MASTERARBEIT / MASTER’S THESIS

"Domain-specific Language for Reports Creation"

verfasst von / submitted by
Mark Biktaev

angestrebter akademischer Grad / in partial fulfillment of the requirements for the degree of
Magister

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:
A066 927

Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:
Individuelles Masterstudium UG2002
DDP Master of Int. Business Informatics

Betreut von / Supervisor:
Univ.-Prof. Dr. Uwe Zdun
Acknowledgement

I would like to express my gratitude to Professor Uwe Zdun for his excellent, continuous and patient supervision through the creation process of this master thesis. His quick and detailed feedbacks as well as his advices and corrections helped me throughout researching and writing this thesis.

Furthermore, I would like to thank to my colleagues Karin Witte and Maria Raskopina for proofreading.

I especially would like to thank my loved ones, who have supported me throughout the entire process, mainly my wife Victoria who believed in me until the last chapter. Thanks to you it all became possible.
Abstract

Reporting is an important part of many business activities which covers all divisions of a company. It allows evaluating, monitoring, predicting and planning the activities of an enterprise. The business report is an essential source of information for analysis and decision making. Typically reports include different tables, diagrams, charts, pictures and descriptive text. Initial data for reports are taken from data sources such as database, specific data file, web services, etc. The main task of a reporting tool is to convert initial data from data sources to obvious, understandable and neatly presented information in the required form for a responsible person. For such data conversion, a report model is created in reporting tools. There are a lot of reporting tools that assist to create business reports. Each of them offers the complex solution for business reporting. Each of these products includes its own meta-model and visual editor for report creation. These tools have similar functionality for report creation, but none of report models can be used in another reporting tool. Report models that have been created in one reporting tool can be suitable for this particular reporting tool. After choosing a reporting tool, a report developer is bound to this tool and its functional and technological features. A developer cannot use the report model in another tool. This master thesis studies business report creation methods in reporting tools based on a design science research paradigm, defines restrictions and problems in their functionality and proposes a new and alternative method for the report creation. An implementation of this method as domain-specific language for report creation has been conducted. This language can be used as an alternative method for report creation with further report generation in existing reporting tools. The language extends the capabilities for creating a report model in reporting tools and helps to improve the process of creating a model, as well as the communication between participants of this process. The functionality of the language leads to an increase in re-use of parts of the report elements. A report model, written in our Report-DSL, can be compatible with many reporting tools. This compatibility decreases the dependence on a software tool for a creator of a report model. The main disadvantages of our Report-DSL are that the specification of a report model and generator require additional effort and support and embedding in or support of other languages (SQL, JavaScript, etc.) in the model specification can be complex. The Report-DSL presented in this master thesis is especially appropriate for either simple report models or report model drafts.
Zusammenfassung

## Table of Contents

1. Introduction ......................................................................................................................11
   1.1. Motivation ..................................................................................................................12
   1.2. Objective ....................................................................................................................13
   1.3. Structure of the thesis ...............................................................................................13
2. Related works ....................................................................................................................14
   2.1. Model-driving engineering and domain-specific languages ....................................14
   2.2. Alternative report creation approaches ....................................................................15
   2.3. Report creation through a domain-specific language ................................................16
3. Research approach ............................................................................................................18
4. Domain of reports creation ...............................................................................................20
   4.1. Overview of reporting tools .......................................................................................21
      4.1.1. Eclipse – BIRT ....................................................................................................21
      4.1.2. Jaspersoft – IReport Designer ........................................................................22
      4.1.3. Pentaho – Report Designer .............................................................................23
      4.1.4. SAP – CrystalReports 2013 SP4 .....................................................................23
      4.1.5. Target Software – Target Decision Suite .........................................................24
      4.1.6. Text Control – TX Text Control .NET ..............................................................24
      4.1.7. GrapeCity – Active Reports Designer ..............................................................25
      4.1.9. Stimulsoft – Reports.Net ................................................................................25
      4.1.10. Fast Reports – FastReport.NET version 2015.3.4 ...........................................26
      4.1.11. Summary .........................................................................................................26
   4.2. Report model creating process ....................................................................................27
   4.3. Report model elements ..............................................................................................28
      4.3.1. Data storage elements .......................................................................................28
      4.3.2. Data presentation elements (elements palette) .....................................................29
      4.3.3. Report structure elements ..................................................................................30
      4.3.4. General properties of data storage elements ......................................................33
      4.3.5. General properties of data presentation and report structure elements ..............34
5. Discussion about report creation process .......................................................................36
   5.1. Models incompatibility and software dependence ....................................................36
   5.2. Alternative method for report model creation ..........................................................38
   5.3. Alternative report model format ................................................................................38
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.</td>
<td>Implementation of domain-specific language</td>
<td>40</td>
</tr>
<tr>
<td>6.1.</td>
<td>Background</td>
<td>40</td>
</tr>
<tr>
<td>6.2.</td>
<td>Concrete syntax</td>
<td>41</td>
</tr>
<tr>
<td>6.3.</td>
<td>Abstract syntax</td>
<td>44</td>
</tr>
<tr>
<td>7.</td>
<td>Comparison of Report-DSL and a visual editor</td>
<td>45</td>
</tr>
<tr>
<td>7.1.</td>
<td>Models compatibility</td>
<td>45</td>
</tr>
<tr>
<td>7.2.</td>
<td>Report creation process</td>
<td>46</td>
</tr>
<tr>
<td>7.3.</td>
<td>Usability</td>
<td>47</td>
</tr>
<tr>
<td>8.</td>
<td>Conclusion</td>
<td>51</td>
</tr>
<tr>
<td>9.</td>
<td>Bibliography</td>
<td>53</td>
</tr>
<tr>
<td>Application 1</td>
<td>Report-DSL grammar : an EBNF-specification</td>
<td>56</td>
</tr>
<tr>
<td>Application 2</td>
<td>Report-DSL generator: a translator to custom report format (Eclipse BIRT Project)</td>
<td>80</td>
</tr>
<tr>
<td>Application 3</td>
<td>The report model, created by using the Report-DSL</td>
<td>109</td>
</tr>
</tbody>
</table>
Figures

Figure 1. Report builder (left) and script example (right) of the report writer language from [18] ...........17
Figure 2. The main window interface of Eclipse – BIRT .................................................................22
Figure 3. The main window interface of Jaspersoft – Ireport Designer ...........................................22
Figure 4. The main window interface of Pentaho – Report Designer ..............................................23
Figure 5. The main window interface of SAP - Crystal Reports ................................................................23
Figure 6. The main window interface of Target Software – Target Decision Suite .........................24
Figure 7. The main window interface of Text Control – TX Text Control .NET ...................................24
Figure 8. The main window interface of GrapeCity – Active Reports Designer .................................25
Figure 9. The main window interface of Microsoft – Report Builder ................................................25
Figure 10. The main window interface of Stimulsoft – Reports.Net ................................................26
Figure 11. The main window interface of Fast Reports .......................................................................26
Figure 12. The model of report in visual editor Eclipse-BIRT (left) and its result in report viewer (right) .28
Figure 13. A conceptual model of a report creating process in existing reporting tools ......................36
Figure 14. A conceptual model of a report creating process in existing reporting tools and alternative way ........................................................................................................................................37
Figure 15. Roles of report designers and their content area ..................................................................41
Figure 16. Pseudo code for concrete syntax of DSL with guidelines of Karsai et al. [35] .....................43
Figure 17. Syntax template for a report model ......................................................................................43
Figure 18. Syntax template for a pseudo element of a report model ..................................................44
Figure 19. Syntax template for properties of a report element ............................................................44
Figure 20. The model fragment of a report created by using the Report-DSL (left) and its results in report viewer of Eclipse–BIRT reporting tool (right) ................................................45
Figure 21. Communication process while creating report model in visual editor (a) and Report-DSL (b) .46
Figure 22. Defining a type and a data-set connection for chart in visual report (a, b) and Report-DSL (c) 47
Figure 23. Defining axes for chart in visual report (a, b) and Report-DSL (c, d) ..................................48
Figure 24. Defining a legend and a color palette for chart in visual report (a, b) and Report-DSL (c) ....49
Figure 25. Defining a color palette for charts in visual report (a) and Report-DSL (b) with reusing of report element ........................................................................................................................................50

Tables

Table 1. List of reporting software ........................................................................................................21
Table 2. General report elements of reporting tools ...........................................................................31
Table 3. General properties of report elements: datasource ..................................................................33
Table 4. General properties of report elements: input parameter .......................................................33
Table 5. General properties or report elements: report parameter, report variable ..........................33
Table 6. General properties of data presentation elements .................................................................34
Table 7. Number of lines of code of report elements .........................................................................50
Table 8. Comparison of visual editor and Report-DSL .......................................................................51
1. Introduction

According to Oxford’s dictionaries the definition of “report” is “an account given of a particular matter, especially in the form of an official document, after thorough investigation or consideration by an appointed person” [1]. A report as a business report “must be informative, factual, understandable, and neatly presented” [2]. The process of creating a report is called reporting. In the period of information age most of the reports are created via software reporting tools. Different reporting tools are widely presented in business intelligence, “a field of the investigation of the application of human cognitive faculties and artificial intelligence technologies to the management and decision support in different business problems” [3]. Usually business reports include different tables, diagrams, charts, pictures and description text. Initial data for reports are taken from data sources such as database, specific data file, web-service, etc. The main task of a reporting tool is to convert initial data from data sources to understandable and neatly presented information in the required form for a user of the report. For that data conversion the report model is created in reporting tools. The report model is a set of instructions and rules for getting and transforming data from data sources and their presentation in the required form as well as the description of different report properties. For example, if data source is a relational database, then the report model normally has the information about the database connecting driver, SQL-statement and tool specific instructions and rules for data conversion to table, chart or other visual form. For table format the following properties can be defined: header, details (with grouping, mapping, highlighting, sorting, filtering options) and footer. For chart form: chart properties, colour palette, axes, series, etc.

In Gartner Magic Quadrant\(^1\) 2015 for business intelligence there are 24 companies [4]. Each of them offers the complex solution for business reporting. All products of these companies have a visual editor for the report model creating. The process of creating consists of defining and customizing of required report elements (data source, title, table, list, chart, etc.) and ordering them in the needed sequence, in the manner a user does it in textual or presentation editors (MS Word, MS PowerPoint or LibreOffice). Depending on the report editor a user should have special knowledge such as SQL, HTML, CSS, Java, .NET Framework, Javascript, C++ or other programming language.

The interfaces of visual editors are represented in Figures 1-10. The brief observation of interfaces and using of reporting tools has shown that all of them look alike and have the similar functionality – to create a report model. The difference is that report model files have a different format that can be suitable for this particular reporting tool. Therefore, after choosing of reporting tool a report developer is joined to this tool and its functional and technological features.

None of the tools developers offer the simple alternative way for the report model creating. An alternative way of visual editors may be a textual one that supports domain-specific language for business reporting, which is explored in this thesis. The domain-

---

\(^1\) “Gartner Magic Quadrant research methodology provides a graphical competitive positioning of four types of technology providers in fast-growing markets: Leaders, Visionaries, Niche Players and Challengers.” (https://www.gartner.com/technology/research/methodologies/magicQuadrants.jsp)
specific language “is a small, usually declarative, language that offers expressive power focused on a particular problem domain” [5].

According to [5] the general benefits of DSLs are:

- “DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs”.
- “DSL programs are concise, self-documenting to a large extent, and can be reused for different purposes”.
- “DSLs enhance productivity, reliability, maintainability, and portability”.
- “DSLs embody domain knowledge, and thus enable the conservation and reuse of this knowledge”.
- “DSLs allow validation and optimization at the domain level”.

The advantages of the report domain-specific language can be the following. It is the goal of this thesis to explore to which extent such advantages can be reached (and what limitations and drawbacks such an approach would have).

- Unified specification of report model for any tool;
- No association to concrete reporting tool;
- No restriction of visual designer;
- Alternative representation of report model;
- Simple reusing of report model or its part;
- Possibility of using language constructions (in textual editor).

This master thesis focuses on an alternative way for business report creation and provides the reader with a unified meta-model of the report and domain-specific language for report creation with further report generation in existing reporting tools.

1.1. Motivation

The existing report creation tools have some restrictions in their functionality; analysis of these tools has discovered the following functions that are not represented in the considered reporting tools:

- Using of unified functionality for managing of report elements inside report model;
- Reusing of report elements or report model in different reporting tools.

The goal of this master thesis is to find an alternative way of visual editors for report model creating that may lift restrictions and also help to improve the process of creating a model by getting compatibility of the report model with many software tools, by decreasing the dependence on a software tool for a creator of a report model, by getting higher readability and visibility of the model report, by reducing the number of lines of code and by increasing the number of re-using parts of report elements.

This master thesis studies the following topics of domain of reports creation:

- Incompatibility of the report models from the different software tools and dependence on the particular software tool;
- Constrained usability of the visual editor and non-existence of alternative way for the report model creation;
• Redundancy of code for the similar elements of a report model in XML-format.

1.2. Objective

The purpose of this work is to conduct the research on specific domain of business report creation methods, to develop a unified meta-model of a report and to specify the domain-specific language for it. This language can be used as an alternative method for the report creation with further report generation in existing reporting tools.

1.3. Structure of the thesis

This master thesis consists of six main parts. The section “Research approach” presents the summary about design science methodology that was used in this thesis. The section “Domain of report creation” covers aspects of report creation by using the existing tools, gives an overview of reporting tools and a representation of the report model creating process, its elements and their properties. In the section “Discussion about report creation process” problems in domain of reports creation are discussed and defined. In the third section “Implementation of domain-specific language” it is shown how the domain-specific language for report creation was implemented. The fourth part “Comparison of the Report-DSL and a visual editor” presents the report creation process in these methods. Finally, the section “Conclusion” presents advantages and disadvantages of Report-DSL and shows topics for further work. The example of report model in the Report-DSL is presented in Application 1; the specification of the Report-DSL and model generator – in Application 2 and 3.
2. Related works

The following section consists of three main parts. The first one presents scientific articles, related to this master thesis, on model-driving engineering and examples of using of domain-specific languages. The second part discusses different report creation approaches. The third one presents report creation approaches using a domain-specific language.

2.1. Model-driving engineering and domain-specific languages

Model-driven engineering is used in many industry sectors such as software development [6, 7, 8], insurance and automotive companies [9, 10], etc. Also domain-specific modeling is used for the following problem domains: telecom services, business processes, industrial automation, platform installation, medical device configuration, machine control, call processing, geographic information system, eCommerce marketplaces, applications in microcontroller, household appliance features, smartphone UI applications, ERP configuration, handheld device applications [11]. The model-driven approach is used for creating new systems as well as for improving and modernisation of existing systems, namely for configuration scripts, parameters or data structures [11], for description of requirements [12, 10, 9], for modelling of web-services [6], for extracting of models from the program code [7], for modelling performance testing of the web application [8] and for using as a rule engine or scripting language [11]. In the literature, there are a number of implementations of domain-specific languages with grammar and code generators. Some of them are presented below:

- The requirements language for the definition of structured scenarios [12] that was implemented in Eclipse Xtext framework. In this language “domain experts and DSML engineers specify requirements via structured natural-language scenarios” [12] in the form of descriptions, which are further “automatically transformed into executable test scenarios providing forward and backward traceability of domain requirements” [12].

- A domain-specific language for modelling of web services [6] that aims to solve problems of modelling web services using UML, because UML is not well-suited for this modelling [6]. The concrete syntax of language and code generator were implemented in Eclipse Xtext framework.

- A domain-specific language for the definition of automotive system requirements that can be presented in form of “the combination of controlled natural language and a domain-specific language” [10] and gives an opportunity to “the user to specify requirements textually with automotive vocabulary and generate artifacts such as functional models” [10]. The DSL are implemented in JetBrains Meta-Programming System.

- A domain specific language for extracting models from program source code that “provides a powerful query language for concrete syntax trees, and mappings between source grammar elements and target meta-model elements are expressed by rules similar to those found in model transformation languages” [7]. The DSL are implemented in Eclipse Xtext framework.
A domain-specific language for modelling performance testing [8] that “allows to automate the generation of test artifacts, such as test cases or test scripts, and improves communication among different teams” [8].

A domain-specific modelling language for specifying and visualizing requirements based on Requirement Abstraction Model for improving requirements of engineering activities [9].

The approach presented in this thesis follows the general model-driving engineering and domain-specific languages approaches, as outlined above, pretty closely. To the best of our knowledge, only a very few approaches have used model-driving engineering and domain-specific languages concepts in the domain of report generation to abstract reporting tools before (see Section 2.3 for details).

2.2. Alternative report creation approaches

Report generation is a frequently used task in applications that use databases [13]. A lot of reporting tools are developed for presentation of information from data-source in a visual form and generating of a report. The approach that is used in this master thesis for creation a report is not the only one. Other alternative approaches of report creation are presented below.

Gjorgjevikj et al. [14] offers an “applicative solution for generating reports from templates” that gives “power to more experienced administrators and simplicity to common end users” [14], which can generate reports from their databases with their own criteria and design. Software that is presented in this article enables the creation of templates containing text and tags by using of a web-based graphical user interface. These tags define the columns and contents of the reports that are recognized and substituted by values retrieved from the database on generation [14]. This software “can be used in public administration, in hospitals for patient’s medical record generating, at schools to easily generate student reports when needed, in warehouses and accounting firms for various tasks” [14].

Srivastava et al. [15] shows one more approach of using a template design for generation of investigation test reports in a Hospital Management Information System are offered in. The solution contains a template designer for result entry and offers a structured format for storing result entry template and data into XML. Report generation has two-layer scheme: generation of XML and generation of PDF. Authors show on experimental example that “the proposed framework significantly outperforms traditional methods of generating reports” [15].

Afonin et al. [16] offers a template language SQLReport “that can be used for rapid development of reports suitable for both reporting and dynamic data exploration” [16]. The features of this language are similar to reporting tools, which are presented in this master thesis in the Section 4. Key-features of the developed system are “supports zero-programming report development, parameterization of SQL queries, interactive results processing by means of client-side JavaScript libraries, and cross-report references” [16]. The result of the query can be presented only in a tabular form in a report [16].
Veytsman et al. [17] describes an approach to report generation that uses the web interface, SQL and \LaTeX. The system that is described in the article is used for the report creation that is “usually made by copying and pasting the daily reports” [17] in a general report. The users by using a web interface launch the report generation that creates a \LaTeX file by selecting the data from database and “create either a PDF report or an editable (e.g., in Microsoft Word) file” [17].

Chan [13] presents a document-driven approach to report generation. In this approach a report generation specifies in terms of a user defined report model that is based on SGML. "Contents of a report can be specified using a transformation language together with queries that retrieve data from different databases". A report output is an editable SGML-document, which can be translated to other formats, for example to HTML.

The approach presented in this thesis has similar task with the approaches mentioned above. Report generation by using model-driving engineering and domain-specific languages concepts is an alternative to approaches that use web interface, template languages, parameterization of SQL queries, document-driven concepts or visual editors that overviewed in the section 4.

2.3. Report creation through a domain-specific language

Application of a domain-specific language for a report model creation is not a new method in the business intelligence software. Dantra et al. [18] presents the development of a visual domain specific visual language (DSVL) and an environment for it that “assists end-users to rapidly design and implement reports sourced from an enterprise system and its database for the printing and graphics art industry” [18]. The language is offered not as an alternative to existing in concrete software approaches, but as a replacement of an existing textual domain-specific language that provides “a more accessible and productive approach to report writing” [18]. The language is created as a meta-modeling framework that “assists developers to modify and add new elements to this reporting language" [18]. The approach to the language design is similar to the design science approach that is used in this thesis (see Section 3), firstly, the domain of the target enterprise system and existing report creation approach are analyzed and described. After that the problems of report creation are defined and the domain specific visual language, environment and framework for it are described. Further the DSVL prototype is implemented by using Microsoft DSL Tools. After implementation the application of its using is demonstrated and evaluated. In Figure 1 a textual script for a report is shown.
The authors of the report writer language declare that it is specific for the enterprise content management and document management software “Prism WIN MIS” and it cannot be used in general reporting tools such as Crystal Reports, FastReport Studio or Eclipse Business Intelligence Reporting Tools, which are presented in the Section 4 of this thesis. The approach presented in this thesis is similar to the report writer language, but a report model, written in our Report-DSL, can be compatible with many reporting tools as opposed to the above mentioned approach.
3. Research approach

This section provides a general overview of the design science research paradigm used as a research method in this thesis. Next, we present the approach that is proposed as an alternative method for the report creation. Finally, the main contribution of this master thesis is described at the end of this section.

According to Hevner et al. [19] there are two paradigms: behavioral science and design science that characterize much of the research in the Information Systems discipline. “The behavioral science paradigm seeks to develop and verify theories that explain or predict human or organizational behavior. The design-science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts” [19] and knowledge, understanding of a problem domain and its solution are achieved by design, development and application of this artifact [19].

The research in this master thesis is based on design science research paradigm. This master thesis looks for extending of capabilities of creating a report model in reporting tools by creating a new and alternative method for report creation.

Design science research methodology can include five [20] or six [21] steps that can repeat in an iterative cycle of research process:

1. Identification and awareness of problem. In this step the specific research problem is defined and the value of a solution is justified. Resources required for this step include "knowledge of the state of the problem and the importance of its solution" [21]. The output is a proposal.

2. Suggestion and definition of objectives of a solution. In this step the goals of a problem solution are inferred and decision about what is possible and feasible is made. Resources required for this step include "knowledge of the state of problems and current solutions, if any, and their efficacy" [21]. The output is a tentative design.

3. Design and development of a solution artefact. The tentative design from the previous step is implemented in an artifact in this step. Such artifacts can be "potentially constructs, models, methods, instantiations or new properties of technical, social or informational resources" [21]. Resources required for development include knowledge of theory that can be brought to bear in a solution. The output is a working artifact.

4. Demonstration of a solution (missing by [20]). The demonstration can be represented as "use in experimentation, simulation, case study, proof, or other appropriate activity" [21]. This demonstration requires resources that include "effective knowledge of how to use the artifact to solve the problem" [21].

5. Evaluation of a solution. Observation and measuring of "how well the artifact supports a solution to the problem" [21] are done in this step. For evaluation a comparison, quantifiable measures, empirical evidence or a logical proof can be used. This step requires "knowledge of relevant metrics and analysis techniques" [21]. At the end of this step it can be decided "whether to iterate back to step three to try to improve the effectiveness of the artifact or to continue" [21] to the next step.

6. Conclusion and communication. In this step an assumption about the importance of the problem, utility and novelty of the artifact, the rigor and effectiveness of its design are concluded. This last step also can produce additional iterations in the research cycle.

Research in this master thesis follows design science research as a systematic approach: concretely, in this thesis the features of reporting tools are analysed, after that they are grouped into common elements and abstracted into a meta-model and then a domain-specific language is built based on the gained insights. As advised by design science research, these steps have been performed in many iterative cycles and during this research process the artifacts have been constantly improved, as new knowledge was discovered.
Applying this methodology, at first the problem domain is defined. This domain covers aspects of report creation by using the existing tools. An overview of reporting tools gives a representation of a report model creating process, its elements and their properties. After that a discussion about the report creation process is possible and problems are identified (Step 1). Next the alternative method for the report creation by using domain-specific language is suggested as solution of these problems (Step 2). Further, this language is implemented and the application is demonstrated in a realistic example (Step 3, 4). The comparison of the suggested alternative and existing reporting tools is conducted as evaluation of a solution (Step 5). As a conclusion step the results of the comparison are consolidated and new problem aspects are detected for the next iteration (Step 6).

The main contribution of this master thesis consists in creating a new and alternative method for the report creation by using the design science research approach, a method that can delete some restrictions of functionality in the existing report creation tools, for example, using of unified functionality for managing report elements inside a report model and reusing of report elements or a report model in different reporting tools. The main output artifact of this thesis is the domain-specific language for report creation (Report-DSL). This language extends the capabilities of creating a report model in reporting tools and helps to improve the process of creating a model, as well as the communication between participants of this process. Functionality of the language allows increasing the number of re-using parts of report elements. A report model, written in Report-DSL, can be compatible with many reporting tools. This compatibility decreases the dependence from a software tool for a creator of a report model. Also this model has alternative readability, visibility and the reduced number of lines of code in comparison with existing report model formats.
4. Domain of reports creation

This section gives an overview about existing reporting tools and summarizes information about the report modelling process and report model elements. Summaries of general properties of different types of report model elements are presented in Tables 2, 3, 4, 5 and 6.

According to Voelter [22] there are two approaches for defining a domain: inductive and deductive. In the inductive or bottom-up approach a domain is defined in terms of existing software used to address a particular class of problems or products. “That is, a domain is identified as a set of programs with common characteristics or similar purpose. ... They could be expressed in any Turing-complete language. Often such domains do not exist outside the realm of software”. In the inductive approach a domain is defined as a subset of programs written in a specific language and can be identified “in the form of their consistent use of a set of domain-specific patterns or idioms” [22]. This makes building a DSL for a domain relatively simple, because it becomes clear what the DSL has to cover and what code to generate from DSL programs. In the deductive or top-down approach, a domain is considered a body of knowledge about the real world, i.e. outside the realm of software.

Inductive approach application enables to make analysis of existing report creation software, after which it will be possible to define main aspects that cover the process of report creation.

In this section software products will be considered as support complex solutions for business analytics and/or business intelligence including reporting tools. Products such as Excel or R-Statistics can also be used as solutions for reporting, but in whole they have another purpose and another market segment position.

There are a lot of different reporting tools. The Web-service www.capterra.com, which connects buyers and sellers of business software, offers more than one hundred software products in the reporting tools area [23]. Some of them (with a number of users greater than one thousand) are presented here:

- Windward Solutions by Windward Studios
- pcFinancials by DSPanel
- Revel iPad POS by Revel Systems
- Tagetik by Tagetik
- MITS Distributor Analytics by Management Information Tools
- Quaestor 7 by AIT Software
- ChristianSteven BI Continuum by ChristianSteven Software
- Docmosis by Docmosis
- Intelicus Reports by Intelicus Technologies
- JReport by Jinfonet Software
- jsreport by jsreport
- Stimulsoft Reports Server by Stimulsoft
- Tableau by Tableau
- Weekdone by Weekdone
- Business Executive Bundle by Mr Dashboard
- Arnica WebReport by Arnica Software
- BI Office by Pyramid Analytics
- BIRT Designers by Actuate
- CALUMO by CALUMO
- CasterStats Reporter by TouchCast
- Cognos by IBM
- DBHawk by Datasparc
- Enth by Enth
- Exago by Exago
- FastReport.Net by Fast Reports
- Flexmonster Pivot Table by Flexmonster
- Fraser Stream Integration by Fraserstream
- Fruition by Juice Analytics
All of them have the same main function: they can create a report model and run it to produce a required output, such as HTML, WORD, XLS or PDF. Ten different software tools with a trial or free/community version for the operating system Microsoft Windows were chosen for analysis. These products and some of their properties are listed in Table 1.

### Table 1. List of reporting software

<table>
<thead>
<tr>
<th>Company</th>
<th>Software product</th>
<th>Gartner MQ 2015</th>
<th>Platform</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>BIRT 4.2</td>
<td>yes</td>
<td>Java</td>
<td>Community</td>
</tr>
<tr>
<td>Tibco</td>
<td>iReports Designer 5.6</td>
<td>yes</td>
<td>Java</td>
<td>Community</td>
</tr>
<tr>
<td>SAP</td>
<td>Crystal Reports 2013</td>
<td>yes</td>
<td>.NET</td>
<td>Trial</td>
</tr>
<tr>
<td>Targit Software</td>
<td>TARGIT Decision Suite 2015</td>
<td>yes</td>
<td>.NET</td>
<td>Trial</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Report Builder 3.0</td>
<td>yes</td>
<td>.NET</td>
<td>Trial</td>
</tr>
<tr>
<td>Pentaho</td>
<td>Report Designer 5.3</td>
<td>yes</td>
<td>Java</td>
<td>Community</td>
</tr>
<tr>
<td>Text Control</td>
<td>TX Text Control 22.0</td>
<td>no</td>
<td>.NET</td>
<td>Trial</td>
</tr>
<tr>
<td>Grapecity</td>
<td>Active Reports Designer 9.2</td>
<td>no</td>
<td>.NET</td>
<td>Trial</td>
</tr>
<tr>
<td>Stimulsoft</td>
<td>Report Designer 2015</td>
<td>no</td>
<td>.NET</td>
<td>Trial</td>
</tr>
<tr>
<td>Fast Reports</td>
<td>FastReport.NET 2015</td>
<td>no</td>
<td>.NET</td>
<td>Trial</td>
</tr>
</tbody>
</table>

### 4.1. Overview of reporting tools


#### 4.1.1. Eclipse – BIRT

“BIRT is an open source software project that provides the BIRT technology platform to create data visualizations and reports that can be embedded into rich client and web applications, especially those based on Java and Java EE. BIRT is a top-level software
project within the Eclipse Foundation, an independent not-for-profit consortium of software industry vendors and an open source community” [24].

4.1.2. Jaspersoft – IReport Designer

“iReport is the free, open source report designer for JasperReports and JasperReports Server. Create very sophisticated layouts containing charts, images, subreports, crosstabs and much more” [25].
4.1.3. Pentaho – Report Designer

“The Report Designer is a desktop reporting tool that provides a visual design environment to easily create sophisticated and rich reports. It is geared towards experienced and power users, who are familiar with the concepts and data-sources used” [26].

4.1.4. SAP – CrystalReports 2013 SP4

“SAP Crystal Reports software is the de facto standard in reporting. With SAP Crystal Reports, you can create powerful, richly formatted, dynamic reports from virtually any data source, delivered in dozens of formats, in up to 24 languages. A robust production reporting tool, SAP Crystal Reports turns almost any data source into interactive, actionable information that can be accessed offline or online, from applications, portals and mobile devices” [27].
4.1.5. Targit Software – Targit Decision Suite

“TARGIT Decision Suite is the only business intelligence platform that offers visual data discovery tools, self-service business analytics, reporting, and stunning dashboards in a single, integrated solution” [28].

![Targit Decision Suite Interface](image1.png)

Figure 6. The main window interface of Targit Software – Targit Decision Suite

4.1.6. Text Control – TX Text Control .NET

“The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor. Users can create documents and templates using ordinary Microsoft Word skills. TX Text Control is completely independent from MS Word or any other third-party application and can be completely integrated into your business application” [29].

![Text Control Interface](image2.png)

Figure 7. The main window interface of Text Control – TX Text Control .NET
4.1.7. GrapeCity – Active Reports Designer

“ActiveReports is a complete reporting solution that includes an extensive .NET reporting library with API, data visualization and navigation controls, designers, and distributable viewers for desktop, web, and mobile” [30].

![Image of GrapeCity – Active Reports Designer](image1)

**Figure 8.** The main window interface of GrapeCity – Active Reports Designer

4.1.8. Microsoft – Report Builder

“Report Builder provides data visualizations that include charts, maps, sparklines, and data bars that can help produce new insights well beyond what can be achieved with standard tables and charts. Use Report Builder to create reports and shared datasets. Publish report parts, and then browse the Report Part Gallery to reuse existing report parts as building blocks for creating new reports quickly with a “grab and go” experience” [31].

![Image of Microsoft – Report Builder](image2)

**Figure 9.** The main window interface of Microsoft – Report Builder

4.1.9. Stimulsoft – Reports.Net

“Stimulsoft Reports.Net is a .NET based report generator which helps you create flexible and feature rich reports. Stimulsoft Reports.Net is delivered with source codes. All
reports are created in the report designer with handy and user-friendly interface. You can use the report designer both at design time and runtime. No royalties for using the report designer at runtime are required. Using Stimulsoft Reports.Net you can create reports on the basis of various data sources” [32].

4.1.10. Fast Reports – FastReport.NET version 2015.3.4

“FastReport.Net is a full-featured reporting solution for Windows Forms, ASP.NET and MVC” [33].

4.1.11. Summary

All above mentioned reporting tools look the same and have the similar functionality – to create a report model in a visual designer. From the technical point of view a report model is a file with commands, instructions and rules, which can be run to connect to data source, to transform data and to present them in the required output format.
After performing the analysis of reporting tools it is possible to define main features of a report designer:

- Support of many data source formats (Flat file, Excel, Database, XML, Web-service);
- Report input parameter;
- Data transformation, data filtering and data sorting;
- Report elements palette;
- Report element formatting;
- Conditional formatting;
- Scripting for better business-specific logic;
- Support of many output formats (HTML, PDF, CSV, XLS, XLS, DOC, PPT, ODS, ODP, ODT).

The following functions are not represented in the considered reporting tools:

- Using of unified functionality for managing of report elements inside report model;
- Reusing of report elements in different reporting tools.

Some software tools have an opportunity to create the report model by means of general-purpose languages (GPLs), such as Java or .NET Framework. These means will not be considered for the purposes of this work, because they are more related to GPLs than to domain-specific languages.

### 4.2. Report model creating process

The process of report model creating consists in the defining and customizing of required report elements (data source, title, table, list, chart, etc.) and ordering them in the needed sequence on layout. Also, depending on the report designer a user should have special knowledge such as SQL, HTML, CSS, Java, .NET Framework, Javascript, C++ or other program languages for using additional functionality of the reporting tool.

After analysis of the above-mentioned reporting tools it is possible to define the following main steps for the report model creating (it means that the report content and structure are already known and the technical implementation is the next task):

1. Defining data source
   a. Choice of data source type
   b. Defining connection parameters

2. Defining report input parameters
   a. Defining parameters type and its data type
   b. Optional: dataset selecting

3. Creating datasets
   a. Defining query-statement to data source
   b. Optional: defining filters, computed columns, grouping, input parameters, etc.

4. Defining data presentation elements
   a. Connection with data sets
   b. Defining filters, grouping, sorting, interactivity, etc.
   c. Formatting elements
5. Ordering report elements on layout
6. Defining general report properties (header, footer, background, etc.)

Now the created report model can be run in the report viewer to get the final report. An example of a report model and its final report is shown in Figure 12.

![Figure 12. The model of report in visual editor Eclipse-BIRT (left) and its result in report viewer (right)](image)

4.3. Report model elements

The main elements of reports in the represented reporting tools are shown in Table 2. The left column contains the short element description. The first row contains the names of reporting tools. The elements names of the corresponding tool or information about the absence or presence of elements are represented as values in the table. Obviously the elements with the same functions can have different names in different reporting tools. General properties of elements from Table 2 are shown in Tables 3 – 6.

On the basis of the above mentioned steps of the report model creation it is possible to divide the report elements into 3 groups: data storage elements, data presentation elements and report structure elements. Data storage elements contain instructions for receiving and primary data transforming for the report creation. Data presentation elements are the instructions for data transforming and presenting data from data storage elements in the end-user needed form. Report structure elements help to locate the data presentation elements on report layout.

4.3.1. Data storage elements

A. External source of data - an element to set up the connection to an external source of data provider, such as database, flat data file (txt, csv), excel file, web-service, xml data, etc. General properties: data source type, URI/ URL, charset/encoding, database connection driver, Schema definition, etc.
B. Set of data from data source – a selection of data from data source which will be used for further visualisation. General properties: data source reference, worksheet/file, SQL-statement, output columns name, computed columns, filter options.

C. Input report parameter – an element to define input values and to prompt a user to provide it when the report runs.

D. Report variable – a variable that can be defined in a report for further using.

4.3.2. Data presentation elements (elements palette)

E. Static text – a static piece of text (normally it contains one line text) that is displayed in the report. General properties: font options, background color, text options, display and break options, padding, border, margin, hyperlink, bookmark, visibility.

F. Block text – a multi-line text element that can be presented in a plain text or HTML-format. It provides the ability to enter a formatted text. General properties: font options, background color, text options, display and break options, padding, border, margin, hyperlink, bookmark, visibility and built-in expression options for HTML formatting.

G. Block text with optional data expressions – a multi-line text element that can display blocks of text from dataset by using an expression that is a combination of constants, controls, functions, literal values, field names and operators. General properties: font options, background color, text options, display and break options, padding, border, margin, hyperlink, bookmark, visibility and built-in expression options for HTML formatting and scripts function input.

H. Data set column or expression result – an element that displays the value from a data set, an expression or a parameter. General properties: font options, background color, text options, display and break options, padding, border, margin, hyperlink, bookmark, visibility and data value type formatting options.

I. Image from a file, URL or database – an element that provides an image from a file, URL or database in one of different image formats such as jpg, png, svg, etc. General options: border, margin, visibility, bookmark, width, height, style, display and break options, reference, alt text.

J. List presentation with header/detail/footer bands – an element that creates a banded presentation of data from data set. General properties that can be applied for all list as well as for each item of list: font options, background color, text options, display and break options, border, hyperlink, bookmark, visibility, width, height, vertical alignment, data binding, data grouping, highlighting, filters, sorting.

K. Table presentation with header/detail/footer bands – an element that presents data in a tabular form with optional grouping, sorting and highlighting of presented data. General properties that can be applied for whole table as well as for each row/column of table: font options, background color, text options, display and break options, border, hyperlink, bookmark, visibility, width, height, vertical alignment, data binding, data grouping, highlighting, filters, sorting.

L. A chart or graph – an element that constructs a chart and presents data in a graphical form. General properties: width, height, display options, output format, chart
type, data binding, data grouping, highlighting, filters, sorting. Properties that depend on chart type parameter: axes, legend, plot, series/categories, sorting and series grouping.

**M.** A spark line chart elements – the same element as the element (L), but with less number of options.

**N.** Cross tabulation – an element that displays data in a cross tabulated, matrix format with optional sorting, highlighting and total values of presented data. General properties that can be applied for whole table as well as for each row/column of table and grand total values: font options, background color, text options, display and break options, border, hyperlink, bookmark, visibility, width, height, vertical alignment, data binding, data grouping, highlighting, filters, sorting, and built-in expression options for aggregating values in rows and columns.

**O.** A barcode chart element – an element that generates the barcode.

**P.** A reference to another report – an element that provides a report that appears inside another report.

**Q.** Map component – an element for displaying the data on the world or local map.

**R.** Post-index, cells – an element for presenting data in ZIP-code format or cells for filling.

**S.** Own components – a possibility to create own components for using in the report.

### 4.3.3. Report structure elements

**T.** Layout for visually arrange of items – an element that can contain other elements and helps to order them for getting output in the report form. General properties: Font, Size, Color, Background color, Text options, Display and Break options, Padding, Border, Margin, Hyperlink, Bookmark, Visibility, Width, Height, Vertical alignment.

**U.** A method of grouping elements – an element or method that helps to group elements for common use.

**V.** Vector graphical elements – the elements that represent graphical primitives such as line, rectangle, ellipse with formatting options.

**W.** Report page – common properties for displaying the report page. General properties: Width, Height, Background color, Orientation, Border, Margin, Header height, Footer height and Header/Footer formatting properties.
Table 2. General report elements of reporting tools

### Data storage elements

<table>
<thead>
<tr>
<th>Reporting tool →</th>
<th>BIRT</th>
<th>Pentaho</th>
<th>Jasper Soft</th>
<th>Active Reports</th>
<th>Crystal Reports</th>
<th>Fast Report</th>
<th>Target</th>
<th>Stimulsoft Reports .Net</th>
<th>Text Control</th>
<th>MS Report Builder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong> External source of data (DB, data file etc.)</td>
<td>Data source</td>
<td>Data source</td>
<td>Data source</td>
<td>Data source</td>
<td>Data source</td>
<td>Data source</td>
<td>Source Data</td>
<td>Data Source</td>
<td>Data Source</td>
<td>Data Source</td>
</tr>
<tr>
<td><strong>B</strong> Set of data from data source</td>
<td>Dataset, Datasource</td>
<td>SQL-Query</td>
<td>Dataset</td>
<td>SQL-Query</td>
<td>Dataset</td>
<td>Dataset</td>
<td>Dataset</td>
<td>Dataset</td>
<td>Master Table</td>
<td>Dataset</td>
</tr>
<tr>
<td><strong>C</strong> Input report parameter</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><strong>D</strong> Report variable</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

### Data presentation elements (elements palette)

<table>
<thead>
<tr>
<th><strong>E</strong> Static text</th>
<th><strong>F</strong> Block text</th>
<th><strong>G</strong> Block text with optional Data Expressions</th>
<th><strong>H</strong> Data set column or expression result</th>
<th><strong>I</strong> Image from a file, url or database</th>
<th><strong>J</strong> List presentation with header/detail/footer bands</th>
<th><strong>K</strong> Tabular presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static text</td>
<td>Label</td>
<td>Label</td>
<td>Static Text</td>
<td>Label</td>
<td>Text object</td>
<td>Label</td>
</tr>
<tr>
<td>Block text</td>
<td>Text</td>
<td>○</td>
<td>HTML component</td>
<td>Text box</td>
<td>Text object</td>
<td>Text</td>
</tr>
<tr>
<td>Block text with optional Data Expressions</td>
<td>Dynamic text</td>
<td>Text Field, Numeric Field</td>
<td>Text Field</td>
<td>Rich Text Box</td>
<td>Text object</td>
<td>Rich Text</td>
</tr>
<tr>
<td>Data set column or expression result</td>
<td>Data</td>
<td>Data Field, Resource elements (Label, Field, Message)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Image from a file, url or database</td>
<td>Image</td>
<td>Image, Image Field</td>
<td>Image</td>
<td>Picture</td>
<td>Image</td>
<td>Image</td>
</tr>
<tr>
<td>List presentation with header/detail/footer bands</td>
<td>List</td>
<td>○</td>
<td>List</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Tabular presentation</td>
<td>Table</td>
<td>Inside</td>
<td>Table</td>
<td>Inside</td>
<td>Inside</td>
<td>Table</td>
</tr>
<tr>
<td>with header/detail/footer bands</td>
<td>canvas</td>
<td>canvas</td>
<td>canvas</td>
<td>canvas</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>L</strong> A chart or graph that shows data graphically</td>
<td>Chart</td>
<td>Chart, Survey Scale</td>
<td>Chart, Spider Chart</td>
<td>Chart</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>M</strong> A spark line chart elements</td>
<td>○</td>
<td>Sparkline</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>N</strong> Cross tabulation to display aggregate data in row and column format</td>
<td>Crosstable</td>
<td>Crosstab-report</td>
<td>Crosstab</td>
<td>Cross table</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>O</strong> A barcode chart element</td>
<td>○</td>
<td>Barcode</td>
<td>Barcode</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P</strong> A reference to another report.</td>
<td>○</td>
<td>Sub Report</td>
<td>Subreport</td>
<td>Subreport</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Q</strong> Map component</td>
<td>○</td>
<td>○</td>
<td>Map</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>R</strong> Post-index, cells</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S</strong> Own components</td>
<td>○</td>
<td>○</td>
<td>Generic element</td>
<td>OLE Object</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Report structure elements

| **T** Layout for visually arrange other items | • | • | • | • | • | • | • | • | • |
| **U** A method of grouping elements | Grid | Band, Message Field | Frame | ○ | ○ | ○ | ○ | ○ | Bands |
| **V** Vector graphical elements | ○ | Horizontal / Vertical Line, Line, Ellipse, Rectangle | Line, Ellipse, Rectangle, Round Rectangle | Line, Cross-section Line/Box, Shape | Line, Rectangle | Line, diagonal line, Rectangle, Dreieck, Ellipse | ○ | Horizontal/ Vertical Line, Shape, Rectangle | ○ |

32
### 4.3.4. General properties of data storage elements

Table 3. General properties of report elements: datasource

<table>
<thead>
<tr>
<th>Element property</th>
<th>A (Datasource)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Excel</td>
</tr>
<tr>
<td>URI/URL</td>
<td>•</td>
</tr>
<tr>
<td>First line as column name</td>
<td>•</td>
</tr>
<tr>
<td>Second line as data type</td>
<td>•</td>
</tr>
<tr>
<td>Charset/Encoding</td>
<td>•</td>
</tr>
<tr>
<td>Separator</td>
<td>•</td>
</tr>
<tr>
<td>Use trailing null columns</td>
<td>•</td>
</tr>
<tr>
<td>Driver class path</td>
<td>•</td>
</tr>
<tr>
<td>User name</td>
<td>•</td>
</tr>
<tr>
<td>Password</td>
<td>•</td>
</tr>
<tr>
<td>Schema definition location</td>
<td>•</td>
</tr>
<tr>
<td>SOAP Endpoint</td>
<td>•</td>
</tr>
</tbody>
</table>

Table 4. General properties of report elements: input parameter

<table>
<thead>
<tr>
<th>Element property</th>
<th>B (Data set)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Excel</td>
</tr>
<tr>
<td>Data source</td>
<td>•</td>
</tr>
<tr>
<td>Worksheet/File</td>
<td>•</td>
</tr>
<tr>
<td>SQL-statement</td>
<td>•</td>
</tr>
<tr>
<td>SQL-statement parameter</td>
<td>•</td>
</tr>
<tr>
<td>Output columns name</td>
<td>•</td>
</tr>
<tr>
<td>Computed columns</td>
<td>•</td>
</tr>
<tr>
<td>Filter options</td>
<td>•</td>
</tr>
</tbody>
</table>

Table 5. General properties or report elements: report parameter, report variable

<table>
<thead>
<tr>
<th>Element property</th>
<th>Possible values</th>
<th>C (Input parameter)</th>
<th>D (Report variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element name</td>
<td>String sequence</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Prompt text</td>
<td>Multi string sequence</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Data type</td>
<td>Boolean, Date, Datetime, Decimal, Float, Integer, String, Time</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Display type</td>
<td>Text box, Combo box, List box, Radio button</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Default value</td>
<td>Value of element</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Static or Dynamic value list</td>
<td>True/False</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Sorting options</td>
<td>Ascending/Descending</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Visibility</td>
<td>True/False</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Required option</td>
<td>True/False</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Allow duplicate values</td>
<td>True/False</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>
### General properties of data presentation and report structure elements

<p>| Element property          | Possible values, units | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |
|---------------------------|------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Element name              | String sequence        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Text font                 | Sans Serif, etc.        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Text size                 | Auto; Large, Larger, Medium, Small, Smaller, X Large, X Small, XX Large, XX Small; Numeric value |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Text color                | Auto; Color Name; RGB(R,G,B) |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Text style                | Bold, Italic, Underline, Text Line Through |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Text alignment            | Left, Centred, Right, Justified |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Text whitespace           | Auto; No Wrapping, Normal, Preformatted |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Style                     | None; Style name        |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Background color          | Auto; Color Name; RGB(R,G,B) |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Display                   | Block, Inline, No Display |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Padding side              | Top, Bottom, Left, Right |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Padding length            | (1…100)% or Numeric value with units: cm, ems, exs, in, mm, picas, points, pixels |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Border Style              | Solid, Dotted, Dashed, Double |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Border Color              | Auto; Color Name; RGB(R,G,B) |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Border Width              | Thin, Medium, Thick; Numeric value in px |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Border Side               | Top, Bottom, Left, Right, All, None |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Margin Side               | Top, Bottom, Left, Right |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Margin length             | (0…100)% or Numeric value with units: cm, ems, exs, in, mm, picas, points, pixels |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Hyperlink location        | URI, Internal bookmark  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |</p>
<table>
<thead>
<tr>
<th>Hyperlink target</th>
<th>Blank, Parent, Self, Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>Break type</td>
<td>Always, Auto, Avoid</td>
</tr>
<tr>
<td>Break location</td>
<td>Before, After, Inside</td>
</tr>
<tr>
<td>Visibility conditions</td>
<td>Scripted expression</td>
</tr>
<tr>
<td>Bookmark</td>
<td>Scripted expression</td>
</tr>
<tr>
<td>Content Type</td>
<td>Auto, Plain, HTML, Expression</td>
</tr>
<tr>
<td>Content expression</td>
<td>HTML expression</td>
</tr>
<tr>
<td>Expression</td>
<td>Scripted expression</td>
</tr>
<tr>
<td>Data format</td>
<td>Number, Date Time, String</td>
</tr>
<tr>
<td>Width</td>
<td>(0…100)% or Numeric value with units: cm, ems, exs, in, mm, picas, points, pixels</td>
</tr>
<tr>
<td>Height</td>
<td>(0…100)% or Numeric value with units: cm, ems, exs, in, mm, picas, points, pixels</td>
</tr>
<tr>
<td>Image Reference</td>
<td>URI, Embedded image name, Data field name (for Image from database)</td>
</tr>
<tr>
<td>Alt Text</td>
<td>Scripted expression</td>
</tr>
<tr>
<td>Vertical alignment</td>
<td>Auto, Bottom, Middle, Top</td>
</tr>
<tr>
<td>Specific properties</td>
<td>Properties that are specific for elements</td>
</tr>
</tbody>
</table>

- item is available
○ item is not applicable or missing
5. Discussion about report creation process

This section covers the aspects of a report creation process such as models incompatibility and software dependence, alternative method for report model creation and alternative report model format. Each aspect is divided into the following parts: discussion, problem, possible solution and expectation.

5.1. Models incompatibility and software dependence

Discussion: On the software market there is a wide range of tools for getting analytical information from different data sources. It is not a simple choice for a company or a developer which tool to use in order to achieve their goals: to get the analytical information from data sources quickly and cheaply. This choice enforces the user to use this particular tool, and switching to another software can be very time-consuming, expensive or even impossible. The reason for this, is that software tools use their own custom report metamodels that are compatible only with this concrete software tool, i.e. the design of a model and the view of a report is possible in that particular tool where this model was created (see Figure 13).

![Figure 13. A conceptual model of a report creating process in existing reporting tools](image)

The overview of existing report creation tools shows (see Section 4.1.) that development of the same business reports is possible by different tools with the same functionality.
Problem: incompatibility of the report models from different software tools and dependence on the concrete software tool.

Possible solution: a unified specification of report meta-model and generator into a custom software format (see Figure 14).

Expectation:
- Compatibility of the report model with many software tools;
- Dependence on a software tool decreases.

Also, users will have an opportunity to choose products (report design tools) that are different by implementations, names, prices, but have the same functionality. That approach, which in the context of model-driven engineering is the translation from Platform Specific Model\(^2\) to Platform Independent Model\(^3\), theoretically allows achieving:
- Independence of a model from development tools that allows implementing a model, based on any software platform.

---

\(^2\) In this case Platform Specific Model is Application Specific Report Model

\(^3\) In this case Platform Independent Model is Application Independent Report Model
Resource saving during software execution for some several software platforms at the same time, i.e., if a developer creates a model once, he will get an opportunity to generate the report model for different software platforms.

5.2. Alternative method for report model creation

Discussion: creating a report model in existing software tools is mainly achieved by using a visual editor (such as WYSIWYG4). It is possible to say that visual editors are standard for creating a model of report documents that contain such elements as tables, diagrams, charts, pictures, and formatting text. The canvas and dialog windows are used in the visual editors for creating a report model where the properties of corresponding elements can be defined.

Sometimes in order to define a property of an element it is necessary to open some overlapping dialogue windows.

It is important for a user to have the high visibility and readability of that model. Existing tools display a model as the visual representation of the model result. That is especially uninformative if it is necessary to read the model quickly and to understand which elements the model consists of and which properties of these elements are defined when the model was created by another user or when the user returns to the development of the model. During further development in order to view element properties, it is necessary to open some overlapping dialogue windows.

The same view can be implemented by using different elements and their properties, but it will be shown in the same form in the visual model. One can understand which kind of form it is (which element is used) only by opening that element and looking at its properties.

If the report model is described by using the formatting text (and not using the visual representation) then the readability and the visibility of the report can improve.

Problem: constrained usability of the visual editor and non-existence of an alternative way for the report model creation.

Possible solution: the textual form of model report view.

Expectation:
- The readability and visibility of the report model improves;
- The speed of report creation process increases.

Also, the user will have an opportunity to use the visual editor after the generation of the textual model report in Application Specific Report Model.

5.3. Alternative report model format

Discussion 3: many reporting tools save the report model in XML-format, which is used by the report generator parser. The report developer can create a model or edit it by changing the XML-file. However, as Martin Fowler noticed in his article «Language

---

4 WYSIWYG - acronym from “what you see is what you get”, denoting the representation of text on-screen in a form exactly corresponding to its appearance on a printout (http://en.oxforddictionaries.com/definition/wysiwyg).
Workbenches: The Killer-App for domain-specific Languages?» (https://www.martinfowler.com/articles/languageWorkbench.htm), «XML makes it easy to parse, although not as easily readable as a custom format might be».

It will be much easier for users to work with the model description made for readability, but not for parsing. Also, by using the custom domain-specific format, there is a possibility to avoid redundancy of the code that is presented in xml-format.

Problem 3: the non-usability of creating a report model in xml-format and the excessive frequency of code for similar elements.

Possible solution 3: domain-specific language with additional functional elements.

Expectation 3:
- The number of lines of code reduces;
- The readability of the report model increases;
- The number of re-using parts of report elements increases;
- The speed of development increases.
6. Implementation of domain-specific language

This section covers the implementation of the domain-specific language created within this thesis. In the first subsection main steps for implementation of domain-specific language are explained. Detailed information about the implementation of concrete syntax and templates for defining a report model by using domain-specific language are presented in the next subsection. In the third subsection an explanation about the abstract syntax is given. Specifications of concrete and abstract syntax are represented in Applications 1 and 2 after the main part.

6.1. Background

Implementation of domain-specific language means “developing a program that is able to read text written in that DSL, parse it, process it, and then possibly interpret it or generate code in another language.” [...] First of all, when reading a program written in a programming language, the implementation has to make sure that the program respects the syntax of that language” [34]. To this aim, the program is broken into tokens. “Each token is a single atomic element of the language; this can be a keyword (such as class in Java), an identifier (such as a Java class name), or symbol name (such as a variable name in Java)” [34]. For instance, in the example on Figure 15, ‘data-set’, ‘label’, ‘chart’ are keywords; the braces ‘{ }’ are operators (which can be seen as special kinds of keywords as well). Elements like 'myReportName', 'myLabelName' are string literals and 17 is an integer literal.

Bettini [34] defines the lexical analysis as “the process of converting a sequence of characters into a sequence of tokens” and the program that executes such process is called “a lexical analyzer, lexer, or simply a scanner” [34]. Having the sequence of tokens is not enough, it is necessary to be “sure that they form a valid statement in our language, that is, they respect the syntactic structure expected by the language. This phase is called parsing or syntactic analysis” [34].

By using the software framework for developing domain-specific languages it is not necessary to implement a lexer or parser by hand. For defining the syntax of the language, it is needed to specify the grammar of the language, and from this specification framework automatically generate the code for the lexer and parser. Such specifications are called grammars or concrete syntax. “A grammar is a set of rules that describe the form of the elements that are valid according to the language syntax” [34].

Next part of DSL implementation is an abstract syntax. The abstract syntax is used for transformations and code generation. The abstract syntax can be changed without any effect on the CS and existing programs [22]. As a rule a framework for DSL developing already has a built-in mechanism for defining abstract syntax and mapping it to concrete syntax.

Xtext, an open-source software framework for developing programming languages and domain-specific languages, will be used for implementation of domain-specific language for report creation (Report-DSL). (https://www.eclipse.org/Xtext/)

As the first activities of language development Karsai et al. [35] offer to identify language uses early and to define roles that develop, review, and deploy the involved
report models. For this case study DSL will be used for documentation of a report model (structure of the report, contents of the report and behaviour of report elements). It is possible to define three main roles which take a part in the report model design process: domain expert, data-source specialist, graphic designer (see Figure 15). The domain expert defines the content of a report, i.e. which elements will represent the data. The graphic designer defines how report elements will be represented, as well as their graphic. The data-source specialist defines content of data storage elements, connections with data-source, SQL-queries, structure of result-sets, etc. A report designer can represent one or more roles creating a report model.

![Diagram of roles in report model design]

**Figure 15. Roles of report designers and their content area**

### 6.2. Concrete syntax

Voelter [22] states that "grammars are the formal definition for concrete textual syntax". Syntax rules of grammar "can be expressed in Backus-Naur Form (BNF), written as $S ::= P_1 \ldots P_n$. This grammar defines a symbol $S$ by a series of pattern expressions $P_1 \ldots P_n$. Each pattern expression can refer to another symbol or can be a literal such as a keyword or a punctuation symbol. If there are multiple possible patterns for a symbol, these can be written as separate productions (for the same symbol), or the patterns can be separated by the ‘$|$’ operator to indicate a choice. An extension of BNF, called Extended BNF (EBNF), adds a number of convenience operators such as ‘?’ for an optional pattern, ‘*’ to indicate zero or more occurrences, and ‘+’ to indicate one or more occurrences of a pattern expression". [22]

For implementation of concrete syntax Karsai et al. [35] offer to use descriptive notations because “a descriptive notation supports both learnability and comprehensibility of a language especially when reusing frequently-used terms and symbols of domain or

---

[22] Backus-Naur Form - a metasyntactic notation used to specify the syntax of programming languages, command sets, and the like (https://www.abbreviations.com/term/76058).
A report model in Report-DSL will be presented as description of the result report and its elements, for example: « A report take data-set "MyDataSet" with fields "A, B, C" from data-source "MyDatasource". A report body has a title "MyTitleText" and a bar chart “MyChart” with overlays bars. X-axis of chart presents values from field "A", Y-axis of chart presents values from field "B"». Pseudo code for concrete syntax of DSL is shown in Figure 1.6 below.

Also Karsai et al. [35] advise to make elements distinguishable, because “easily distinguishable representations of language elements are a basic requirement to support understandability”. For separation of kinds of elements in textual languages, keywords are usually used that “have to be placed in appropriate positions of the concrete syntax”. These keywords should be efficient for writing and not for parsing because “models are much more often read than written and therefore to be designed from a readers point of view” [35]. For definition and description of report elements common distinguishable keywords, like label, chart, table, row, cell, width 15 cm, margin 5 pc, align centre and others will be used.

Karsai et al. [35] propose to use comments and to provide organizational structures for models. For explaining design decisions it is essential to write comments on model elements, because “this makes models more understandable and simplifies or even enables collaborative work” [35]. For comments a widely accepted standard form of line comments, like // ... and grouped comments, like /* ... */ will be used. For organizing of hierarchies in a report model structure elements like grid or table will be used. For organizing of similar report model elements syntactic blocks like “data-sources { ... }”, “data-sets { .. }”, “body { ... }”, etc. will be used. Also it will be possible to separate a model into separate files.

Karsai et al. [35] advise to keep the balance between compactness and comprehensibility, because “usually a document is written only once but read many times. Therefore, the comprehensibility of a notation is very important, without too much verbosity. On the other hand, the compactness of a language is still a worthwhile and important target in order to achieve effectiveness and productivity while writing in the language. Hence, a short notation is more preferable for frequently used elements rather than for rarely used elements” [35]. The short compact form can be used for the notation of report element properties. If the property of element has unambiguous value, then it is possible to use only this value for its declaration, without describing the name of this property. For example, next two declarations have similar semantics and comprehensibility, but the second is more compact:

- chart myChartName type bar subtype overlay {}
- chart myChartName bar overlay {}
Some syntax templates for defining a report model by using report domain-specific language are shown in the figures below.

The root element of the report model is ‘Report’ (see Figure 17), which can have group of same elements such as ‘Data-sources’, ‘Data-sets’, ‘Parameters’, ‘Styles’, ‘Page-setups’, ‘Element properties’ and ‘Report body’. A group of elements is a place for definition of corresponding elements, such as ‘Data-source’, ‘Data-set’, ‘Parameter’, etc. In the section ‘Report body’ a structure of the report will be defined by using such elements as ‘Grid’, where report elements such as ‘Label’, ‘Text’, ‘Dynamic text’, ‘Image’, ‘List’, ‘Table’, Chart’, etc. will be placed.

```
report  myReportName {
  data-sources {
    data-source myDataSourceName {
      }
  }
  data-sets {
    data-set myDatasetName { fields { A, B, C } }
  }
  //The report body starts here
  body {
    label myLabelName content “myTitleText”
    { font-size 17px }
    chart myChartName bar overlay {
      x-axe text { myDatasetName.A }
      y-axe linear { myDatasetName.B }
    }
  }

```  

**Figure 16. Pseudo code for concrete syntax of DSL with guidelines of Karsai et al. [35]**

An element can be defined by using a syntax template that is shown in Figure 18. The element can have ‘Base value properties’ which define the element type, ‘Graphic properties’ and/or ‘Content properties’.

```
Report
{
  Group of elements
  {
    Elements
  }
  Report body
  {
    Elements
  }
}

```  

**Figure 17. Syntax template for a report model**
Element properties can be aggregated in 'Group of properties', for example some properties of the element 'Chart' can be organized into 'Major-grid' or 'Minor-grid'. Element properties can be defined without grouping as well. 'Simple property', such as 'Name', 'Font', 'Text-size', etc., can be defined with the corresponding keyword with value. 'Multi value property' of element, such as 'Color', 'Style', 'Padding', 'Border', etc., can have list of values (see Figure 19).

The grammar of the Report-DSL is specified in Xtext-file and shown in Application 1.

6.3. Abstract syntax

Abstract syntax is a translator of code of a report model written in the Report-DSL into the custom report model format of a software tool. The translator of the Report-DSL model into the Eclipse BIRT report model is implemented in the Xtend-file and shown in Application 2. Karsai et al. [35] advise to use the following guidelines for implementation of abstract syntax: “Align abstract and concrete syntax”, “Prefer layout which does not affect translation from concrete to abstract syntax”, “Enable modularity”.

Figure 18. Syntax template for a pseudo element of a report model

```
Element  Base value properties
{
  Graphic properties
}
{
  Content properties
}
```

Figure 19. Syntax template for properties of a report element

```
Group of properties
{
  Simple property Value
  Multi value property (Value, ..., Value)
}
```

The grammar of the Report-DSL is specified in Xtext-file and shown in Application 1.

6.3. Abstract syntax

Abstract syntax is a translator of code of a report model written in the Report-DSL into the custom report model format of a software tool. The translator of the Report-DSL model into the Eclipse BIRT report model is implemented in the Xtend-file and shown in Application 2. Karsai et al. [35] advise to use the following guidelines for implementation of abstract syntax: “Align abstract and concrete syntax”, “Prefer layout which does not affect translation from concrete to abstract syntax”, “Enable modularity”.

Figure 18. Syntax template for a pseudo element of a report model

```
Element  Base value properties
{
  Graphic properties
}
{
  Content properties
}
```

Figure 19. Syntax template for properties of a report element

```
Group of properties
{
  Simple property Value
  Multi value property (Value, ..., Value)
}
```

The grammar of the Report-DSL is specified in Xtext-file and shown in Application 1.

6.3. Abstract syntax

Abstract syntax is a translator of code of a report model written in the Report-DSL into the custom report model format of a software tool. The translator of the Report-DSL model into the Eclipse BIRT report model is implemented in the Xtend-file and shown in Application 2. Karsai et al. [35] advise to use the following guidelines for implementation of abstract syntax: “Align abstract and concrete syntax”, “Prefer layout which does not affect translation from concrete to abstract syntax”, “Enable modularity”.

Figure 18. Syntax template for a pseudo element of a report model

```
Element  Base value properties
{
  Graphic properties
}
{
  Content properties
}
```

Figure 19. Syntax template for properties of a report element

```
Group of properties
{
  Simple property Value
  Multi value property (Value, ..., Value)
}
```

The grammar of the Report-DSL is specified in Xtext-file and shown in Application 1.

6.3. Abstract syntax

Abstract syntax is a translator of code of a report model written in the Report-DSL into the custom report model format of a software tool. The translator of the Report-DSL model into the Eclipse BIRT report model is implemented in the Xtend-file and shown in Application 2. Karsai et al. [35] advise to use the following guidelines for implementation of abstract syntax: “Align abstract and concrete syntax”, “Prefer layout which does not affect translation from concrete to abstract syntax”, “Enable modularity”.

Figure 18. Syntax template for a pseudo element of a report model

```
Element  Base value properties
{
  Graphic properties
}
{
  Content properties
}
```

Figure 19. Syntax template for properties of a report element

```
Group of properties
{
  Simple property Value
  Multi value property (Value, ..., Value)
}
```

The grammar of the Report-DSL is specified in Xtext-file and shown in Application 1.

6.3. Abstract syntax

Abstract syntax is a translator of code of a report model written in the Report-DSL into the custom report model format of a software tool. The translator of the Report-DSL model into the Eclipse BIRT report model is implemented in the Xtend-file and shown in Application 2. Karsai et al. [35] advise to use the following guidelines for implementation of abstract syntax: “Align abstract and concrete syntax”, “Prefer layout which does not affect translation from concrete to abstract syntax”, “Enable modularity”.

Figure 18. Syntax template for a pseudo element of a report model

```
Element  Base value properties
{
  Graphic properties
}
{
  Content properties
}
```

Figure 19. Syntax template for properties of a report element

```
Group of properties
{
  Simple property Value
  Multi value property (Value, ..., Value)
}
```

The grammar of the Report-DSL is specified in Xtext-file and shown in Application 1.

6.3. Abstract syntax

Abstract syntax is a translator of code of a report model written in the Report-DSL into the custom report model format of a software tool. The translator of the Report-DSL model into the Eclipse BIRT report model is implemented in the Xtend-file and shown in Application 2. Karsai et al. [35] advise to use the following guidelines for implementation of abstract syntax: “Align abstract and concrete syntax”, “Prefer layout which does not affect translation from concrete to abstract syntax”, “Enable modularity”.

Figure 18. Syntax template for a pseudo element of a report model

```
Element  Base value properties
{
  Graphic properties
}
{
  Content properties
}
```

Figure 19. Syntax template for properties of a report element

```
Group of properties
{
  Simple property Value
  Multi value property (Value, ..., Value)
}
```

The grammar of the Report-DSL is specified in Xtext-file and shown in Application 1.

6.3. Abstract syntax

Abstract syntax is a translator of code of a report model written in the Report-DSL into the custom report model format of a software tool. The translator of the Report-DSL model into the Eclipse BIRT report model is implemented in the Xtend-file and shown in Application 2. Karsai et al. [35] advise to use the following guidelines for implementation of abstract syntax: “Align abstract and concrete syntax”, “Prefer layout which does not affect translation from concrete to abstract syntax”, “Enable modularity”.

Figure 18. Syntax template for a pseudo element of a report model

```
Element  Base value properties
{
  Graphic properties
}
{
  Content properties
}
```

Figure 19. Syntax template for properties of a report element

```
Group of properties
{
  Simple property Value
  Multi value property (Value, ..., Value)
}
```

The grammar of the Report-DSL is specified in Xtext-file and shown in Application 1.
7. Comparison of Report-DSL and a visual editor

In this section, aspects such as compatibility of models, report creation process and usability in Report-DSL in comparison with a visual editor will be discussed based on an example. The report model “Sales report” was created as an example of the application of the Report-DSL. The sample database of Eclipse – BIRT Project (www.eclipse.org/birt/documentation/sample-database.php) was used as data source for the report. Calendar year was used as input parameter. Four data sets were defined. Data from them were used in corresponding line, pie or bar charts, which were arranged in the grid of a report body.

The report model, created by using the Report-DSL, is shown in Application 3 and a fragment of this model – in Figure 20.

Figure 20. The model fragment of a report created by using the Report-DSL (left) and its results in report viewer of Eclipse–BIRT reporting tool (right)

7.1. Models compatibility

The developed specification of the Report-DSL and the generator into a custom software format allows solving the problem of incompatibility of the report models from the different software tools and reducing the dependence on the concrete reporting tool. The report model created by using the Report-DSL can be translated into different custom
report models of corresponding reporting tools if a generator into this reporting tool model format is available. Generators for that case can be developed in Xtext framework as it was implemented for Eclipse–BIRT reporting tool in Application 2.

7.2. Report creation process

The process of the report model creating can be considered as the communication between participants of process (see Figure 20): analyst (A), graphic designer (D) and data-source specialist (P).

I need a bar chart with dynamic of sales by month

I have defined color palette: rgb(86 207 255), rgb(182 228 255), rgb(97 184 141), rgb(126 218 173), rgb(213 149 148), rgb(249 166 132)

Ok. I make a dataset and create the model in visual editor as you have said.

I need a bar chart with dynamic of Sales By Months

I have defined color-palette: rgb(86 207 255), rgb(182 228 255), rgb(97 184 141), rgb(126 218 173), rgb(213 149 148), rgb(249 166 132)

Ok. I make a dataset and create the model with Report-DSL as you have written.

chart SalesByMonths bar side-by-side
dataset SalesByMonths
{
  legend {}
  x-axe text {SalesByMonths.Month} {}
  y-axe linear {SalesByMonths.Revenue} {}
  color-palette {
    rgb(86 207 255)
    rgb(182 228 255)
    rgb(97 184 141)
    rgb(126 218 173)
    rgb(213 149 148)
    rgb(249 166 132)
  }
}

Figure 21. Communication process while creating report model in visual editor (a) and Report-DSL (b)

Each participant performs his own part: analyst specifies content of a report, data-source specialist specifies data storage elements and graphic designer specifies styles of report elements. If a report model is created in a visual editor, then usually only one of the above mentioned participants uses a reporting tool. It is the report designer who implements requirements from other participants while creating a report model. It takes a while to transform these informal requirements to the custom report model format. If a
A report model is created in a textual editor by using the Report-DSL then those informal requirements in textual form can be used directly in a report model. Terms and definitions specified in Report-DSL have common language forms (like words of everyday language) that are clear to all participants and not only to the report designer. As the result, the communication process improves, and that may allow increasing the speed of the report creation process.

### 7.3. Usability

The Report-DSL can be used as an alternative way for the report model creation that has different usability in defining of report model elements and their properties. A report designer doesn’t need to open a lot of overlapping windows for defining properties of an element any more. They can be written simply as list of pairs "property-value", because a report model has textual form. This application may increase the speed of the report creation process. The approaches of creating an element “Chart” and their properties such as “Data-set”, “Y-axis”, “X-axis”, “Legend” and “Color-palette” in the visual editor and in Report-DSL are represented in Figures 22-25.

```plaintext
chart SalesByMonths line overlay
dataset SalesByMonths
{
  legend {}
  x-axe text {SalesByMonths.Month} {}
  y-axe linear {SalesByMonths.Revenue} {}
  color-palette {}
}
```

Figure 22. Defining a type and a data-set connection for chart in visual report (a, b) and Report-DSL (c)
Figure 23. Defining axes for chart in visual report (a, b) and Report-DSL (c, d)
By using Report-DSL it is easier to reuse properties of report elements, because the property can be defined separately from its element. In the Report-DSL the special syntax element "Element-properties" was specified, in which any property can be defined for further using. For using such property only, the reference to it in the corresponding element needs defining. In the example, shown in Figure 25 (b) the property "Color-palette" of the element "Chart" with the name "SixColorPalette" was defined. Further, this property is used in four different charts of the report. If colors of the palette need changing, then it is enough to change this property only in block "Element-properties". All color palettes in the charts already have the reference to it. In the visual editor on the other hand, in order to change one color of palette it is necessary to open three overlapping windows (make four mouse clicks), set the color code and save the new result (see Figure 25 (a)). A report designer needs to make about one hundred mouse clicks for changing the palette of six colors in four charts. This way, the speed of creation of a report in Report-DSL and further support may be easier and take less time.
element properties
{
  color-palette SixColorPalette
  {
    rgb(86 207 255)
    rgb(182 228 255)
    rgb(97 184 141)
    rgb(126 218 173)
    rgb(213 149 148)
    rgb(249 166 132)
    rgb(197 197 197)
  }
}

chart SalesByMonths line overlay
{
  color-palette { SixColorPalette }
}

chart SalesByProductLines line overlay
{
  color-palette { SixColorPalette }
}

chart SalesByTerritories pie
{
  color-palette { SixColorPalette }
}

chart SalesByOffices bar side-by-side
{
  color-palette { SixColorPalette }
}

Figure 25. Defining a color palette for charts in visual report (a) and Report-DSL (b) with reusing of report element

Table 7 shows the average numbers of lines of code of corresponding elements from the report model in Application 3. Elements of a report model, written in the Report-DSL, have up to 55% fewer rows in average for their defining. The number of rows of the model is equal to approximately 11% of the number of rows of the model in XML format for this report.

Table 7. Number of lines of code of report elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Eclipse–BIRT model (XML format)</th>
<th>Report-DSL model (textual format)</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>70</td>
<td>20</td>
<td>29%</td>
</tr>
<tr>
<td>Parameter</td>
<td>32</td>
<td>8</td>
<td>25%</td>
</tr>
<tr>
<td>Label</td>
<td>7</td>
<td>6</td>
<td>86%</td>
</tr>
<tr>
<td>Chart</td>
<td>860</td>
<td>70</td>
<td>8%</td>
</tr>
<tr>
<td>Grid</td>
<td>19</td>
<td>14</td>
<td>74%</td>
</tr>
<tr>
<td>Total report</td>
<td>3542</td>
<td>400</td>
<td>11%</td>
</tr>
</tbody>
</table>
8. Conclusion

In this master thesis research on business report creation methods was made and an alternative way for creation a report model was offered. The analysis of the reporting software has shown that all of these tools have similar functionality for report creation. Also they have similar sets of report elements and report model creating processes in different tools look alike. Each of these tools has its own report model format and none of the report models are compatible with any other tool. Problems in the domain of reports creation were identified, such as report model incompatibility, dependence on software, constrained usability of the visual editor and the excessive frequency of code of report model elements in xml-format. In order to solve these problems, the domain-specific language was offered as an alternative way for the report model creation. Such language Report-DSL was implemented by means of Xtext, an open-source software framework for developing domain-specific languages. Concrete syntax of language was specified for often used report elements and the generator to Eclipse-BIRT model format was implemented. The report model in Report-DSL was created as an example of its application. Compared to a visual editor, the using of Report-DSL may improve communication process between report developers, increase the speed of the report creation process by using the textual representation for a report model and by reusing element properties. Also, the number of lines of code of a report model, written in Report-DSL, has considerably decreased in comparison with XML model format.

The usage of the Report-DSL has not only advantages, but also some limitations and drawbacks. The main disadvantages of the Report-DSL are that the specification of a report model and generator require additional effort and support, and embedding in or supporting of other languages (SQL, JavaScript, etc.) in the model specification can be complex. The Report-DSL presented in this master thesis thus is at the moment mainly applicable for either simple report models or report model drafts.

In Table 8 main differences of the Report-DSL from visual editors are presented.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Visual editor</th>
<th>Report-DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatible model</td>
<td>no</td>
<td>possible</td>
</tr>
<tr>
<td>Additional abstraction of model</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Independence on model editor</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>User-friendly textual format</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Elements properties reusing</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Reducing of code duplicating</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Additional support of model specification and generator</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Embedding and support of other languages</td>
<td>not required</td>
<td>problematic</td>
</tr>
<tr>
<td>Full integration in report development process</td>
<td>not required</td>
<td>problematic</td>
</tr>
</tbody>
</table>

The compatibility of the report model can be supported in the Report-DSL, but at the same time additional abstraction provided in the report model is higher than in the visual editors. This could lead to extra time and costs required for learning of the Report-DSL. A report creator is independent from a reporting tool, when he or she creates a report model.
in the Report-DSL. A model of a report in the Report-DSL has a more user-friendly format than the XML-format used in reporting tools. Duplication of code of a model in the Report-DSL is reduced by reusing element properties. A report model specification as well as a generator need additional support in case of changing of the custom report format in reporting tools. Also the embedding of other languages, which are used in report model development, can be problematic due to additional implementations required for these specific languages that may complicate the Report-DSL. The full integration of the Report-DSL in a report development process may be problematic, because other components (e.g. the report viewer) of this process will need changing. Basically the report development process includes two main steps: design of a report model and deployment of this model. The Report-DSL can be partly integrated into a model design process, because it is a model creation method. But the model in the Report-DSL format cannot be used on the deployment step yet because components that use this model are able to read only the software-specific formats, but not the Report-DSL model format. That is, those other tools would need to be changed to integrate the Report-DSL.

Further work on research concerning the report creation domain and developing the Report-DSL may include aspects, which were not considered in this thesis, namely:

- embedded support of SQL in data storage elements, as well as dialects of SQL;
- embedded support of script languages, that are used for defining element behaviour;
- full use of textual editor features such as color assistants, code builders, etc.;
- close integration with reporting tool for continuous report development;
- cross-generation from custom model report format to the Report-DSL format.
9. Bibliography


[28] "One Business Intelligence Platform fpr All your BI Needs," [Online]. Available:


Application 1. Report-DSL grammar : an EBNF-specification

```
grammar org.xtext.example.mydsl.Rdsl with org.eclipse.xtext.common.Terminals
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

generate rDsl "http://www.xtext.org/example/mydsl/RDsl"
```

Root: Report
  root = Report ;

Report:
  'report' name = ID
  '{' ( datasources = Datasources?
  & datasets = Datasets?
  & parameters = Parameters?
  & styles = Styles?
  & pagesetup = PageSetup?
  & body = Body?
  & elementProperties = elementProperties?
  )
  '}'
elementProperties:
'element properties' '{'
properties += (Font
| FontSize
| ColorPalette
| HAlignment
| VAlignment
| Whitespace
| StyleRef
| Display
| Border
| Padding
| Margin
| Visibility
| PageBreak
| Bookmark
| Width
| Height
| AltText
)+
'}'
;

Datasources:
{Datasources} 'data-sources' '{' datasource += Datasource* '}'
;

Datasource:
'data-source' name = ID 'type' type = (DatasourceSample | DatasourceJDBC | DatasourceCSV)
;

DatasourceSample:
'type' = 'sample'
;

DatasourceJDBC:

'jdbc' '{
'driver' driver = STRING &
'url' url = STRING &
'user' user = STRING &
'password' password = STRING
}'
;

DatasourceCSV: //TODO Generator statement
'csv' '{
'url' url = STRING &
'charset' charset = STRING &
'delimiter' delimiter = STRING & //TODO String+
'headerline' headerline = (BooleanLiteral)
}'
;

Datasets:
'data-sets' '{ dataset += Dataset* '}'
;

Dataset:
'dataset' name = ID '{
'datasource' datasource = [Datasource | QualifiedName] &
('fields' '{' field+=DatasetField (',' field+=DatasetField)* '}'
)
`parameters` '{' parameter+=[Parameter] (',', parameter+=[Parameter])* '}'?
&
// (`computed fields` '{' computedfield += ComputedDatasetField* '}')?
`query` '{' query = STRING? '}' // #TODO SelectStatement
);

DatasetField:
  SimpleDatasetField | ComputedDatasetField
;

SimpleDatasetField:
  name=ID
;

ComputedDatasetField:
  name=ID aggregation=AggregatorFunction datatype = DataType argument = JSExpression (`filter` filter = JSExpression)?
;

Parameters:
  `parameters` '{' parameter += Parameter* '}'
;

StaticParameterValues:
  `value` += StaticParameterValue* (`sort by` sort = StaticParameterSort sorttype = SortType)?
;

StaticParameterValue:
  (label = STRING ':', value = STRING (default ?= 'default'))?
DynamicParameterValues:
'{" value += DynamicParameterValue '}
;

DynamicParameterValue:
'dataset' dataset = [Dataset]
(label = JSExpression ':' value = JSExpression)
('default' default += JSExpression*)? &
('sort by' sort = [DatasetField | QualifiedName] sorttype = SortType)?
;

Parameter:
controltype = ParameterControlType name = ID
'{
  ('promt-text' promt = STRING)?
  & ('help-text' help = STRING)?
  & ('hidden' hide = BooleanLiteral)?
  & ('required' required = BooleanLiteral)?
  & ('duplicate' duplicate = BooleanLiteral)?
  & ('values' valuetype = ParameterValueType
    paramtype = ParameterType
    datatype = DataType
    values = ParameterValue
  )
}'};
ParameterValue:
  (DynamicParameterValues | StaticParameterValues);

Styles:
  'styles' '{ style += Style* }';

PageSetup:
  'page-setup' '{ properties += PageProperties }'?

Body:
  (Body) 'body' '{ elements += ReportElements* }'

ReportElements:
Label | Text | DynamicText | xData | Image | Chart | List | Grid | Table | CrossTable

```
Label
   'label' name = ID
   content = STRING
   (('{' properties += LabelProperties ',')')?

Text
   'text' name = ID
   content = STRING // #TODO HTML
   (('{' properties += LabelProperties '}')')?

DynamicText
   'dynamic-text' name = ID
   content = JSExpression
   (('{' properties += LabelProperties '}')')?

xData
   'data' name = ID
   content = (JSExpression)
   (('{' properties += LabelProperties '}')')?

Image
   'image' name = ID
   'type' type = ('file' | 'url')
   'content' content = JSExpression
   (('{' properties += LabelProperties '}')')?
```
Chart:

'chart' name = ID
(type = (ChartWithAxesType) subtype = ChartSubType | type =
ChartWithoutAxesType)
'dataset' dataset = [ Dataset | QualifiedName ]
('' properties += LabelProperties
( title = Title   ( legend = Legend )? 
( plot = Plot)?
(axe = NoAxe | (xaxe = XAxe & 
yaxe = YAxe))
( colorpalette = ColorPalette)?
''})?
;

Legend:

'legend' place = Place orientation = OrientationType
(''
(title = Title)? &
(visibility = LegendVisibility)? &
('direction')? &
{'show value'}?
''
;
Plot: 'plot' '{' '}'

XAxe: 'x-' value=Axe

YAxe: 'y-' value=Axe

Axe: 'axe' type = ('linear' | 'text' | 'logarithmic' | 'date-time') data = JSExpression ('grouping by' groupingby = JSExpression)? '{' ( (title = Title)? & (line = Line)? & ('major-' majorgrid = ChartGrid)? & ('minor-' minorgrid = ChartGrid)? )'}'

Line: 'line' '{' value=BorderLine '}'

ChartGrid: 'grid' '{' ( ('scale' scale = INT) & value=BorderLine & tick=Tick )'}'
Tick:  
'tick' place=('above' | 'across' | 'below') ("{ value=BorderLine }")?

ColorPalette:  
'color-palette' (name=ID)? '{ (value += (StringColor | HexColor | RgbColor)+) | (linkto={ColorPalette}) }'

Place:  
(position = ('left' | 'right') anchor = ('top' | 'middle' | 'bottom'))  
| (position = ('above' | 'below') anchor = ('left' | 'middle' | 'right'))

Title:  
'title' value = STRING place=('left' | 'right' | 'above' | 'below')  
("{ properties = ChartElementProperties }")?
ChartElementProperties:
  ( padding = Padding?
  & border = Border?
  & visibility = Visibility?
  & font = Font?
  & fontSize = FontSize?
  & color = Color?
  & backgroundColor = BackgroundColor?
  & textStyle = TextStyle?
  & hAlign = HAlignment?
  & vAlign = VAlignment?
  & whiteSpace = Whitespace?
  & style = StyleRef?
  & display = Display?)
;
enum ChartWithAxesType returns STRING:
    bar | line | area
;

diagram diagram

enum ChartWithoutAxesType returns STRING:
pie
;

diagram diagram

enum ChartSubType:
sbs = 'side-by-side' | stacked | percent = 'percent-stacked' | overlay
;

diagram diagram

enum PositionType:
    North = 'top' | South = 'bottom' | West = 'left' | East = 'right'
;

diagram diagram

enum OrientationType:
    vertical | horizontal
;

diagram diagram
List:
'list' name = ID
('(\ properties += LabelProperties ')')?
'(\ ('header' '{\ headerelements += ReportElements* '}')')?
'(\ ('detail' '{\ detailelements += ReportElements* '}')')?
'(\ ('footer' '{\ footerelements += ReportElements* '}')')?
''
;

Grid:
'grid' name = ID
('(\ properties += LabelProperties ')')?
'(\ ('columns += Column+ rows += Row+ ')')?
''
;

Table:
'table' name = ID
('(\ properties += LabelProperties ')')?
'(\ ('dataset' dataset = [ Dataset | QualifiedName ]))?
'(\ ('columns += Column+ ')
  ('header' '{\ headerrows += Row+ '}')')?
  ('detail' '{\ detailrows += Row+ '}')?
  ('footer' '{\ footerrows += Row+ '}')?
  ')''
;

CrossTable:
'cross-table' name = ID
('(\ properties += LabelProperties ')')?
''
Column:  
{Column} 'column' (order = INT)?  
{"' properties += LabelProperties '})?  
';

Row:  
'row' (order = INT)?  
{"' properties += LabelProperties '})?  
'[' cells += Cell+ ']'  
';

Cell:  
{Cell} 'cell' (order = INT)?  
{"' properties += LabelProperties '})?  
'[' elements += ReportElements* ']'  
';

PageProperties  
: font = Font?  
& fontSize = FontSize?  
& color = Color?  
& backgroundColor = BackgroundColor?  
& textStyle = TextStyle?  
& hAlign = HAlignment?  
& vAlign = VAlignment?  
& whiteSpace = Whitespace?  
& style = StyleRef?  
& display = Display?  
& border = Border?  
& padding = Padding?  
& margin = Margin?  
& visibility = Visibility?  
& pagebreak = PageBreak?  
& toc = Toc?  
& bookmark = Bookmark?  
& width = Width?  
& height = Height?  
';

LabelProperties  
: (font = Font?  
& fontSize = FontSize?  
& color = Color?  
& backgroundColor = BackgroundColor?  
& textStyle = TextStyle?  
& hAlign = HAlignment?  
& vAlign = VAlignment?
'color' (StringColor | HexColor)
;
BackgroundColor → bg-color

BackgroundColor returns Color:
'bg-color' (StringColor | HexColor)
;

StringColor → ColorType

StringColor:
value = ColorType
;

HexColor → HEX_COLOR

HexColor:
value = HEX_COLOR
;

RgbColor → rgb(INT INT INT)

RgbColor:
'rgb( r = INT g = INT b = INT )'
;

TextStyle

TextStyle:
(TextStyle) 'text-style' (name=ID)? (((I ?= 'italic'? & B ?= 'bold'? & U ?= 'underline'? & T ?= 'through-line')? ) )
;

HAlignment

HAlignment → h-alignment

HAlignment:
'h-alignment' (name=ID)? (value = HalignmentType | (linkto=[HAlignment]))
BorderSide:
('right' '{' right = BorderLine '}')? & ('left' '{' left = BorderLine '}')?
& ('top' '{' top = BorderLine '}')? & ('bottom' '{' bottom = BorderLine '}')?
;

BorderLine:
'style' style=BorderLineStyleType & 'width' width = BorderWidth & color = Color
;

BorderWidth:
value = (WidthTypeValue | SizeValue)
;

WidthTypeValue:
value = WidthType
;

Padding:
'padding' '{' (IndentAll | IndentSide) '}'
;

Margin returns Padding:
'margin' '{' (IndentAll | IndentSide) '}'
;

IndentAll:
'all' IndentValue
;
IndentSide:
('right' right=IndentValue)? & ('left' left=IndentValue)? & ('top' top=IndentValue)? & ('bottom' bottom=IndentValue)?
;

IndentValue:
value = (SizeValue | SizeValuePercent)
;

PageBreak:
'page-break' '{
    ('before' before=PageBreakValue)? & ('after' after=PageBreakValue)? & ('inside' inaide=PageBreakValue)? & ('interval' interval=IntLiteral)? & ('repeat-header' repeat=BooleanLiteral)?
} '}'
;
Visibility:
'visibility' (show ?= 'true' | 'hide for' (HideAll | HideForOne | HideForMore))

LegendVisibility:
'visibility' value = ('true'|'hide')

HideAll:
'all' (expression = JSExpression)?

HideForOne:
'{' value += HideForValue+ '}'

HideForValue returns HideAll:
format = OutputFormatType (expression = JSExpression)?

HideForMore:
'(' format += OutputFormatType+ ')'(expression = JSExpression)?

JSExpression:
'{'(value = Literal) '}' //TODO Builder Expression
enum SizeUnit:
  M = 'medium'
  L = 'large'  | Lr = 'larger'  | XL = 'x-large'  | XXL = 'xx-large'
  S = 'small'  | Sr = 'smaller' | XS = 'x-small' | XXS = 'xx-small';
enum LengthUnit: cm | ems | exs | in | mm | pc | pt | px;
enum HalignmentType: left | center | right | justify;
enum ValignmentType: bottom | middle | top;
enum WhitespaceType:
    nowrap = 'no-wrapping' | normal = 'normal' | pre = 'preformatted';

denum DisplayType: block = 'block' | inline = 'inline' | none = 'no';

denum BorderLineStyleType: solid | dotted | dashed | double;

denum WidthType: thin | medium | thick;

denum OutputFormatType: xlsx | postscript | docx | pptx | pdf | xls | ppt | doc | html | odp | odt | ods;

denum PageBreakType:
    pageBreakAfter = 'after' | pageBreakBefore = 'before' | pageBreakInside = 'inside';

denum PageBreakValue: auto | always | avoid;

denum SortType: asc = 'asc' | desc = 'desc';

denum BorderType:
    R = 'right' | L = 'left' | T = 'top' | B = 'bottom';

denum TargetType:
    Blank = 'blank' | Parent = 'parent' | Self = 'self' | Top = 'top';

denum DrillThroughType:
    NewWindow = 'new-window' | SameFrame = 'same-frame' | ParentFrame = 'parent-frame' | WholePage = 'whole-page';

denum ColorType:
    aqua | blue | black | fuchsia | gray | green | lime | maroon | navy | olive | orange | purple | red | silver | teal | yellow | white;

denum ParameterControlType:
    chb = 'check-box' | tb = 'text-box' | lb = 'list-box' | rb = 'radio-button' | cb = 'combo-box';

denum ParameterValueType: dynamic | static;

denum ParameterType: multi | simple;

denum DataType:
    boolean | date | datetime | decimal | float | integer | string | time;

denum StaticParameterSort: label | value;

denum AggregatorFunction: count | sum | max | min;

/** *************************************
 * T E R M I N A L S
 ** *************************************
 */
terminal ID: '^'?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;

terminal INT returns ecore::EInt: ('0'..'9')*;

 terminal STRING:
    "" { "" . /* 'b'|'t'|'n'|'f'|'r'|'u'"" | "" | "" /* 'b'|'t'|'n'|'f'|'r'|'u'"" | "" */ | !("\"" | "") |
\n\n```
"*/" ( '"' . / "b' | 't' | 'n' | 'f' | 'r' | 'u' | 'm' | 'n' | 't' +/ | !('"' | "") ) */
```

\n\n```
terminal ML_COMMENT: '/\*' -> '*/';
```

```
terminal SL_COMMENT: '//' !('"' | 'r')* ('r' ? 'n')?;
```

```
terminal WS: ( ' ' | 't' | 'r' | 'n')+;
```

```
terminal ANY_OTHER: .;
```

```
terminal FLOAT: ('0'..'9')+'.'('0'..'9')*;
```

```
terminal HEX_COLOR: "#" ('0'..'9'|'a'|'f'|'A'|'F') (0.'9'|'a'|'f'|'A'|'F') ('0'..'9'|'a'|'f'|'A'|'F') ('0'..'9'|'a'|'f'|'A'|'F') ('0'..'9'|'a'|'f'|'A'|'F');
```

```
Literal : FloatLiteral
    | IntLiteral
    | StringLiteral
    | BooleanLiteral
    | NullLiteral
    | ({$ID} value = [DatasetField | QualifiedName]);
```

IntLiteral: value=INT;

FloatLiteral: value=FLOAT;

StringLiteral: value=STRING;

BooleanLiteral: value= ('true' | 'false');

NullLiteral: (NullLiteral) 'null';

```
Application 2. Report-DSL generator: a translator to custom report format (Eclipse BIRT Project)

/*
* generated by Xtext
*/
package org.xtext.example.mydsl.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IFileSystemAccess
import org.eclipse.xtext.generator.IGenerator
import org.xtext.example.mydsl.rDsl.Report
import org.xtext.example.mydsl.rDsl.Datasource
import org.xtext.example.mydsl.rDsl.Dataset
import org.xtext.example.mydsl.rDsl.ReportElements
import org.xtext.example.mydsl.rDsl.Grid
import org.xtext.example.mydsl.rDsl.Label
import org.xtext.example.mydsl.rDsl.Column
import org.xtext.example.mydsl.rDsl.Row
import org.xtext.example.mydsl.rDsl.Cell
import org.xtext.example.mydsl.rDsl.Text
import org.xtext.example.mydsl.rDsl.DynamicText
import org.xtext.example.mydsl.rDsl.Image
import org.xtext.example.mydsl.rDsl.Chart
import org.xtext.example.mydsl.rDsl.List
import org.xtext.example.mydsl.rDsl.xData
import org.xtext.example.mydsl.rDsl.Table
import org.xtext.example.mydsl.rDsl.CrossTable
import org.xtext.example.mydsl.rDsl.LabelProperties
import org.xtext.example.mydsl.rDsl.SizeValue
import org.xtext.example.mydsl.rDsl.SizeValuePercent
import org.xtext.example.mydsl.rDsl.SizeValueUnit
import org.xtext.example.mydsl.rDsl.HexColor
import org.xtext.example.mydsl.rDsl.StringColor
import org.xtext.example.mydsl.rDsl.BorderAll
import org.xtext.example.mydsl.rDsl.BorderSide
import org.xtext.example.mydsl.rDsl.BorderLine
import org.xtext.example.mydsl.rDsl.Font
import org.xtext.example.mydsl.rDsl.FontStyle
import org.xtext.example.mydsl.rDsl.TextStyle
import org.xtext.example.mydsl.rDsl.BorderStyle
import org.xtext.example.mydsl.rDsl.WidthTypeValue
import org.xtext.example.mydsl.rDsl.Padding
import org.xtext.example.mydsl.rDsl.IndentValue
import org.xtext.example.mydsl.rDsl.IndentAll
import org.xtext.example.mydsl.rDsl.IndentSide
import org.xtext.example.mydsl.rDsl.Visibility
import org.xtext.example.mydsl.rDsl.HideAll
import org.xtext.example.mydsl.rDsl.HideForOne
import org.xtext.example.mydsl.rDsl.JSExpression
import org.xtext.example.mydsl.rDsl.HideForMore
import org.xtext.example.mydsl.rDsl.Display
import org.xtext.example.mydsl.rDsl.PageBreak
import org.xtext.example.mydsl.rDsl.Toc
import org.xtext.example.mydsl.rDsl.Bookmark
import org.xtext.example.mydsl.rDsl.Style
import org.xtext.example.mydsl.rDsl.PageSetup
import org.xtext.example.mydsl.rDsl.PageProperties
import org.xtext.example.mydsl.rDsl.StyleRef
import org.xtext.example.mydsl.rDsl.Width
import org.xtext.example.mydsl.rDsl.Height
import org.xtext.example.mydsl.rDsl.AltText
import org.xtext.example.mydsl.rDsl.StringLiteral
import org.xtext.example.mydsl.rDsl.DatasourceSample
import org.xtext.example.mydsl.rDsl.DatasourceJDBC
import org.xtext.example.mydsl.rDsl.Parameter
import org.xtext.example.mydsl.rDsl.DatasetField
import org.xtext.example.mydsl.rDsl.BooleanLiteral
import org.xtext.example.mydsl.rDsl.StaticParameterValues
import org.xtext.example.mydsl.rDsl.DynamicParameterValues
import org.xtext.example.mydsl.rDsl.DynamicParameterValue
import org.xtext.example.mydsl.rDsl.StaticParameterValue
import org.xtext.example.mydsl.rDsl.ComputedDatasetField
import org.xtext.example.mydsl.rDsl.ColorPalette
import org.xtext.example.mydsl.rDsl.RgbColor
import org.xtext.example.mydsl.rDsl.Title
import org.xtext.example.mydsl.rDsl.Axe
import org.xtext.example.mydsl.rDsl.ChartGrid
import org.xtext.example.mydsl.rDsl.ChartElementProperties
import org.xtext.example.mydsl.rDsl.Legend
import org.xtext.example.mydsl.rDsl.YAxe
import org.xtext.example.mydsl.rDsl.HAlignment
import org.xtext.example.mydsl.rDsl.VAlignment
import org.xtext.example.mydsl.rDsl.Whitespace
import org.xtext.example.mydsl.rDsl.LegendVisibility

/**
 * Generates code from your model files on save.
 * see http://www.eclipse.org/Xtext/documentation.html#TutorialCodeGeneration
 */
class RDslGenerator implements IGenerator {

// @Inject extension IQualifiedNameProvider

override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    fsa.generateFile(e.name+
        "_gen.rptdesign".
        e.compile.toString.replaceAll("\r\n", "").replaceAll("\t", "")
//.toString.replaceAll()
}

int A = 0 // Variable for id's count
int P = 0 // Variable for dataset parameters count

def compile(Report r) {

    "<?xml version="1.0" encoding="UTF-8"?>
    <report xmlns="http://www.eclipse.org/birt/2005/design" version="3.2.23"
     id="A=A+1x">rDSL-User</property>
     <property name="createdBy"></property>
     <property name="units">in</property>
     <property name="layoutPreference">fixed layout</property>
     «IF r.datasources != null»
     <data-sources>
     «FOR ds : r.datasources.datasource»
     "ds.compile"
     «ENDFOR»
     </data-sources>
     «ENDIF»
     «IF r.datasets != null»
     <data-sets>

```
«FOR dt : r.datasets.dataset»
«dt.compile»
«ENDFOR»
</data-sets>
«ENDIF»
«IF r.parameters != null»
<parameters>
 «FOR p : r.parameters.parameter»
 «p.compile»
 «ENDFOR»
</parameters>
«ENDIF»
«IF r.styles != null»
<styles>
 «FOR s : r.styles.style»
 «s.compile»
 «ENDFOR»
</styles>
«ENDIF»
«IF r.pagesetup != null»
<page-setup>
 «(r.pagesetup).compile»
</page-setup>
«ENDIF»
«IF r.body != null»
<body>
 «FOR b : r.body.elements»
 «b.compile»
 «ENDFOR»
</body>
«ENDIF»
</report>

/*****************************/
/* DATA ELEMENTS */
/*****************************/

def compile(Datasource d)

«IF d.type instanceof DatasourceSample»
 <oda-data-source extensionID="org.eclipse.birt.report.data.oda.sampledb" name="xd.name" id="A=A+1"/>
 «ENDIF»
«IF d.type instanceof DatasourceJDBC»
 <oda-data-source extensionID="org.eclipse.birt.report.data.oda.jdbc" name="xd.name" id="A=A+1"/>
 <list-property name="privateDriverProperties">
 <ex-property>
 <name>metadataBidiFormatStr</name>
 <value>ILYNN</value>
 </ex-property>
 <ex-property>
 <name>disabledMetadataBidiFormatStr</name>
 </ex-property>
 <ex-property>
 <name>contentBidiFormatStr</name>
 <value>ILYNN</value>
 </ex-property>
 <ex-property>
 <name>disabledContentBidiFormatStr</name>
 </ex-property>
</list-property>
```
<property name="odaDriverClass">ʻ(d.type as DatasourceJDBC).driverʻ</property>
<property name="odaURL">ʻ(d.type as DatasourceJDBC).urlʻ</property>
<property name="odaUser">ʻ(d.type as DatasourceJDBC).userʻ</property>
<encrypted-property name="odaPassword" encryptionID="base64">ʻ(d.type as DatasourceJDBC).passwordʻ</encrypted-property>
</oda-data-source>

```python
def compile(Dataset d):
    ...
    oda-data-set
    extensionID="org.eclipse.birt.report.data.oda.jdbc.JdbcSelectDataSet"
    name="ʻd.nameʻ" id="ʻA=A+1ʻ">
    <list-property name="parameters"/>
    <list-property name="parameters">
        ʻFOR p : d.parameterʻ
        ʻP=P+1ʻ
        <structure>
            <property name="name">ʻPʻ</property>
            <property name="paramName">ʻp.nameʻ</property>
            <property name="nativeName"/>
            <property name="dataType">integerʻ</property>
            <property name="nativeDataType">4ʻ</property>
            <property name="position">ʻPʻ</property>
            <property name="isOptional">trueʻ</property>
            <property name="allowNull">trueʻ</property>
            <property name="isInput">trueʻ</property>
            <property name="isOutput">falseʻ</property>
        </structure>
        ʻENDFORʻ
    </list-property>
    ʻP=ʻ
    <property name="dataSource">ʻd.datasourceeref.nameʻ</property>
    <list-property name="computedColumns">
        ʻIF c instanceof ComputedDatasetFieldʻ
        ʻc.compileʻ
        ʻENDIFʻ
        ʻENDFORʻ
    </list-property>
    <xml-property name="queryText"><![CDATA[ʻd.query.toStringʻ]]></xml-property>
</oda-data-set>

```
def compile(DynamicParameterValues p)

   «(v as DynamicParameterValue).compile»
   «ENDFOR»
```python
def compile(StaticParameterValues p):
    #<simple-property-list name="defaultValue">
    <FOR v : p.value>
        <IF v.default>
            <value type="javascript">(v as StaticParameterValue).value</value>
        </IFDEF>
    </ENDFOR>
    </simple-property-list>
    <property name="sortBy">p.sort</property>
    <property name="sortDirection">p.sorttype</property>
    <list-property name="selectionList">
        <FOR v : p.value>
            (v as StaticParameterValue).compile
        </ENDFOR>
    </list-property>
    
def compile(DynamicParameterValue p):
    #<property name="dataSetName">p.dataset.name</property>
    <expression name="valueExpr" type="javascript">dataSetRow["p.value.compile.toString.trim"]</expression>
    <expression name="labelExpr" type="javascript">dataSetRow["p.label.compile.toString.trim"]</expression>
    <expression name="sortByColumn">dataSetRow["(p.sort.compile).toString.trim"]</expression>
    <property name="sortDirection">p.sorttype</property>
    #<simple-property-list name="defaultValue">
    <FOR d : p.default>
        <value type="javascript">d.compile</value>
    </ENDFOR>
    </simple-property-list>
    
def compile(StaticParameterValue p):
    #<structure>
    <property name="value">p.value</property>
    <property name="label">p.label</property>
    </structure>
    
    /***************************************************************
    * END OF PARAMETERS
    ***************************************************************/
    
    /***************************************************************
    * END OF DATA ELEMENTS
    ***************************************************************/
    
    def compile(ReportElements e):
    <IF e instanceof Grid>e.compile<ENDIF>
    <IF e instanceof Label>e.compile<ENDIF>
    <IF e instanceof Text>e.compile<ENDIF>
    <IF e instanceof DynamicText>e.compile<ENDIF>
    <IF e instanceof xData>e.compile<ENDIF>
    <IF e instanceof Image>e.compile<ENDIF>
    <IF e instanceof Chart>e.compile<ENDIF>
    <IF e instanceof List>e.compile<ENDIF>
    <IF e instanceof Table>e.compile<ENDIF>
    <IF e instanceof CrossTable>e.compile<ENDIF>
    
    /***************************************************************
    * REPORT ELEMENTS
    ```
def compile(Grid g)
    '''
    <grid name="g.name" id="A=A+1">
        <FOR p : g.properties>
            <p.compile>
        </FOR>
        <FOR c : g.columns>
            <c.compile>
        </FOR>
        <FOR r : g.rows>
            <r.compile>
        </FOR>
    </grid>
    '''
def compile(Column c)
    '''
    <column id="A=A+1">
        <FOR p : c.properties>
            <p.compile>
        </FOR>
    </column>
    '''
def compile(Row r)
    '''
    <row id="A=A+1">
        <FOR p : r.properties>
            <p.compile>
        </FOR>
        <FOR c : r.cells>
            <c.compile>
        </FOR>
    </row>
    '''
def compile(Cell c)
    '''
    <cell id="A=A+1">
        <FOR p : c.properties>
            <p.compile>
        </FOR>
        <FOR e : c.elements>
            <e.compile>
        </FOR>
    </cell>
    '''
def compile(Table t)
    '''
    <table name="t.name">
        <FOR p : t.properties>
            <p.compile>
        </FOR>
        <IF t.dataset != null>
            <t.dataset.getBoundDataset>
        </IF>
        <FOR c : t.columns>
            <c.compile>
        </FOR>
        <FOR hr : t.headerrows>
            <header>
                <hr.compile>
            </header>
        </FOR>
        <FOR dr : t.detailrows>
            </FOR>
    </table>
<detail>
  «dr.compile»
</detail>
«ENDFOR»
«FOR fr : t.footerrows»
  <footer>
    «fr.compile»
  </footer>
«ENDFOR»
</table>

def compile(List l)
  ...
  <list name="«l.name»" id="«A=A+i»">
    «FOR p : l.properties»
      «p.compile»
    «ENDFOR»
    «FOR hr : l.headerelements»
      <header>
        «hr.compile»
      </header>
    «ENDFOR»
    «FOR dr : l.detailelements»
      <detail>
        «dr.compile»
      </detail>
    «ENDFOR»
    «FOR fr : l.footerelements»
      <footer>
        «fr.compile»
      </footer>
    «ENDFOR»
  </list>

def compile(CrossTable t)
  ...
  <cross-table>
    «FOR p : t.properties»
      «p.compile»
    «ENDFOR»
  </cross-table>

def compile(Label l)
  ...
  <label name="«l.name»" id="«A=A+i»">
    <text-property name="text">«l.content»</text-property>
    «FOR p : l.properties»
      «p.compile»
    «ENDFOR»
  </label>

def compile(Text t)
  ...
  <text id="«A=A+i»">
    <property name="contentType">html</property>
    <text-property name="content">«t.content»</text-property>
    «FOR p : t.properties»
      «p.compile»
    «ENDFOR»
  </text>

def compile(DynamicText d)
...<text-data id="\xA=A+1">
<expression name="valueExpr">d.content.compile</expression>
<property name="contentType">html</property>
<FOR p : d.properties>
  p.compile
<ENDFOR>
</text-data>
...

def compile(xData d):
...
<data id="\xA=A+1">
<property name="resultSetColumn">d.content.compile</property>
<FOR p : d.properties>
  p.compile
<ENDFOR>
</data>
...
def compile(Image i):
...
<image id="\xA=A+1">
<FOR p : i.properties>
  p.compile
<ENDFOR>
<IF i.type != null>
  <property name="source">i.type.toString</property>
  <expression name="uri" type="javascript">i.content.compile</expression>
<ENDIF>
</image>
...
def compile(Chart c):
...
<extended-item extensionName="Chart" name="\xc.name" id="\xA=A+1">
<xml-property name="xmlRepresentation"><![CDATA[
<IF c.type.toString == 'pie'><model:ChartWithoutAxes
<ELSE><model:ChartWithAxes
<ENDIF>
<model:ChartWithoutAxes
 xmlns:attribute="http://www.birt.eclipse.org/ChartModelAttribute"
<IF c.type.toString != 'pie'>xmlns:data="http://www.birt.eclipse.org/ChartModelData"</IF>
xmlns:layout="http://www.birt.eclipse.org/ChartModelLayout"
xmlns:model="http://www.birt.eclipse.org/ChartModel"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsi:attribute="http://www.birt.eclipse.org/ChartModelAttribute"
<IF c.type.toString != 'pie'>xmlns:data="http://www.birt.eclipse.org/ChartModelData"</IF>
<Version>2.6.1</Version>
<Type>"c.type.toString.toFirstUpper"</Type>
<SubType>"IF c.type.toString == 'pie'>standard"</SubType>
<Children>
<Children xsi:type="layout:TitleBlock">"c.title.chartTitleBlock"</Children>
<Children xsi:type="layout:Plot">"chartPlotBlock"</Children>
<Children xsi:type="layout:Legend">"c.legend.chartLegendBlock"</Children>
<Bounds>
  <Left>0.0</Left>
  <Top>0.0</Top>
  <Width>212.0</Width>
  <Height>130.0</Height>
</Bounds>
<Insets>
<Top>3.0</Top>
<Left>3.0</Left>
<Bottom>3.0</Bottom>
<Right>3.0</Right>
</Insets>
<Row>-1</Row>
<Column>-1</Column>
<Rowspan>-1</Rowspan>
<Columnspan>-1</Columnspan>
<Outline>
<Style>Solid</Style>
<Thickness>1</Thickness>
<Color>
<Transparency>255</Transparency>
<Red>0</Red>
<Green>0</Green>
<Blue>0</Blue>
</Color>
<Visible>false</Visible>
</Outline>
<Visible>true</Visible>
</Block>
<Dimension>Two_Dimensional</Dimension>
<Units>Points</Units>
<SeriesThickness>10.0</SeriesThickness>
«IF c.type.toString == 'pie'»<GridColumnCount>0</GridColumnCount>«ENDIF»
<ExtendedProperties>
>Name>enable.area.alt</Name>
<Value>false</Value>
</ExtendedProperties>
<SampleData>
<BaseSampleData>
<DataSetRepresentation>'A','B','C','D','E'</DataSetRepresentation>
</BaseSampleData>
<OrthogonalSampleData>
<DataSetRepresentation>6,4,12,8,10</DataSetRepresentation>
<SeriesDefinitionIndex>0</SeriesDefinitionIndex>
</OrthogonalSampleData>
</SampleData>
<Interactivity/>
«emptyMessage»
«IF c.type.toString == 'pie'»
<SeriesDefinitions>
<Query>
<Definition></Definition>
</Query>
<SeriesPalette>
«IF c.colorpalette != null»«c.colorpalette.compile»«ENDIF»
</SeriesPalette>
«ELSE» ««ChartWithAxes
<Axes>
<Type>c.xaxe.value.type.toFirstUpper</Type>
<Title>
«IF (c.xaxe.value as Axe).title != null»«(c.xaxe.value as Axe).title.compile»«ELSE»«nullTitle»«ENDIF»
</Title>
<TitlePosition>«IF (c.xaxe.value as Axe).title != null»«(c.xaxe.value as Axe).title.place.toFirstUpper»«ENDIF»</TitlePosition>
<AssociatedAxes>
<Type>c.yaxe.value.type.toFirstUpper</Type>
<Title>
«IF (c.yaxe.value as Axe).title != null»«(c.yaxe.value as Axe).title.compile»«ELSE»«nullTitle»«ENDIF»
</Title>
<TitlePosition>«IF (c.yaxe.value as Axe).title != null»«(c.yaxe.value as Axe).title.place.toFirstUpper»«ENDIF»</TitlePosition>
«ENDIF»

<SeriesDefinitions>
  <Query>
    «IF ((c.yaxe.value as Axe).groupingby) != null»
    <Definition>
      «(c.yaxe.value as Axe).groupingby.compile»
    </Definition>
    <Grouping>
      <GroupType>Text</GroupType>
    </Grouping>
    «ELSE»
    <Definition></Definition>
    «ENDIF»
  </Query>
  <SeriesPalette>
    «IF c.colorpalette != null»«c.colorpalette.compile»«ENDIF»
  </SeriesPalette>
  <Series xsi:type="type:«c.type.toString.toFirstUpper»Series">
    <Visible>true</Visible>
    <Label>
      «nullLabel»
    </Label>
    <DataDefinition>
      «IF (c.yaxe.value as Axe).data != null»«(c.yaxe.value as Axe).data.compile»«ENDIF»
    </DataDefinition>
    <Grouping>
      <GroupType>Text</GroupType>
      <AggregateExpression>Sum</AggregateExpression>
    </Grouping>
  </Series>
  <SeriesIdentifier></SeriesIdentifier>
  <DataPoint>
    <Components>
      <Type>Orthogonal_Value</Type>
    </Components>
    <Separator>, </Separator>
  </DataPoint>
  <LabelPosition>Outside</LabelPosition>
  <Stacked>false</Stacked>
  <Triggers>
    <Condition>onmouseover</Condition>
    <Action>
      <Type>Show_Tooltip</Type>
      <Value xsi:type="attribute:TooltipValue">
        <Text></Text>
        <Delay>200</Delay>
      </Value>
    </Action>
    <Triggers>
      «IF c.type.toString == 'bar'»
      <Riser>Rectangle</Riser>
      «ELSEIF c.type.toString == 'line'»
      <Markers>
        <Type>Box</Type>
        <Size>4</Size>
        <Visible>true</Visible>
        <Outline>
          <Visible>true</Visible>
        </Outline>
      </Markers>
    </Triggers>
  </Triggers>
</SeriesDefinitions>
<Markers>
<LineAttributes>
<Style>Solid</Style>
<Thickness>1</Thickness>
<Color>
<Transparency>255</Transparency>
<Red>0</Red>
<Green>0</Green>
<Blue>0</Blue>
</Color>
<Visible>true</Visible>
</LineAttributes>
<PaletteLineColor>true</PaletteLineColor>

«ELSEIF c.type.toString == 'pie'»
<Exploration>0</Exploration>
<Title>
<Caption>
<Value></Value>
<Font>
<Size>16.0</Size>
<Bold>true</Bold>
</Font>
</Caption>
<Background xsi:type="attribute:ColorDefinition">
<Transparency>0</Transparency>
<Red>255</Red>
<Green>255</Green>
<Blue>255</Blue>
</Background>
<Outline>
<Style>Solid</Style>
<Thickness>1</Thickness>
<Color>
<Transparency>255</Transparency>
<Red>0</Red>
<Green>0</Green>
<Blue>0</Blue>
</Color>
</Outline>
<Insets>
<Top>0.0</Top>
<Left>2.0</Left>
<Bottom>0.0</Bottom>
<Right>3.0</Right>
</Insets>
<Visible>true</Visible>
</Title>
<TitlePosition>Below</TitlePosition>
<LeaderLineAttributes>
<Style>Solid</Style>
<Thickness>1</Thickness>
<Visible>true</Visible>
</LeaderLineAttributes>
<LeaderLineLength>10.0</LeaderLineLength>
«ENDIF»
</Series>
<Grouping>
<GroupType>Text</GroupType>
<AggregateExpression>Sum</AggregateExpression>
</Grouping>
</SeriesDefinitions>
«IF c.type.toString != 'pie'»
<Orientation>Vertical</Orientation>

<LineAttributes>
  <Style>Solid</Style>
  <Thickness>1</Thickness>
  <Color>
    <Transparency>255</Transparency>
    <Red>0</Red>
    <Green>0</Green>
    <Blue>0</Blue>
  </Color>
  <Visible>true</Visible>
</LineAttributes>

<Label>
  <Caption>
    <Value></Value>
    <Font>
      <Alignment/>
    </Font>
  </Caption>
  <Background xsi:type="attribute:ColorDefinition">
    <Transparency>0</Transparency>
    <Red>255</Red>
    <Green>255</Green>
    <Blue>255</Blue>
  </Background>
  <Outline>
    <Style>Solid</Style>
    <Thickness>1</Thickness>
    <Color>
      <Transparency>255</Transparency>
      <Red>0</Red>
      <Green>0</Green>
      <Blue>0</Blue>
    </Color>
  </Outline>
  <Insets>
    <Top>0.0</Top>
    <Left>2.0</Left>
    <Bottom>0.0</Bottom>
    <Right>3.0</Right>
  </Insets>
  <Visible>true</Visible>
</Label>

<LabelPosition>Left</LabelPosition>

<MajorGrid>
  «IF (c.yaxe.value as Axe).majorgrid != null»«((c.yaxe.value as Axe).majorgrid as ChartGrid).compile»
  «ELSE»«nullGrid»«ENDIF»
</MajorGrid>

<MinorGrid>
  «IF (c.yaxe.value as Axe).minorgrid != null»«((c.yaxe.value as Axe).minorgrid as ChartGrid).compile»
  «ELSE»«nullGrid»«ENDIF»
</MinorGrid>

<Scale>
  «ENDIF»
  «ENDIF»
```xml
<Axes>
  <PrimaryAxis>true</PrimaryAxis>
  <CategoryAxis>true</CategoryAxis>
  <Percent>false</Percent>
</Axes>
<Orientation>Vertical</Orientation>
<UnitSpacing>50.0</UnitSpacing>
<Rotation>
  <Angles>
    <XAngle>-20.0</XAngle>
    <YAngle>45.0</YAngle>
    <ZAngle>0.0</ZAngle>
    <Type>None</Type>
  </Angles>
</Rotation>
<IF c.type.toString == 'pie'>
  <model:ChartWithoutAxes>
</model:ChartWithoutAxes>
<ELSE>
  <model:ChartWithAxes>
</model:ChartWithAxes>
<ENDIF>
<FACE>
  <property name="outputFormat">SVG</property>
  <property name="inheritColumns">true</property>
  <FOR p : c.properties>
    <p.compile>
  </FOR>
  <IF c.dataset != null>
    <c.dataset.getBoundDataset>
  </IF>
</FACE>

def chartTitleBlock(Title t)
  """<getNullBounds""
  <Insets>
    <Top>3.0</Top>
    <Left>3.0</Left>
    <Bottom>3.0</Bottom>
    <Right>3.0</Right>
  </Insets>
  <Row>-1</Row>
  <Column>-1</Column>
  <Rowspan>-1</Rowspan>
  <Columnspan>-1</Columnspan>
  <Outline><getOutlineProperties/></Outline>
  <Visible><IF t != null>true</ELSE>false</ENDIF><Visible>
  <Label>
    <IF t != null>t.compile</IF>
    <ELSE>nullLabel</ELSE><ENDIF>
  </Label>
  """
  def chartPlotBlock()
  """<getNullBounds""
  <Insets>
    <Top>3.0</Top>
    <Left>3.0</Left>
    <Bottom>3.0</Bottom>
    <Right>3.0</Right>
  </Insets>
  <Row>-1</Row>
  """
```
```python
def chartLegendBlock(Legend l):
    «getNullBounds»
    «Anchor»
    IF l.place.anchor == 'left' » West
    ELSEIF l.place.anchor == 'right' » East
    ELSEIF l.place.anchor == 'top' » North
    ELSEIF l.place.anchor == 'bottom' » South
    ELSEIF (l.place.anchor == 'middle' && (l.place.position == 'right' || l.place.position == 'left')) » West
    ELSEIF (l.place.anchor == 'middle' && (l.place.position == 'above' || l.place.position == 'bellow')) » South
    ELSE » South » ENDIF
    «EndAnchor»
    «Insets»
    Top 3.0
    Left 3.0
    Bottom 3.0
    Right 3.0
    «EndInsets»
    «Row»
    «Column»
    «Rows</Rowspan»
    «Columnspan»
    «Outline»
    «Visible»
    «ClientArea»
    «Insets»
    Top 2.0
    Left 2.0
    Bottom 2.0
    Right 2.0
    «EndInsets»
    «ClientArea»
    «Text»
    «Value»
    «Font»
    «Alignment»
    «EndFont»
    «EndText»
    «Orientation»
    IF l.orientation != null »
    l.orientation.toString().toFirstUpper » ELSE Vertical » ENDIF
    «EndOrientation»
    «Direction»
    Top_Bottom
    «Separator»
    «Style»
    Solid
    «Thickness»
```

96
<Color>
  <Transparency>255</Transparency>
  <Red>0</Red>
  <Green>0</Green>
  <Blue>0</Blue>
</Color>
<Visible>true</Visible>
</Separator>
<Position>«IF l.place.position != null»«l.place.position.toString().toFirstUpper»«ELSE»Right«ENDIF»</Position>
<ItemType>«IF (l.eContainer as Chart).type.toString == 'line'»Series«ELSE»Categories«ENDIF»</ItemType>
<Title>«IF l.title != null»«l.title.compile»«ELSE»nullTitle«ENDIF»</Title>
>TitlePosition>Above</TitlePosition>

'''
def compile(LegendVisibility v)
  '''
  «IF v != null»«IF v.value == 'hide'»false«ELSE»true«ENDIF»
  '''
  def getOutlineProperties()
    '''
    <Style>Solid</Style>
    <Thickness>0</Thickness>
    <Color>
      <Transparency>255</Transparency>
      <Red>0</Red>
      <Green>0</Green>
      <Blue>0</Blue>
    </Color>
    <Visible>false</Visible>
    '''
    def xAxeGrid()
      '''
      '''
      def yAxeGrid()
        '''
        '''
        /*******************************
        * END OF REPORT ELEMENTS
        *****************************/
        /* PROPERTIES GROUP
        *****************************/
        def compile(LabelProperties p)
          '''
          «IF p.font != null»<property
            name="fontFamily">«p.font.compile</property>«ENDIF»
          «IF p.fontSize != null»<property
            name="fontSize">«p.fontSize.compile»«ENDIF»
          «IF p.backgroundColor != null»<property
            name="backgroundColor">«p.backgroundColor.compile»«ENDIF»
          «IF p.color != null»<property
            name="color">«p.color.compile»«ENDIF»
          «IF p.HAlign != null»<property
            name="textAlign">«p.HAlign.compile»«ENDIF»
          '''
        '''
      /*************/
      */
      def compile(LegendVisibility v)
        '''
        «IF v != null»«IF v.value == 'hide'»false«ELSE»true«ENDIF»
        '''
        def getOutlineProperties()
          '''
          <Style>Solid</Style>
          <Thickness>0</Thickness>
          <Color>
            <Transparency>255</Transparency>
            <Red>0</Red>
            <Green>0</Green>
            <Blue>0</Blue>
          </Color>
          <Visible>false</Visible>
          '''
          def xAxeGrid()
            '''
            '''
            def yAxeGrid()
              '''
              '''
              /*******************************
              * END OF REPORT ELEMENTS
              *****************************/
              /* PROPERTIES GROUP
              *****************************/
              def compile(LabelProperties p)
                '''
                «IF p.font != null»<property
                  name="fontFamily">«p.font.compile</property>«ENDIF»
                «IF p.fontSize != null»<property
                  name="fontSize">«p.fontSize.compile»«ENDIF»
                «IF p.backgroundColor != null»<property
                  name="backgroundColor">«p.backgroundColor.compile»«ENDIF»
                «IF p.color != null»<property
                  name="color">«p.color.compile»«ENDIF»
                «IF p.HAlign != null»<property
                  name="textAlign">«p.HAlign.compile»«ENDIF»
                '''
              /*************/
              */
            /*************/
            */
          /*************/
          */
        /*************/
        */
      /*************/
      */
    /*************/
    */
  /*************/
  */
/*************/
*/
```python
def compile(PageProperties p):
    ...
    """
    ""
    def compile(Font f):
        ...
        """
```
```python
«IF f.linkto != null»
«IF f.linkto.value instanceof SizeValue»
	«(f.linkto.value as SizeValue).compile»
«ENDIF»
«IF f.linkto.value instanceof SizeValuePercent»
	«(f.linkto.value as SizeValuePercent).value.toString»
«ENDIF»
«IF f.linkto.value instanceof SizeValueUnit»
	«(f.linkto.value as SizeValueUnit).value.toString»
«ENDIF»
«ELSE»
«IF f.value instanceof SizeValue»
	«(f.value as SizeValue).compile»
«ENDIF»
«IF f.value instanceof SizeValuePercent»
	«(f.value as SizeValuePercent).value.toString»
«ENDIF»
«IF f.value instanceof SizeValueUnit»
	«(f.value as SizeValueUnit).value.toString»
«ENDIF»
«ENDIF»
```
```javascript
var c = (b.linkto.value as BorderLine).color.compile
var s = (b.linkto.value as BorderLine).style
var w = (b.linkto.value as BorderLine).width.compile

<property name="borderBottomColor">«c»</property>
<property name="borderBottomStyle">«s»</property>
<property name="borderBottomWidth">«w»</property>
<property name="borderLeftColor">«c»</property>
<property name="borderLeftStyle">«s»</property>
<property name="borderLeftWidth">«w»</property>
<property name="borderRightColor">«c»</property>
<property name="borderRightStyle">«s»</property>
<property name="borderRightWidth">«w»</property>
<property name="borderTopColor">«c»</property>
<property name="borderTopStyle">«s»</property>
<property name="borderTopWidth">«w»</property>

ENDIF

IF b.value instanceof BorderSide
  IF (b.linkto.value as BorderSide).bottom != null
    <property name="borderBottomColor">«((b.linkto.value as BorderSide).bottom as BorderLine).color.compile»</property>
    <property name="borderBottomStyle">«((b.linkto.value as BorderSide).bottom as BorderLine).style»</property>
    <property name="borderBottomWidth">«((b.linkto.value as BorderSide).bottom as BorderLine).width.compile»</property>
  ENDIF
  IF (b.linkto.value as BorderSide).left != null
    <property name="borderLeftColor">«((b.linkto.value as BorderSide).left as BorderLine).color.compile»</property>
    <property name="borderLeftStyle">«((b.linkto.value as BorderSide).left as BorderLine).style»</property>
    <property name="borderLeftWidth">«((b.linkto.value as BorderSide).left as BorderLine).width.compile»</property>
  ENDIF
  IF (b.linkto.value as BorderSide).right != null
    <property name="borderRightColor">«((b.linkto.value as BorderSide).right as BorderLine).color.compile»</property>
    <property name="borderRightStyle">«((b.linkto.value as BorderSide).right as BorderLine).style»</property>
    <property name="borderRightWidth">«((b.linkto.value as BorderSide).right as BorderLine).width.compile»</property>
  ENDIF
  IF (b.linkto.value as BorderSide).top != null
    <property name="borderTopColor">«((b.linkto.value as BorderSide).top as BorderLine).color.compile»</property>
    <property name="borderTopStyle">«((b.linkto.value as BorderSide).top as BorderLine).style»</property>
    <property name="borderTopWidth">«((b.linkto.value as BorderSide).top as BorderLine).width.compile»</property>
  ENDIF
ELSE
  IF b.value instanceof BorderAll
    var c = (b.value as BorderLine).color.compile
    var s = (b.value as BorderLine).style
    var w = (b.value as BorderLine).width.compile
    <property name="borderBottomColor">«c»</property>
    <property name="borderBottomStyle">«s»</property>
    <property name="borderBottomWidth">«w»</property>
    <property name="borderLeftColor">«c»</property>
    <property name="borderLeftStyle">«s»</property>
    <property name="borderLeftWidth">«w»</property>
    <property name="borderRightColor">«c»</property>
    <property name="borderRightStyle">«s»</property>
    <property name="borderRightWidth">«w»</property>
    <property name="borderTopColor">«c»</property>
    <property name="borderTopStyle">«s»</property>
    <property name="borderTopWidth">«w»</property>
  ENDIF
ELSE
```
```python
def compile(BorderWidth b):
    if b.value instanceof BorderSide:
        if (b.value as BorderSide).bottom != null:
            borderBottomColor = (b.value as BorderSide).bottom as BorderLine.color
            borderTopStyle = (b.value as BorderSide).bottom as BorderLine.style
            borderTopWidth = (b.value as BorderSide).bottom as BorderLine.width
        endif
        if (b.value as BorderSide).left != null:
            borderLeftColor = (b.value as BorderSide).left as BorderLine.color
            borderLeftStyle = (b.value as BorderSide).left as BorderLine.style
            borderLeftWidth = (b.value as BorderSide).left as BorderLine.width
        endif
        if (b.value as BorderSide).right != null:
            borderRightColor = (b.value as BorderSide).right as BorderLine.color
            borderRightStyle = (b.value as BorderSide).right as BorderLine.style
            borderRightWidth = (b.value as BorderSide).right as BorderLine.width
        endif
        if (b.value as BorderSide).top != null:
            borderTopColor = (b.value as BorderSide).top as BorderLine.color
            borderTopStyle = (b.value as BorderSide).top as BorderLine.style
            borderTopWidth = (b.value as BorderSide).top as BorderLine.width
        endif
    endif

    if b.value instanceof SizeValue:
        s = (b.value as SizeValue).compile
    endif
    if b.value instanceof SizeValuePercent:
        s = (b.value as SizeValuePercent).value
    endif

    def compile(SizeValue s):
        s.value.toString
    endif

    def compile(IndentValue i):
        if i.value instanceof SizeValue:
            s = (i.value as SizeValue).compile
        endif
        if i.value instanceof SizeValuePercent:
            s = (i.value as SizeValuePercent).value
        endif
```

101
def compile(Padding p)

    «IF p instanceof IndentAll»
    «(p as IndentAll).compile»
    «ENDIF»
    «IF p instanceof IndentSide»
    «(p as IndentSide).compile»
    «ENDIF»

def compile(IndentAll i)

    «var value = (i as IndentValue).compile»
    «IF i.eContainingFeature.EContainingClass.name == 'PageProperties'»
        «property name="top" i.eContainingFeature.name.toString.toFirstUpper">«value"</property»
        «property name="left" i.eContainingFeature.name.toString.toFirstUpper">«value"</property»
        «property name="bottom" i.eContainingFeature.name.toString.toFirstUpper">«value"</property»
        «property name="right" i.eContainingFeature.name.toString.toFirstUpper">«value"</property»
    «ELSEIF i.eContainingFeature.EContainingClass.name == 'ChartElementProperties'»
        «Insets»
        «Top»«value"</Top»
        «Left»«value"</Left»
        «Bottom»«value"</Bottom»
        «Right»«value"</Right»
        «/Insets»
    «ENDIF»

def compile(IndentSide i)

    «IF i.eContainingFeature.EContainingClass.name == 'ChartElementProperties'»
        «property name="" i.eContainingFeature.name.toString.toFirstUpper">«(i as IndentValue).compile"</property»
    «ELSE»
        «FOR e : i.eContents»
            «property name=""»
            «IF i.eContainingFeature.EContainingClass.name == 'PageProperties'»
                «e.eContainingFeature.name»
                «i.eContainingFeature.name.toString.toFirstUpper"»
            «ELSEIF i.eContainingFeature.EContainingClass.name == 'LabelProperties'»
                «i.eContainingFeature.name.toString"»
            «ENDIF»
            «/property»
        «ENDFOR»
    «ENDIF»

def compile(Visibility v)

    «list-property name="visibility"»
```python
<IF v.show>«ENDIF»
<IF v instanceof HideAll>
  «(v as HideAll).compile»
ENDIF
<IF v instanceof HideForOne>
  «FOR x : v.value»
  «(x as HideAll).compile»
ENDIF
<IF v instanceof HideForMore>
  «v.compile»
ENDIF
</list-property>

... def compile(HideAll h) ...
... <structure>
  «IF h.format == null»<property name="format">all</property>«ENDIF»
  «IF h.format != null»<property name="format">«h.format»</property>«ENDIF»
  «IF h.expression == null»<expression name="valueExpr" type="javascript">true</expression>«ENDIF»
  «IF h.expression != null»<expression name="valueExpr" type="javascript">«h.expression.compile»</expression>«ENDIF»
</structure>
... def compile(HideForMore h) ...
... «FOR f : h.format»
... <structure>
  «IF h.format != null»<property name="format">«f»</property>«ENDIF»
  «IF h.expression == null»<expression name="valueExpr" type="javascript">true</expression>«ENDIF»
  «IF h.expression != null»<expression name="valueExpr" type="javascript">«h.expression.compile»</expression>«ENDIF»
</structure>
... def compile(Display d) ...
... <property name="display">«IF d.linkto != null»«d.value = (d.linkto as Display).value»«ENDIF»«d.value.getName»</property>
... def compile(PageBreak p) ...
... «IF p.after != null»<property name="pageBreakAfter">«p.after»</property>«ENDIF»
  «IF p.before != null»<property name="pageBreakBefore">«p.before»</property>«ENDIF»
  «IF p.inside != null»<property name="pageBreakInside">«p.inside»</property>«ENDIF»
  «IF p.repeat != null»<property name="repeatHeader">«x.p.repeat.value»</property>«ENDIF»
  «IF p.interval != null»<property name="pageBreakInterval">«p.interval.value»</property>«ENDIF»
... def compile(Toc t) ...
... <structure name="toc">
  «expression name="expressionValue" type="javascript">«t.value.compile»</expression>
  «IF t.style != null»
  <property name="TOCStyle">«t.style.refname.name»</property>
  «ENDIF»
</structure>
```
```python
def compile(Bookmark b):
    <expression name="bookmark" type="javascript">«b.expression.compile»</expression>

def compile(StyleRef s):
    <property name="style">«s.refname.name»</property>

def compile(Style s):
    <style name="«s.name»" id="xA=A+1">
        «FOR p : s.properties»
            «p.compile»
        «ENDFOR»
    </style>

def compile(Width w):
    <property name="width">«w.value.compile»</property>

def compile(Height h):
    <property name="height">«h.value.compile»</property>

def compile(AltText a):
    <expression name="altText" type="javascript">«a.expression.compile»</expression>

def compile(JSExpression j):

def compile(DatasetField f):
    «f.name»

def getBoundDataset(Dataset d):
    <property name="dataSet">«d.name»</property>
    <list-property name="boundDataColumns">
        «FOR f : d.field»
            «f.getBoundDataColumns»
        «ENDFOR»
    </list-property>

def getBoundDataColumns(DatasetField f):
    «structure»
```
<property name="name">«f.name»</property>
<expression name="expression">dataSetRow["«f.name»"]</expression>
</structure>

def compile(ColorPalette c)
    «
    «IF c.linkto != null»
    «FOR e : c.linkto.value»
    «IF e instanceof RgbColor»
    «(e as RgbColor).compile»
    «ENDIF»
    «ENDFOR»
    «ELSE»
    «FOR e : c.value»
    «IF e instanceof RgbColor»
    «(e as RgbColor).compile»
    «ENDIF»
    «ENDFOR»
    «ENDIF»
    
    def compile(RgbColor c)
    
    def compile(Title t)
    
    <Caption>
    <Value>«t.value»</Value>
    <Font>
    <Size>16.0</Size>
    <Bold>true</Bold>
    <Alignment>
    <horizontalAlignment>Center</horizontalAlignment>
    <verticalAlignment>Center</verticalAlignment>
    </Alignment>
    </Font>
    <Background xsi:type="attribute:ColorDefinition">
    <Transparency>0</Transparency>
    <Red>255</Red>
    <Green>255</Green>
    <Blue>255</Blue>
    </Background>
    <Outline>
    <Style>Solid</Style>
    <Thickness>1</Thickness>
    <Color>
    <Transparency>255</Transparency>
    <Red>0</Red>
    <Green>0</Green>
    <Blue>0</Blue>
    </Color>
    <Visible>false</Visible>
    </Outline>
    <Insets>
    <Top>0.0</Top>
```python
def nullTitle()
    «getNullBounds»
    <Insets>
        <Top>3.0</Top>
        <Left>3.0</Left>
        <Bottom>3.0</Bottom>
        <Right>3.0</Right>
    </Insets>
    <Row>-1</Row>
    <Column>-1</Column>
    <Rowspan>-1</Rowspan>
    <Columnspan>-1</Columnspan>
    <Outline>
        <Style>Solid</Style>
        <Thickness>1</Thickness>
        <Color>
            <Transparency>255</Transparency>
            <Red>0</Red>
            <Green>0</Green>
            <Blue>0</Blue>
        </Color>
        <Visible>false</Visible>
    </Outline>
    <Visible>false</Visible>
    <Label>
        «nullLabel»
    </Label>
    ...
    def compile(ChartGrid c)
    ...
    <LineAttributes>
        <Style>Solid</Style>
        <Thickness>1</Thickness>
        <Color>
            <Transparency>255</Transparency>
            <Red>196</Red>
            <Green>196</Green>
            <Blue>196</Blue>
        </Color>
        <Visible>true</Visible>
    </LineAttributes>
    <TickStyle>Across</TickStyle>
    <TickAttributes>
        <Style>Solid</Style>
        <Thickness>1</Thickness>
        <Color>
            <Transparency>255</Transparency>
            <Red>196</Red>
            <Green>196</Green>
            <Blue>196</Blue>
        </Color>
        <Visible>true</Visible>
    </TickAttributes>
    ...
    def nullLabel()
    ...
```
<Caption>
  <Value>Chart Title</Value>
  <Font>
    <Bold>false</Bold>
    <Alignment>
      <horizontalAlignment>Center</horizontalAlignment>
      <verticalAlignment>Center</verticalAlignment>
    </Alignment>
  </Font>
</Caption>

<Background xsi:type="attribute:ColorDefinition">
  <Transparency>0</Transparency>
  <Red>255</Red>
  <Green>255</Green>
  <Blue>255</Blue>
</Background>

<Outline>
  <Style>Solid</Style>
  <Thickness>1</Thickness>
  <Color>
    <Transparency>255</Transparency>
    <Red>0</Red>
    <Green>0</Green>
    <Blue>0</Blue>
  </Color>
</Outline>

<Insets>
  <Top>0.0</Top>
  <Left>2.0</Left>
  <Bottom>0.0</Bottom>
  <Right>3.0</Right>
</Insets>

<Visible>false</Visible>

def nullGrid()
  ...

def emptyMessage()
  ...

<EmptyMessage>
<Caption>
<Value>This chart contains no data.</Value>
</Font>
</Alignment>
</Caption>
<Background xsi:type="attribute:ColorDefinition">
<Transparency>64</Transparency>
<Red>127</Red>
<Green>127</Green>
<Blue>127</Blue>
</Background>
<Outline>
<Color>
<Transparency>128</Transparency>
<Red>127</Red>
<Green>127</Green>
<Blue>127</Blue>
</Color>
<Visible>true</Visible>
</Outline>
<Insets>
<Top>10.0</Top>
<Left>10.0</Left>
<Bottom>10.0</Bottom>
<Right>10.0</Right>
</Insets>
<Visible>false</Visible>
</EmptyMessage>

```python
def getNullBounds():
    <Bounds>
    <Left>0.0</Left>
    <Top>0.0</Top>
    <Width>0.0</Width>
    <Height>0.0</Height>
    </Bounds>
```

/*******************************
* END OF PROPERTIES
*******************************/
Application 3. The report model, created by using the Report-DSL

```report
  sales_report {
    element properties {
      font FontForTitle 'Bookman Old Style'
      font FontForChart 'PT Sans'

      font-size FontSizeForTitle 75.px
      font-size FontSizeForChart 15.px

      h-alignment hAlignForTitle center
      v-alignment vAlignForTitle bottom

      border BorderForTitle {all {color blue style dotted width medium}}

      display DisplayStyleForChartLabel block

      color-palette SixColorPalette {
        rgb(86 207 255)
        rgb(182 228 255)
        rgb(97 184 141)
        rgb(126 218 173)
        rgb(213 149 148)
        rgb(249 166 132)
        rgb(197 197 197)
      }
    }
  }
```

```data-sources {
  data-source SampleDB type sample
}
```

```data-sets {
  dataset SalesByProductLines {
    datasource SampleDB
    fields (Year, Month, ProductLine, Revenue)
    parameters {SelectedYear}
    query "SELECT YEAR(ORDERS.ORDERDATE) AS "Year",
       MONTH(ORDERS.ORDERDATE) AS "Month",
       PRODUCTLINE AS "ProductLine",
       SUM( PRICEEACH*QUANTITYORDERED )/100 AS "Revenue"
FROM
    ORDERS, ORDERDETAILS, PRODUCTS
WHERE
    ORDERDETAILS.PRODUCTCODE = PRODUCTS.PRODUCTCODE
    AND ORDERDETAILS.ORDERNUMBER = ORDERS.ORDERNUMBER
    AND YEAR(ORDERS.ORDERDATE) = ?
GROUP BY
    YEAR(ORDERS.ORDERDATE), MONTH(ORDERS.ORDERDATE), PRODUCTS.PRODUCTLINE"
}
```

```dataset SalesByMonths {
  datasource SampleDB
  fields (Year, Month, Revenue)
  parameters {SelectedYear}
  query "SELECT YEAR(ORDERS.ORDERDATE) AS "Year",
     MONTH(ORDERS.ORDERDATE) AS "Month",
     SUM( PRICEEACH*QUANTITYORDERED )/100 AS "Revenue"
```

109
FROM ORDERDETAILS, ORDERS
WHERE
ORDERDETAILS.ORDERNUMBER = ORDERS.ORDERNUMBER
AND YEAR(ORDERS.ORDERDATE) = ?
GROUP BY
YEAR(ORDERS.ORDERDATE), MONTH(ORDERS.ORDERDATE) "
}

dataset SalesByTerritory {
datasource SampleDB
fields {Year, Territory, Revenue}
parameters {SelectedYear}
query {"SELECT YEAR(ORDERS.ORDERDATE) AS "Year",
OFFICES.TERRITORY AS "Territory",
SUM( PRICEEACH*QUANTITYORDERED )/100 AS "Revenue"
FROM ORDERDETAILS, ORDERS, CUSTOMERS, EMPLOYEES, OFFICES
WHERE
ORDERDETAILS.ORDERNUMBER = ORDERS.ORDERNUMBER
AND ORDERS.CUSTOMERNUMBER = CUSTOMERS.CUSTOMERNUMBER
AND CUSTOMERS.SALESREPEMPLOYEENUMBER = EMPLOYEES.EMPLOYEENUMBER
AND OFFICES.OFFICECODE = EMPLOYEES.OFFICECODE
AND YEAR(ORDERS.ORDERDATE) = ?
GROUP BY
YEAR(ORDERS.ORDERDATE), OFFICES.TERRITORY "
}

dataset SalesByEmployeers {
datasource SampleDB
fields {Year, LastName, FirstName, Revenue}
parameters {SelectedYear}
query {"SELECT
YEAR(ORDERS.ORDERDATE) AS "Year",
EMPLOYEES.LASTNAME as "LastName",
EMPLOYEES.FIRSTNAME as "FirstName",
SUM( PRICEEACH*QUANTITYORDERED )/100 AS "Revenue"
FROM ORDERDETAILS, ORDERS, CUSTOMERS, EMPLOYEES
WHERE
ORDERDETAILS.ORDERNUMBER = ORDERS.ORDERNUMBER
AND ORDERS.CUSTOMERNUMBER = CUSTOMERS.CUSTOMERNUMBER
AND CUSTOMERS.SALESREPEMPLOYEENUMBER = EMPLOYEES.EMPLOYEENUMBER
AND YEAR(ORDERS.ORDERDATE) = ?
GROUP BY
YEAR(ORDERS.ORDERDATE), EMPLOYEES.LASTNAME, EMPLOYEES.FIRSTNAME "
}

dataset SalesByOffices {
datasource SampleDB
fields {Year, City, Revenue}
parameters {SelectedYear}
query {"SELECT YEAR(ORDERS.ORDERDATE) AS "Year",
OFFICES.CITY as "City",
SUM( PRICEEACH*QUANTITYORDERED )/100 AS "Revenue"
FROM ORDERDETAILS, ORDERS, CUSTOMERS, EMPLOYEES, OFFICES
WHERE
ORDERDETAILS.ORDERNUMBER = ORDERS.ORDERNUMBER
AND ORDERS.CUSTOMERNUMBER = CUSTOMERS.CUSTOMERNUMBER
AND CUSTOMERS.SALESREPEMPLOYEENUMBER = EMPLOYEES.EMPLOYEENUMBER
AND YEAR(ORDERS.ORDERDATE) = ?
GROUP BY
YEAR(ORDERS.ORDERDATE), OFFICES.CITY "
}
GROUP BY
  YEAR(ORDERS.ORDERDATE), OFFICES.CITY
}
}

parameters {
  list-box SelectedYear {
    prompt-text 'Please select a year'
    required true
    values
      static multi integer { 'Y11': '2011'
      'Y12': '2012' default
      'Y13': '2013' }
  }
}

body {
  grid Header {}
  {
    column {}
    row {
      cell {}
      {
        label Title "Sales report" {
          font FontForTitle
          font-size FontSizeForTitle
          h-alignment hAlignForTitle
          display DisplayStyleForChartLabel
        }
      }
    }
  }
  grid Charts {}
  {
    column {}
    row {
      cell {}
      chart SalesByMonths line overlay
dataset SalesByMonths
      {
        width 20.cm height 10.cm
        font FontForChart
title "Sales by year" left
legend below middle vertical {
  visibility hide
}
x-axe text {SalesByMonths.Month} {
  line {color aqua style solid width thin}
major-grid {
  scale 1
  color aqua
  style dashed
  width thin
  tick above
}
minor-grid {
  scale 1
  color fuchsia
  style dashed
  width thin
  tick above
}
title "Month" below
}
y-axis linear {SalesByMonths.Revenue} {
  line {color aqua style solid width thick}
  major-grid {
    scale 1
    color black
    style dashed
    width thin
    tick above
  }
  minor-grid {
    scale 1
    color fuchsia
    style dashed
    width thin
    tick above
  }
  title "Revenue" left
}
color-palette {
  SixColorPalette
}
}
}
row {
  cell {
    chart SalesByProductLines line overlay
    dataset SalesByProductLines
    {
      width 20.cm height 10.cm
      font "sans-serif"
      title "Sales by product lines" left
      legend below middle horizontal {
        // title "Chart Legend Title" above
        visibility true
      }
    }
    x-axis text {SalesByProductLines.Month} {
      line {color aqua style solid width thick}
      major-grid {
        scale 1
        color aqua
        style dashed
        width thin
        tick above
      }
      minor-grid {
        scale 1
        color fuchsia
        style dashed
        width thin
        tick above
      }
      title "Month" below
    }
    y-axis linear {SalesByProductLines.Revenue}
    grouping by {SalesByProductLines.ProductLine}
    {
      line {color aqua style solid width thick}
      major-grid {
        scale 1
        color black
        style dashed
        width thin
      }
    }
  }
}
tick above
}
minor-grid {
  scale 1
  color fuchsia
  style dashed
  width thin
  tick above
}
title "Revenue" left
}
color-palette {
  SixColorPalette
}
}
}
row {
  cell {
    chart SalesByTerritories pie
dataset SalesByTerritory
    {
      width 20.cm height 10.cm
      font "sans-serif"
      title "Sales by territories" left
      legend below middle horizontal {
        title "Chart Legend Title" above
      }
      x-axis text {SalesByTerritory.Territory} {
        line {color aqua style dashed width thick}
        major-grid {
          scale 1
          color aqua
          style dashed
          width thin
          tick above
        }
        minor-grid {
          scale 1
          color fuchsia
          style dashed
          width thin
          tick above
        }
        title "Territory" below
      }
      y-axis linear {SalesByTerritory.Revenue} {
        line {color aqua style dashed width thick}
        major-grid {
          scale 1
          color black
          style dashed
          width thin
          tick above
        }
        minor-grid {
          scale 1
          color fuchsia
          style dashed
          width thin
          tick above
        }
        title "Revenue" left
      }
    }
  }
}
```javascript
row {  
cell {  
  chart SalesByOffices bar overlay  
  dataset SalesByOffices  
  {  
    width 20.cm height 10.cm  
    font "sans-serif"  
    title "Sales by offices" left  
    legend below middle horizontal {  
      visibility hide  
    }  
    x-axe text {SalesByOffices.City} {  
      line {color aqua style dashed width thick}  
      major-grid {  
        scale 1  
        color aqua  
        style dashed  
        width thin  
        tick above  
      }  
      minor-grid {  
        scale 1  
        color fuchsia  
        style dashed  
        width thin  
        tick above  
      }  
    }  
    y-axe linear {SalesByOffices.Revenue} {  
      line {color aqua style dashed width thick}  
      major-grid {  
        scale 1  
        color black  
        style dashed  
        width thin  
        tick above  
      }  
      minor-grid {  
        scale 1  
        color fuchsia  
        style dashed  
        width thin  
        tick above  
      }  
      title "Revenue" left  
    }  
  }  
  color-palette {  
    SixColorPalette  
  }  
}  
}
```