to improve existing tools, or to detect faulty behavior when developing new techniques. The benchmark data has to be defined in a standardized way, relying on a meaningful ground truth, which should contain all relevant properties and information that

[1] http://sourceforge.net/apps/mediawiki/droid

are necessary to describe a digital object.
To obtain such ground truth, current approaches are using collections of real world data and defining the properties for each object. It is also common to select objects conforming to a certain distribution for various properties (e.g. file size). This process is executed manually and thus very time and resource-consuming. This fact results in incomplete benchmark data regarding coverage of properties and formats [2, 29]. Additionally, it is almost impossible to generate ground truth with a reasonable size that is also as close as possible to an aspired distribution when using a manual approach.

To improve this situation, the idea of a flexible model driven approach has been developed. Instead of using real objects, model representations of it are used as input for an automatic workflow, that generates multiple files and its corresponding ground truth. After defining a generic source model of a digital object, its properties are diversified to create a data set that meets certain distributions. In the next step, platform specific models are generated, again using (domain specific) distributions in order to generate files for various formats. Depending on the domain, the platform specific models are transformed into source code or actual files. In case of source code generation it is necessary to compile and/or execute the code in order to generate the actual files.
For example a generic document could be generated as a source model by defining its elements and properties, which in the end results in multiple files of different formats like Microsoft Word (in different versions), OpenOffice Writer, WordPerfect etc. A more detailed description of the workflow can be found in Section 3.2
The described approach provides the following advantages:

- The process can be executed automatically, without the need for time-consuming user actions
- Ground truth can be generated in a large scale
- The process that generates the data is well documented and traceable
- Statistical input from real world data can be used to gain well-defined distribution of properties
- The process is configurable and extensible and allows fine tuning of the workflow in order to create specific output, that exactly corresponds to the users needs
- By defining one source model it is possible to create output for multiple formats within one workflow
- When defining digital objects by using a model, there is no dependency on a specific format or tool that could get obsolete sometime in future

To provide such a framework, the following research questions need to be addressed:

- **How have quality assurance activities been carried out so far in digital preservation?**
  To improve the current situation it must be identified first. Therefore a basic literature research about the state of the art in this area and about related topics must performed. This could provide useful information that might be used as input for the development of a concept.

- **What are the requirements of a concept that fills the gap between digital preservation and quality assurance?**
  Before developing a new concept it is mandatory to analyze its requirements and goals. After defining the relevant features, an approach for solving the resulting problems can be provided. Also the requirements of the generated output are part of this research question and contains issues such as how to generate the ground truth, and how to yield data that is as close as possible to real world data collections.

- **How can we prove the feasibility of the developed concept?**
  To answer the question if the developed concept can be carried out in practice while meeting the defined requirements, a prototype can be implemented as a proof of concept. The reference implementation of such a prototype could provide important conclusions about the developed concept such as limitations and special cases for certain domains or platforms.

- **Does an implementation of this concept yield meaningful data within a reasonable runtime?**
  The reference implementation can be used to verify if the generated output meets the expected requirements regarding quality and performance. This can be done by processing various experiments and analyzing their results.

This thesis provides a prototype framework for this model driven approach and investigates whether it is feasible to generate realistic benchmark data and a meaningful ground truth for quality assurance purposes. The goal is to close the gap between digital preservation, using characterisation tools for migration purposes, and quality assurance. This gap should be closed by means of model driven engineering.
The work focuses on the domain of page based documents, which are digital representations of documents (e.g. books, letters, newspapers, scientific papers, etc.) divided into single pages, where each page is defined as an area with a predefined size and therefore with a predefined amount of space for content. In comparison, pageless documents are documents without a predefined boundary for the content such as web pages.

The structure of this thesis is as follows:
Chapter 2 gives an overview of related work for digital preservation, quality assurance, model driven engineering and information models. It also contains an evaluation of the current situation in digital preservation regarding benchmark data that is used for quality assurance issues.
Chapter 3 contains a detailed description of the underlying concept for a model driven approach.

The whole workflow and its building blocks are introduced, using the domain of paged based documents as an example. Also the background of the stochastic processes that are used to diversify the properties are outlined in this chapter.
A reference implementation, that has been developed as a proof of concept for the model driven approach, is documented in Chapter 4. It contains details about the concrete implementation, used techniques, problems and alternatives for each step in the workflow.
To verify the results of the reference implementation from Chapter 4, Chapter 5 contains various experiments where a representative example is taken as base for the execution of the whole workflow. These experiments consist of numerous testruns including performance measures and analysis of the generated output.
In Chapter 6 an outlook on further work such as approaches for different domains and object types is given, and possible improvements of the current reference implementation are illustrated. Finally the thesis is summarized and the lessons learned are described.

CHAPTER 2

Related Work

The approach briefly described in the last chapter combines different scientific areas such as digital preservation, model driven engineering and quality assurance. This chapter will outline topics that are related to the approach, and will give basic information about each area, in order to provide a better understanding for it. Also the current situation regarding availability of benchmark data for quality assurance in digital preservation is issued in this chapter.

2.1 Digital Preservation

The amount of digital artifacts is growing rapidly each day. The market research firm International Data Corporation (IDC) assumes that until 2020 the amount of digital data will about double every two years to an amount of 40 trillion gigabytes [17]. On the one hand, more and more data is only available in digital form, also referred to as born-digital. On the other hand, analog information is increasingly being digitized. This data contains our cultural heritage, scientific knowledge, but also vital personal information such as medical records or administrative documents, and last but not least, data with subjective value such as images and other private documents.
Digital objects are often depending on specific software or hardware. Rothenberg and Garrett et al. are providing various examples, to illustrate that past experiences have shown that environments are subject to changes and might evolve or become obsolete faster than expected [55, 66]. Although Rosenthal claims that in the last fifteen years the risk of format obsolescence has become almost negligible, there are still various techniques necessary *„for ensuring the future legibility of digital documents“* [54].
For this reason, it is essential to introduce activities that provide long term accessibility of digital artifacts in each domain and for each data type. But factors such as the heterogeneity of data [3] or the difficult handling of proprietary formats [22], makes this a challenging task.

In order to enable long term accessibility and interoperability of digital objects over time and changing environments, the area of digital preservation is introduced. Digital preservation

„refers to the series of managed activities necessary to ensure continued access to digital materials for as long as necessary“ [30]. It encompasses technical strategies, as well as organizational activities and policies, and is pushed by various institutions including libraries, archives, museums, as well as scientific institutions.

The preservation of digital objects is exposed to different levels of threats [47]:

- Physical layer: preserving the sequence of bits in a file
- Logical layer: preserve objects over changing environments
- Conceptual/semantic layer: correct interpretation of data

A major challenge is the preservation of the logical layer, which includes the ability of opening a file and using it in the manner it was created for. In contrast to the physical preservation, where the key task is to be enable accessing the bits sometime in future, the logical preservation deals with understanding these bits. Digital preservation provides several approaches to overcome the threat on the logical layer [38]. Some of the techniques used today are:

- Emulation: Using software to open and run a file on a current system by recreating the behavior of the original environment including hardware and software. The advantage of this approach is that the look and feel and the original functionality is not lost (assuming a perfect emulation). This approach is widely used today, especially for games and old programs. [18, 56]
- Migration: this common strategy, which includes the transformation from a certain file format into another one, is described in more detail in subsection Migration.
- Technology preservation: keeping the original environment that is necessary to open and use the digital objects including hardware and software. The advantage of this approach is that it guarantees the preservation of data in its original form. But at the same time it is difficult to keep environments runnable over years, and also the costs and required space are disadvantages. This solution is only viable for particular cases and for short-term preservation. [25]
- Encapsulation: Wrapping metadata around the files which includes information how to interpret the data and how it can be accessed. Actually this is just a basis for digital preservation but does not solve the original problem of long term accessibility. [67]
- Analog strategies: Transferring the content of digital objects to analog media by printing it on paper or microfilm. There is no need for a specific environment to view the data and for example microfilms can last up to 500 years. But this approach is only usable for object types like images or documents. [23]

Beside emulation, the migration of digital data is the most common strategy used in digital preservation today. The approach described in this thesis has the goal to provide benchmark data for quality assurance purposes, applied on characterisation tools (cf. Section 2.1). Since

one major application of characterisation tools is the verification and validation of migration processes, this technique is going to be described in detail.

Migration

The task force on digital archiving describes migration as *„a set of organized tasks designed to achieve the periodic transfer of digital materials from one hardware/software configuration to another, or from one generation of computer technology to a subsequent generation.“* [66]
As for all digital preservation strategies described above, the goal is to enable the usage of data over changing environments and preserve the logical layer. Therefore the intellectual content of a potentially obsolete file is transformed into a digital object, which is a representation of the same content, but stored in a different form. The format is adapted to meet the requirements of the new environment without changing the content and concepts of the artifacts.

It is not just reasonable to migrate from an old format to a current one, but it is also useful to migrate data from proprietary into open formats. In best case a format is used that is standardized and supported by different programs, in order to reduce the risk of losing data if environments are getting obsolete. For example Microsoft Office files like Word documents, Excel spreadsheets or PowerPoint presentations could be migrated into the corresponding OpenDocument format, which is an open XML based format for office applications, developed by OASIS [28].

The goal of migration tools is to provide the same look and feel and the same features for the output, like available in the original object. But as stated by the Council on Library and Information Resources (CLIR), migration is basically a translation, and as for all translations it is difficult to avoid loss of information during a migration process [37]. Especially for complex formats this is almost impossible. Although this can lead to unusable data for certain formats, in general differences are tolerable to a certain degree. It usually depends on the delta between the content of the original file and the output of the migration. Therefore it is essential to measure the differences respectively analyze the content before and after the migration. For that purpose digital preservation uses characterisation tools, which are described in the following section.

Characterisation Tools

To identify a digital object and answer the question if it is really the object it is expected to be, its metadata has to be analyzed. Metadata in the context of digital preservation can imply different things [46, 65]. It could contain information like the structure, actual content, technical information like file format, descriptive data like author, domain specific information like codec for videos etc.
The characterisation of metadata is important to validate and verify digital artifacts. As mentioned in the previous section, it can be used to compare different files in order to detect the differences. This is necessary to identify if a migration process was successful and how much information has been lost or has been changed.

For that purpose, characterisation tools are introduced in digital preservation. These tools are able to analyze the content and properties of digital artifacts and can be used for extraction of metadata. It is impossible to provide an universal definition of the produced output, since every characterisation tool offers different features and analyzes on different levels, with different detailedness. Usually these tools are at least able to identify the file format, but for the validation of migration processes it is essential to gain significant properties for a comparison. Moreover, it is also important to verify if a file format meets the required specifications, otherwise migration actions could fail.

Characterisation tools can be divided into tools that are able to identify the file format and into tools for metadata extraction. For example the tool DROID[1] (Digital Record Object Identification) developed by The National Archives, is an open source software used for identification of file formats. Also FIDO[2] (Format Identification for Digital Objects) developed by Open Planets Foundation is designed for that purpose. Both tools are using the PRONOM[3] database, which is an information system for file formats. JHOVE[4] (JSTOR/Harvard Object Validation Environment) in contrast, additionally provides functionality for characterisation of format specific properties, and therefore is able to extract technical metadata. Also the Metadata Extraction Tool[5] developed by the National Library of New Zealand, as its name implies, is able to extract the metadata for a range of supported file formats. FITS[6] (File Information Tool Set) is another example for a tool that can extract technical metadata by wrapping third-party tools and consolidating their output. The eXtensible Characterisation Languages (XCL) [61] developed by the University of Cologne goes one step further and theoretically supports even the extraction of the actual content of digital objects.
Moreover various solutions exist that support the extraction of metadata only for specific file formats. For example docx4j[7] or Apache POI[8] are Java libraries that can be used for manipulation of Microsoft Office files, but also provide the functionality to programmatically extract and analyze the content of files with such a file format.

2.2 Quality Assurance

Independent of the domain, quality plays a crucial role in software engineering. The Institute of Electrical and Electronics Engineers (IEEE) defines software quality as *„the degree to which a system, component, or process meets specified requirements“*, respectively meets the users or customers needs or expectations [62].
To ensure that software is developed with a proper quality, it is necessary to introduce different quality assurance methods. Quality assurance, in the context of software engineering, can be

[1] http://www.nationalarchives.gov.uk/information-management/projects-and-work/droid.htm
[2] http://www.openplanetsfoundation.org/software/fido
[3] http://www.nationalarchives.gov.uk/PRONOM
[4] http://jhove.sourceforge.net
[5] http://meta-extractor.sourceforge.net
[6] http://code.google.com/p/fits/
[7] http://www.docx4java.org
[8] http://poi.apache.org/

defined as *„a systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of a software system product conforms to established functional technical requirements“* [16].

To ensure that the requirements are fulfilled according to the specification, it is necessary to introduce methods for verification and validation. A common quality assurance strategy for that purpose is software testing. The tests can be executed manually or in an automated way, the scope can vary from single units to whole systems, and the goal is the approval of behavior and output of the objects under test.
The classification of software tests can be divided into white box tests and black box tests. While white box testing means that the internal structure of the code is investigated, black box testing in contrast does not observe the code that is creating an output. Instead, some input is used to verify if the actual result conforms to the expected output.

Quality Assurance in Digital Preservation

Also software that is used in digital preservation such as characterisation tools has to be tested in the same manner as in other disciplines. The difficulty in this domain is the acquisition of meaningful test data sets that can be used as input for quality assurance actions such as black box testing. The National Library of Australia lists some collections that contain test data for different file formats and domains like e.g. Govdocs1[9], Prometheus Ingest [8] or Pandora Web Archive[10]. They also state that obviously no data set *„that has many files, is accurately characterised, is representative of real data, and is freely distributable“* seems to exist at the moment [27].

When collecting benchmark data for software testing, the major problem is the inability of verifying the result by comparing it to an expected output due to absence of attached metadata. According to Becker and Rauber, it is essential that benchmarks rely on a known ground truth [2]. Current approaches are using real-world collections, where a meaningful subset of benchmark data is selected and annotated, in order to cover specific test cases.
As described by Neumayer et al., the definition of ground truth is essential to verify the correctness of a system [43]. The annotation of each file from the benchmark data, which in the end forms the ground truth, usually has to be added manually. The manual definition of ground truth however, increases the risk of inconsistent metadata. Therefore the current situation could be improved by using an automated approach for the annotation of benchmark data.

Neumayer et al. also list different challenges digital preservation is faced with when it comes to generation of benchmark data. To provide a meaningful stratification, the elements within a collection used as test data should reflect real-world distributions. The goal is to ensure, *„that all essential groups of digital objects are represented according to their relevance“*. In the SCAPE project for example, a set of existing scientific publications is used as input data, in order to ana-

[9] http://digitalcorpora.org/corp/nps/files/govdocs1/
[10] http://pandora.nla.gov.au/

lyze various characterisation tools [29]. When using existing collections as basis for test data, it might be necessary to inject additional data to reach that goal. Also in the SCAPE project it was necessary to add larger files manually to the test data set, in order to achieve a better coverage of properties.
Generally, benchmark data used for testing should be selected according to its application scenario, respectively the domain it is intended for. The size of the collection and the files should be assembled, so that all features of a specific test case are covered. In many cases it might be easier to reach that goal by designing benchmark data from scratch and not using real data at all. According to Seadle et al., however, this strategy has the disadvantage that *„artificially manufactured data tend to fail to represent the variety and complexity of real data“* [40].
Moreover, both solutions are sharing another disadvantage, which is the time consuming process of manipulating, respectively creating benchmark data manually. A solution that combines the advantages of both strategies, while processing without time consuming user intervention, would be an automated approach that manufactures artificial data which reflects the characteristics of real world collections. This output could meet statistical distributions as detailed as possible, without the need of manipulating each single file manually, which is not economical regarding resource management.

2.3 Model Driven Engineering

To implement an automated approach that manufactures artificial benchmark data, it is necessary to define some kind of source object. This object is reproduced multiple times with varying properties, in order to generate benchmark data in a large scale. The approach described in this thesis uses model driven engineering for the definition of the source object, which in that case is a model representation of an actual object.

One reason for using a model driven approach is that there exists no adequate alternative for solving this issue. When using the domain of page based documents exemplarily, one could use Microsoft Word or OpenOffice Writer or some other arbitrary word processor, to generate a specific text document as source object. But when using this approach, the user could already add features to the document that are not available for every word processor and therefore cannot be implemented by each platform. Instead, it should be mandatory to create a source document that is generic and could be generated by each word processor. This is necessary since one of the core concepts and benefits of this approach is the possibility to generate benchmark data for different platforms, out of one generic source object. But this is only possible by using an independent way of designing such a document. In return, this approach has the disadvantage that the user is not able to explicitly define features of a specific platform when designing the source object. These elements are automatically set by the workflow when transforming the source object into a platform specific representation (cf. Section 4.3).
Using some kind of domain specific languages (DSL), which allows the definition of platform independent source objects, would be an option. Fowler defines DSLs as *„a computer programming language of limited expressiveness focused on a particular domain“* [12]. Common examples are HTML, CSS, regular expressions, SQL, rake, etc. But in comparison to a model

driven approach, plain DSLs entail some disadvantages. Kourouthanassis and Giaglis argue that the expense for learning a new language is high [34] and also van Deursen et al. list the cost of education for the users of the DSL as a problem [9]. That means specific knowledge is necessary to be able to use that language, and depending on the domain this can end up in very complex structures.
Using a model in contrast, seems to be the most natural way to design an object. Brambilla et al. describes models as a simplified representation of real world objects [4]. When using models as core artifacts within a software engineering process, we are speaking about model driven engineering (MDE). According to Brambilla et al. also the term model driven architecture (MDA) is often used in this context, which in contrast refers to a subset of MDE activities, standardized by the Object Management Group (OMG).
MDE is also using DSLs, more precisely a domain specific modeling language (DSML), but MDE also offers graphical modeling tools in many cases. These tools can be used to design models in a convenient and intuitive way, without the need of learning a new programming language.

As described above, the source object needs to be generic, or in other words platform independent. According to OMG, a platform in the sense of model driven architecture refers to *„technological and engineering details that are irrelevant to the fundamental functionality of a software component“* [48]. The platform independent source object in a model driven approach conforms to a platform independent model (PIM), which the OMG defines as *„a formal specification of the structure and function of a system that abstracts away technical detail“*.
There are techniques available in MDE that allow actions such as transformation of a model into another model or into text. In our approach the model transformation technique is used to generate a broader base for the benchmark data, by automatically providing data for multiple platforms. The input for such a model transformation is the platform independent source model, which is transformed into various platform specific models. According to Mellor et al., a platform specific model (PSM) conforms to *„the result of weaving a PIM with the platforms on which it depends“* [41].

Before a model, whether platform independent or platform specific, can be designed, as a very first step the base for the model has to be defined. That includes which elements and attributes are available for a certain model, how elements are related to each other, and how a valid model looks like. For that purpose a metamodel has to be created for each model instance. According to Mellor et al., a metamodel in the context of MDE is defined as *„a model of a modeling language“*, which *„defines the structure, semantics, and constraints for a family of models“* [42], or as described by Frankel simply *„a model of a model“* [13].

Information Models

A model driven approach is only as good as its underlying models. Therefore, a comprehensive and clear specification of these models is of particular importance, maybe even the most important step of the whole workflow. There is already a wide range of information models and ontologies for all kind of different domains available, which can be used for a formal description of digital objects. This section provides some existing solutions, especially focusing on the

domain of paged based documents.

Different terms are used in literature to describe the formal specification of digital objects. An ontology, as defined by Gruber, is *„an explicit specification of a conceptualization“*, including the vocabulary to describe objects and their relationship [20]. According to Pidcock and Uschold the distinction between ontologies and metamodels is tough [49]. They state that if a vocabulary from an ontology is used to create a model, it is equivalent to a metamodel. But in contrast not all ontologies are explicitly modeled as metamodels.
As defined by Lee an information model is *„a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse“* [39]. This definition is again very close to the definition of ontologies. Rector et al. describe the difference with the distinction, that information models are located at the level of data structures and code, while ontologies are the „models of meaning“ [52].

The first example is an ontology approach provided by FORTH-ICS, that implements a solution for objectifying the symbol structure and sensory impression of information objects [10]. The conceptual model consists of a physical and a digital model, where the symbol structure is the central element. It contains a set of symbols and the relation between each one. To describe information objects such as text documents, the model can also contain information about the positioning of the symbols by defining arrangement rules and additional information that describes the appearance, such as symbol type, or information format. Beside text documents this approach can also be applied to other information objects such as music, images or websites.

Another ontology is introduced as part of the Planets XCL (eXtensible Characterization Language) project [51]. The XCL ontology is used to describe file properties and their connections to each other for XCL based software. XCL is a characterisation software, used to validate and verify migrations in digital preservation. When migrating a digital object from one file format to another one, the syntax of the same property in both files will most likely differ. To enable the comparison of such properties, this ontology provides a vocabulary to find matching properties, in order to detect changes during the migration process. Currently properties for images and text are fully supported.
Regarding properties for the domain of page based documents, the InSPECT project funded by JISC provides a report[11] that contains a collection of significant properties, including information about possible values and ranges. This report can be used as input when creating a new information model. InSPECT also provides such a report for the domains of audio recordings, raster images and emails.

The NEPOMUK project[12], which is an ontology based open source solution that supports the exchange of knowledge within social networks, provides ontologies for different resources such as files or emails. The core ontology is the NEPOMUK Information Element Ontology

[11] http://www.significantproperties.org.uk/structuredtext-testingreport.pdf
[12] http://nepomuk.semanticdesktop.org/

(NIE)[13], which contains vocabulary for common information objects on the semantic desktop. One of these vocabularies is the NEPOMUK File Ontology (NFO)[14] that provides a solution for files such as documents, software, websites and media. This ontology has a comprehensive support for different file types, and focuses on basic informations on meta level. For text documents it supports information such as number of characters, lines and words, but it is not going deeper into the content.

The TEI (Text Encoding Initiative) guidelines provide an open format that can be used as information model for text based documents [6]. The first version was based on SGML, which, according to Hackos, was the first standard used for building information models [21]. The current version of TEI is based on XML, which is a widely accepted standard for defining data structures and in Hackos' view has mainly superseded SGML as basis for information models today.
By using the TEI schema it is possible to draw on a wide range of textual concepts on different levels of granularity. The schema provides various features for modeling complete documents, including platform specific formats such as Microsoft Word or OpenOffice Text. Moreover TEI supports customization, in order to enable its usage for specific domains and platforms.

Using information models as base for the manufacturing of information that simulates real world data is not a new invention, Shannon used such an approach already back in 1948 [57]. Similar to our solution, he used a mathematical model to generate messages, that are close to real messages regarding statistical properties. Thus, also the communication problems could be reduced to the problems of real messages. In our approach this is done by using statistical data and an information model as input, in order to generate benchmark data that is also as close as possible to real data.

2.4 Summary

The usage of quality assurance activities in digital preservation, particularly for characterisation tools, is of significant importance for the development of new, or the verification of exiting preservation techniques. The performance of systematic tests in this area is hindered by the non existence of meaningful benchmark data. As a result, there is currently a gap between digital preservation and quality assurance. To close this gap, different ontologies and information models are available as base for a model driven approach which uses real world statistics to manufacture data. A concept for such an approach, that generates benchmark data with a known ground truth, is described in the next chapter.

[13] http://www.semanticdesktop.org/ontologies/nie/
[14] http://www.semanticdesktop.org/ontologies/nfo/

CHAPTER 3

Model Driven Benchmark Creation

This chapter contains a detailed description of the underlying concept for the model driven approach that should fill the gap between digital preservation and quality assurance. Before a global picture and a detailed description of each building block from the intended workflow is given, the challenges and goals of this concept are addressed. To be able to generate realistic data, the usage of statistical methods is a central issue, which is also introduced in this chapter. As a last point the options to verify if the workflow yields useful benchmark data and a meaningful ground truth are discussed, including considerations about the objects under test.

3.1 Challenges and Goals

The main goal is the development of a concept that generates benchmark data with a known ground truth in an automatable way, in order to provide a solid base for quality assurance activities. A major challenge in digital preservation is the variety of different domains and object types. Therefore at least the basic concept should be generic enough to be applicable for all these types.
A workflow that implements such a concept should be configurable, to enable fine tuning of the output and to allow various experiments. This is helpful to verify the operability of characterisation tools and to reach a comprehensive test coverage with the generated test data. According to that, the contribution of this concept for digital preservation is the facilitation of the development process for such tools.
To reach that goal, it is also important that the generated data corresponds to real world data. Therefore, the statistical properties of generated collections should be similar to the statistical properties of representative real world collections. For this purpose a requirement of the concept is the ability of adding statistical data as input to the workflow. It must be possible to create and modify such an input in an easy and intuitive way, either automated or manually.
The acquisition of this input must be carefully planned, since the quality and applicability of generated benchmark data strongly depends on the selection of meaningful statistical data.

As described in the previous chapter, the concept should be based on a model driven approach. An important goal for that issue is to find metamodels that are suitable for the particular domain. Each metamodel should contain all relevant features to enable the modeling and generation of realistic benchmark data.
In general, a model driven approach consisting of model transformations and code generation is quite common and well documented for different applications. The creation of the actual files out of the generated code in contrast is a challenging task. There exists no unique approach that fits for all domains, and even for the same domain there can be significant differences between single platforms. In some cases it might be necessary to develop very complex and innovative approaches in order to implement this step in the workflow.

Another challenge is to provide adequate scalability within the workflow. This includes the ability to deal with multiple platforms within one run of the workflow. It should be possible to add new platforms to the framework on demand, without affecting the existing implementation. That means that the platform specific parts need to be decoupled from the main workflow. In another context scalability also refers to the goal, not having any limitations regarding number of generated test data. It should be possible to generate benchmark data in a large scale within one execution of the workflow. Even then the performance should be stable regarding duration, usage of resources such as memory, and quality of the output.

Within the generated ground truth, each element should be unambiguously identifiable. This additional information enables the traceability throughout the entire workflow and comparability between generated files. Enabling traceability has the benefit that it is possible to reconstruct the behavior of the workflow, respectively the transformation of each element, starting from source model via different steps, through to the actual file that is part of the generated benchmark data.

3.2 Building Blocks of the Data Generation Framework

To illustrate the underlying workflow of a generation framework for ground truth data, the domain of page based documents will be used as an example. As depicted in Figure 3.1, the workflow consists of multiple building blocks. In this section each building block, its dependencies and input data will be described in detail.

As a starting point the source model has to be defined. Therefore, a PIM relying on an own metamodel must be created as a representation of a real document. This could happen manually or in a (semi-)automated way by using distributions that conform to real world data.
When modeling the PIM manually, each element and its properties must be defined, according to the elements of the document that should be created. This could be either a real document or a representation of a document in the user's mind. The degree of detail and which information can be defined in this model is a matter of the chosen approach. For example when adding a text block, the content could either be defined manually or a dummy text could be used, where according to a certain distribution different text could be inserted (varying from very short to very long text). When using an automated approach, the source model and its elements could be

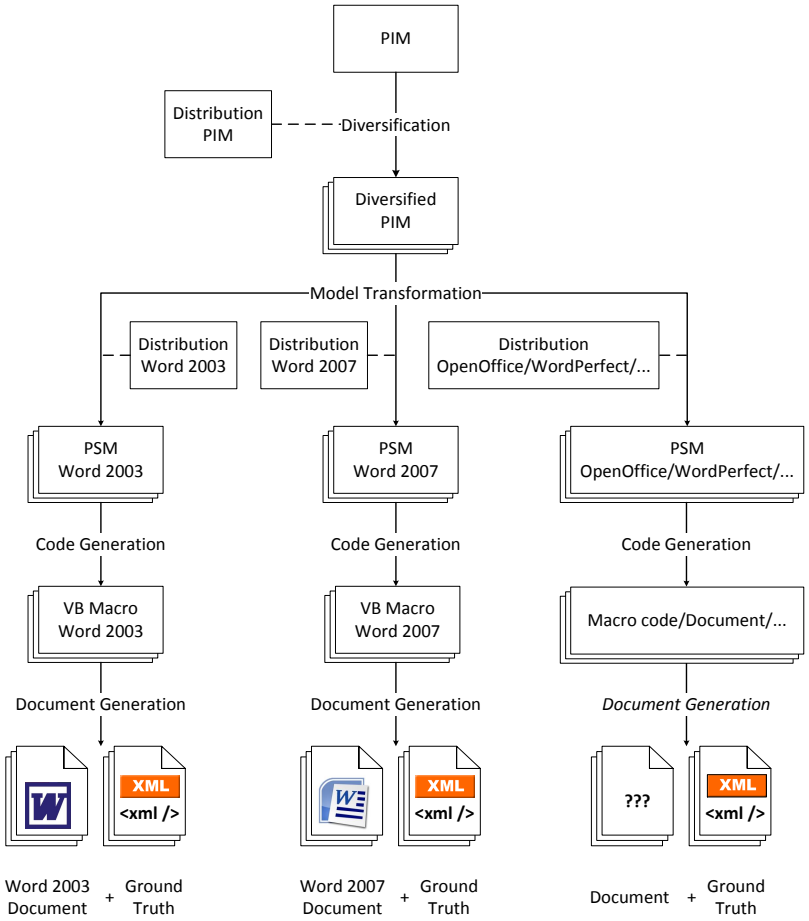


Figure 3.1: Technical workflow

defined automatically by using statistics from real world collections. Another possible solution is a semi-automated approach, where all elements that should be included in the document are being defined, but the number and/or position is set automatically, according to distributions and statistics.

To get a wider base of ground truth data, some elements defined in the PIM can be diversified by creating multiple models out of one source model. These elements can have different values in each diversified model, according to a certain distribution. For example pages can have different formats, tables can have different widths, and so on.
Possibly each element could have its own distribution. That's why an input model is used to represent different kinds of distributions for different elements, where each distribution can have different input parameters. Some elements might have a uniform distribution and others might have more complex distributions with multiple input parameters (cf. Section 3.3). The advan-

tage of using models as base for the statistical input is the homogeneous integration into the workflow. Further benefits are the flexible way of modifying and modeling this data.
There are two options to define the number of models that should be generated at this step. First, a configurable setting could be used where the number of generated models is defined manually. Second, all permutations, i.e. all possible combinations of the diversifiable elements and its values, are generated. According to that decision and setting also the number of documents that are finally being generated is defined in that step.

After the diversification step each PIM will be transformed into a PSM, with each platform having its own transformation rules relying on a separate metamodel. Again some elements will be diversified in this step according to a certain distribution model. The distribution model will be a different one for each platform, since every platform might have specific elements and even for the same elements different distributions, which might additionally depend on the occurrence of the element within the file. For example the size of text will probably have a different distribution, depending if it is located in the footer of a document or if its the title of a chapter.
Generally it's tough to draw the line between diversification which is independent from the platform and diversification which is platform specific. Depending on that decision, the diversification will take place at the PIM diversification step or at the model transformation from PIM to PSM.
If we take a look at the example with page format, generally every document will have some kind of page format (i.e. width and height) but on the other hand there will probably be a different distribution for Word documents compared to PowerPoint documents. Values which are definitely platform specific are e.g. the implementation of a table, which could be in case of the platform Word 2003 a usual Word table or an embedded Excel sheet.
As a result of the model transformation step, PSMs in the same number as diversified PIMs are being generated, multiplied with the number of platforms.

After model transformations, the code generation step is performed. Depending on the platform of the PSM, different kind of code can be generated in this step. For example for Word platforms or OpenOffice Writer, macro code could be generated while for some platforms it might even be possible to produce the actual documents directly by means of code generation.
In the last step of the workflow, each platform will have different processes and use different tools. In this step the actual test data, which in our case is some kind of document, is being generated. Therefore, specific programs will be used to execute the code that has been generated in the previous step. For example, Word platforms could use a tool which runs the generated macro code directly within the native application. For other platforms where the actual documents can be generated in the code generation step, this part of the workflow will of course be skipped.

Along with the actual files also its ground truth is generated in that step, at least for some cases where the output cannot be created in the code generation step. For this purpose the information is stored in an own file. The format should be non proprietary such as XML and the data could be stored by using key-value pairs. It is suggested that the ground truth structure relies on an own metamodel, which has the advantage that the ground truth from different domains

and platforms can easily be compared and processed. Some typical values that can be added as ground truth data are number of pages, number of words, number of characters, number of specific elements like e.g. images and tables, the file extension, just to mention a few. Additionally to that information also the hierarchical structure of the source model including all elements and attributes, the diversified PIM, and the PSM could be added to the ground truth. This allows users to analyze or trace the models throughout the whole workflow.

To enable traceability it is necessary to add some kind of unique identifier to each element. This identifiers must be assigned to the elements as soon as the source model is defined. Furthermore, the exact value must be passed to the corresponding element in each model instance throughout the whole workflow. In case an element changes in the model transformation from PIM to PSM, the identifier should be set in a meaningful way. That means, if e.g. the element table in a document is transformed to a Microsoft Excel spreadsheet, the identifier should also be passed from the table element to the spreadsheet element.
Like for the generation of actual files, also the ground truth generation step can be very different for each platform. For some platforms the ground truth file can be generated with the same tool that is used for file generation and for other platforms the code generation tool from the previous step could be more suitable.

For other domains, such as pageless documents like websites, or for graphics etc., a similar workflow could be used. In many cases data such as web pages or images could be generated directly in the code generation step. The first steps however, i.e. the diversification of the PIM and the model transformation into a PSM, would have exactly the same concept as described for page based documents.

3.3 Stochastic Processes for Creating Variance of Properties

During model transformation, the properties of the model's elements can be diversified at domain level and at technical level. This section demonstrates how to yield results that are as close as possible to real objects regarding variation of properties by using stochastic processes and analysis of real world data.

When observing a real collection of digital data, each contained information will most likely have different values or characteristics throughout the whole collection. The distributions used to represent that variance can be different for each property and element within the file. All distributions can be represented by using a model where relevant parameters can be defined. This can vary from very simple solutions to complex constructs that contain many parameters, in order to describe the particular distribution.

For less complex cases the group of uniform distributions can be used. This type of distribution, also referred to as rectangular distribution, can be defined as *„symmetric distribution that occurs when the frequency of each value on the X axis is the same“* [58]. One of the simplest uniform distributions is the continuous uniform distribution, where random numbers can assume

any value within a given range. In a discrete uniform distribution in contrast, only a finite number of values can be assigned to the random number.
Both distributions can be extended for a more meaningful usage within the workflow. By dividing the sample space into multiple parts, each having a different mass, it is possible to introduce a set of predefined options with different probabilities. This can be used for information, that can only be represented through a limited amount of different values. Considering the domain of page based documents exemplarily, the horizontal or vertical alignment of text would match this case. Vertical alignment for example can only have particular values like top, center or bottom, each having a different probability.
Further extensions could be the assignment of lower and upper limits to the sample space. Additionally the single steps between available random numbers can be defined for discrete distributions. These refinements can be applied not only to simple distributions but also to more complex ones.
Another example of a common continuous distribution is the normal distribution. According to Taneja, this distribution is *„the most important continuous probability distribution in the field of statistics“* [60]. The distribution is characterized by its perfect symmetric bell shape and *„is mainly used to study the behaviour of continuous random variables“* [64].

The described distributions can be sufficient, depending on given domain and particular element. However in many cases such basic distributions will not yield the best result, respectively will not reflect real world behavior of elements in an object. That's why distributions such as skewed distributions, Poisson distributions, chi-squared distributions, bimodal distributions etc. might be necessary, or even more complex ones. Usually statistical models are getting more complex and more extensive, the closer they are to the actual representation. Therefore the goal is to find a balance between complexity and significance of the models. It should still be possible to model a particular distribution with a reasonable effort.

3.4 Evaluation

In order to verify that the developed concept yields the expected output and meets all requirements, an evaluation of the implemented workflow can provide details about its success. This can be achieved by running representative and well assembled tests, using an implementation of the workflow. As a first step it is important to create a source model that contains a meaningful set of features and can be used as realistic input for the workflow. Also the distribution models that contain the statistical input should yield realistic values.
After modeling the input, the workflow can be executed several times, potentially with slightly changed configuration settings for some runs. The performance of each run can be tested regarding completeness, correctness and quality of the generated files. Additionally also elapsed time and usage of resources, such as main memory or CPU can be measured and compared.
The result of the process where actual files are generated can be compared with the ground truth that is also generated by the workflow. This provides information about the quality of the ground truth, but also about the created files itself.
A detailed description of an actual evaluation process, performed with the reference implemen-

tation of our concept, can be found in Chapter 5.

3.5 Summary

In this chapter the concept for a workflow that is able to fill the gap between digital preservation and quality assurance, by generating benchmark data and its corresponding ground truth, was introduced. After defining the goals and challenges of this concept, each step of the workflow was described in detail. This model driven approach contains the definition of a source model, the diversification of its elements in order generate output conforming to real world statistics, the transformation into models for different platforms, and finally the code generation step that either generates the actual output or files that must can be used to generate the actual output. In the latter case additionally a document generation step must be included to the workflow, which is necessary for certain platforms. The integration of statistical data as input for the workflow, as well details about evaluation of the concept in order to verify its success, have been addressed in this chapter.
In the next chapter this concept will be used as base for an actual prototype in order to provide further details about the practical usage of the developed approach such as limitations and possible alternatives.

CHAPTER 4

Reference Implementation

To prove the feasibility of the concept from the previous chapter, a reference implementation has been implemented in form of a prototype that contains the workflow for a specific domain, which in our case is the domain of page based documents. Each step of this workflow is described in detail, including used technologies, problems, limitations, alternative solutions and results.
The workflow has been developed by using the software development environment Eclipse[1] for the implementation of the model driven approach. Eclipse is a widely-used open source platform with a huge community, established in commercial, and academic areas [7]. It provides various frameworks, tools and implementations for model driven engineering such as Eclipse Modeling Framework (EMF), Model-to-model transformation components like ATL or QVT, Model-to-text technologies for code generation such as Java Emitter Templates (JET) or Xpand, all combinable in one homogeneous solution.

The reference implementation provides only basic features, some of them even simplified, just to demonstrate how the single parts of the workflow could be implemented, using specific technologies. Which features are supported by the prototype can be found in Section 4.1. The available platforms are limited to Microsoft Word 2003 and Microsoft Word 2007 and thus only cover a small range, respectively only one specific approach. But for the sake of simplicity it should be sufficient to show one exemplary solution with representative features. Moreover, most steps are quite similar for different platforms, as mentioned before only the step where the actual documents are created can vary notably.
The chosen approach is certainly not the only way to solve this problem and in some cases there is room for improvement and optimization within the prototype. For a comprehensive implementation, which covers most features of the given domain, the prototype would have to be extended in several ways. The current implementation in its simplicity is basically intended for decision support and exposing limitations for future work. Therefore, alternatives to some of

[1] http://www.eclipse.org/

the used approaches, including advantages and disadvantages, are going to be described in this chapter as well.

4.1 Metamodel

At the beginning of the process a domain specific, platform independent metamodel has to be created. This metamodel, which is the base for each page based document model, will be described in this section. Also a domain specific metamodel for each platform is necessary, in the reference implementation this is one metamodel for Microsoft Word 2003 and Microsoft Word 2007. Since there are hardly any differences regarding available features between these two platforms, there exists just one metamodel for both platforms, which is also described below. The only feature that is new in Word 2007 is the possibility to set global themes for a document. Therefore, in an advanced version of the reference implementation, two separate metamodels could be used as it is the case for each other platform. Additionally, the metamodel for the statistical input data, that is used for calculating the distributions, is introduced in this section.

Before any step of the workflow can be conducted, it is necessary to provide a basis for the models first. Since models generally rely on a metamodel, the first task in the process is to create such a metamodel for all model types. For that purpose the Eclipse Modeling Framework (EMF) is used, which is a powerful framework for modeling, unifying XML, UML and Java (cf. [59]). It provides features such as designing and editing of metamodels using a graphical user interface, generating of editors to create and manipulate instances of the metamodel, and generation of model representations in Java code.
Metamodels that are designed using EMF are based on the Ecore metamodel[2], that is comparable to the Meta-Object Facility (MOF) standard[3]. The models are serialized and stored using the XML Metadata Interchange (XMI) format. For an example how serialization of models works in EMF, compare the graphical representation of a metamodel in Figure 4.3 with its XMI serialization in Listing 4.1

As mentioned at the beginning of this chapter the introduced metamodels are just simplified models that cover a basic subset of features. A more advanced metamodel might look completely different. In the following subsections the metamodels for the PIM, PSM for Microsoft Word 2003/2007, and for the distribution model are described in detail.

Metamodel of Platform Independent Model

Figure 4.1 contains the complete metamodel of the PIM that is used in the reference implementation. The root element of this metamodel forms the *Model* element. When the source model is designed by the user, the *Model* element has exactly one *Document* element. Since the diversified PIMs share the same metamodel, after the diversification process (cf. Section 4.4) it usually has multiple *Document* elements, each representing one diversified document. Each document

[2] http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html
[3] http://www.omg.org/spec/MOF/

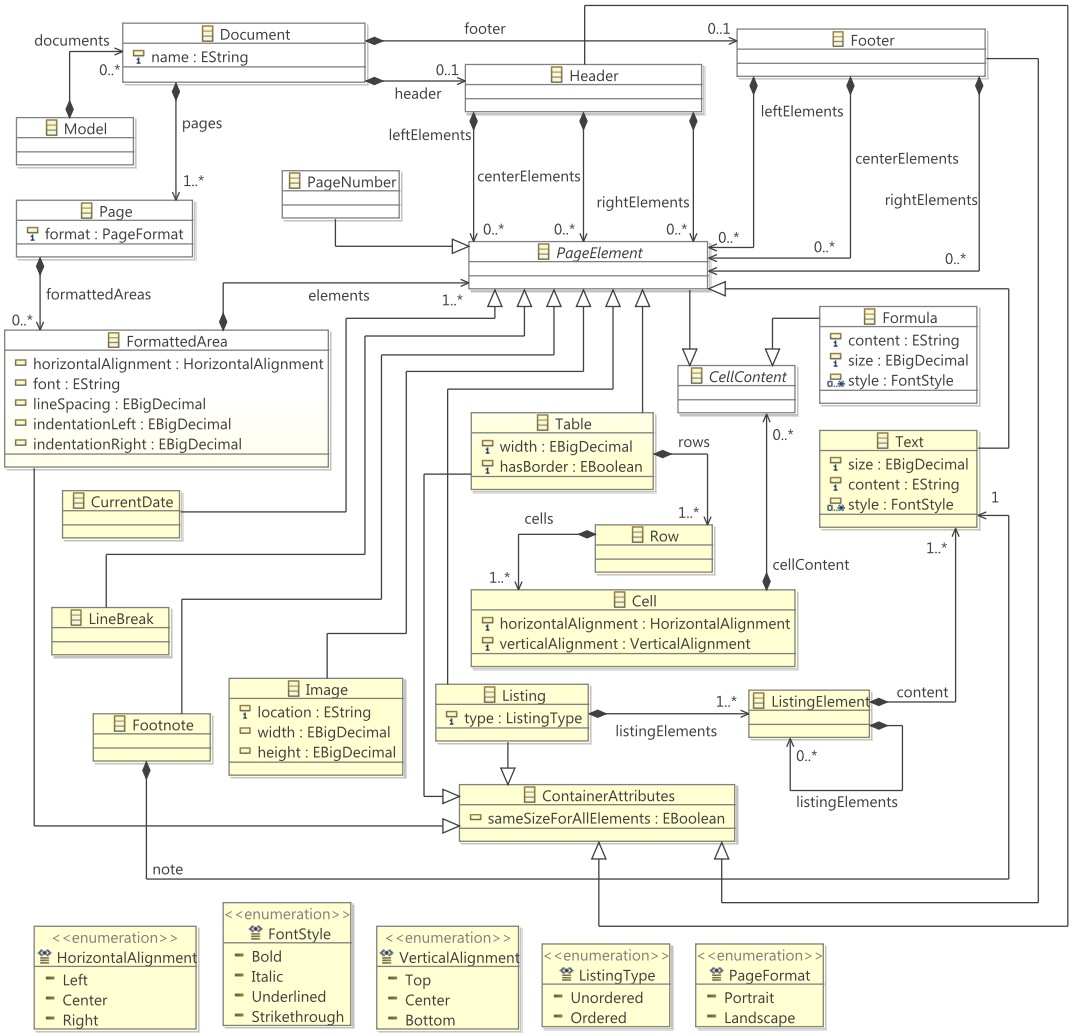


Figure 4.1: Metamodel of platform independent model

has a name, one or many pages, and it could have a maximum of one header and one footer. In most page based documents it is possible to assign different header and footer for each page, but for the reference implementation the header and footer is set for the whole document for the sake of simplicity. Of course it is also possible to not assign header or footer or just a header without footer and vice versa. Another simplification is the assumption that header and footer can consist of three parts: a left area, a center area and a right area. Each area can have none or multiple *Page Elements*.
The *Page* element has a format which is either *Portrait* or *Landscape*. Each page can contain multiple *Formatted Areas*. A formatted area is characterized by the same font, horizontal alignment, line spacing, indentation from left and from right side. The horizontal alignment of the formatted area can either be left, centered or right and applies to the whole content of the formatted area, but not to the formatted area itself. The line spacing and font applies to the text inside the formatted area, where line spacing is a rational number, which defines the space between each line, and the font is set as arbitrary string defined by the unique name of the font. The width of the formatted area results from the available width of the page minus size of the left and right indentation.

Formatted areas contain one or many page elements. The *Page Element* is an abstract class which can either be *Table*, *Text*, *Listing*, *Image*, *Footnote*, *Page Number*, *Current Date* or *Line Break*. Also the elements in footer and header are of type *Page Element*. *Page Number*, *Current Date* and *Line Break* are stand-alone elements and do not have any attributes. A footnote has a text, which is represented by a *Text* element. *Text* elements contain the content itself in form of a string representation and can be defined by its font size and by a *Font Style*. The style of a text can either be *Bold*, *Italic*, *Underlined*, *Strikethrough* or there is no specific style set for a *Text* element.
Images are defined by their width, height and location. The location is a mandatory attribute which is the path to the file that represents the image. If the workflow is distributed over multiple machines (cf. Section 4.6), the user must be careful to use a valid path, that is reachable from the machine where the document generation takes place. Images can also have a width and a height. In an advanced version an image does not necessarily have a location, since it could be drawn within the document for example. But again for the sake of simplicity the *Image* element always refers to a file in the reference implementation.

The page element *Table* can be designed by adding an arbitrary number of *Row* elements and for each row an arbitrary number of *Cell* elements can be added. In this simplified version it is intended that the number of cells for each row is the same. Additionally, the width can be defined for the *Table* element as percentage share of the pages width, as well as the information if a border should be added or not. For the sake of simplicity it is not possible to define different border settings for single cells or to change the border style (solid, dashed, dotted, etc.), the color, or the thickness of the border. The setting if a border is used or not is set globally for the whole table and per default a border is set. For each cell the horizontal alignment (*Left*, *Center* or *Right*) and the vertical alignment (*Top*, *Center* or *Bottom*) can be set.
Cells contain one or many abstract *Cell Content* elements, which are either page elements or a

Formula element. In case of page elements, each element described above can be the content of a cell, even another table. Formula elements can have the same attributes as text, which is the font size, an optional style, and the content of the formula. The content are formulas as used in Microsoft Excel like e.g. *=SUM(A1:B5)*.

Listings are defined by a listing type, which is either an *Unordered* or an *Ordered* list. A list consists of one to many *List Elements*, where each element can have again one to many elements. If a list element contains another list element, this element belongs to the next level of the listing. Each listing element contains one to many text elements with the attributes described above.

The *Container Attributes* class is a special element where all container classes are inheritors. By assigning this class to an element, it is possible to define if all containing elements should have the same font size. Elements that inherit from *Container Attributes* class are *Listing*, *Table*, *Formatted Area*, *Header* and *Footer*. For example, if a table has the attribute *sameSizeForAllElements* set to true, every text or formula in each cell will have the same font size assigned during the diversification process. Per default this behavior is turned off and therefore each child element in a container can have a different font size.

Metamodel of Platform Specific Model for Microsoft Word03/Word07

As described in the introduction of this section there exists just one metamodel for both platforms Microsoft Word 2003 and Microsoft Word 2007 in the prototype, which can be found in the diagram of Figure 4.2. Basically, the elements and attributes from the PIM metamodel are just a subset of the features from the PSM metamodel for Word03/Word07. For this reason only the differences to the PIM metamodel will be described in this subsection.

For the Word platforms it is possible to turn on track revision and auto correction. Both attributes are set globally at the *Document* element. Additionally, the language setting for the document can be set by defining the language name on this level. The *Page Elements* are basically the same as for the PIM metamodel, but some can have different implementations.
The *Current Date* element for example is defined as abstract class that can be added as an *Auto Date* element or as *Manual Date*. It is possible to define the format of the date for both elements by setting its string representation accordingly. For this reason this attribute resides on the abstract class. If the *Auto Date* is used, this means the built-in quickpart from Microsoft Word is inserted for that purpose. If the *Manuel Date* is used instead, a simple text including the current date in the predefined format is inserted. The same case applies to the *Page Number* element, where an abstract element can be implemented either as text in case *Manual Page Number* is used, or as Microsoft quickpart in case the *Auto Page Number* element is inserted.

Also the *Listing* element is an abstract class in the PSM metamodel, either inserted as *Ordered List* element or as *Unordered List* element. If an unordered list is used also the bullet style can diversify. The possibility of adding listing elements in a nested way and the content of listing elements in form of Text elements, works exactly as for the PIM metamodel.

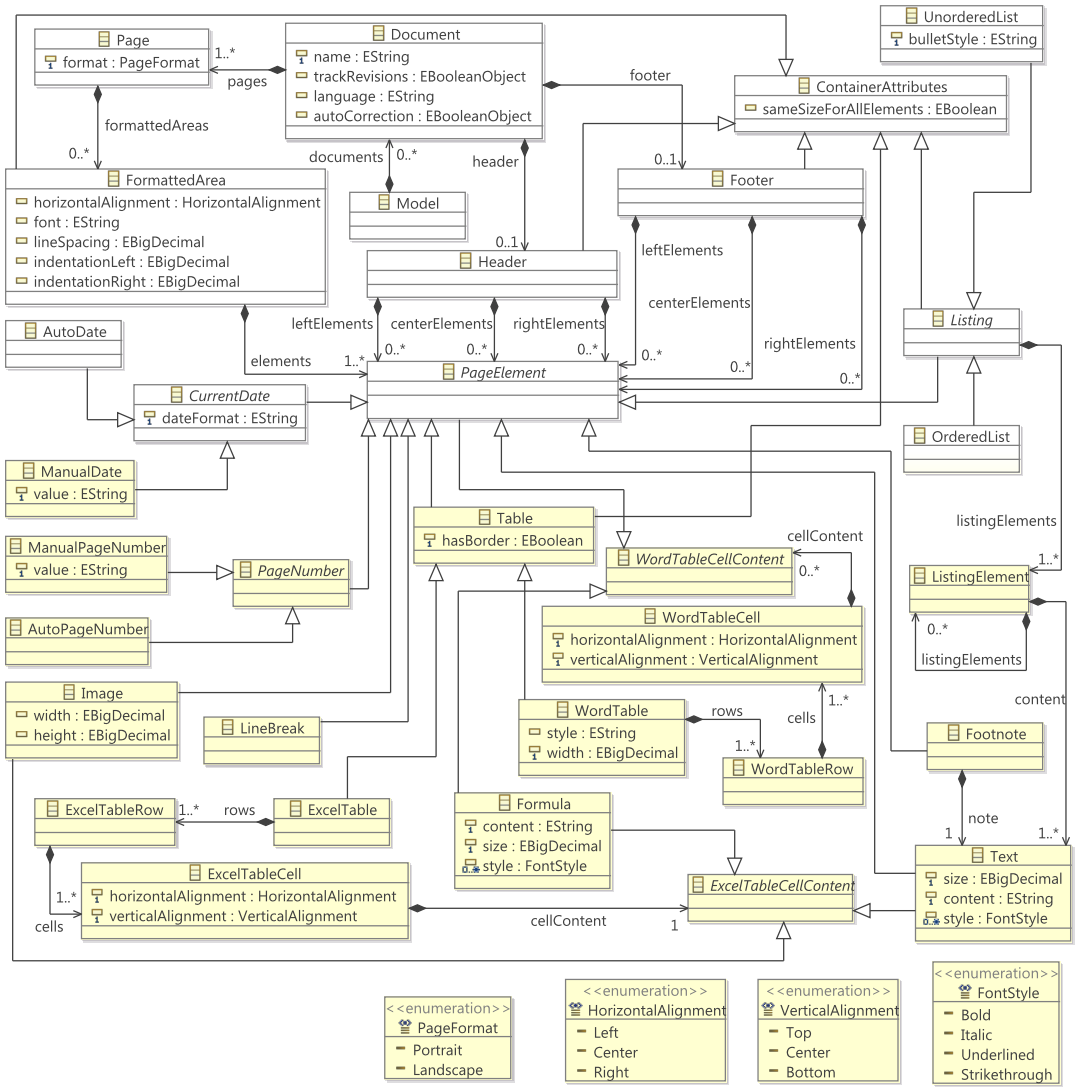


Figure 4.2: Metamodel of platform specific model for Word03/Word07 platform

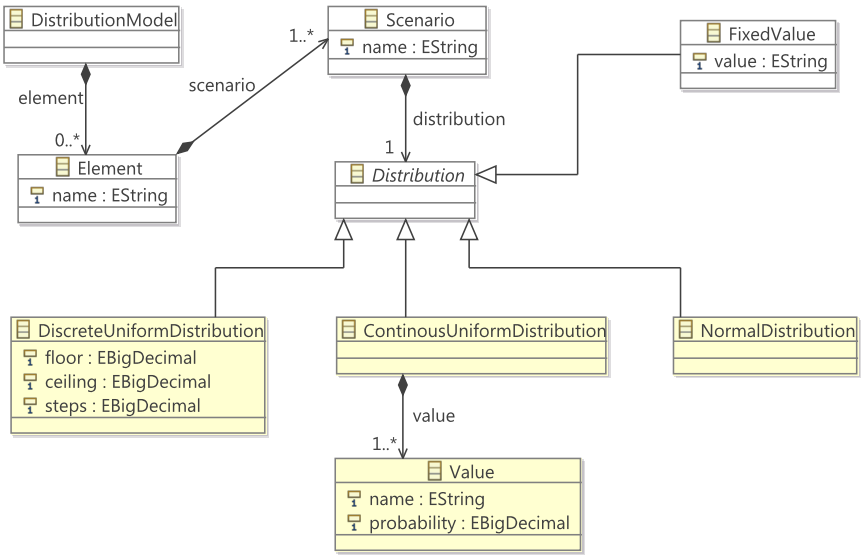


Figure 4.3: Metamodel of distribution model

There are various ways a table can be implemented in Microsoft Word, such as standard Word table, as embedded spreadsheet, using tabbed values etc. In the reference implementation the *Table* element can be implemented as *Word Table* element, which is the standard table in Microsoft Word, or as *Excel Table*, which conforms to an embedded spreadsheet from Microsoft Excel.
A Word table can have a style corresponding to the predefined styles available in Microsoft Word. In contrast to the PIM metamodel only word tables can have a width, since the width of embedded Microsoft Excel sheets can not be changed in Microsoft Word. As in the PIM metamodel, each type can have one to many rows. Depending on the implementation this is either a *Word Table Row* element or an *Excel Table Row* element. The same applies to cells, where the element can be a *Word Table Cell* element or an *Excel Table Cell* element, each having a horizontal and a vertical alignment, just as in the PIM metamodel. The possible content of cells is different for the particular implementation. Excel table cells can either contain an *Image* element, a *Text* element or a *Formula* element. A Word table cell in contrast can be any *Page Element* or a *Formula* element.

Metamodel of Distribution Model

The distribution model is used to introduce the statistical input data that is used in the diversification process. Also this model conforms to a metamodel, which is shown in Figure 4.3. The root element of this model, the *Distribution Model*, can have multiple *Element* classes. Each element is defined by its name, which should be unique within the model. An element is corresponding to a class or attribute that can be diversified. Examples are font size, page format, technical implementation of a table etc.

For each element different scenarios can be available. Depending on the scenario, the values or even the distribution can be different. For example, the element *Font Size* will have different values for the scenario *Header* than for the scenario *Footnote*, which means that the text where the size should be diversified is used either inside a header element or, in the latter case, inside a footnote. Also, the scenario has a unique name, and for elements that only have a single scenario, the name could either be set to *Default* or left empty depending on the implementation of the particular diversification (cf. Section refsec:diversification).

There is exactly one distribution assigned to each scenario. The actual distribution, which inherits from the abstract *Distribution* class, could be any distribution from simple distributions such as discrete uniform distribution to more complex distributions such as Poisson distribution. In the reference implementation only simple distributions are used for the sake of simplicity, which are discrete uniform distribution and continuous uniform distribution. Additionally, it is possible to use the *Fixed Value* element in order to define an attribute that should not change its value.
For the *Discrete Uniform Distribution* element the parameters floor, ceiling and steps have been added as refinement for the common discrete uniform distribution. The *Continuous Uniform Distribution* element, in contrast, has been extended by adding multiple options, each with a certain probability. For that purpose one to many *Value* elements are assigned, each having a unique name, and the corresponding probability. For example the value of a vertical alignment (*Top*, *Center* or *Bottom*) could be calculated using a continuous uniform distribution.
The *Normal Distribution* element is just a placeholder and not yet implemented. Additional distributions can be easily added by inheriting from abstract *Distribution* class and by adding the parameters that are necessary to calculate the values for the particular distribution as attributes. By using this solution, a flexible and decoupled way of extending the statistical input has been created.

Listing 4.1: XMI serialization of the distribution metamodel

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
5     name="distribution" nsURI="http://distribution/1.0"
6     nsPrefix="distribution">
7    <eClassifiers xsi:type="ecore:EClass" name="DistributionModel">
8      <eStructuralFeatures xsi:type="ecore:EReference"
9        name="element" upperBound="-1"
10       eType="#//Element" containment="true"/>
11   </eClassifiers>
12   <eClassifiers xsi:type="ecore:EClass" name="Element">
13     <eStructuralFeatures xsi:type="ecore:EAttribute"
14       name="name" lowerBound="1"
15       eType="ecore:EDataType
16            http://www.eclipse.org/emf/2002/Ecore#//EString"/>
```

```
17    <eStructuralFeatures xsi:type="ecore:EReference"
18       name="scenario" lowerBound="1"
19       upperBound="-1" eType="#//Scenario" containment="true"/>
20  </eClassifiers>
21  <eClassifiers xsi:type="ecore:EClass" name="Scenario">
22    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
23       lowerBound="1"
24       eType="ecore:EDataType
25            http://www.eclipse.org/emf/2002/Ecore#//EString"
26       defaultValueLiteral=""/>
27    <eStructuralFeatures xsi:type="ecore:EReference" name="distribution"
28       lowerBound="1" eType="#//Distribution" containment="true"/>
29  </eClassifiers>
30  <eClassifiers xsi:type="ecore:EClass" name="Distribution" abstract="true"/>
31  <eClassifiers xsi:type="ecore:EClass" name="DiscreteUniformDistribution"
32     eSuperTypes="#//Distribution">
33    <eStructuralFeatures xsi:type="ecore:EAttribute" name="floor"
34       lowerBound="1"
35       eType="ecore:EDataType
36            http://www.eclipse.org/emf/2002/Ecore#//EBigDecimal"/>
37    <eStructuralFeatures xsi:type="ecore:EAttribute"
38       name="ceiling" lowerBound="1"
39       eType="ecore:EDataType
40            http://www.eclipse.org/emf/2002/Ecore#//EBigDecimal"/>
41    <eStructuralFeatures xsi:type="ecore:EAttribute" name="steps"
42       lowerBound="1"
43       eType="ecore:EDataType
44            http://www.eclipse.org/emf/2002/Ecore#//EBigDecimal"/>
45  </eClassifiers>
46  ...
47 </ecore:EPackage>
```

4.2 Model Definition

As described above, the source model is generated manually in our current approach. Beside that, also the statistical input in form of distribution models is modeled by the user. In an improved version of the workflow both models could be created in an automated or semi-automated way. Especially the injection of statistical input should not rely on interaction with the user. At least the source of the used distribution data must be collected from representative real world collections. In the prototype simplified values are used as well as very basic distributions. A more detailed description of possible improvements in that area can be found in Section 6.2.

When creating a model instance from a particular metamodel in EMF, there are two different approaches available in Eclipse. First the dynamic instance tool can be used, which is the most

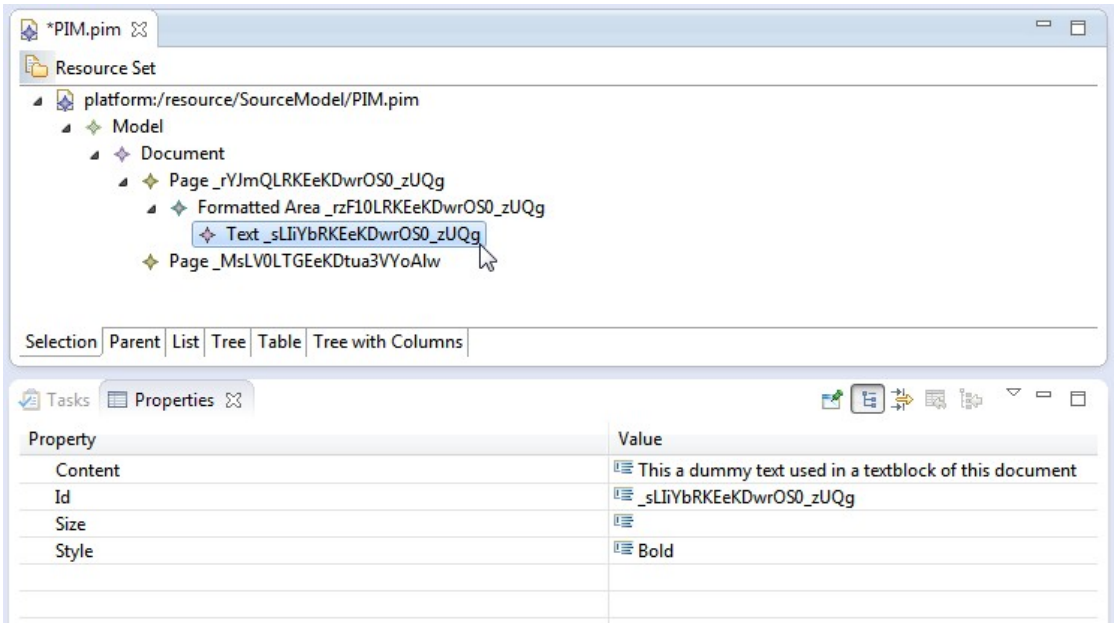


Figure 4.4: EMF editor for designing model instances

simple way, and the second solution is to use a reflective editor. In the reference implementation the latter approach is used for reasons described in Section 4.7.

In both cases EMF provides a wizard and an editor with a graphical user interface to design instances of a metamodel. As shown in Figure 4.4, the graphical user interface (GUI) of the editor uses a tree view in order to display the structure of the created model instance. In this GUI the user can add new elements by right-clicking on the particular node, and add or edit attributes in the properties view.

The elements and attributes that are designed in the source model will also be available in this form in the generated documents. That means that the user can influence the output by defining attributes explicitly. In the prototype it is also necessary to actually insert the content of text elements. Instead of this, an alternative approach would be to use dummy text that is inserted during the workflow using statistical data as basis.

Only attributes that are available in the metamodel of the PIM can be defined manually. Attributes that are only available in the platform specific metamodels are exclusively set during the model transformation step. Therefore the only way to influence the setting of such an attribute is to change its statistical values in the distribution model. That means, for example, that it is not possible to manually set the language of a Word document or the implementation of a table.

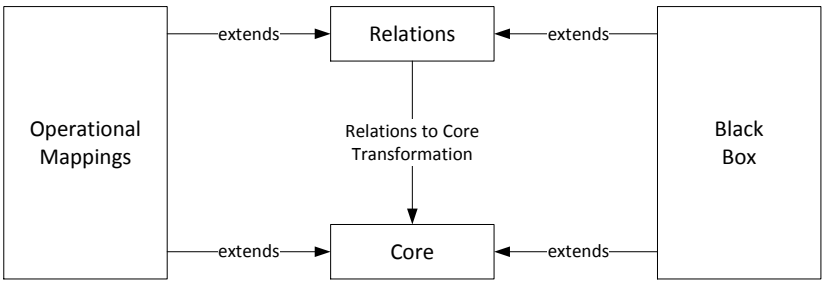


Figure 4.5: QVT architecture

4.3 Model Transformation

After the manual steps have been completed, the automated part of the workflow starts with the model transformation steps. In the reference implementation the model transformation is used to generate diversified instances of the source model on one hand, as well as for transforming PIMs into PSMs on the other hand. The diversification step will be discussed in more detail in the next section. First the base of the diversification process will be described in this section.

One goal for the implementation of the workflow is to develop a homogeneous and reliable approach by using Eclipse elements wherever possible. Since the common model transformation techniques in Eclipse are ATL and QVT, these solutions were potential candidates for the model transformation part.

QVT, which stands for Query/View/Transformation, generally is a standard language used for model transformation, developed by the OMG[4]. According to Kurtev, QVT languages consist of three layers that are core language, relations language, and operational mappings [36] (cf. Figure 4.5). The core language defines the basis for the semantics of the relations language in a declarative way. With the relations language, the relations between models can be defined, again in a declarative way. This is used for the specification of the actual model transformations. The operational mappings extend the relations language by using OCL, and an imperative language, which enables the use of constructs such as loops or conditions.

An appropriate alternative to QVT for model transformations is the ATLAS transformation language (ATL). As described by Jouault and Kurtev, ATL is a hybrid language, offering the possibility to define transformations in an imperative way as well as in a declarative language, whereby the usage of the latter is recommended whenever possible [32]. According to Jouault et al., there exists a similarity to QVT since a large part of features are also available in ATL [31]. The model transformations are specified by means of modules. Each module consists of a header, which contains the definition of the input and output models, an optional import

[4]http://www.omg.org/spec/QVT/

section, which allows the usage of existing libraries, and the transformation rules. Transformation rules can either be implemented as matched rules, which are declarative rules that define the relation between source and target patterns for specific elements, or as imperative constructs. The latter ones can be used in cases where algorithms are too complex for using a declarative solution.

In the reference implementation QVT is used as technology for model transformations as a result of benefits regarding available features. The disadvantage of ATL towards QVT is the fact that ATL does not provide a built-in function for copying the source model in order to generate multiple models within the diversification process. This is necessary since the approach for the diversification step relies on a process that makes a one-to-one copy of the source model for each target model to be created. QVT provides the *deepclone* function for that purpose, in ATL such a method is not available.
Another benefit of QVT towards ATL is the possibility of using external functions, which is described below in subsection Black Box Implementation. Without that feature it is not possible to fulfill the given requirement of using distributions as input. Since neither QVT nor ATL provides advanced mathematical functions, or at least randomize functions, calculating the distributions cannot be realized directly within the model transformation code. ATL obviously does not support the usage of external functions and therefore cannot be used for that purpose.

Generally, there are different implementations of QVT available with differing support regarding specified features. The reference implementation uses Operational QVT (QVTo), which is an official open source implementation of the operational mappings language in Eclipse. According to Dvorak, this implementation was developed for designing transformations that are able to build complex target models by defining explicit steps in an imperative respectively procedural language [11]. Commonly, a source model conforming to a metamodel A is converted to a target model conforming to a metamodel B. But as it is partly used in the reference implementation, it can also transform a source model to a target model, both conforming to the same metamodel.
A QVTo model transformation consists of the declaration of input and output models, a main method which is the entry point of each transformation, and mapping operations where the mapping between single elements of input and output models are defined. Moreover, helper classes can be imported, which are either other transformation files or libraries such as black box functions. Additionally also query operations can be used, which are read-only helper functions within the transformation file. Within the mapping definitions, imperative OCL constructs such as loops, switches, execution controls, or exceptions can be used.

In the reference implementation an Operational QVT project is used, including one transformation file for the diversification of the PIM, and theoretically one transformation file for the transformation from PIM to PSM for each platform. Since the two platforms Word03 and Word07 share the same metamodel, there is also just one transformation file for both platforms. As depicted in the first lines of the example in Listing 4.2, the declaration of each transformation consists of two input models and one output model. In this example, which contains the first few

lines of the model transformation file where the PIM is diversified, the input and output files conform to the same metamodel. In the model transformation file for the conversion from PIM to PSM, the output model refers to another metamodel, and therefore an additional modeltype definition is included for the PSM (cf. Listing 4.3).
Additionally to the modeltype definitions, a metamodel mapping has to be added to the project. Therefore, the used metamodels must be inserted into the QVT settings in the projects properties by defining source model URI and target model URI for the particular metamodel. The source model URI corresponds to the one used in the transformation file (e.g. http://pim/1.0) and the target model URI is the actual path to the metamodel.
As mentioned above, the `main()` operation acts as entry point for the transformation process. It contains the invocation of the root element's mapping, which in the case of the prototype is the *Model* element. The reason for having a root element one level below the *Document* element, is the fact that this solution requires the definition of all generated target models within a single model. This is necessary since in the code generation step it is much easier to iterate through elements from one model, than iterating through documents, each residing in an own model and therefore in an own file. This functionality is inherently not available in the chosen code generation technique, and would have to be resolved by means of batch processing or something similar.

For each element of the PIM metamodel there is a mapping operation available. In case of the PIM diversification step, the type of input and output element are always of the same kind (cf. Listing 4.2). For the PIM to PSM transformation this is a mapping to the corresponding element from the PSM metamodel. As described in the previous section some elements are exclusively set during this step. For example the language attribute of a document is not available as attribute in the PIM since this feature might be missing for some word processors. Therefore, the value of this attribute is automatically set within the model transformation step by using its specific distribution.
In the mapping operation `Model2Model()` of the PIM diversification, which contains the conversion of the root element, the diversified copies of the target model are being generated. Therefore, as mentioned above, the *deepclone()* function is used to copy the target model with all its elements, attributes and references, which are forwarded as input for all mapping operations of the transformation file. An example is the `Document2Document()` mapping in Listing 4.2, where the original document is used as input, and its containing elements such as the references to header, footer and pages are mapped to the newly created output document. Additionally, the document's name is set by adding a unique suffix to each document, which is necessary when the documents are being generated in the last step of the workflow to avoid files with the same name. In the mapping operation `Page2Page()` a first example of an attribute diversification is given. By calling the helper function `GetValueFromDistribution()`, the value for the current document is returned, which conforms to a particular distribution. A detailed description of that process can be found in Section 4.4.
The mapping of the attribute *id* is necessary to fulfill the requirement of traceability. As described in Section 4.7, unique identifiers are automatically set in the source model during its generation step. In order to use the same identifier for the same element throughout the whole

model chain, this mapping is available in all mapping operations. In general all other mapping operations are following the same principle regarding mapping of attributes and references.

Listing 4.2: Part of the QVTo model transformation for the diversification of the PIM

```
1  modeltype PIM uses pim('http://pim/1.0');
2  modeltype DIST uses distr('http://distribution/1.0');
3
4  transformation PimDiversification(in source:PIM, in distr:DIST, out target:
       PIM);
5
6  main()
7  {
8     source.rootObjects()[PIM::Model]->map Model2Model();
9  }
10
11 mapping PIM::Model::Model2Model() : PIM::Model
12 {
13    while (i := 0; i < numberOfDocuments)
14    {
15       var sourceDocument := self.documents.deepclone()[Document];
16       documents += sourceDocument->map Document2Document();
17       i := i + 1;
18    }
19 }
20
21 mapping PIM::Document :: Document2Document() : PIM::Document
22 {
23    if (self.name = null) then name := defaultDocumentName + '_' +
          documentNameSuffix.toString()
24    else name := self.name + '_' + documentNameSuffix.toString() endif;
25    pages += self.pages->map Page2Page();
26    header := self.header->map Header2Header()->asSequence()->first();
27    footer := self.footer->map Footer2Footer()->asSequence()->first();
28
29    documentNameSuffix := documentNameSuffix + 1;
30 }
31
32 mapping PIM::Page :: Page2Page() : PIM::Page
33 {
34    id := self.id;
35    formattedAreas += self.formattedAreas->map FormattedArea2FormattedArea();
36    var pageFormatAsStr := GetValueFromDistribution("PageFormat", "", self.
          format).repr();
37
38    switch
39    {
```

```
40          case(pageFormatAsStr = "Portrait"){format := PageFormat::Portrait};
41          case(pageFormatAsStr = "Landscape"){format := PageFormat::Landscape };
42      };
43  }
```

The mapping operation for the root element `Model2Model()` in the PIM to PSM model transformation does not contain any copy logic as in the PIM diversification file (cf. Listing 4.3). The generated *Document* elements from the previous transformation step are simply used, and converted into corresponding Word03, respectively Word07 elements. In that step, again, some elements are diversified in the same manner as in the previous transformation step, but in that case on platform specific level. Apart from that, the logic behind the mappings is similar to the logic that is used for the PIM diversification.

Listing 4.3: Part of the QVTo model transformation from PIM to PSM

```
1  modeltype PIM uses pim('http://pim/1.0');
2  modeltype PSM uses psm_word07('http://psm_word07/1.0');
3  modeltype DIST uses distribution('http://distribution/1.0');
4
5  transformation PimToPsm_Word07(in source:PIM, in distribution:DIST, out
       target:PSM);
6
7  main()
8  {
9      source.rootObjects()[PIM::Model]->map Model2Model();
10 }
11
12 mapping PIM::Model::Model2Model() : PSM::Model
13 {
14     documents := self.documents->map Document2Document();
15 }
16
17 mapping PIM::Document :: Document2Document() : PSM::Document
18 {
19     name := self.name;
20     pages += self.pages->map Page2Page();
21     header := self.header->map Header2Header()->asSequence()->first();
22     footer := self.footer->map Footer2Footer()->asSequence()->first();
23     language := GetValueFromDistribution("Language", "", null).repr();
24     autoCorrection := GetValueFromDistribution("AutoCorrection", "", null).
           repr().toBoolean();
25     trackRevisions := GetValueFromDistribution("TrackRevisions", "", null).
           repr().toBoolean();
26 }
```

An advantage of QVTo is the fact that it supports polymorphism in terms of overloading mapping operations. Since heritage is widely used in the metamodels, some elements can assume different forms. *Page Element*, for example, is a central element in all metamodels, which can exist in various object types. If overloading would not be supported, this would result in transformations with many if-conditions and therefore less readability and maintainability. By using overloaded mapping operations, this behavior can be implemented in a neat solution.
As illustrated in Listing 4.4, queries are used for that purpose. It is important that the mapping operation for the abstract element is also available, otherwise the transformation code does not compile.

Listing 4.4: Overloading of mapping operations in QVTo

```
1  mapping PIM::FormattedArea :: FormattedAreaMapping() : PSM::FormattedArea
2  {
3     elements += self.elements.PageElement2PageElement();
4  }
5
6  query PIM::PageElement :: PageElement2PageElement() : PSM::PageElement
7  {
8     log("Should not reach this point: PageElement is an abstract element");
9     return null;
10 }
11
12 query PIM::Text :: PageElement2PageElement() : PSM::PageElement
13 {
14    return self->map Text2Text()->asSequence()->first();
15 }
16
17 query PIM::LineBreak :: PageElement2PageElement() : PIM::PageElement
18 {
19    return self->map LineBreak2LineBreak()->asSequence()->first();
20 }
```

Black Box Implementation

The built-in functions and libraries of QVT Operational are strongly limited. To solve more complex tasks QVT provides so called black box functions. These functions can be implemented in any programming language and can then be invoked from QVT code. This makes QVT much more powerful, and finally applicable for our use cases [45]
To describe the practical usage of black box functions and to demonstrate how these functions can be implemented in QVTo, an example from the reference implementation is used:
A major problem was the calculation of values for the distributions during the diversification process since mathematical operations in QVTo are limited to basic arithmetical operations. But for most distributions it is necessary to apply some kind of random function such as `Math.random()` in Java.

Java class	OCL type
java.lang.Object	OclAny
java.lang.String	String
java.lang.Boolean	Boolean
java.lang.Integer	Integer
java.lang.Double	Real
java.lang.Short	Integer
java.lang.Long	Integer
java.lang.Float	Real
java.util.Collection	Collection
java.util.List	Sequence
java.util.Set	Set
java.util.LinkedHashset	OrderedSet
org.eclipse.ocl.util.Bag	Bag

Table 4.1: Mapping between Java classes and OCL types for most common types

For this purpose a plug-in project with the namespace *GroundTruthGenerator.transform.blackboxlib* has been added to the reference implementation in Eclipse. To provide functionality for the calculation of distribution values, the class *DistributionCalculator* has been created, which contains several statistical calculation methods.As for all Java methods, different objects are accepted as parameters and an object can be returned. The data types that are transfered between Java code and QVT code are mapped as described in Table 4.1, which is taken from Eclipse documentation[5].
When creating a new project it might be helpful to start with the black box example project from QVT. Therefore *File* > *New* > *Example* > *Black Box Library Definition* must be selected, the project name must be set, and finally the classes and methods must be implemented with the particular Java functionality. The advantage of this approach is that all necessary libraries for QVT black box implementations are already included in the project.

To make black box implementations available in the QVT project it is necessary to add appropriate extension points to the plug-in. This can be done either by opening *plugin.xml* with the default editor in Eclipse and using the graphical user interface provided in the *Extensions* tab, or by adding the extension point and the corresponding unit entry manually in a text editor. In both cases an entry similar to Listing 4.5 should exist in *plugin.xml* afterwards.
To export the plug-in by creating a .jar file out of it, *plugin.xml* must be opened with the default editor and the *Export Wizard* must be used from the *Export* section in the *Overview* tab. In this wizard it is important to select the proper plug-in, which is *GroundTruthGenerator.transform.blackboxlib* in case of the prototype. The folder which contains the plug-in directory of Eclipse must be selected as destination directory.
After restarting Eclipse the black box functions can be used inside the QVT project by adding

[5] http://help.eclipse.org/juno/topic/org.eclipse.m2m.qvt.oml.doc/references/blackboxing.html

`import GroundtruthGenerator.transform.DistributionLib;` to the appropriate transformation file.

Listing 4.5: Black box extension point configuration

```
<extension point="org.eclipse.m2m.qvt.oml.javaBlackboxUnits">
   <unit name="DistributionLib" namespace="GroundtruthGenerator.transform">
      <library class="GroundTruthGenerator.transform.blackboxlib.
          DistributionCalculator" name="DistributionLib">
      </library>
   </unit>
</extension>
```

4.4 Diversification

In this section the actual implementation of the theoretical approach from Section 3.3 is described in detail. The first step of the model transformation process is entirely used for the diversification of elements in the source model. Beside transforming the diversified models from PIM to PSM, also in the second model transformation step the diversification of the elements, in that case at platform level, is an important task. The process of diversification is the same for both steps. The transformation file *DistributionHelper* is included to both transformations as external helper class by using the *import* functionality of QVTo. This file consists entirely of helper operations, which again use overloading of operations.

Listing 4.6: QVTo helper function for identifying the current scenario of an element

```
helper GetFontSizeScenario(text:PIM::Text) : String
{
   switch
   {
      case (text.container().oclIsKindOf(Header)) return "Header";
      case (text.container().oclIsKindOf(Footer)) return "Footer";
      case (text.container().oclIsKindOf(Footnote)) return "Footnote";
      else return "Default";
   };

   return "Default";
}
```

First the helper function `GetValueFromDistribution()` from the *DistributionHelper* file is called from within a mapping operation (cf. `Page2Page()` operation in Listing 4.2). This function accepts either two or three parameters, which are *diversifiedObjectName* and *currentScenario* in both cases, and additionally *assignedValue* in case of three input parameters.
The first parameter is the name of the object that should be diversified. This name must have

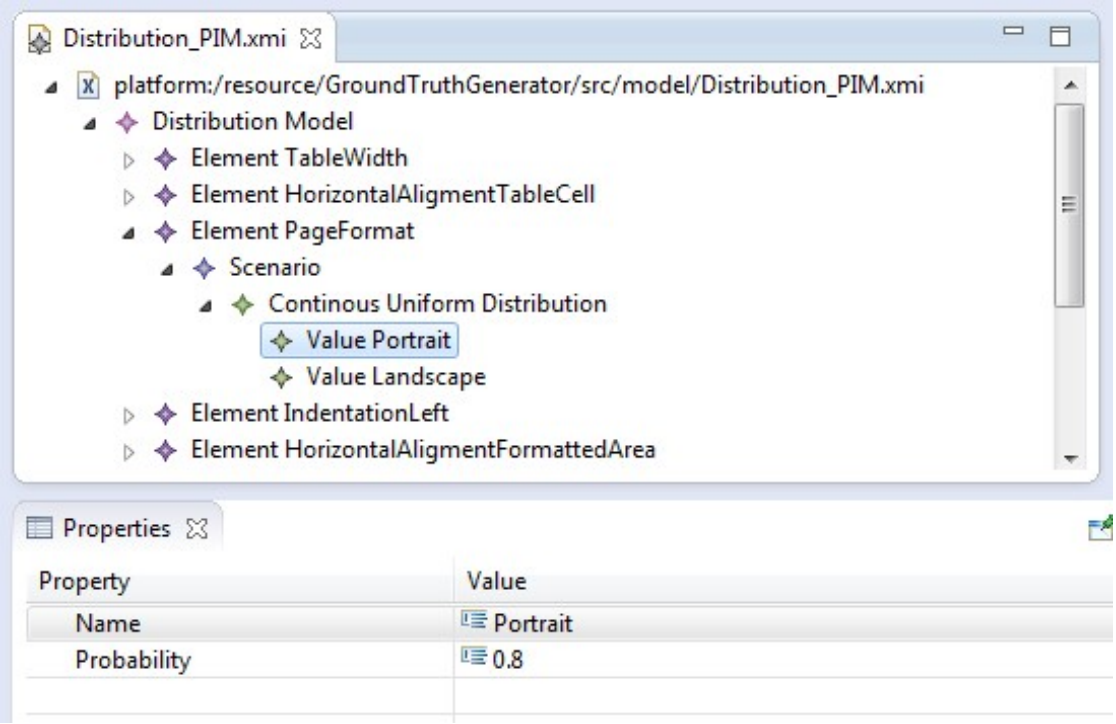


Figure 4.6: Distribution model

a corresponding element in the distribution input file, such as *PageFormat* in Figure 4.6. This model confirms to the metamodel defined in Figure 4.3. The second parameter is the scenario that applies to the current element. This additional information is necessary since elements can have different distributions depending on the occurrence and state of the element. In this example no specific scenario is defined, and therefore an empty string is used. But for other elements multiple scenarios can exist, each defined by a unique name.

The scenario that is relevant for the current situation is identified by means of helper functions, residing in the particular transformation file. The content of such helper functions are varying depending on the element. In any case either the position within the document, or the properties of the element itself, such as type of child elements are inspected in that function. In Listing 4.6 an example of a helper function that identifies the current scenario for the attribute *Font Size* of a *Text* element can be found.
The third attribute of the function with a signature of three parameters, is the value that has been assigned by the user. For most attributes that are diversified during the workflow, the user has the option to assign a value when designing the source model. In that case this attribute is not diversified, but the assigned value is used. Therefore the function `GetValueFromDistribution()` immediately returns the assigned value again, instead of calculating a value from the distribution. If the assigned value is empty, the distribution is calculated as usual.

With the knowledge of the element name and the current scenario, the proper distribution can be selected from the distribution model. Depending on the current transformation step, and thus on the input model that has been defined in the transformation file, either the distribution model for the PIM diversification, or a distribution model for the particular platform is used. In the example of Figure 4.6, the result would be a *Continuous Uniform Distribution* element.
In the *DistributionHelper* file each distribution has an associated overloaded helper function `ValueFromDistribution()` that is called from the `GetValueFromDistribution()` operation. Each distribution will have its own logic, and depending on the complexity more or less is implemented directly in the helper function. In many cases, however, a corresponding Java code will be called, which implements the complex part of the calculation. In Figure 4.7, the function for discrete uniform distributions is illustrated. The parameters *floor*, *ceiling* and *steps* are selected from the distribution model as input for the `getValueForDiscreteUniformDistribution()` method from the corresponding external Java class *DistributionCalculator*, which is imported by means of black box functions.

Listing 4.7: QVTo helper function for calculating the discrete uniform distribution

```
1 query DIST::DiscreteUniformDistribution :: ValueFromDistribution() : OclAny
2 {
3    var currentDist = self;
4    var floor := currentDist.floor;
5    var ceiling := currentDist.ceiling;
6    var steps := currentDist.steps;
7    var value := getValueForDiscreteUniformDistribution(floor.repr().asInteger
         (), ceiling.repr().asInteger(), steps.repr().asFloat());
8    return value;
9 }
```

How external functions are integrated into the model transformation step has already been described in Section 4.3. In the particular example of a discrete uniform distribution, the actual implementation within Java from the reference implementation can be found in Figure 4.8. The parameter for an upper and lower border, as well as the definition of steps are usually not part of a common discrete uniform distribution. In the prototype these parameters has been added to yield more realistic results.

Listing 4.8: External Java method for calculating the discrete uniform distribution

```
1 public static double getValueForDiscreteUniformDistribution(int floor, int
      ceiling, double steps)
2 {
3    int maxValueForCalc = (ceiling - floor) * 10;
4    double stepsForCalc = steps * 10;
5    maxValueForCalc += stepsForCalc;
6    double randomNumber = Math.random() * (maxValueForCalc / stepsForCalc);
7    int iRandomNumber = (int)randomNumber;
```

Diversification step	Diversification level	Example
PIM diversification	Domain-specific level	Page format
PIM to PSM transformation	Domain-specific level	Font size
PIM to PSM transformation	Technical level	Table implementation

Table 4.2: Different levels of diversification

```
8     iRandomNumber *= stepsForCalc;
9     double result = (double)iRandomNumber / 10;
10    return result + floor;
11 }
```

In the reference implementation two distributions are currently implemented. Beside the discrete uniform distribution, which is used for attributes such as table width, width and height for images, font size, etc., the continuous uniform distribution is available. In that case however, a specific application has been chosen for the continuous uniform distribution within the prototype. The calculated random variable of a continuous range between 0 and 100 is used to select a value out of several predefined options. Each option is assigned to a value range within the sample space, and the mass of each option is corresponding to a certain probability.
An example of such a definition within the distribution model can be found in Figure 4.6. The element *Page Format* can have two values, each having a different mass. In this particular example the value *Portrait* has a probability of 80 percent. This means a mass of 80 percent from the sample space are reserved for that value, and if the continuous uniform distribution yields a random number of that mass, the value *Portrait* is returned.
Additionally to these distributions it is also possible to define a static value. Therefore the distribution *Fixed Value* must be added to the distribution model, and there the value can be set accordingly.

The diversification process can also be split into diversification on domain-specific level and on technical level. The domain-specific diversification only takes place at the PIM diversification step, while in the second step both, domain-specific and technical diversification, are executed (cf. Table 4.2).
Parameters which are diversified on domain-specific level influences the appearance of a document. Common examples are font type, font size or line spacing. In contrast, diversification on technical level influences how elements are implemented. For example, a table can be implemented in different ways, depending on the platform. In Microsoft Word a common table can be added, or it can be implemented as embedded spreadsheet from Microsoft Excel. With a few constraints such as usage of borders, tables could be added as simple text, either by using tab stops or by using multiple text columns.
It is important to include both diversification levels to maximize variety of documents. Especially different technical implementations are extremely useful to verify correctness of characterisation tools regarding support of different features. As in real world documents where different approaches can be used to implement specific features, this variety should also be represented in

the ground truth created by the reference implementation.

4.5 Code Generation

The code generation step can be slightly different for various platforms, at least regarding generated output. For some domains and platforms this step could already create the actual benchmark data. In the particular case of the reference implementation however, another step is necessary which executes the generated output from the code generation step. More precisely the platform specific target models from the PIM to PSM model transformation step are used as input, in order to generate Visual Basic (VB) macro code for the platforms Microsoft Word 2003 and Microsoft Word 2007.

For code generation, which is basically a Model-to-text transformation, Eclipse offers various solutions such as JET, Xpand or Acceleo. The prototype uses Xpand as language for the code generation step, since it is a powerful but yet easy to use solution that offers many advantages. Xpand is a statically-typed domain-specific template language, originally developed by openArchitectureWare (oAW), which is now part of the Eclipse Modeling Project. The syntax of the language is limited, but still satisfies most requirements in the area of code generation. According to Gronback, it relies on Xtend, which is a high level programming language from oAW framework, and the underlying expression language, additionally it supports aspect-oriented programming [19]. The execution of Xpand is handled by the Modeling Workflow Engine (MWE), which is responsible for model parsing and text generation. As described by Klatt, Xpand can be used independent of the model type [33]. The parsers for EMF, UML, textual models, and even for models created in Microsoft Visio, are supported natively. Moreover, a post processor for generated code is available within the workflow, which can be included by defining so called beautifier. These processors are responsible for transforming the generated code, according to code conventions of languages such as Java or XML.

Major benefits of Xpand are support of polymorphism and type safety. Similar to QVT the overloading of template definitions is possible, which enables a more sophisticated programming style. Listing 4.9 contains an example from the reference implementation, where overloading is used to dynamically insert code, depending on the current page element.
According to Jugel and Preußner, further advantages of Xpand are the excellent tool support in Eclipse, which provides features such as syntax highlighting and code-completion within the editor [35].

Listing 4.9: Overloading of template definitions in Xpand

```
1 «DEFINE PageElements FOR Table»
2    numberOfTables = numberOfTables + 1
3    «EXPAND BuildTable FOR this»
4 «ENDDEFINE»
5
6 «DEFINE PageElements FOR Text»
```

```
   Selection.Font.Size = «((Text)this).size.toInteger()»
   Selection.TypeText Text:="«((Text)this).content»"
«ENDDEFINE»

«DEFINE PageElements FOR LineBreak»
   Selection.TypeParagraph
«ENDDEFINE»
```

The code generation process in the reference implementation is started by iterating through all *Document* elements of the PSM in the MWE file. For each document a VB macro for Microsoft Word is created within the executed Xpand template file. The content is the same as from a macro that is recorded manually within Microsoft Word 2003 or 2007. The template file is divided into a main body that creates the basic structure of a VB *Function* procedure, and into multiple *DEFINE* blocks that are executed from the main body by means of *EXPAND* directives. As illustrated in the example of Listing 4.10, this allows a neat template code, which is easy to debug and to maintain.

Listing 4.10: Basic structure of an Xpand template

```
«IMPORT psm_word07»

«DEFINE main FOR Document»
«FILE name + "_Word07.bas"»
Function «name + "_Word07"»(outputPath As String) As String()

   Documents.Add Template:="Normal", NewTemplate:=False, DocumentType:=0

   «LET pages.collect(p|p.format) AS pageOrientation»
   «FOREACH pages AS page ITERATOR iter»

      «EXPAND FormattedAreas FOREACH page.formattedAreas»

      «IF !iter.lastIteration»
      Selection.InsertBreak Type:=wdPageBreak
      «ENDIF»

   «ENDFOREACH»
   «ENDLET»

End Function
«ENDFILE»
«ENDDEFINE»

«DEFINE FormattedAreas FOR FormattedArea»

   «FOREACH ((FormattedArea)this).elements AS element ITERATOR iter»
   «EXPAND PageElements FOR element»
```

```
29      «ENDFOREACH»
30
31      Selection.TypeParagraph
32      Selection.Style = ActiveDocument.Styles("Standard")
33
34  «ENDDEFINE»
```

Theoretically there are also other approaches available for the platform of Microsoft Word. Instead of generating VB macro code, it would be possible generate .NET source code for example in the programming language C#. In .NET there are libraries available for each version of Word, which allow the assembling of Word documents. Thus, it is possible to generate documents directly out of .NET. The advantage of this solution is that the complex process of executing the generated VB macro code is no longer required. The advantage of the macro solution in contrast is that a macro can implement every feature of a particular Word version, since every manual step in Word can be recorded by means of macros. For libraries, however, it cannot be assured that every single feature is supported. That's why this approach has not been taken into account for the reference implementation, but the macro approach has been used.
Another option would be the generation of the actual documents in this step. This would undoubtedly be the most elegant solution. But for Microsoft Word this is, depending on the version, by no means or not without further ado possible. Since Microsoft Word 2007 uses the XML based Office Open XML format, the content should be human readable and therefore producible by means of code generation. But the actual .docx documents are container packages, that contain the human readable XML part in a zipped form. For that reason, again, an additional step would be necessary to create the zipped package file. Microsoft Word 2003 documents are using a binary format that is not human readable. This fact makes it impossible to generate documents for this version by means of code generation.

4.6 Document Generation

Since the whole workflow should be runnable in an automated way, also the macro code should be executed automatically. A detailed research of available tools that are able to automatically process VB macros for Word 2003 and Word 2007, did not bring any result. That's why an own application has been developed for this purpose. Since the host applications are Microsoft products for both platforms, it was obvious to use a Microsoft technology for that issue as well. Therefore .NET has been chosen as framework for the development of a tool that is able to transform VB macros into actual Word documents. The implementation of that tool should yield an executable that can be started from within the workflow.

Different considerations have been made to find an appropriate solution. The first idea was to inject the macro code into Normal.dot for Word 2003, respectively Normal.dotm for Word 2007. These files are the templates that are used as base for each new document in Microsoft Word. Any macro that is available in these templates can be processed by executing the command line call *winword.exe /m<macroname>*. The problem with that approach was the inability to open

and edit the templates from outside Microsoft Word, and therefore it was not possible to inject the macro code. An approach that automatically imports macro files from a predefined directory into the template file did not work, since the import process in Visual Basic is not automatable. An article[6] in the knowledge base of *microsoft.com* finally led the process into the right direction. This approach is opening an instance of Microsoft Word, imports a macro file to Visual Basic at runtime, and finally runs the macro code. As described in this article, it is important to grant access to the Visual Basic project. This can be done by setting the parameter *Trust Access to the VBA project object model* in the options of Word onetime, more precisely in *Trust Center Settings*. For Word 2003 additionally the registry must be extended in order to be able to execute the logic. Therefore the DWORD key *AccessVBOM* must be added to *HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Word\Security* with the value set to 1. The last problem that had to be solved was the error *Programmatic access to Office VBA project is denied*. The reason for having that problem is the inability to access the standard template Normal.dot and Normal.dotm from outside the host application. The solution to that issue was to manually create a template that is exclusively used for the document generation tool. This template is stored in a predefined directory and used inside the source code by adding the line `Application.Documents.Add(<path and name of template>)`.
Listing 4.11 shows the relevant steps that are executed in the final version of the document generation tool. This tool has been implemented in Microsoft Visual Studio using C# as programming language. Since every version of Word uses different libraries for the implementation of the process to import and execute macro code, also different executables must be created for each version. That's why there are two executables available, which are named as *MacroRunnerWord03.exe* and *MacroRunnerWord07.exe*.

Listing 4.11: C# code to import and run VB macros

```
1  Application = new Application("Word.Application");
2  Application.Documents.Add("D:\PBDML.dotm");
3  Application.Visible = false;
4
5  var macros = Directory.GetFiles(MacroFileLocation, "*.bas").ToList<string>();
6
7  foreach (var macro in macros)
8  {
9     var macroName = Path.GetFileNameWithoutExtension(macro);
10    var addedMacro = Application.VBE.ActiveVBProject.VBComponents.Import(
          string.Format("{0}.bas", Path.Combine(MacroFileLocation, macroName)));
11    var groundTruthFromMacro = Application.Run(macroName, OutputPath);
12
13    var groundTruth = new GroundTruth()
14    {
15       NumberOfPages = Int32.Parse(groundTruthFromMacro[0]),
16       NumberOfTables = Int32.Parse(groundTruthFromMacro[1]),
17       NumberOfImages = Int32.Parse(groundTruthFromMacro[2]),
```

[6] http://support.microsoft.com/kb/219905

```
18          NumberOfWords = Int32.Parse(groundTruthFromMacro[3]),
19          NumberOfCharacters = Int32.Parse(groundTruthFromMacro[4]),
20          NumberOfCharactersWithSpaces = Int32.Parse(groundTruthFromMacro[5]),
21          NumberOfParagraphs = Int32.Parse(groundTruthFromMacro[6]),
22          NumberOfLines = Int32.Parse(groundTruthFromMacro[7]),
23          FileExtension = DocExtension
24      };
25
26      var groundTruthGenerator = new GroundTruthGenerator(SourceModelPath,
            DiversifiedModelPath, TargetModelPath, OutputPath);
27      var documentName = GetDocumentName(macroName);
28      groundTruthGenerator.CreateGroundTruth(documentName, macroName,
            groundTruth);
29
30      Application.VBE.ActiveVBProject.VBComponents.Remove(addedMacro);
31  }
32
33  object saveChanges = false;
34  Application.Quit(ref saveChanges);
```

After retrieving all macro files that need to be executed within the current run, each macro is imported into the Word application, respectively its current VB project by calling `Application.VBE.ActiveVBProject.VBComponents.Import(<macro>)`. It is crucial to remove the macro again after its execution, otherwise the VB application will not be able to execute any more macro code after a certain limit has been reached. At this point, which occurs after adding approximately 900 files, an error would be thrown and the whole run would abort. That's why the line `Application.VBE.ActiveVBProject.VBComponents.Remove(addedMacro)` has been added to the loop, in order to prevent such errors.

By executing `Application.Run(<macroname>, <outputPath>)`, the imported macro file, that has been generated in the code generation step, is processed. The first parameter of this method, which is the name of the macro file, is mandatory. Each additional parameter is optional and forwarded to the macro, which in the case of the prototype is the output directory where the generated files are stored. The macro code consists of a VB *Function* procedure that adds all elements sequentially to the Microsoft Word document, and finally saves this document in the forwarded output directory. An example of the basic structure that is necessary to create a new Word document can found in Listing 4.12. In this example an empty Word 2007 document is created and its ground truth is returned.

The syntax of the macro code for Word 2003 and Word 2007 is quite similar, but there exist some differences. Therefore it is usually not possible to run a macro for Word 2003 by using a Word 2007 application and vice versa.

Listing 4.12: Visual Basic macro that creates an empty Microsoft Word document

```
Function Invoice_1_Word07(outputPath As String) As String()

   Dim numberOfImages As Integer
   Dim numberOfTables As Integer

   Documents.Add Template:="Normal", NewTemplate:=False, DocumentType:=0

   Dim groundTruth() As String
   ReDim groundTruth(0 To 7)

   Dim numberOfPages As String
   Dim numberOfWords As String
   Dim numberOfCharacters As String
   Dim numberOfCharactersWithSpaces As String
   Dim numberOfParagraphs As String
   Dim numberOfLines As String

   ActiveDocument.Repaginate
   numberOfPages = ActiveDocument.BuiltInDocumentProperties(wdPropertyPages)
   numberOfWords = ActiveDocument.Range.ComputeStatistics(wdStatisticWords)
   numberOfCharacters = ActiveDocument.Range.ComputeStatistics(
       wdStatisticCharacters)
   numberOfParagraphs = ActiveDocument.Range.ComputeStatistics(
       wdStatisticParagraphs)
   numberOfLines = ActiveDocument.Range.ComputeStatistics(wdStatisticLines)

   groundTruth(0) = numberOfPages
   groundTruth(1) = CStr(numberOfTables)
   groundTruth(2) = CStr(numberOfImages)
   groundTruth(3) = numberOfWords
   groundTruth(4) = numberOfCharacters
   groundTruth(5) = numberOfCharactersWithSpaces
   groundTruth(6) = numberOfParagraphs
   groundTruth(7) = numberOfLines

   If (Dir(outputPath, vbDirectory) = "") Then
      MkDir (outputPath)
   End If

   ActiveDocument.SaveAs FileName:=outputPath + "\Invoice_1_Word07.docx"
   ActiveDocument.Close

   Invoice_1_Word07 = groundTruth

End Function
```

Using Different Versions of Microsoft Word Parallel on One Machine

The approach described above could fail under certain circumstances, which makes this part of the workflow very volatile. In case more than one version of Microsoft Word is installed on a machine, which in the case of the prototype is a necessary requirement, it is not guaranteed which version is opened and used to import and run a macro file. The problem is that in such a case only one Word version can be active at the same time, and this version is used as standard when running the document generation tool. If the wrong version is used for that purpose it will probably fail due to different syntax of the generated macro code.
To solve this problem, the tool offers the possibility to start *winword.exe* with the argument */r* before continuing with the logic to import and run the macros. This has the effect that the setup for the current version of Word is started and the registration of that version is performed. This process defines the particular Word as standard version, which usually results in using the correct version for the current part of the workflow.

If for any reason a Word process is active in background, which does not necessarily mean that a Word application is currently opened, the version of this active process is used for generating the documents. That means if for example an active version of Word 2007 is running in background and the workflow tries to generate documents for Word 2003, the execution of the macro code will most likely fail. The only solution to this problem is to end all running Microsoft Word processes before starting the document generation step. This could be implemented as an automatically triggered task within the Macro Runner, but it might be problematically if the user has Microsoft Word opened with an unsaved document accidentally. This would result in discarding unsaved data and in a frustrated user. To avoid such a case, the workflow could be stopped here if the Macro Runner discovers a running Microsoft Word process with an unsaved document. However, this is neither a neat solution, nor does is guarantee that for each run the correct version is used.

In summary the only reliable solution to that problem is to run the document generation process for each Microsoft Word platform on a different machine, where only the particular Microsoft Word version is installed. Therefore different remote machines have been set up for running the workflow of the reference implementation, which is specified in Section 4.8.

4.7 Ground Truth Generation

There are also several options available when it comes to the generation of files that contain the ground truth data which corresponds to the generated documents. On the one hand XSLT (Extensible Stylesheet Language Transformations) could be used to transform the models after the model transformation step into appropriate ground truth files. On the other hand the ground truth could be generated in the code generation step. In the reference implementation another solution has been picked, where the ground truth data is generated as part of the document generation step. The reason for that decision is the fact that beside information from the models also data from the host application itself should be added to the ground truth. This solution has

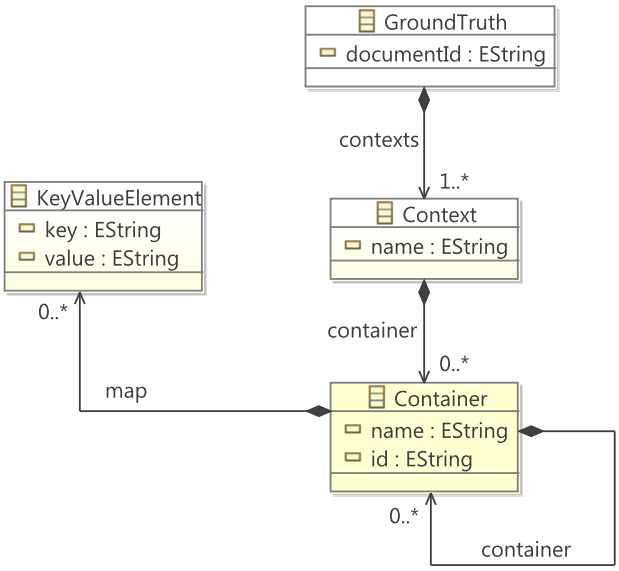


Figure 4.7: Metamodel of ground truth model

the advantage that information from the whole processing chain, starting from the source model right through to the actual document, is covered within the ground truth. This additionally gives an overview how the document evolves during the process and allows better validation and verification of the output.

To retrieve information directly from the host application, the information must be acquired during the execution of the macro code. Listing 4.12 depicts how ground truth data such as number of pages, number of words, number of characters, etc. are selected by the macro, added to a collection, and assigned as return value of the function in form of a string array.
When executing the `Application.Run(<macroname>, <outputPath>)` method in the C# code of the document generation tool (cf. Listing 4.11), this array is returned as parameter and added to a *GroundTruth* object. Along with source model, diversified PIM, and PSM, this object is used as input parameter for the *GroundTruthGenerator*. This class is assembling the final output for the ground truth file by using the LINQ to XML technology [14] to generate an XML structure, conforming to an own metamodel. This metamodel, which is shown in Figure 4.7, is necessary to allow better compatibility, comparison and exchange with other applications.

Ground truth can be defined on different levels, which in the reference implementation are PIM level, diversified PIM level, PSM level, and host application level. In the ground truth model these levels are defined by using the *Context* element, where each context conforms to a level. Moreover, each context consists of a range of container elements where each container can contain another container. Containers can either be information units like the collection of various metadata, or elements from the document such as *Page*. Container elements can have a

collection of attributes in form of key-value pairs. By using this container concept almost every structure can be modeled and also additional information can be added to the ground truth in a flexible way.

Traceability Within the Ground Truth

A requirement for the ground truth is the ability to trace model elements throughout the entire workflow. That means a particular element of the source model has to be identifiable as the same element also in the diversified PIM, as well as in the PSM. To enable this functionality, several issues had to be considered for the reference implementation.
First, a unique identifier must be assigned to each element. In EMF two options are available for that purpose, which are intrinsic identifier and extrinsic identifier. Extrinsic IDs are not directly available as attribute of an element, they are stored external to an object and are maintained by the XMLResource [59]. The problem with this option is that it is not possible to pass such an ID from an element in the source model to the corresponding element in the target model when performing a model transformation. This is why intrinsic IDs are used to assign an unique identifier. Intrinsic IDs are modeled as common attribute, with the parameter *isID* set to true. Since this is actually a common attribute, it is possible to transform it like any other value from the model.

An idea how to set the values of the identifiers was to assign it during the first model transformation step. The problem with that approach is that the identifiers can only be set for the target model, which in that case is the diversified PIM. But since identifiers must already be set for the source model, it is necessary to assign them at creation time of the source model. Therefore the Java implementation of the EMF model must be adapted, which can be handled inside Eclipse. In the constructor of the base class *BaseClassImpl* the method `setId(EcoreUtil.generateUUID())` must be added, which automatically sets an identifier for each created element. By using the *generateUUID* method from the *EcoreUtil* class, a universally unique identifier is generated for that purpose.
Usually the simplest way to design a source model is to create a dynamic instance of the PIM metamodel. This can be done by right clicking on the *Model* element in the PIM.ecore metamodel in Eclipse and choosing *Create Dynamic Instance*. This approach has the disadvantage that the model code is not executed in background and therefore the identifiers are not set automatically. That's why the reflective editor from EMF must be used instead. For that purpose EMF provides functionality to generate an editor plug-in out of a metamodel. This plug-in contains Java code that allows users to design model instances of the metamodel. To start the editor it is necessary to run the generated editor project as a new Eclipse instance. In this instance the wizard and editor are available for the model generation of the metamodel's type.
An additional adaption was necessary for that approach, since resolving the metamodel's path in the reflective editor does not work as for the solution using the dynamic instance. Neither an absolute, nor a relative declaration of the path worked well, so the solution was to use the platform resource URI that points to a specific directory within the project's workspace. Before starting the model transformations, an ant task is responsible for copying the generated source

model into that directory.

The traceability in the reference implementation is limited to elements, that means tracing attributes is excluded from that feature. The reason lies in the underlying XML structure, since XML attributes consists of a key-value pair with no possibility for an additional information like an identifier. Therefore a completely different solution would be necessary for that case, which has not been considered for the prototype.
A disadvantage of the current approach is the inability to copy elements when using the reflective editor. Technically this is not a problem at all, but when an element is copied, also its identifier is copied with the element and is therefore not unique anymore. If, for example, a table is designed with numerous rows that are similar, it would be much faster to model the first row and then copy and paste it multiple times. Currently this would result in a traceability problem, since each row and its elements would have the same identifiers.

4.8 Running the Workflow

To execute the whole workflow as a single automated task, Apache Ant has been chosen to run all building blocks successively in the reference implementation. In the current version only the modeling process of the source model is a manual step. Right after this step has been finished, the user can run a batch file which executes all Ant scripts, in order to start the workflow. After its completion the result is a new directory containing all documents and its corresponding ground truth files of the current run, respectively for the modeled source model.
In detail following Ant tasks are used in the script:

- QVT model transformations: For model transformations using QVT, there is a separate Ant task available where the transformation file, all input and output models, and potential configuration parameters are defined (cf. Listing 4.13). This task is used to execute the transformation for the PIM diversification and for each transformation from the diversified PIM to the PSM. Additionally, before starting the first transformation from PIM to a diversified PIM, the copy task from Ant is used to copy the manually generated source model into a predefined directory. This is step is necessary to keep the location where the source files are stored flexible and to guarantee that the schema location of the source model refers to the proper metamodel. For the copy task it is important to set the parameter *overwrite* to *true*. Otherwise changes in the source model without changing the its name won't be recognized by the workflow.

- Xpand code generation: Xpand provides the class *WorkflowAntTask*, which is used to run a code generation workflow from Ant. For that purpose the class must be added to the classpath of the Ant task and also the corresponding workflow file must be defined. In order to avoid conflicts with old macro files, an Ant task is used which deletes all generated macro files from the previous run before starting the current code generation workflow.

- MacroRunner document and ground truth generation: Since an executable is used to start the document and ground truth generation process, the Ant script for starting this step is simply using the Exec task including the required command line argument.

Actually all tasks could be run from a single Ant file without using a batch file, but the QVT Ant task only works if running in the same JRE as the Eclipse workspace. Therefore it is not possible to run it from outside Eclipse in the same workflow with other Ant tasks as usual. A solution to this problem is to execute the Ant script from command line, using a headless Eclipse instance that must be started with a call similar to this:

```
java -Xms128m -Xmx1024m -XX:MaxPermSize=256m
   -jar ./plugins/org.eclipse.equinox.launcher_<version>.jar
   -data <WORKSPACE_PATH> -nosplash -consolelog
   -application org.eclipse.ant.core.antRunner -f <ANTSCRIPT>
```

The easiest way to programmatically execute such a command is to use a batch file, which first calls the QVT task as depicted above and after that, calling the subsequent Ant files. This can be done either by using the command *call ant* to start the file build.xml, which in turn executes the remaining Ant files, or by calling the command *call ant -file* followed by the particular Ant file name for each of the remaining Ant files. In both cases it is important to use the *call* command, otherwise the Ant tasks are not returning to the calling batch file.
A drawback of the approach using headless Eclipse to call the QVT Ant task is its error-proneness regarding compatibility of Eclipse, Java, and operating system. To avoid any problems it is highly recommended to use the same bit version for each of it. If it is not possible to achieve this, and depending on the operating system used and bit version of the Java Virtual Machine, it might be necessary to add additional settings to the command line call above such as architecture (-arch), operating system (-os) and window system (-ws), in order to configure the Eclipse runtime[7] accordingly. The user also has to be very careful with setting the heap size (parameters Xms, Xmx and XX:MaxPermSize) to avoid errors such as *Could not reserve enough space for object heap*.

Listing 4.13: QVT Ant task

```
1  <project name="project" default="main" xmlns:qvto="http://www.eclipse.org/qvt
       /1.0.0/Operational">
2     <target name="main">
3        <qvto:transformation uri="platform:/resource/ProjectA/Transform.qvto">
4           <in uri="platform:/resource/ProjectB/model/DiversifiedPIM.xmi" />
5           <in uri="platform:/resource/ProjectB/model/Distr_Word07.xmi" />
6           <out uri="platform:/resource/ProjectB/src/model/PsmWord07.xmi" />
7           <configProperty name="numberOfDocuments" value="10" />
8           <configProperty name="defaultDocumentName" value="Test" />
9        </qvto:transformation>
10    </target>
11 </project>
```

[7] http://help.eclipse.org/juno/topic/org.eclipse.platform.doc.isv/reference/misc/runtime-options.html

Eclipse also provides an alternative, where QVT model transformations can be launched programmatically in Java[8], including the advantage that the transformation can be run standalone as a Java application outside Eclipse. But this approach has limitations too and in the specific case of the reference implementation it cannot be used without modifying classes in the library *org.eclipse.m2m.qvt.oml*. The problem is that this library, which is responsible for compiling and interpreting the operational mapping language of QVT, does not support black box implementations in the latest version (3.2.0). But as described in the Section 4.3, these black box functions are mandatory to support the calculation of various distributions within the diversification process. Possibly this functionality will be added in a future release, but currently only a patch for *org.eclipse.m2m.qvt.oml* would solve this problem. That's why the approach using the Ant script inside a batch file has been chosen for the reference implementation.

Running the Workflow on Different Machines

As described in Section 4.6, the only way to guarantee the usage of the correct Microsoft Word version when processing the macros is to use a different machine for each target platform. For the reference implementation three machines have been set up, where one is responsible for running the main workflow, while the others are used to generate the documents for the particular version of Microsoft Word. For that putpose it is important that on each machine where documents are being generated only the particular version of Microsoft Word is installed.

In Figure 4.8 the configuration of the used system architecture is illustrated. The components that are responsible for model transformation and macro code generation reside on the main machine. First the QVT transformations and the Xpand code generation are started as for the single machine approach. After these processes are finished, a client is called, which starts the document and ground truth generation on the remote machines. In the reference implementation this client is an executable, developed with .NET framework 4.0.
The communication between the main machine and the machines where the document and ground truth generation takes place, is implemented by means of Representational State Transfer (REST) [53]. To trigger the file generation process, first the models that are generated during the current run and the generated macro files are sent to the corresponding remote machine as input files. This is done by calling the POST method of the MacroRunnerService that resides on each remote machine. The input files are attached to the message send by the client through HTTP to the endpoint of the RESTful service. In the reference implementation this service is hosted in IIS 7, developed with ASP.NET Web API from MVC 4 Framework [15].

For example on machine *Word 2007* the POST method of the MacroRunnerService is awaiting the source model, the diversified models, and the PSMs, as well as the macro scripts as input files. Once the transmission is completed and all files are copied into a configurable directory on the remote machine, the service is starting the MacroRunner executable for Microsoft Word 2007 (cf. description above). Since only Microsoft Word 2007 is installed on the specific machine, it is not required to start some kind of configuration to define Microsoft Word 2007 as

[8] http://wiki.eclipse.org/QVTOML/Examples/InvokeInJava

standard version.
After Microsoft Word 2007 is used to import the macros and execute the code, and after generating the corresponding ground truth data, the generated Word documents and its ground truth files (i.e. the output of the whole workflow process) are sent by the MacroRunnerService back to the main machine. This is done by using the Distributor service, which is also hosted in IIS and developed as ASP.NET Web API application. Again files are transmitted as payload of the request, received by the Distributor service, which simply copies the sent files to a configurable output directory. Through the whole process, the current timestamp in the format yyyyMMddHHmm is used to generate a new directory for each run for the input files, as well as for the output files on each machine.

The workflow for Microsoft Word 2003 would be exactly the same, except for using a different remote machine where Microsoft Word 2003 instead of the 2007 version is exclusively installed and used to generate the documents. There are two different approaches how generated macro code can be forwarded and executed on a remote machine. The first option is to iterate through all macro files and send each file in a single service call, and therefore start the document and ground truth generation process separately. The other option is to send all generated macro files within one message and execute the document and ground truth generation process on the remote machine as single step within a loop. An advantage of the first approach is that there are no issues regarding timeouts or problems with the size of attached content when having a huge collection of input data. Another one is that files could theoretically be processed in parallel, which can improve the performance regarding duration of the runs. But in practical use it emerged that sending multiple request within a short period of time could result in losing messages. Also the performance argument has two sides, since starting the Macro Runner for each single macro code implies that the Word application has to be started for every file. This in turn has disadvantages regarding total duration compared to opening the application once and then importing and executing each macro code in a loop, which is the case if all macro files are sent within one request.

When installing and configuring the components of the workflow, there are a few issues the user has to pay attention to. Since the MacroRunnerService is calling the MacroRunner and the MacroRunner on the other hand is calling Microsoft Word, it is important under which user the service is running. Either a valid Windows user has to be configured as identity of the service's application pool in IIS, or the LocalUser can be set. Otherwise the service will not have the permission to access Microsoft Word. Additionally a Windows user that has the permission to open Microsoft Word and to access Visual Basic projects must be set as identity in the *Component Services* for the Microsoft Word application. If this configuration is missing, the programmatic access to Visual Basic project is not trusted, even if the security settings in Microsoft Word has been set correctly. Additionally, the limits for the requests over the web must be configured on server side, in order to enable the generation of benchmark data in a large scale. To deal with messages that contain many files, especially the parameters from Listing 4.14 must be accordingly set in the *web.config* of the service. According to this, there exists a technical limit for files that can be generated at once. Since the maximum value for these

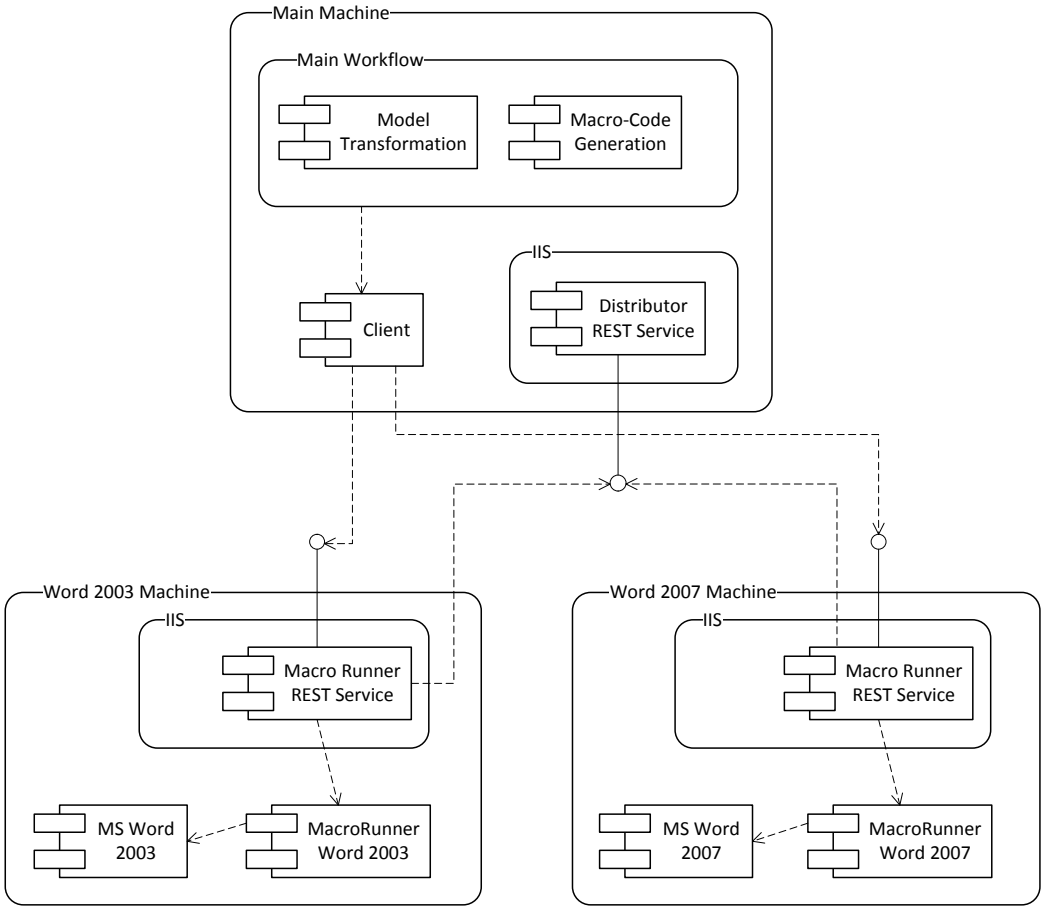


Figure 4.8: Deployment diagram

parameters are 2 GB for *maxRequestLength* respectively 4 GB for *maxAllowedContentLength*, the theoretical maximum number of generated documents is approximately at around 130.000. In case this limit is to low, the current implementation would have to be changed in order to create the data within multiple iterations. But most likely no user would start a run that creates such a high number of documents, since the workflow would run a few days.

Listing 4.14: Relevant parameters in the web.config to allow large data sets

```
1 <system.web>
2    <httpRuntime maxRequestLength="2097151" />
3 </system.web>
4 <system.webServer>
5    <security>
6       <requestFiltering>
```

```
7           <requestLimits maxAllowedContentLength="4294967295" />
8       </requestFiltering>
9     </security>
10 </system.webServer>
```

As for the non distributed approach, again a combination of batch file and Ant scripts is used to orchestrate the workflow in the prototype. The starting point is a batch file that first starts the headless Eclipse instance to do the transformation steps, just as described in Section 4.8. Also the macro code generation is again started using an Ant script analogous to a single machine. After that step another Ant script is invoked that executes the clients, which communicate with the RESTful services on the remote machines.
If the ground truth and documents are being generated for different platforms within one workflow, the performance can be improved by sending the generated macro files in parallel running threads. For this purpose the Ant script that invokes the clients on the main machine is using the *parallel*[9] Ant task for the invocations.

Configuring the Workflow

In order to create a flexible output, there are some configurations that influence the result of the workflow. Following parameter can be set:

- The platforms for which the workflow should generate documents including its corresponding ground truth
- Name and directory of the source model that should be used as input for the next run
- Number of documents that should be created
- Name that is used as prefix for documents if no document name is defined in the source model
- Date format that is used for *Current Date* elements in the PSM for Word03/Word07
- The output is mostly influenced by the input data provided by the distribution models

Additionally it is possible to configure some parameter that do not change the output, but are useful to make the workflow more flexible:

- Directory where the source model is stored
- Path of the distribution models used for the diversification process
- Path where the diversified PIMs and the target models, which are created in the model transformation step, are stored
- Path where the generated macro files are stored

[9] http://ant.apache.org/manual/Tasks/parallel.html

- URI of remote machines in the distributed approach
- Path where input and output files on remote machines are stored in the distributed approach

4.9 Results

The prototype's output of a single run are usually numerous documents for Word 2003 and Word 2007 and its corresponding ground truth files. The exact number of files depends on the setting in the Ant file of the model transformation. Additionally, it is possible to exclude single platforms from the run by removing the corresponding parts from the Ant files. Each Word document has a corresponding XML file, which has the same name as the document and contains the ground truth data. The structure of the ground truth conforms to the ground truth metamodel as described in Section 4.7 and is shown in Listing 4.15, where also the traceability of elements is vividly illustrated.
This example shows a document with the name *BasicTest* that contains only a blank page. During the modeling of the source model the unique identifier *_EDK-AOM8EeKTKpgVKZVoYg* has been assigned to the *Page* element. This ID has been forwarded to the diversified PIM, and to the PSM during the model transformation steps, and therefore has the same value for each context. Each context contains the whole model structure, which in that simple case only consists of the document's attributes and the single *page* element. Additionally each context also has a *Summary* container that can be filled with different informations such as metadata or properties of the document. In this example only the number of pages is stored there, except for the *Host Application* context, which also includes the file format. The information that is available for a particular context can differ. For the first three levels where the base of the context is a model instance, the whole model structure is available but the metadata is strongly limited to information that can be retrieved during the model transformation step. In the context of the host application, in contrast, the metadata can be much more detailed, since the information can be retrieved directly from the document, respectively the macro execution. The model structure, however, is not available at this level, since the inspected object is not a model anymore.

Listing 4.15: Content of a ground truth file

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <groundtruth>
3     <contexts name="PIM">
4        <container name="documents">
5           <map key="name" value="BasicTest" />
6           <container name="pages" id="_EDK-AOM8EeKTKpgVKZVoYg" />
7        </container>
8        <container name="summary">
9           <map key="numberOfPages" value="1" />
10       </container>
11    </contexts>
12    <contexts name="Diversified PIM">
13       <container name="documents">
```

```
14          <map key="name" value="BasicTest_2" />
15          <container name="pages" id="_EDK-AOM8EeKTKpgVKZVoYg" />
16       </container>
17       <container name="summary">
18          <map key="numberOfPages" value="1" />>
19       </container>
20    </contexts>
21    <contexts name="PSM">
22       <container name="documents">
23          <map key="name" value="BasicTest_2" />
24          <map key="language" value="wdGerman" />
25          <map key="autoCorrection" value="true" />
26          <map key="trackRevisions" value="false" />
27          <container name="pages" id="_EDK-AOM8EeKTKpgVKZVoYg" />
28       </container>
29       <container name="summary">
30          <map key="numberOfPages" value="1" />
31       </container>
32    </contexts>
33    <context name="HostApplication">
34       <container name="summary">
35          <map key="numberOfPages" value="1" />
36          <map key="fileExtension" value="docx" />
37       </container>
38    </context>
39 </groundtruth>
```

Figure 4.9 contains an overview of the artifacts that are generated throughout the whole workflow, which is demonstrated by using a very basic example. The source model consists of a single page with a formatted area that contains one line of text. This source model is multiplied five times as part of the diversification process. In the PIM to PSM model transformation step the size of the text is set and from these PSMs the macro code is generated during the code generation step. When executing the macro code, the final output is created, which are actual Word documents and the XML files that contain the corresponding ground truth.

The generated Word documents can now be used as input for characterisation tools in order to verify and validate the results of the characterisation process. The output of this process can be compared with the corresponding ground truth file, in order to detect differences and errors. The practical application of such an activity is content of the next chapter.

4.10 Summary

This chapter contains a detailed description of a reference implementation that has been developed to prove the feasibility of the concept from the previous chapter. This prototype is able to generate benchmark data and its corresponding ground truth for the domain of page based

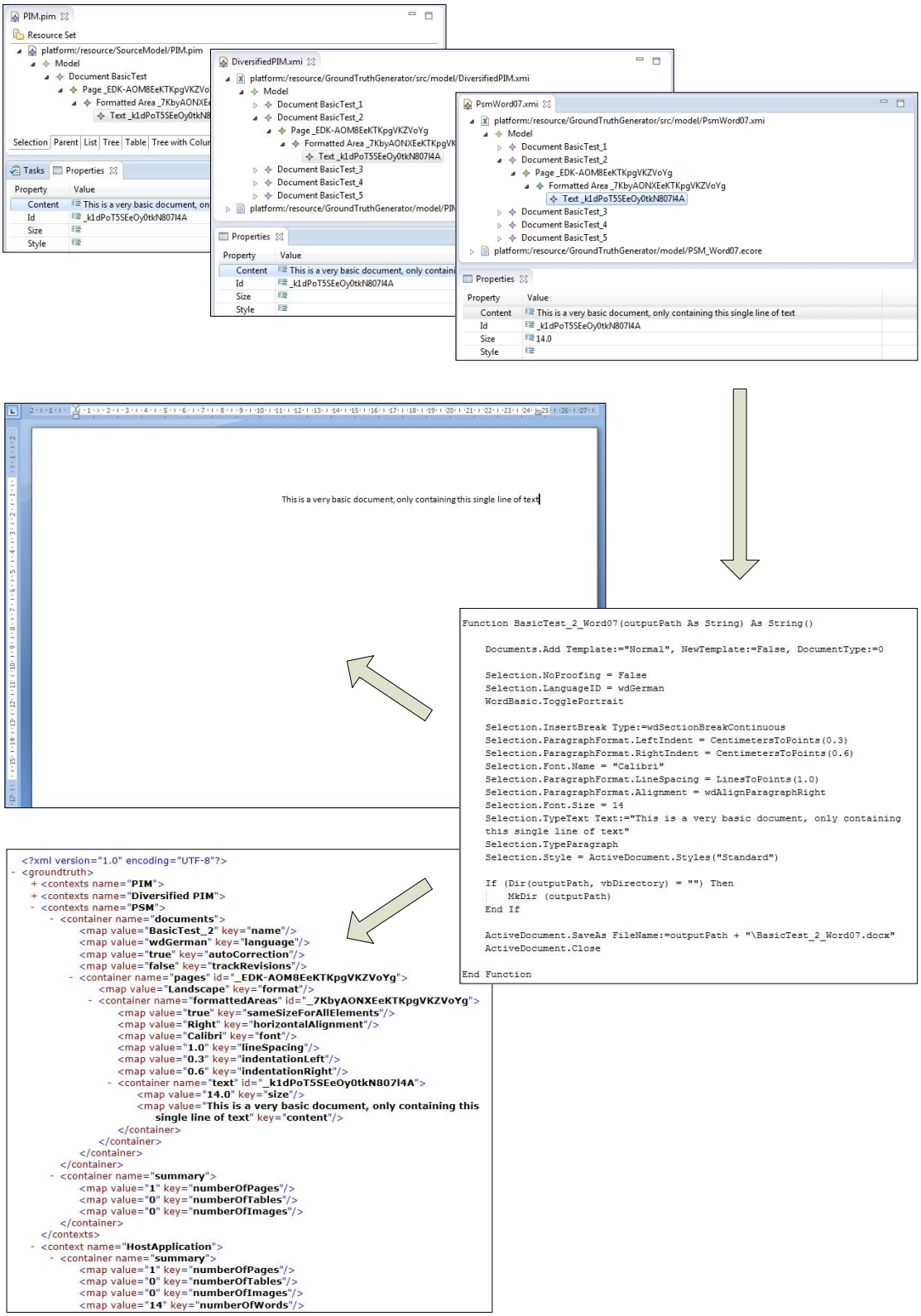


Figure 4.9: Overview of the workflow and its artifacts

documents and currently supports the platforms Word 2003 and Word 2007. Each step of the workflow has been described in detail including a brief introduction of the used technologies as well as limitations and decisions regarding actual implementation. In the next chapter the results that are generated by this prototype are used as input for an evaluation of the reference implementation.

CHAPTER 5

Experiments and Quantitative Evaluation Results

In order to prove the integrity and the results of the reference implementation described in the previous chapter, this chapter contains an exemplary run of the whole workflow, which is described step by step. This procedure is performed for the approach using a single machine as well as for the distributed approach. The results are compared and analyzed regarding performance and completeness.

5.1 Example Workflow

For the example workflow a document has been designed that covers as many features as possible. An additional goal was to define an example that is as realistic as possible regarding structure and content, and for that purpose an invoice has been selected. This invoice consists of one page containing a header and a footer. In the header there is a centered logo image, which is stored as .gif on the local hard drive. The footer contains the company name and its banking information on the left side, the current page number in the middle, and the current date on the right side. The letterhead is defined as a table with no border and two columns, where the first column includes name and address of the customer and the second column includes invoice number, customer ID, and billing date. The main part of the invoice starts with the title *Invoice*, that is defined as bold text, followed by some invoice text in the next line. After that text comes a listing with two items consisting of text. At the end of the document there is a table which contains the actual invoice. This table consists of a table header, three items that are defined by item number, description, amount and price, and in the last row the total amount of all prices. The total amount is defined as *Formula* element, which sums up the price values from the 3 rows containing the items. The text of the table header is defined as centered and bold, the item numbers and item descriptions are all aligned left, the amounts and prices are aligned right, and the total amount is defined as bold and aligned right. An example of a complete invoice document

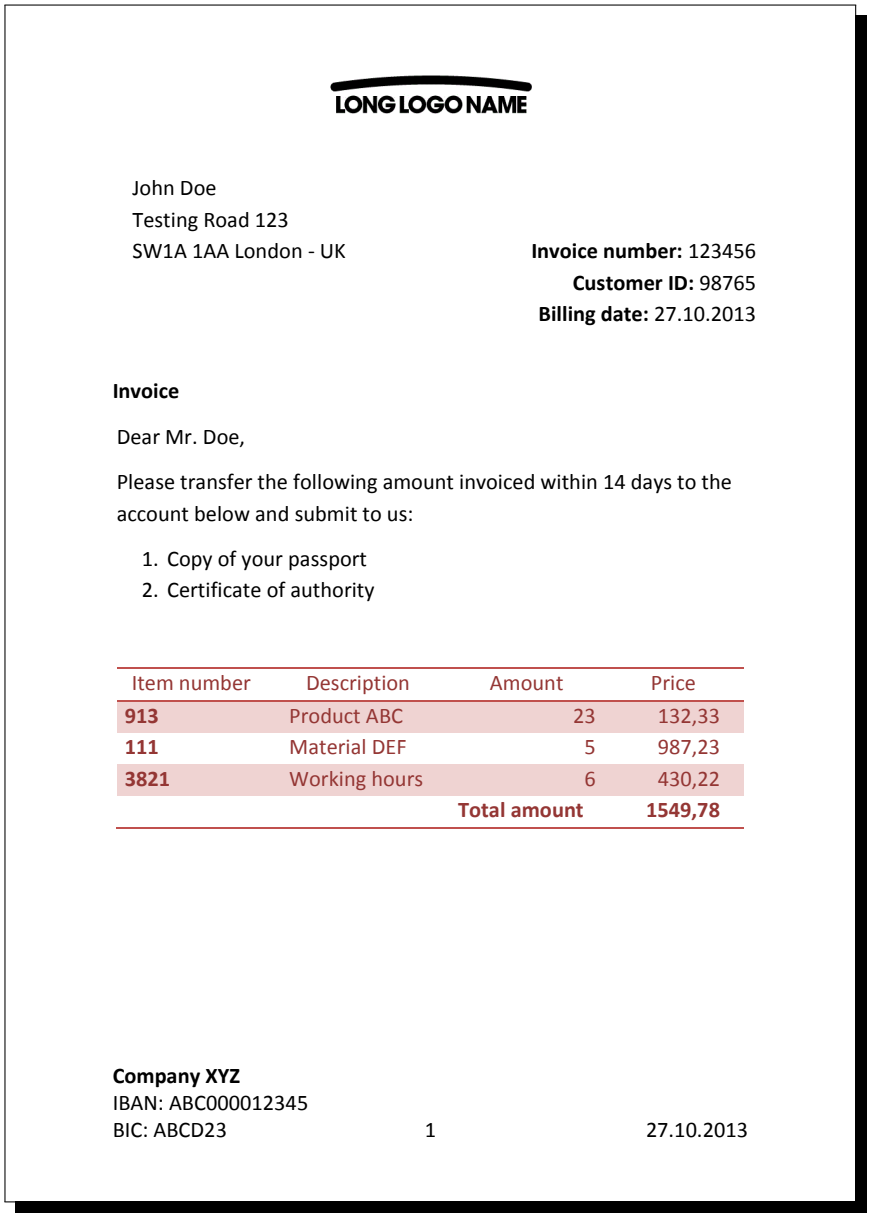


Figure 5.1: Invoice used as input for experiments

as described above can be found in Figure 5.1.

This example includes a trade-off for the invoice model between the limitations of diversified objects and realistic implementation. The problem is that there are many attributes already predefined and therefore they are not diversified during the model transformation step. The point is that the less is predefined, the more variability is available in the final output. Since the in-

voice example should be realistic, there is not so much degree of freedom for some elements and attributes. If every value would be diversifiable, the structure of some documents would not conform to a realistic invoice document.

To create the source model for the invoice, the reflexive EMF editor of the PIM metamodel is used as described in the previous chapter. The directory where this model is stored must be configured in the Ant file *modelTransformation.xml* by setting the parameter *sourceModel.dir*. Another configuration that has to be set before starting the workflow is the number of documents to be generated, which is defined in the property *numberOfDocuments* in the same Ant file. For every run documents are generated for the platforms Word 2003 and Word 2007. That means the number of documents that are actually generated can be multiplied with a factor of two. The parameter in the Ant file, respectively the specification of generated documents in the performance measures in Section 5.2, always refers to the number that is defined in the *modelTransformation.xml* Ant file and therefore refers to the number of generated documents per platform.
At this point there are two options for starting the workflow. To run it on a single machine, the batch file *run_workflow.bat* must be executed. As a first step the manually created source model is copied into a predefined working directory inside the Eclipse workspace. This model is used as input for the diversification process, together with the PIM diversification model *Distribution_PIM.xmi*. The source model is copied as many times as defined in the *numberOfDocuments* property, and for each document the relevant attributes are diversified. As output of this step, the model *DiversifiedPIM.xmi* is created, which contains a *Model* root element that consists of multiple *Document* elements, including the diversified content of the invoice.
This model is used as input for the next model transformation step, together with the statistical input from the models *Distribution_Word03.xmi* for the Word 2003 platform and *Distribution_Word07.xmi* for the Word 2007 platform. More precisely, the PIM to PSM model transformation is executed twice, one time for each platform. First the models *DiversifiedPIM.xmi* and *Distribution_Word03.xmi* are used as input to create the output model which can be found in the file *PsmWord03.xmi*. This file contains the same number of *Document* elements, each taken from the diversified model and transformed into a corresponding PSM. During that process some additional elements and attributes are diversified. The same applies to the Word 2007 platform, where *DiversifiedPIM.xmi* and *Distribution_Word07.xmi* is used as input in order to create the output model *PsmWord03.xmi*.
The next step, which is the code generation of the VB macros, is called once for each platform. For Word 2003 the model *PsmWord03.xmi* is used as input for the Xpand template *Word03.xpt*. In this template each *Document* element is used as input to create an own macro file and to save it to a predefined directory, where only the macro files for Word 2003 are stored. This directory is defined in the Xpand workflow file *Word03.mwe*, more precisely in the property *src-gen*. Again, the same process is also executed for Word 2007, where *PsmWord07.xmi* is used as input for the *Word07.xpt* template, which saves the macro files to the directory defined in *Word07.mwe*.
Finally *MacroRunnerWord03.exe* and *MacroRunnerWord07.exe* are executed with the parameter *all*, in order to create a word document and its corresponding ground truth for each macro file that is found in the predefined directories mentioned above. The output of this step is saved to the directory defined in the configuration file *MacroRunnerWord03.exe.config*, respectively

MacroRunnerWord07.exe.config in the parameter *OutputPath*. There, the result of each run can be found in a directory that is named as the timestamp of the current run.

The second option is to start the workflow in a distributed way, as described in Section 4.8. For that purpose the batch file *run_distributed_workflow.bat* must be executed. This file is performing exactly the same steps and using the same input and settings as described for the single machine approach, except for the last step. At that point the *MacroRunnerRestClient.exe* is executed, first with the Parameter *Word03* and then with the parameter *Word07*. Depending on this parameter either the generated macros for Word 2003 or for Word 2007 are send to the remote machine. Which machine is used for that purpose also depends on the platform parameter. The *MacroRunnerService* on the particular remote machine executes either *MacroRunnerWord03.exe* or *MacroRunnerWord07.exe* as for the single machine approach, and finally returns the generated output.

5.2 Quantitative Evaluation Results

Both options described above for running the workflow have been executed numerous times as part of experiments. In total 80 test runs have been performed, in order to receive meaningful data regarding performance of the output. The results and the corresponding configurations are described in detail in this section.
To get reference values for the change of performance under a changing number of generated documents, tests have been performed, using the prototype and shifting the number of generated documents for each run. Additionally, each run has been repeated several times with the same setting for the number of generated documents, in order to get an average value and therefore more realistic values for the duration of the workflow.

For the distributed approach, virtual machines have been used as remote machines. These have been set up using the virtualization software package *Oracle VM VirtualBox*[1] version 4.2.12. The environment settings used for testing the performance of the single machine approach, as well as the settings of the tests using virtual machines for the distributed approach, can be found in Table 5.1.

The experiments revealed that the tests using a single machine are quite error-prone as a consequence of the problems described in Section 4.6. In some cases Word 2003 was used to generate Word 2007 documents and vice versa, and therefore the result of these runs could not be evaluated.
Figure 5.2 contains the results of performance measures for the duration of complete runs. It shows the relation between number of generated documents and the amount of time in seconds that is necessary to create this data. Despite the overhead that results from the communication between different machines, the distributed approach is faster than the single machine approach. This is mainly because the distributed approach uses parallelism for the document generation for

[1] https://www.virtualbox.org/

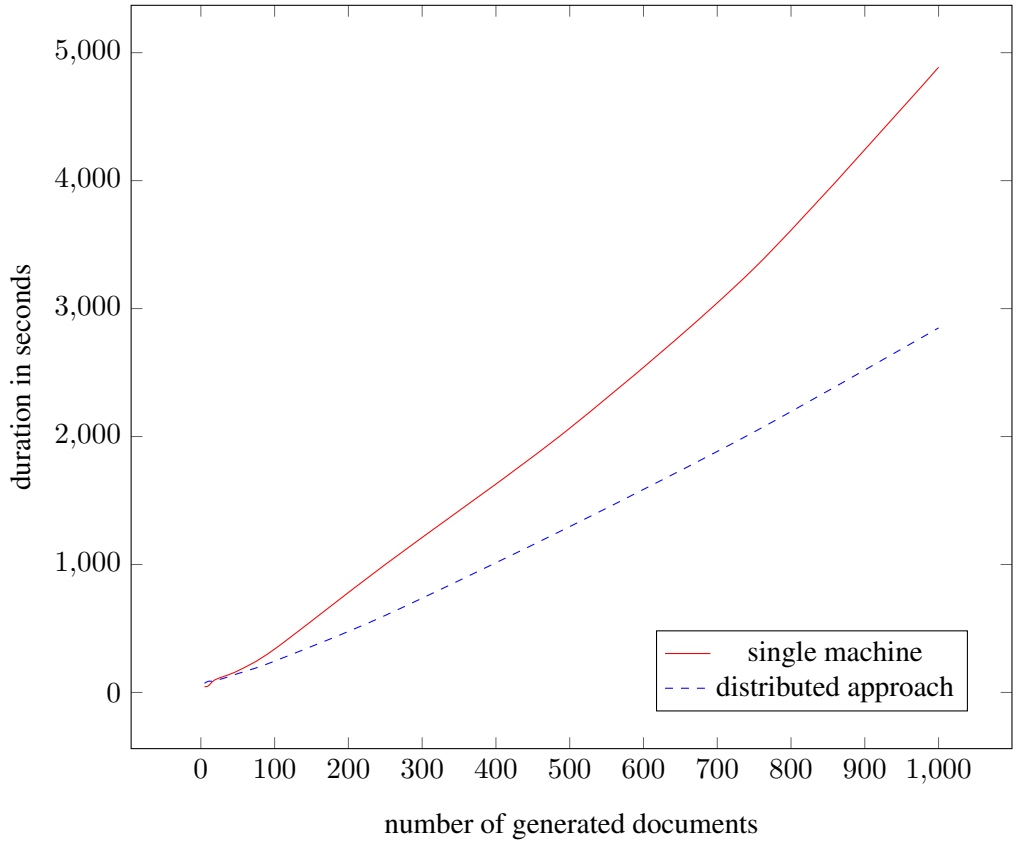
Hardware - local machine	
CPU	Intel Core i5 with 2,40 GHz
Memory	4096 MB
Software - local machine	
Operating system	Windows 7 Home Premium 64 bit SP 1
Java version	Build 1.7.0_07-b11
.NET framework version	4.0
Eclipse version	Juno Service Release 1
Hardware - virtual machines	
CPU	Intel Core i5 with 2,40 GHz
Memory (Word2003 and Word2007 machines)	512 MB
Memory (main machine)	1024 MB
Software - virtual machines	
Operating system	Windows 7 Professional 32 bit
Java version	Build 1.7.0_21-b11
.NET framework version	4.0
Eclipse version (main machine)	Juno Service Release 2

Table 5.1: Environment settings used for tests

different platforms. That means the documents for Word 2003 and for Word 2007 are generated simultaneously, where the document generation process for Word 2003 is faster than the process for Word 2007. According to the test results it seems that the difference is getting even bigger, the more documents are generated.
Generally, the runtime largely depends on the content of the documents. An element that has noticeable influence on the duration of a run is for example the *Table* element. Depending on its implementation, a table can either be set up relatively fast, which is the case for common Word tables, or it can extend the runtime, in case an embedded spreadsheet is used. Accordingly, the total duration of a workflow also depends on the statistical input. This data affects the probability and therefore the frequency of occurrence of a particular feature, and some features can be processed faster than others when it comes to the execution of the macro code.

Another important result of the experiments is the percentage distribution of each step within the workflow. As illustrated in Figure 5.3 the relation between model transformation, code generation, and document generation is changing rapidly with an increasing amount of generated documents. When creating only a few documents, the duration of the document generation step is just slightly higher than the duration of the model transformation step. As shown in Figure 5.4, the code generation step in contrast changes insignificantly with an increasing number of documents. With a growing number of created output data, the document generation step occupies the major part of the total runtime, and the first two steps of the workflow only make up a fraction of the whole process.

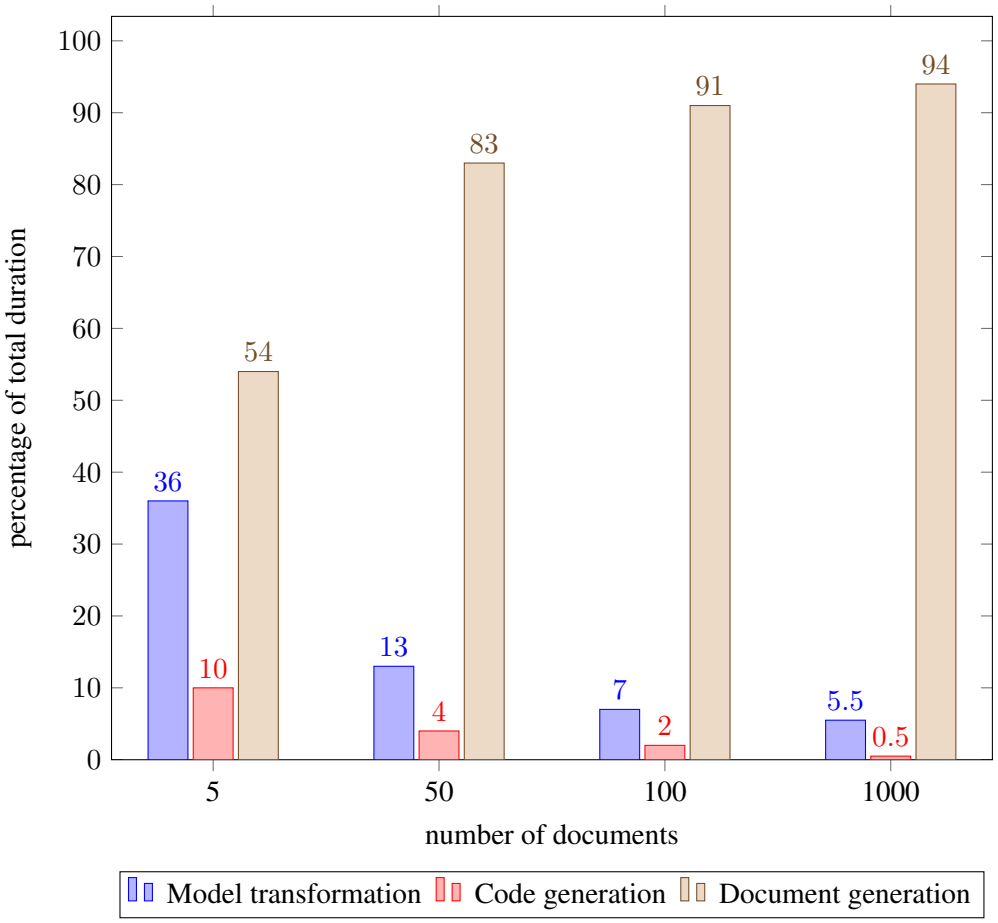
Figure 5.2: Result of performance measures



The average consumption of harddisk space is 26.8 KB for a document that has been generated for Word 2007, 30 KB for a Word 2003 document, and 46 KB for a ground truth file. The usage of memory and CPU has a very high peak at the model transformation step and a high peak at the code generation step and is relatively steady for the document generation process with around 50 percent usage for both of the test environments. Certainly the runtime of the process could be improved by using high-performance hardware when running the workflow.

All randomly reviewed documents contain the defined elements from the source model, and also the structure of the files meets the expected result. A review of the generated ground truth approves this outcome. The model structure in these files are consistent through all model steps and the structure and elements are also available in the corresponding documents in the same form. When comparing the *summary* section from the ground truth through the entire chain from PIM context to the host application context, all results are the same, except for the number of pages. The value from the host application, which corresponds to the actual number of pages in the generated files for all reviewed files, does not correspond to the values in the model contexts in every case. That issue is not specific for the invoice example, but a general problem of the page based documents in combination with modeling. In that domain no real boundaries

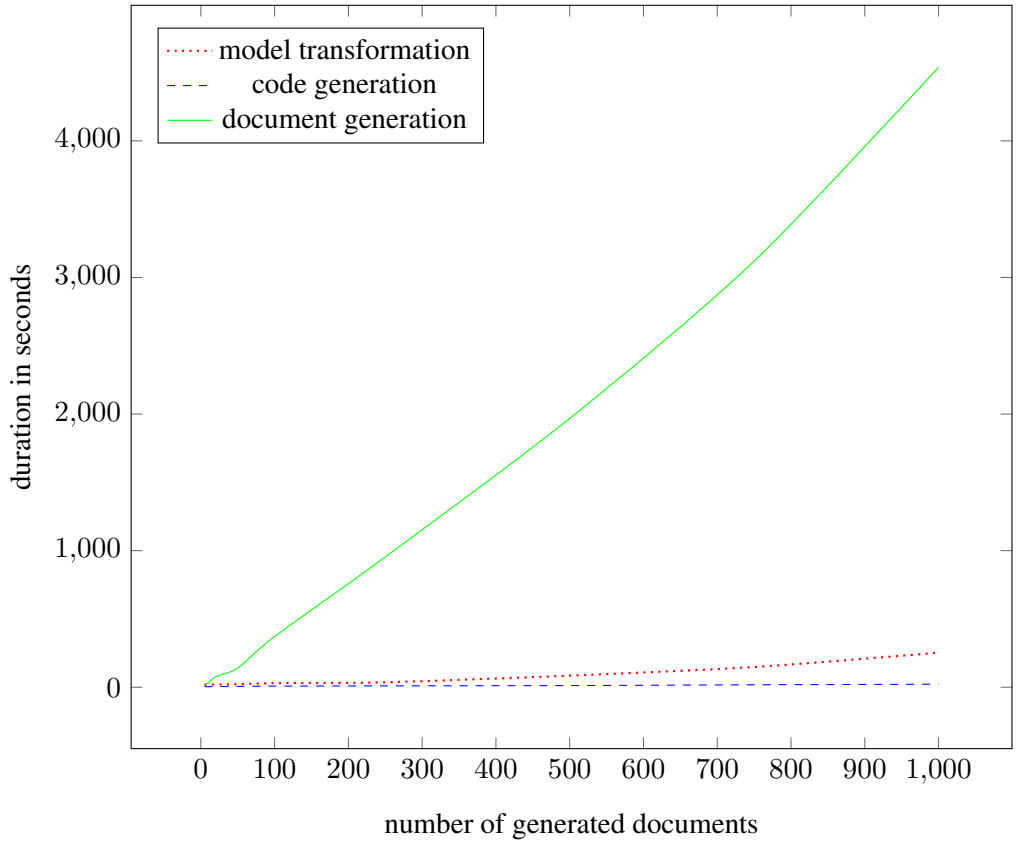
Figure 5.3: Percentage distribution of workflow steps regarding runtime



exist for elements, in contrast to domains where files have a predefined space in each direction such as videos or images. For example *Text* elements have no restriction for the content and can therefore exceed a page, even if only one page is modeled in the document. Additionally, the size of a text is usually not known beforehand, but is set in the diversification step, which can also lead to an „overflow“ of a page unexpectedly.

Reviewing the ground truth also revealed that there are some differences in the metadata values of the host application between different documents. The number of words for example depends on the implementation of the table that contains the actual invoice. For all documents where the table is available as common Word table the value is 72, and for documents where embedded spreadsheets are used instead the value is 49. When counting all words in this invoice table, where also a number is counted as single word, the delta exactly conforms to the difference of these two values. This result shows that obviously the host application does not count the words that are available in embedded objects. Therefore, also other informations such as number of characters, number of paragraphs, or number of lines are different for files with

Figure 5.4: Runtime of each step within the workflow



identical content, but different implementation of particular elements.
Another example for that behavior is the implementation of a listing element. When comparing two documents with the same content where one contains a listing that is implemented by using an unordered list, while the other one contains a listing that is implemented as ordered list, the values in the ground truth does exactly match, except for the number of characters. The delta between the values corresponds exactly to the number of list items, which points out that the number that is used as bullet point for the items is counted as character by the host application, but graphical bullet points are not, which absolutely makes sense. The word count however, is not affected by this behavior. That means characters as bullet points are not defined as words, but only as single characters.

The number of counted words is, except for the difference described above, the same for all files. Experiments showed that only text that is not located in footer or header is considered by the host application. Therefore even if more pages are generated than defined in the source model due to „overflowing“ content, the number of words and characters are usually the same, even tough the footer and header elements are available in a different number. When considering this behavior, the result of counted words corresponds to the result of manual counting, and

therefore seems to deliver the correct value.

Generally, behavior that results in differences between ground truth and the actual object is a problem, as it would invalidate the entire approach and therefore should be avoided. There are various options to deal with that problem which should be considered for a more advanced version of the implementation. One option is to remove specific fields from the ground truth that are not relevant for most characterisation tools but might differ under certain circumstances as described above, such as number of words. But for fields such as number of pages, which have a higher relevance for some characterisation tools, another solution must be found. In that particular case it might be reasonable to ensure that the content that is added to a page does not exceed its available space. This could be realized by adding limits to specific elements such as number of characters in a text, text size, number of rows in a table etc. Additionally, it would be necessary to define limits for the number of elements that could be added to a page, otherwise limits for single elements and attributes would be senseless. In any case this is an issue that requires further investigation and improvement.

Except for the file format itself, there is neither a visible difference between Word 2003 and Word 2007 documents, nor differences within the ground truth can be found. The general quality of the content regarding realistic presentation of invoices is varying, which technically is no malfunction and results from the different values that are available due to the diversification of elements and attributes.

The experiments also revealed the technical bottleneck of the reference implementation. While the model driven part of the workflow using Eclipse technologies runs flawless, the self-developed document generation tool still shows some weaknesses, since it is at an early stage in the development process. In case of errors during the macro execution, the whole run is aborted and in some individual cases inexplicable errors occurred. However, most errors occurred during the execution of the single machine approach as a result of platform conflicts. Therefore the usage of different machines already makes the implementation quite stable, but further development could improve the current implementation to provide an even more reliable solution.

5.3 Summary

The reference implementation from the previous chapter has been used to execute numerous test runs for a representative example. In this chapter a report about each step of a particular run can be found. Additionally, the environment that has been used for the experiments, as well as the actual execution of the tests, have been described in detail. Finally we added the results of the analysis of the output and of the performance measures.
In the last chapter the results from the development of the concept, from the implementation of the prototype, and from the analysis of the evaluation are used as input for a summarizing conclusion of the topic and as base for an outlook for further work.

CHAPTER 6

Summary and Outlook

The last chapter contains an outlook of extensions that could be implemented in future work in order to improve the reference implementation. Additionally, the applications and limitations of the current prototype are described. Also possible approaches for implementing the workflow for other object types than page based documents are introduced, and finally the thesis is concluded by providing summary and outlook.

6.1 Applications and Limitations of the Current Approach

Generally, the intended application of the approach described in this thesis is the validation and verification of characterisation tools. Moreover, the application of the conceptual approach are digital objects of any domain. That means that the concept should theoretically be applicable for files containing websites, images, audio, video, etc. The only constraint is that it must be possible to design the objects in a model driven way. For some object types this can be realized easily while for other objects this is a quite complex task.
The concrete application of the particular reference implementation are tools that are able to identify and analyze page based documents. More precisely only tools with support for Microsoft Word documents for the versions Word 2003 and Word 2007 derive benefit from the current development stage. For any tool that implements processes with Word documents as input and information about the content of these documents as output, the current approach can make a contribution by improving quality assurance measures.
Either for the development of new applications, the implementation of new modules for existing tools, or for the extension of existing solutions, having a wide range of realistic benchmark data with known ground truth facilitates the development and provides more reliability. Especially the ability to create new benchmark data on demand that exactly meet specific requirements, based on configurable statistical input, is extremely helpful for the development process.
Since the ground truth contains both, the complete structure of a document as well as informations on metadata level, it can be used either for tools that only provide low level information, but

also for tools that provide more detailed analysis of the content. Beside performing rapid manual tests it should also be possible to create scripts for automated testruns, since the generated ground truth relies on a predefined structure that conforms to a metamodel.

Limitations

Since the main purpose of the prototype is to verify if the developed concept is applicable for actual deployment, its features are strongly limited. There are many features that could improve the current implementation and make the output even more meaningful, respectively increase the quality of benchmark data.Some of these limitations are described in the following section.

A major limitation is included in the metamodels themselves. There are just a handful of features available compared to the ones that actually exist for page based documents. For specific platforms, in the particular case Microsoft Word, this gap is even bigger. Moreover, the available features are in many cases modeled in a simplified way, and do not allow the whole range of functions. Also the distributions that are supported by the prototype are strongly limited and relatively inflexible regarding implementation of realistic data.

For quality assurance issues it is recommended to use not only valid test data, but to add invalid data in order to test the behavior of a software on error. In the specific case of characterisation tools this includes processing a corrupted file, or a file with faulty content. Due to the actual implementation of the workflow, it is not possible to create such files. The reason is that using macro code that contains faulty code, or at least code that creates faulty files, would already fail and abort when executing the macro. The generation process would not run to the end where the files are saved and effectively created, and therefore generating faulty documents is not available in the current implementation. A future release could be improved by finding a way to create corrupted files, or at least enable the injection of invalid or faulty objects.

Another limitation is the inability to model events within the workflow. Such an approach could be based on a specific event model where actions are defined that are triggered while manufacturing a document. The goal would be to simulate user interaction like it happens when real documents are created. An example is the change tracking feature provided in Word documents, which could be dynamically turned on and off, and already generated parts of the document could be changed again while having change tracking turned on. Such a behavior cannot be implemented in the current prototype.

6.2 Possible Extensions of the Reference Implementation

Beside the described limitations that are not available in the reference implementation for the sake of simplicity, there are also various extensions and improvements that would add valuable functionality to the current approach. These functions, that are mainly used for making the implementation even more flexible and powerful, are listed and described in this section.

An improvement of the prototype would be the possibility to let the user decide whether to generate the source model manually or in an automated or semi-automated way. This optional solution could use an upstream generation process, that designs the source model automatically by using statistical data. Therefore an approach similar to the diversification step could be used, however considerably more complex.
The first option for this improvement is a completely automated generation process of the source model. That means, beside distribution of values and attributes, also decisions regarding appearance of the element itself such as positioning and quantity is taken from statistical data that is collected from real world collections. In a very first step this statistical data could be used as input for a model transformation tool or something similar, that assembles the source model according to the collected distributions.
The second option is an approach that generates the source model also automatically, but using input from the user as base for decisions regarding number and position of particular elements. That means the user selects the elements that should be contained in the source model, but the position and actual number of selected elements are added automatically, again by using various distributions.

It's not just the definition of the source model that can be improved, also its format could be integrated in a more flexible way. By drawing on the advantages of a model driven approach, different model transformation files could be provided to support various input formats such as TEI. As a result, the source model could be designed by using any information model, as long as there is a corresponding model transformation available for it, which transforms the source model into a PIM as it is used in the current approach.
It is also imaginable that, independent of the chosen approach that is used to generate the source model, multiple models are generated and used as input for the workflow. Currently only one source model can be used for that purpose, but if multiple source models would be an option, the size of the output of one run could be increased by the number of additional source models multiplied with the possible output of one source model.

When adding content to text blocks or other elements that contain text within the source model, it is currently necessary to define the actual text. In an alternative approach the user could decide if inserted text should be used or if text should be inserted automatically by the workflow. In the latter case, again statistical input could be used to calculate the amount of letters, words, lines, whitespaces, specific characters etc. and insert it as content. The semantic of the inserted text has no relevance when using the generated files as input for quality assurance activities, and therefore any dummy text could be inserted, as long as it meets the given distributions.

In the prototype the number of generated documents conforms to a configuration parameter, where the user can manually define the number of files generated in the diversification step. This number is then multiplied with the number of platforms. Another option would be to calculate the number of generated files, so that all a possible combinations of the included elements are covered. Specifically that refers to such elements that can have different implementations. For

example page format could be either landscape or portrait, a vertical alignment can be top, centered or bottom, and a table could be a common table, an embedded spreadsheet or a tab-spaced text.
When trying to find all possible permutations of these elements, a decision tree could help to find the solution to the problem. This process is non-trivial, and depending on the used technology, it can end up in a very complex algorithm. Depending on the number of elements included in a document, and depending on the possible representations the elements can have, this can result in a huge collection of output data. A time- and resource-consuming process is offset by the possibility to create benchmark data with a complete test coverage regarding included features. In addition the completeness can only be reached partly, since attributes with continuous values such as table width or text size are excluded from that algorithm.

As described in the previous section, the implemented distributions are strongly limited and therefore there is a lot of space for improvements in this area. First, the definition of the statistical input, which is currently a manual process, could be replaced with an approach that automatically collects real world data and generates the distribution input files out of it. Therefore some kind of import process would be necessary that is able to transform statistical data from a database or an input file into the distribution model that is used within the workflow.
Since only very basic distributions are available in the current prototype, which can not reproduce the distribution of real world data, a major requirement for a more realistic workflow would be to add more advanced distributions. Therefore the metamodel of the distribution model as well as the blackbox functions for QVT would have to be extended by providing more complex algorithms with more input parameter. Also further statistical methods such as stratification could be used to improve the current situation.

Beside revision of the current metamodels and extension of the distributions, adding additional platforms would be the most important improvement for the workflow. Details about possible approaches for some relevant platforms can be found in the next section.

6.3 Approaches for Different Object Types

For the domain of page based documents there are numerous word processors available, where some are open source solutions and some are proprietary. By adding Microsoft Word the most widely used word processor is already available in the prototype. It is additionally valuable that both, the old *.doc* format as well as the new Open XML format, are available for Word in the reference implementation. But also tools such as OpenOffice Writer are widely used and therefore also this platform should be added in a future release. At a first glance the solution for OpenOffice Writer could be quite similar to the one used for the creation of Word documents since OpenOffice also supports the usage of macro code. OpenOffice is using XBA macro files, which is basically an XML file that contains macro code written in the programming language *OpenOffice BASIC*[1], which is quite similar to the VB code used in Word macros. For this type

[1] https://wiki.openoffice.org/wiki/Documentation/BASIC_Guide

of macro code however, the code generation tool that has been implemented for Word macros cannot be used, since it is a specific tool that only works for this particular document type. Instead, a completely different approach must be developed for the import and execution process. Pitonyak provides detailed information about OpenOffice macros, including potential starting points for solving that issue [50].
Also other word processing applications such as WordPerfect, LibreOffice Writer, or Calligra Words support some kind of macro and could be possibly added as platforms by implementing a concept that relies on a macro-based solution. For other tools, such as AbiWord or Google Docs, there is no such functionality available and therefore the generation of actual documents must rely on a completely different approach. In some cases it might even be possible that documents of a particular platform cannot be created at all when using this approach. For the most common platforms however, it should be possible to include them to the workflow somehow.

The basic concept of the model driven approach described in this thesis should also be applicable for other domains. For some object types the integration into the workflow should work smoothly for others it might be highly complex, inconvenient, or even impossible. The domain of pageless documents such as websites is a good example for a domain that fits perfectly for this concept. The actual documents could be generated directly within the code generation step and the definition of a source model would be quite similar to the one used for page based documents. The implementation of the workflow for the domain of graphics is already more complicated. The definition of a source model could eventually be based on the concept of vector graphics. In the model transformation steps various properties of the graphic could be diversified and the transformation into different platforms such as Windows Bitmap, Graphics Interchange Format or JPEG could take place. The application of the code generation step, however, is already highly complex, and also the process where the actual images are generated could be a complex process. For object types such as music or video files, the definition of a source mode as well as the creation of the actual output are completely open regarding practicability.

6.4 Summary and Outlook

This thesis introduces the concept of a model driven approach for generating benchmark data and its corresponding ground truth.
This concept provides a promising solution to fill the gap between quality assurance and digital preservation. By creating files that can be designed in an intuitive and tool-independent way, the user has the possibility to generate a wide range of test data on demand, which can be used as input for black box testing. Moreover, by integrating statistical input data into the workflow, the generated output can be as close as possible to real world collections and therefore enables comprehensive test coverage for many different applications.

In the first chapter the idea behind this concept was introduced and the intended strategy to prove and demonstrate its application was briefly described.
The second chapter contains relevant information about the topics that are involved in the approach. Moreover, the state of the art regarding usage of benchmark data in digital preservation

has been addressed. The outcome, that there is currently no solid base available that can be used for quality assurance actions in digital preservation, led to the conclusion that such an approach would provide a valuable contribution to digital preservation.
Therefore, in Chapter 3 the concept of this approach, which has been introduced in the first chapter, was described in detail.
The reference implementation, that has been developed in order to provide a proof of concept, was discussed in Chapter 4. It contains detailed information about all decisions that affected the development process and the selection of the chosen technologies.
To figure out if the reference implementation operates as expected, various testruns have been executed, which are documented in Chapter 5, including detailed information about the results of the experiments.
In the current chapter the applications and limitations, as well as possible improvements have been described, and preliminary considerations about the usage of the concept for other object types and different domains have been made.

The evaluation of the prototype identified some issues that might leave the impression that the generated ground truth is not yet perfectly fine-tuned. But for some properties that are available in the ground truth of the reference implementation, such as number of words or number of characters, the relative importance is rather low. Not all properties are relevant for testing characterisation tools and therefore the focus should be set on important significant properties such as page count or number of images for the domain of page based documents. For most of these focused benchmarks the current implementation already provides reliable data.

A primary challenge for further development will be the task of increasing the coverage of the workflow by enhancing the metamodels with a more comprehensive feature set. Also some technical details should be refined in order to make the implementation more stable. Especially the document generation step can be improved in several respects.
After these improvements are finished, the next task will be the integration of additional platforms and domains to provide a broader range of supported object types. This would raise the contribution to digital preservation and make the implementation more attractive for potential users. At this point further analysis must be started to find out which domains and object types are really applicable for this approach, or if the domain of page based documents is a special field which can exclusively utilize the benefits of this model driven approach in its entire scope.

Generally, the conclusion of the reference implementation's development and the execution of the experiments is the insight that for the domain of page based documents such an approach can provide meaningful data in an easy to use way. Since the reference implementation is yet a prototype with limited features it is natural that for such a complex domain there are still some improvements necessary until a complete implementation is available. Nevertheless, the underlying concept provides a very useful approach for the generation of benchmark data for certain applications, even though not covering every special case, it is still applicable in a focused area. Moreover, the current prototype is a good starting point for further development and does already produce meaningful output for a specific domain.

Bibliography

[1] Stephen Abrams, Sheila Morrissey, and Tom Cramer. What? So What?: The Next-Generation JHOVE2 Architecture for Format-Aware Characterization. *The International Journal of Digital Curation*, 4(3):123–136, 2009.

[2] Christoph Becker and Andreas Rauber. Decision criteria in digital preservation: What to measure and how. *Journal of the American Society for Information Science and Technology*, 62(6):1009–1028, June 2011.

[3] Uwe M Borghoff, Peter Rödig, and Jan Scheffczyk. *Long-Term Preservation of Digital Documents, Principles and Practices*. Springer-Verlag Berlin Heidelberg, online-ausg. edition, 2006.

[4] M. Brambilla, J. Cabot, and M. Wimmer. *Model-driven Software Engineering in Practice*. Synthesis digital library of engineering and computer science. Morgan & Claypool, 2012.

[5] Adrian Brown. Selecting file formats for long-term preservation. Technical report, August 2008.

[6] TEI Consortium. TEI P5: Guidelines for electronic text encoding and interchange. `http://www.tei-c.org/Guidelines/P5/`. Accessed: 2013-09-29.

[7] Jim D'Anjou, Scott Fairbrother, Dan Kehn, John Kellerman, and Pat McCarthy. *Java(TM) Developer's Guide to Eclipse, The (2nd Edition)*. Addison-Wesley Professional, 2004.

[8] Nicholas del Pozo, Douglas Elford, and David Pearson. Prometheus: managing the ingest of media carriers. *National Library of Australia Staff Papers*, 2009.

[9] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *ACM SIGPLAN NOTICES*, 35:26–36, 2000.

[10] Martin Doerr and Yannis Tzitzikas. Information carriers and identification of information objects: An ontological approach. *The Computing Research Repository*, abs/1201.0385, 2012.

[11] Radomil Dvorak. Model transformation with operational qvt. `http://help.eclipse.org/juno/topic/org.eclipse.m2m.qvt.oml.doc/references/M2M-QVTO.pdf`. Accessed: 2013-11-24.

[12] Martin Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 2010.

[13] D.S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. OMG. Wiley, 2003.

[14] Adam Freeman. Linq to xml. In *Introducing Visual C# 2010*, pages 925–963. Springer, 2010.

[15] Adam Freeman. *Pro ASP.NET MVC 4*. Apress, Berkely, CA, USA, 4th edition, 2013.

[16] D. Galin. *Software Quality Assurance: From Theory to Implementation*. Alternative Etext Formats. Pearson Education Limited, 2004.

[17] John Gantz and David Reinsel. The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east. Technical report, IDC, December 2012.

[18] Stewart Granger. Emulation as a Digital Preservation Strategy. *D-Lib Magazine*, 6(10).

[19] R.C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Eclipse Series. Pearson Education, 2009.

[20] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993.

[21] JoAnn T. Hackos. What is an information model & why do you need one? Technical report, The Gilbane Report, 2002.

[22] Douglas R. Harvey. *Preserving digital materials*. Saur, München, 2005.

[23] Margaret Hedstrom. Digital Preservation: A Time Bomb for Digital Libraries. *Language Resources and Evaluation*, 31(3):189–202, May 1997.

[24] Margaret Hedstrom. It's About Time: Research Challenges in Digital Archiving and Long-Term Preservation. Technical report, NSF and Library of Congress, Arlington, VA, 2003.

[25] Tony Hendley. *Comparison of Methods & Costs of Digital Preservation*. eLib Studies. Library Information Technology Centre, South Bank University, 5 1998.

[26] Jane Hunter and Sharmin Choudhury. Implementing preservation strategies for complex multimedia objects. In *Seventh European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2003*, pages 473–486. Springer, 2003.

[27] Matthew Hutchins. Testing software tools of potential interest for digital preservation activities at the national library of australia. *National Library of Australia Staff Papers*, 2012.

[28] International Organization for Standardization. Information technology – open document format for office applications (opendocument) v1.0. ISO/IEC 26300:2006, November 2006.

[29] Johan van der Knijff and Carl Wilson. Evaluation of charaterisation tools part 1: Identification. Technical report, SCAPE Project, September 2011.

[30] M.J. Jones, N. Beagrie, Archives Resource: The Council for Museums, Libraries, and British Library. *Preservation management of digital materials: a handbook*. The British Library for Resource, the Council for Museums, Archives and Libraries, 2001.

[31] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. Atl: a qvt-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720. ACM, 2006.

[32] Frédéric Jouault and Ivan Kurtev. Transforming models with atl. In *Satellite Events at the MoDELS 2005 Conference*, pages 128–138. Springer, 2006.

[33] Benjamin Klatt. Xpand: A closer look at the model2text transformation language. *Language*, 10(16):2008, 2007.

[34] Panos E. Kourouthanassis and George M. Giagli. *Pervasive Information Systems*. M.E.Sharpe, 2008.

[35] Frank Alexander Kraemer and Peter Herrmann, editors. *System Analysis and Modeling: About Models - 6th International Workshop, SAM 2010, Oslo, Norway, October 4-5, 2010, Revised Selected Papers*, volume 6598 of *Lecture Notes in Computer Science*. Springer, 2011.

[36] Ivan Kurtev. State of the art of qvt: A model transformation language standard. In *Applications of Graph Transformations with Industrial Relevance*, pages 377–393. Springer, 2008.

[37] Gregory W. Lawrence, William R. Kehoe, Oya Y. Rieger, William H. Walters, and Anne R. Kenney. Risk Management of Digital Information: A File Format Investigation. Technical report, June 2000.

[38] Kyong-Ho Lee, Oliver Slattery, Richang Lu, Xiao Tang, and Victor Mccrary. The State of the Art and Practice in Digital Preservation. *Journal of Research of the National Institute of Standards and Technology*, 107(1):93–106, January 2002.

[39] Y. Tina Lee. Information modeling: From design to implementation. In *Proceedings of the Second World Manufacturing Congress*, pages 315–321, 1999.

[40] N.Y. McGovern and K. Skinner. *Aligning National Approaches to Digital Preservation*. Educopia Institute Publications. Educopia Institute, 2012.

[41] Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest editors' introduction: Model-driven development. *IEEE Software*, pages 14–18, 2003.

[42] Stephen J. Mellor, Scott Kendall, Axel Uhl, and Dirk Weise. Mda distilled, 2004.

[43] Robert Neumayer, Hannes Kulovits, Andreas Rauber, Manfred Thaller, Eleonora Nicchiarelli, Michael Day, Hans Hofmann, and Seamus Ross. On the need for benchmark corpora in digital preservation. In *Proceedings of the 2nd DELOS Conference on Digital Libraries*, 2007.

[44] Heike Neuroth, Achim Oß wald, Regine Scheffel, Stefan Strathmann, and Mathias Jehn. *nestor-Handbuch: Eine kleine Enzyklopädie der digitalen Langzeitarchivierung*. 2009.

[45] Siegfried Nolte. *QVT - Operational Mappings*. Springer, 2010.

[46] OCLC/RLG. Preservation metadata for digital objects: A review of the state of the art, a white paper by the oclc/rlg working group on preservation metadata technical report. Technical report, OCLC/RLG, 2001.

[47] National Library of Australia and Unesco. Information Society Division. *Guidelines for the Preservation of Digital Heritage*. National Library of Australia, 2003.

[48] OMG. Model driven architecture (mda). Technical report, Document number ormsc/2001-07-01, 2001.

[49] Woody Pidcock and Michael Uschold. What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta-model? `http://infogrid.org/trac/wiki/Reference/PidcockArticle`. Accessed: 2013-09-28.

[50] A. Pitonyak, C.R. Pearsall, and A. Boyer. *OpenOffice.org Macros Explained*. Hentzenwerke Series. Hentzenwerke Publishing, 2004.

[51] Johanna Puhl. Final version of file characteristics ontology. `http://planetarium.hki.uni-koeln.de/planets_cms/sites/default/files/PC2-D14.pdf`. Accessed: 2013-09-29.

[52] A. L. Rector, R. Qamar, and T. Marley. Binding ontologies and coding systems to electronic health records and messages. *Applied Ontology*, 4(1):51–69, January 2009.

[53] Leonard Richardson and Sam Ruby. *Restful web services*. O'Reilly, first edition, 2007.

[54] David S.H. Rosenthal. Format obsolescence: assessing the threat and the defenses. *Library Hi Tech*, 28(2):195–210, 1 January 2010.

[55] Jeff Rothenberg. Ensuring the longevity of digital information. `http://http://www.clir.org/pubs/archives/ensuring.pdf`. Accessed: 2013-09-12.

[56] Jeff Rothenberg. *Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation*. Council on Library & Information Resources, 1999.

[57] Claude Elwood Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423,623–656, 1948.

[58] Chris Spatz. *Basic Statistics: Tales of Distributions: Tales of Distributions*. Wadsworth, 2010.

[59] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2nd edition, 2009.

[60] H.C. Taneja. *Statistical Methods for Engineering and Sciences*. I.K. International Publishing House Pvt. Limited, 2009.

[61] Manfred Thaller, editor. *The eXtensible Characterisation Languages - XCL*. Number 3 in Kölner Beiträge zu einer geisteswissenschaftlichen Fachinformatik. Verlag Dr. Kovač, Hamburg, 2009.

[62] The Institute of Electrical and Eletronics Engineers. IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, December 1990.

[63] Malcolm Todd. Technology watch report: File formats for preservation. Technical report, DPC, October 2009.

[64] Mukesh Trehan VK Ohri TR Jain, SC Aggarwal. *Business Statistics*. Vk Publications, 2008.

[65] National Information Standards Organization (U.S.). *Information Standards Quarterly: Digital Preservation. Special issue*. National Information Standards Organization, 2010.

[66] Donald Waters and John Garrett. Preserving Digital Information: Report of the Task Force on Archiving of Digital Information. Technical report, 1996.

[67] Andrew Waugh, Ross Wilkinson, Brendan Hills, and Jon Dell'oro. Preserving digital information forever. In *Proceedings of the fifth ACM Conference on Digital Libraries*, DL '00, pages 175–184, New York, NY, USA, 2000. ACM.