German Medical Data Sciences: Shaping Change – Creative Solutions for Innovative Medicine
R. Röhrig et al. (Eds.)

doi:10.3233/SHTI190814

Spreadsheet Model Generator (SMOG): A Lightweight Tool for Object-Spreadsheet Mapping

Alexandr UCITELI[a,1], Christoph BEGER[a,b], Stefan KROPF[a], and Heinrich HERRE[a]
[a] *Institute for Medical Informatics, Statistics and Epidemiology (IMISE), University of Leipzig, Germany*
[b] *Growth Network CrescNet, University of Leipzig, Germany*

Abstract. In the life science domain, experts are usually familiar with spreadsheet software and often use it in their daily work to collect and structure required domain knowledge. The processing and analysis of spreadsheet data is an important task that must be supported by efficient software solutions. A typical application scenario is for example an integration of spreadsheet data (specified or derived) in an ontology to provide reasoning or search. Different converter tools were developed to support a spreadsheet-to-ontology transformation. Such tools allow often only a relatively simple structure of the spreadsheet template or they require complex mapping processes to map the spreadsheet and ontological entities. In many cases, it is impossible to use the existing converter tools because the spreadsheet data must be processed first before the derived data can be integrated into the ontology. In such cases, an efficient and fast development of customized software solutions is of great importance. In this paper, we present a general spreadsheet processing framework to efficiently read and write spreadsheet data. The Spreadsheet Model Generator (SMOG) provides a simple mechanism to automatically generate the Java object model and mappings between object code and spreadsheet entities. Our solution greatly simplifies the implementation of spreadsheet access methods and enables an efficient development of spreadsheet-based applications. The SMOG has already been used successfully in several biomedical projects under real world conditions.

Keywords. Spreadsheet model, object mapping, reverse engineering, programming model generation

1. Introduction

Nowadays, ontologies play a central role in various research as well as industrial projects (see section 4.1). Developing a domain ontology requires not only knowledge of ontological formalisms, but also background knowledge about the domain. The cooperation between ontologists and domain experts is a much debated problem in the literature and has to face some challenges. Because domain experts (such as biologists, statisticians, quality managers or authority employees) are familiar with spreadsheet software and often use it in their daily work, this solution is often the method of choice to simplify the acquisition of domain knowledge. There are a lot of converter tools that

[1] Alexandr Uciteli, IMISE, University of Leipzig, Härtelstraße 16-18, 04107 Leipzig, Germany; E-mail: auciteli@imise.uni-leipzig.de

allow a more or less direct (semi-) automatic transformation of spreadsheet data into an ontology. These solutions have two major issues. Either they allow only a relatively simple structure of the spreadsheet template, which degrades the usability by domain experts, or they require complex mapping processes to map the spreadsheet data to ontological entities. Besides, it is often not possible to directly integrate the spreadsheet data into the ontology because they have to be processed beforehand (see examples in section 4.1). For such cases, we developed the Spreadsheet Model Generator (SMOG)[2], a new approach to efficiently read and write spreadsheet data without having to be familiar with existing spreadsheet software libraries. The SMOG provides a general framework, which can be utilized by developing customized spreadsheet-based applications (also ontology-independent), as well as by existing converter tools. Our solution significantly simplifies the implementation of spreadsheet access methods and enables an efficient development of spreadsheet-based applications.

2. State of the art

There are a lot of methodologies and tools [1–5] supporting transformation of spreadsheet entries in OWL, which are already used in different life science projects. Some of these tools support population of existing ontologies, some allow the generation of spreadsheet templates from the ontology, and some provide expressive languages to map spreadsheet and ontology entities. Such solutions do not directly compete with our approach and could even benefit from it. In contrast to existing converter tools, our approach focusses on use cases, where the spreadsheet data must be processed first before the derived data can be integrated into the ontology. In such cases, it is often impossible to use the existing converter tools, so that a customized software solution must be developed. The software to support this task must consist of three modules: (1) reading/writing spreadsheets, (2) reading/writing ontologies and (3) a data processing module. The SMOG presented in this paper enables generation of a spreadsheet handling module (1) and supports efficient processing of complex spreadsheet templates.

3. Concept

The SMOG works similarly to Object-Relational Mapping (ORM) tools (such as Hibernate [6]). The ORM tools try to overcome the paradigm mismatch between the tabular data representation in relational databases systems and the object-graph representation by object-oriented languages. The ORM connects object code to a relational database and simplifies development through automatic data conversation between database and object model.

By reverse engineering a spreadsheet schema (specified by an annotated spreadsheet template), the SMOG provides a simple mechanism to automatically generate the associated object model and mappings between object code and spreadsheet entities (such as tables, fields and trees). Programmers of spreadsheet-based applications get access to the spreadsheet entries directly via the generated object

[2] https://github.com/Onto-Med/smog

model. This approach abstracts from a concrete spreadsheet library and enables more efficient development.

4. Implementation

4.1. Usage under real world conditions

Our solution has already been used successfully in several biomedical projects (Leipzig Health Atlas (LHA) [7,8], OntoMedRisk [9], OntoPMS [10]) [3] under real world conditions. In this section we introduce two of these projects.

4.1.1. OntoMedRisk: Development of an ontology-based software solution for perioperative risk minimization

In the OntoMedRisk project, methods and tools were developed to identify and analyze risks in perioperative processes in hospitals [9]. The developed software suite is based on the Risk Identification Ontology (RIO), which provides a framework for risk specification and reasoning. The specification of risks is a complex task that needs to be performed by domain experts (medical staff). To facilitate this task, an Excel template was developed which is intelligible to medical staff and can be used for the software tools to transfer the specified information into the ontology. **Figure 1** shows a small part of the OntoMedRisk template to specify a possible risk.

	A	B	C	D	F	G	H	I	J
1	Class:Risk								
2	Field: Risk Name	***Infection_Risk_001***							
3									
4	Field: Risk Specification Rule	(c1 OR c2 OR c3) AND (c4 OR c5) AND c6							
5									
6	Field: Risk Description	Risk of **Dura Mater Bacteria Infection (Meningitis)**.							
7	Field: Recommended Action	Please check antibiotic administration and vaccination status (pneumococcus, meningococcus and haemophilus influenzae type b).							
8	Table: Conditions					Table: Adverse Situations			Table: Treatment Phases
9	Condition Nr.	KPI	Operator	Values		Adverse Situation	Probability		Treatment Phase
10	c1	Age_in_months	IN	[0; 5)		Dura_Mater_Bacteria_Infection	0,09		Indication
11	c2	Skull_bone_thickness	IN	[0; 2]					
12	c3	Ear_structure	==	"abnormal"					
13	c4	Vaccination_status	==	"no"					
14	c5	Vaccination_status	==	"unknown"					
15	c6	Antibiotic_prevention	==	false					

Figure 1. OntoMedRisk template

The ontology generator tool RIOGen developed in the OntoMedRisk project includes the SMOG as a component. The complete risk programming model was automatically generated and used to efficiently read and integrate the risk specifications in the ontology. A system was developed, which collects risk-relevant data from various sources and provides it for the ontology-based risk identification and analysis. The results of such an analysis are transmitted to the medical staff in form of context-sensitive hints and alerts. About 30 risks related to cochlear implantation (e.g., infection risks, **Figure 1**) were implemented.

[3] Funded by the German Ministry of Education and Research (BMBF). LHA: 031L0026, OntoMedRisk: 01IS14022, OntoPMS: 01IS15056B.

4.1.2. OntoPMS: Development of an ontology-supported risk identification software for the observation of medical products on the market (post-market surveillance, PMS)

During PMS, a wide range of sources has to be monitored in order to identify all accessible data about the safety and performance of medical devices. The objective of the OntoPMS project was to support PMS by a clever search strategy and the possibility to create complex search queries by domain experts [10]. We developed the Search Ontology (SON), a new approach for modelling search queries and to aid the preparation of the document corpus. For the application of the SON in a particular domain, it has to be extended by a Domain-specific Search Ontology (dSON) to model domain-relevant search queries. In addition, we developed an Excel template to specify the information required to create a dSON, which significantly simplifies the ontology development by domain experts. **Figure 2** presents a part of the template to model a taxonomy of relevant search concepts described by various simple terms (labels).

	A	B	C	H	I
1	Class: Facet				
2	TreeTable: Concept Tree			Simple Terms (en) [StringL] <,>	Simple Terms (de) [StringL] <,>
3	Medical_Device			medical device	medizinprodukt
4		Clip		clip	clip
5			Endoscopic_Clipping_System	endoscopic clipping system, endoscopic clipping device, endoclip, endoscopic clip, "over-the-scope-clip", "through-the-scope-clip"	endoskopisches clipsystem, endoclip, endoskopischer clip, "over-the-scope-clip", "through-the-scope-clip"

Figure 2. OntoPMS template

For the automatic generation of a dSON (including complete queries) from the Excel template, we implemented the Search Ontology Generator (SONG) based on the SMOG. Using the SMOG, the implementation of reading and writing query specifications has been significantly simplified.

4.2. Spreadsheet Entities

The SMOG supports four main types of spreadsheet entities: fields, tables, trees and classes. The entities are marked with entity identifiers “Field:”, “Table:”, “TreeTable:” and “Class:” followed by the entity name.

Fields are single name-value pairs, like “Risk Name = Infection_Risk_1” (**Figure 1**). Tables consist of a head row with column names and the record rows (e.g., Table: Conditions, **Figure 1**). Trees represent hierarchical structures or taxonomies. Child nodes are placed in the subsequent rows in the next column regarding to their parent. For instance, the node “Clip” is a child node of the node “Medical_Device” and is placed in the next row and next cell relative to its parent node (**Figure 2**). Additionally, trees can be combined with tables in such a way that each table record contains certain information about the corresponding tree node (e.g., simple terms/labels describing search concepts, **Figure 2**). If multiple sheets have the same structure and are intended to collect information about objects of the same class, each of these sheets is marked with the entity identifier “Class:” followed by the class name. For example, various risks are specified in the same manner, each risk in a separate sheet with the same structure (Class: Risk, **Figure 1**). The SMOG uses this information to generate one class (e.g., Risk) and multiple objects (one per sheet, e.g., Risk1, Risk2, etc.) instead of one class per sheet.

Furthermore, it is possible to specify datatypes of fields or table values in the corresponding title or head cells. Following datatypes are implemented: String, Integer,

Double, Boolean and Date as well as the corresponding list types (with suffix L, e.g., StringL). The datatypes are specified in square brackets (e.g., [Integer]). For list types, an additional separator character can be defined in angle brackets (e.g., [StringL] <,> **Figure 2**). The default separator character is "|". Alternatively, the datatypes (except for list datatypes) can be defined by formatting cells using the spreadsheet program (e.g., as number or date).

4.3. SMOG

By reverse engineering a spreadsheet schema (specified by a spreadsheet template, annotated with entity identifiers), the SMOG generates the following types of classes: workbook class, sheet class, table record class and tree node class (column: Class Type, **Table 1**), which contain methods to access spreadsheet entities (col.: Access, **Table 1**).

A workbook class represents the whole spreadsheet and has methods to access single sheets (sheet objects) or sheet classes (lists of sheet objects). E.g., `getRiskList()` (**Table 1**) returns all risk specifications (**Figure 1**) as objects of the class `Risk`.

A sheet class implements methods to read and write entities (fields, tables and trees) specified on a concrete sheet. Return value types and names of the generated methods correspond with the defined datatypes and names of the spreadsheet entities. For example, `getRiskDescription()` returns the value of the Field: Risk Description (**Figure 1**).

Field access methods provide access to single field values; table access methods – to table record objects; and tree access methods – to root tree node objects (**Table 1**). Table record and tree node classes offer read and write methods for the single record values similarly to the field access methods, as well as methods to delete or add records and nodes. Additionally, tree node classes contain a `getChildren()` method to list all specified child nodes as tree node objects. Methods for reading and writing list values return lists of values of defined datatypes and require an array as input parameter (e.g., `List<String> getSimpleTermsEn(), setSimpleTermsEn(String... values)`).

The generated model follows a high abstraction level approach to access spreadsheet entries. Starting with a workbook object, which is initialized with the spreadsheet file, the sheet objects can be accessed via respectively named methods. For example, the call `wb.getRiskList()` returns a list with objects of the class `Risk` representing the specified risk types. Then, read and write methods for desired entities can be called. The call `risk.getRiskName()` returns the name of the corresponding risk type (e.g., the string "Infection_Risk_1", **Figure 1**). Tables and trees can be accessed in the same way. For instance, `risk.getConditions()` returns a list of `RiskConditionsRecord` objects, which can be iterated to access fields, such as `cond.getConditionNr()` (**Figure 1**).

The SMOG is developed with Java 8 and is based on Apache POI 4.0.1 (Java API for Microsoft Documents) [11]. The generator can be used either as an API call in the source code to generate the model in the current workspace, or as a command line tool to generate the model as a Maven artefact to be included as dependency into a Java project.

Table 1. Generated methods (examples) based on the spreadsheet templates in **Figure 1** and **Figure 2**

Figure 1		
Class Type	**Method**	**Access**
Workbook	`List<Risk> getRiskList()`	Sheet
Sheet	`Optional<String> getRiskDescription()`	Field
Sheet	`void setRiskDescription(String value)`	Field
	`void clearRiskDescription()`	Field
Sheet	`boolean isEmptyRiskDescription()`	Field
Sheet	`List<RiskConditionsRecord> getConditions()`	Table
Sheet	`void clearConditions()`	Table
Sheet	`boolean isEmptyConditions()`	Table
Sheet	`RiskConditionsRecord addRecordInConditions()`	Table
Sheet	`void deleteLastRecordInConditions()`	Table
Table Record	`Optional<String> getConditionNr()`	Record Field
Table Record	`void setConditionNr(String value)`	Record Field
Table Record	`void clearConditionNr()`	Record Field
Table Record	`boolean isEmptyConditionNr()`	Record Field
Table Record	`RiskConditionsRecord insertRecordAfter()`	Record Field
Table Record	`void deleteRecord()`	Record Field
Figure 2		
Class Type	**Method**	**Access**
Workbook	`List<Facet> getFacetList()`	Sheet
Sheet	`List<FacetConceptTreeNode> getConceptTree()`	Tree
Sheet	`boolean isEmptyConceptTree()`	Tree
Sheet	`void clearConceptTree()`	Tree
Sheet	`FacetConceptTreeNode addRootNodeInConceptTree(String name)`	Tree
Tree Node	`List<String> getSimpleTermsEn()`	Record Field
Tree Node	`void setSimpleTermsEn(String... values)`	Record Field
Tree Node	`void clearSimpleTermsEn()`	Record Field
Tree Node	`boolean isEmptySimpleTermsEn()`	Record Field
Tree Node	`List<FacetConceptTreeNode> getChildren()`	Record Field
Tree Node	`FacetConceptTreeNode addChild(String name)`	Record Field
Tree Node	`void deleteNode()`	Record Field

5. Discussion

Developing convenient ways to extract data from spreadsheets is a subject of ongoing research. Many scientists focused on syntactical analysis [12,13] and data validation [14,15] of spreadsheets, whereas our aim was to support developers to access the data.

CData Software provides a non-free commercial driver package (ODBC Drivers) [16], which features read and write access to many file formats and databases as well as compatibility with extract, transform and load tools. In comparison to our solution, the ODBC Driver for spreadsheets does not support multiple tables per sheet and other structures like hierarchies or taxonomies. Data in sheets must be structured according to a relational database and developers have to write structured query language (SQL) queries to access and manipulate the data. The ODBC Driver provides the user with a GUI based reverse engineering tool for manual mapping of spreadsheets, whereas SMOG automatically tailors a ready to use API for any number of spreadsheets.

Hermans et al. [12] proposed an approach to generate Unified Modelling Language class diagrams (UML) from spreadsheets automatically. The authors implemented a C#.net based tool called Gyro and gave an outlook on the possible creation of a domain-specific model, which was out of the scope of their publication. SMOG could

also be extended to generate UML automatically, but there was no need for this feature in the presented applications in Section 4.1. We may consider adding it in the future.

Engels et al. [15] used a top-down approach to build spreadsheet templates for specific domains. The advantage is that these templates are more robust to input errors from domain experts, but the approach cannot be used for existing spreadsheets and there is no programmatic way to access the input data. A bottom-up approach like ours is more viable in the life science domain.

Currently, the SMOG does not perform error checks during the input, which is a limitation of our tool. Instead, exceptions are thrown at runtime if, for example, the datatype of an entry conflicts with the specified one and they can be caught in the application to prompt the user to correct the error. Nevertheless, all information required for input checks is specified either as annotations of the template or using the particular spreadsheet software, so that we could generate the checks (e.g., as Visual Basic procedures for Excel) in the future. A further possible extension is an implementation of plugins for IDEs to generate spreadsheet programming models at the touch of a button.

6. Conclusion

In this paper we introduce a new tool to transform spreadsheet templates into Java-based object models. The presented approach greatly simplifies the implementation of spreadsheet access methods and enables efficient development of spreadsheet-based applications. Our tool was successfully used in several projects under real world conditions.

Conflict of Interest

The authors state that they have no conflict of interests.

References

[1] M.J. O'Connor, C. Halaschek-Wiener, and M.A. Musen, Mapping Master: A Flexible Approach for Mapping Spreadsheets to OWL, in: P.F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J.Z. Pan, I. Horrocks, and B. Glimm (Eds.), Semantic Web – ISWC 2010, Springer Berlin Heidelberg, 2010: pp. 194–208.

[2] A. Blfgeh, J. Warrender, C.M.U. Hilkens, and P. Lord, A document-centric approach for developing the tolAPC ontology, *J. Biomed. Semant.* **8** (2017) 54.

[3] S. Jupp, M. Horridge, L. Iannone, J. Klein, S. Owen, J. Schanstra, K. Wolstencroft, and R. Stevens, Populous: a tool for building OWL ontologies from templates, *BMC Bioinformatics*. **13** (2012) S5.

[4] K. Tahar, J. Xu, and H. Herre, Expert2OWL: A Methodology for Pattern-Based Ontology Development, *Stud. Health Technol. Inform.* **243** (2017) 165–169.

[5] K. Tahar, M. Schaaf, F. Jahn, C. Kücherer, B. Paech, H. Herre, and A. Winter, An Approach to Support Collaborative Ontology Construction, *Stud. Health Technol. Inform.* **228** (2016) 369–373.

[6] Hibernate ORM, (n.d.). http://hibernate.org/orm/.

[7] A. Uciteli, C. Beger, K. Rillich, F.A. Meineke, M. Loeffler, and H. Herre, Ontology-Based Modelling of Web Content: Example Leipzig Health Atlas, in: T. Hoppe, B. Humm, and A. Reibold (Eds.), Semantic Appl., Springer Vieweg, Berlin, Heidelberg, 2018: pp. 111–123.

[8] C. Beger, A. Uciteli, and H. Herre, Light-Weighted Automatic Import of Standardized Ontologies into the Content Management System Drupal, *Stud. Health Technol. Inform.* **243** (2017) 170–174.

[9] A. Uciteli, J. Neumann, K. Tahar, K. Saleh, S. Stucke, S. Faulbrück-Röhr, A. Kaeding, M. Specht, T. Schmidt, T. Neumuth, A. Besting, D. Stegemann, F. Portheine, and H. Herre, Ontology-based specification, identification and analysis of perioperative risks, *J. Biomed. Semant.* **8** (2017) 36.
[10] A. Uciteli, S. Kropf, T. Weiland, S. Meese, K. Graef, S. Rohrer, M.O. Schurr, W. Bartussek, C. Goller, P. Blohm, R. Seidel, C. Bayer, M. Kernenbach, K. Pfeiffer, W. Lauer, J.-U. Meyer, M. Witte, and H. Herre, Ontology-based specification and generation of search queries for post-market surveillance, *J. Biomed. Semant.* **10** (2019) 9.
[11] Apache POI, (n.d.). https://poi.apache.org/.
[12] F. Hermans, M. Pinzger, and A. van Deursen, Automatically Extracting Class Diagrams from Spreadsheets, in: T. D'Hondt (Ed.), ECOOP 2010 – Object-Oriented Program., Springer Berlin Heidelberg, 2010: pp. 52–75.
[13] L.V.S. Lakshmanan, S.N. Subramanian, N. Goyal, and R. Krishnamurthy, On querying spreadsheets, in: Proc. 14th Int. Conf. Data Eng., 1998: pp. 134–141.
[14] R. Mittermeir, and M. Clermont, Finding high-level structures in spreadsheet programs, in: Ninth Work. Conf. Reverse Eng. 2002 Proc., 2002: pp. 221–232.
[15] G. Engels, and M. Erwig, ClassSheets: Automatic Generation of Spreadsheet Applications from Object-oriented Specifications, in: Proc. 20th IEEEACM Int. Conf. Autom. Softw. Eng., ACM, New York, NY, USA, 2005: pp. 124–133.
[16] ODBC Drivers | CData Software, *CData Softw.* (n.d.). https://www.cdata.com/odbc/.