# A Framework to Model Metadata for Knowledge Management Tools

Daniel Lüdtke, Sinan Mece, Meenakshi Deshmukh, Michael Bock, Andreas Schreiber, Andreas Gerndt
Simulation and Software Technology
German Aerospace Center (DLR)
Berlin, Köln, Braunschweig; Germany
{daniel.luedtke, sinan.mece, meenakshi.deshmukh, m.bock, andreas.schreiber, andreas.gerndt}@dlr.de

*Full Paper* — **In recent years many kinds of knowledge management tools are being developed. Most of them have in common that they provide an interface to acquire artifacts along with a certain set of metadata. In this paper, a new framework is presented to model metadata for knowledge management tools and to generate metadata-related program code from this model for different components of the target tool. These components include graphical user interfaces for desktop applications, data base schemes, and metadata-related web client code. Finally, two applications from the knowledge management domain are presented, where this framework is successfully integrated to reduce development time substantially.**

***Keywords-metadata; code generation; domain-specific language***

## I. INTRODUCTION

Knowledge has long been a key asset for organizations [1]. This insight led to the active and major research field of knowledge management (KM). The main task of KM is to support the creation, assimilation and dissemination, and finally the application of knowledge [2].

In projects where multidisciplinary teams are working together on complex tasks, the transfer of tacit and explicit knowledge is important to the success of a project. In particular, in domains like aerospace, where large-scale projects are common, an effective and efficient knowledge transfer can help to reduce costs, risks, and enhance product quality at once. Additionally, KM is a critical factor to retain knowledge in an organization when staff is leaving [3].

Along with the research on knowledge management, many KM tools are being developed to aid and support KM tasks such as lessons-learned systems, especially in the aerospace domain [4].

No matter which KM process a tool addresses, they usually have in common that they provide an interface to acquire artifacts that contain knowledge in a certain way. These artifacts can be of arbitrary types like documents, spreadsheets, databases, simulation models etc. Along with these artifacts, typically a certain set of metadata, like authors, categories, or a context must be specified as well. Especially for random data that do not contain a structure for easy extraction of important metadata, the user must be asked to provide the necessary information manually.

It is crucial that KM tools should provide a smart user interface. This can help to improve user acceptance as well as to increase the effectiveness for solving KM tasks regardless of their complexity [5]. Declining user acceptance can be caused by long questionnaires of required metadata that the user has to provide. The interface should be simple and only present the currently required metadata fields. However, such tools yield generally better results if a large set of metadata is available. Hence, a smart user interface is desirable which adapts to the type of artifact and the information a user has already provided. For example, if a spreadsheet is added to the KM tool, no additional information should be inquired about unrelated aspects like bibliographic information of a journal paper. Building such smart user interfaces conventionally requires complex program logic. Moreover, metadata should be generic and adaptable to future changes independent of program logic.

In this paper, we present the new framework *Language for Metadata Based Applications* (LAMBDA) to model metadata for knowledge management tools. It provides a modeling language and code generators for several components of the KM tool. This approach has the advantage that a single point of information—the metadata model—is present from which each part of metadata-related software components can be changed automatically. Thus, consistency is ensured throughout the complete tool chain.

The next section introduces the framework. Section III presents two KM tools where the LAMBDA framework is currently used. Finally, Section IV concludes the paper and gives an outlook to future development.

## II. FRAMEWORK DESCRIPTION

In this section, the framework is introduced by looking at the class of knowledge management applications where LAMBDA can improve the productivity of development. Furthermore, the metadata description language is defined and it is shown how program code can be automatically generated.

### A. Classes of Applications

The framework is intended to be used by developers of KM tools. It shall ease the process of ensuring a consistent data model over the whole set of components that constitutes a KM tool. LAMBDA was designed with the following components of a KM tool in mind:

- Desktop client with a graphical user interface (GUI) to add new or edit existing artifacts with their associated metadata; the metadata is collected by a wizard (see screenshot in Fig. 1) or can be edited with specialized editor components
- Server to store metadata and KM artifacts with indexing capabilities
- Web frontend to search in metadata (possibly also in KM artifacts) and retrieve items

LAMBDA is especially suitable for KM tools if a complex set of metadata is stored along with the actual artifacts. LAMBDA provides a single point of reference for the metadata model and all parts of the application that are depending on the metadata model are generated even if they are realized with different technologies. This makes it easy to create and change sophisticated metadata models during the development process by ensuring consistency throughout the whole toolset. A change to the metadata model does not require manual changes of program code at different locations and components.

Metadata in LAMBDA is structured in profiles and attributes. A KM tool has a set of profiles and each profile encapsulates a set of attributes. An attribute represents a data field, which could be, for example, a name of a knowledge artifact, the name of the author, links to websites, or an internal hash value that is not shown to the user. Profiles group a small set of related attributes together.

Additionally, LAMBDA supports dynamic behavior of metadata entry wizards and editors. It is possible to hide profiles from the user or deactivate ("grey out") attributes dependent on inputs, which users have already provided at other attributes. These features are necessary to create smart metadata entry dialogues, which ask only the question that are relevant for the given context.

For example in the tool SimMoLib (see Section III.A), users need to provide information of their simulation model. Depending on the selected simulation platform in the first profile (first page in the wizard), only the specific profiles for the simulation platform in question are presented. This prevents long lists of metadata fields where just a small set is relevant for the user in a given context.

Incorporating LAMBDA in the development process allows the tailoring of a general-purpose KM tool to a specific domain. For instance, the same tool can adapted to model and analyze air transportation data as well as ground traffic data or even to set up the database for a lessons learned system. This can be accomplished without the need of substantially changing the program code, besides the name and some artwork. One has probably just to provide a new metadata model for a given application.

### B. Overview of the LAMBDA Framework

The presented framework LAMBDA consists of several components that can be integrated completely or just some selected ones in a target KM tool. The core of LAMBDA consists of the metadata modeling language to define the metadata fields—grouped into profiles—and dependencies between fields.

The metadata modeling language is a domain specific language (DSL). A DSL is "a computing programming language of limited expressiveness focused on a particular domain" [6]. The advantage of DSLs compared to other general-purpose description languages, like XML, is simply readability by humans, support of minimum features and focusing only on the specific domain at hand.

LAMBDA's DSL module can be integrated in the common open source integrated development environment (IDE) Eclipse or used as a standalone product along with other development environments like Visual Studio. From the metadata model, program code for several components of the target product can be generated.

Currently, LAMBDA supports generation of:

- Wizards and editors for the Eclipse Rich Client Platform (RCP),
- Java-based code for storing and retrieving the metadata in a JSON [7] representation,
- HTML and JavaScript code for the web interfaces and index servers.

Nevertheless, LAMBDA can be easily extended to generate other modules as well.

### C. Defining the Metadata Model

To implement LAMBDA's DSL part, the widespread open source framework for DSL development Xtext [8] is used. Xtext is fully integrated in Eclipse. A DSL is described in Xtext by using an EBNF (Extended Backus-Nauer Form) style grammar. From this, a parser for the DSL with an internal representation and a code editor is generated. The generated DSL editor supports syntax highlighting, code folding, content assist, and inline error markers. The editor can be used as part of the Eclipse IDE for integration in the development process of the target KM tool. Additional checks, which could not be derived from the grammar itself, can be added by providing simple Java extensions to the Xtext project. The same approach is used to extend the content assist feature or code formatting capabilities.

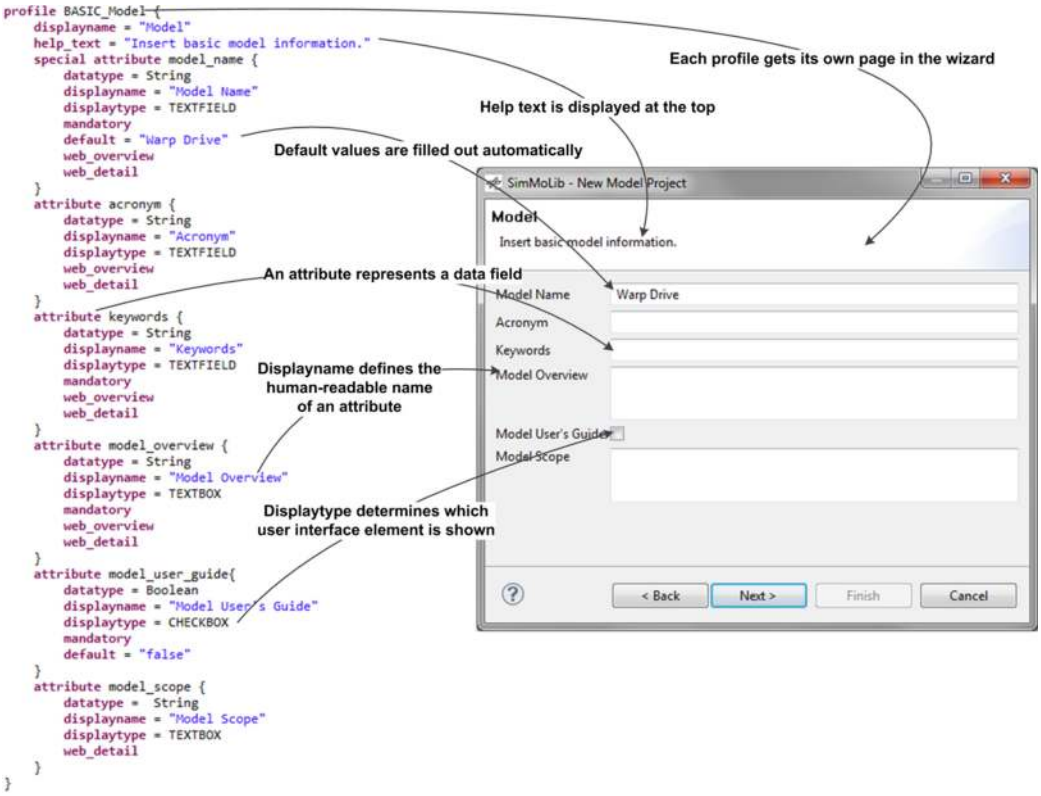


Figure 1. Mapping of metadata model to a metadata entry wizard in a KM tool GUI

The LAMBDA metadata description language was specifically designed to define complex metadata models with a human readable syntax. A metadata description consists of a preamble where data types (e.g. strings, dates, email addresses etc.) and display types are defined. The preamble is somehow similar to type declarations in a common programming language like C. Display types refer to common GUI elements like text boxes, text areas, dropdown menus etc. It has to be defined which GUI element is suitable for a given data type. For instance, a date picker widget is not appropriate to enter an email address.

After the preamble the profile definition is given. Each profile represents a single page in the wizard of the target application with a number of entry fields, the so-called attributes. Additional parameters can be provided in the DSL to define, for instance, human-friendly names that will be displayed or help texts for a more extensive description what an entry field means. A keyword is available to indicate that an entry field is mandatory for the user to provide input. In addition, special keywords are available to influence the appearance of attributes in the web interface: One can select whether an attribute should appear on the search result page or just in a detailed overview page. Fig. 1 depicts the mapping of some of the DSL features to their counterparts in the generated GUI elements.

A special syntax in the DSL is available to support dynamic behavior of the metadata entry wizards. It is possible to hide complete profiles from the user depending on the input the user has given. Additionally, single fields can be deactivated depending on inputs in other fields. With this, intelligent wizards for metadata gathering can easily be modeled that only ask the user the questions that are necessary depending on the information that the user has already given.

Having a simple human-readable domain specific language and a powerful editor at hand, domain experts outside of the software developer team can define the metadata model on their own. The editor provides instant feedback if the model was correctly defined. Thus, the model editor ensures that the description provided by the domain experts can be translated to runnable program code. The error-prone step of converting a model description from a spreadsheet or a text document can thus be omitted.

From our experience, even non-programmers do not need to overcome major obstacles to provide complex metadata models. If the procedure is integrated in an automated build process and the target KM tool has an auto-update function, the modeler can see the results in a couple of minutes or hours without the need of having the whole development environment installed. Just the LAMBDA modeling editor and a development version of the KM tool is sufficient.

### *D. Code Generation*

If a syntactically and semantically correct definition of the metadata model is saved within the DSL editor, the code generation is automatically triggered in the background. For code generation the Xtend framework [9] is used, which is related to Xtext. Xtend is a statically typed programming

language, which can be integrated in Java projects. Besides nice simplifications compared to the Java language, it has strong template features to create code generators.

In LAMBDA, Xtend templates are used to generate code for the different target platforms that are needed. For the GUI wizards and editors Eclipse-compatible Java code is generated. For server communications and storage schemas, Java code that conforms to the JavaBeans conventions is produced. The web interface parts that are metadata-related have their own templates that generate JavaScript and HTML code. All these different code fragments result from the same metadata description.

LAMBDA's goal is to provide as many reusable components for KM tools as possible. The code generators are not restricted to the Java language as demonstrated with the JavaScript/HTML generator parts.

## III. APPLICATIONS

In the following, we present two KM tools that are currently developed using the LAMBDA framework.

### A. Simulation Model Library

The application of computer-aided simulation and calculation models in all phases of space systems development gains more and more importance in recent years. Creating high-quality models is a time-consuming task. The complex models represent a wealth of knowledge that needs to be preserved for future projects. The project Simulation Model Library (SimMoLib) addresses the issues concerning the preservation of knowledge that lies within simulation and calculation models for the space domain [10]. Within the project SimMoLib guidelines and best practices regarding model development, model documentation, validation and verification, as well as model reviews to establish a collection of reusable models are developed. To efficiently catalogue models, a new software system is created to support collaborative development, submission, archiving, reviewing, searching, and utilization of models across department borders.

SimMoLib consists of a web client for model searching and downloading and a full featured desktop client for collaborative model development and management. Fig. 2 depicts a screenshot of the web view for searching models that is embedded in the desktop client. On the right-hand side of the figure one can see a wizard to create new models.

Since the different simulation platforms are used by different disciplines during space system development, establishing a software-based common model library in this domain is a challenging task. Especially creating a complex metadata model with many iterations during development is time consuming and error-prone. For each supported simulation tool, another set of metadata is necessary. Integration of LAMBDA in the development process has simplified this task tremendously. With the single point of truth in the form of the metadata model, an agile development approach could be established. We

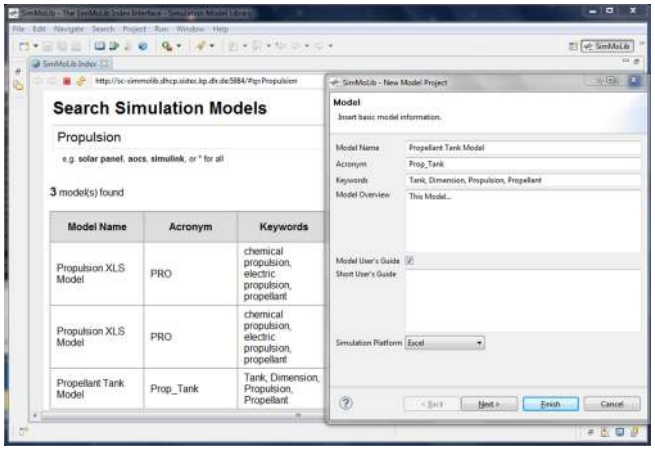


Figure 2. Screenshot of SimMoLib's internal web view and a New Model Wizard

experimented with variations of metadata models to find an optimal solution by having a working version of the SimMoLib client with up-to-date metadata components. This approach definitely saves development time and avoids code duplication.

### B. XPS for CFD

*XPS for CFD* is a second tool where the integration of the DSL-based metadata framework is currently underway. It is a knowledge management system for users of computational fluid dynamics (CFD) that provides specific guidelines, best practices, rules, standards for individual codes, algorithms, and deployable workflows [11].

The KM system consists of two independent, complementary parts. The desktop client captures and annotates CFD data described above and the web interface that serves as a search platform for the data captured by the desktop client. The web interface supports CFD users also by providing best practices from experts so it also constitutes a lessons-learned system.

In contrast to SimMoLib, the set of metadata in XPS for CFD is less comprehensive. However, the metadata model is relatively complex due to the various data types in the KM system. Another important requirement in the case of XPS for CFD is that the customers of this tool want to change the metadata model easily. This requirement is important because of the fact that standards and experiences evolve and thus an updated metadata model needs to be adapted at specific points in time.

This required flexibility for highly dynamic metadata and models led to a web framework, which is in use now by different customers from different domains. Fig. 3 shows a use case from a project dealing with global air transportation topics [12]. The underlying web engine differs significantly from a technical viewpoint to the web client approach of SimMoLib. However, for both cases, a simple adaptation of the code generator templates allows a fast evaluation of different metadata models.

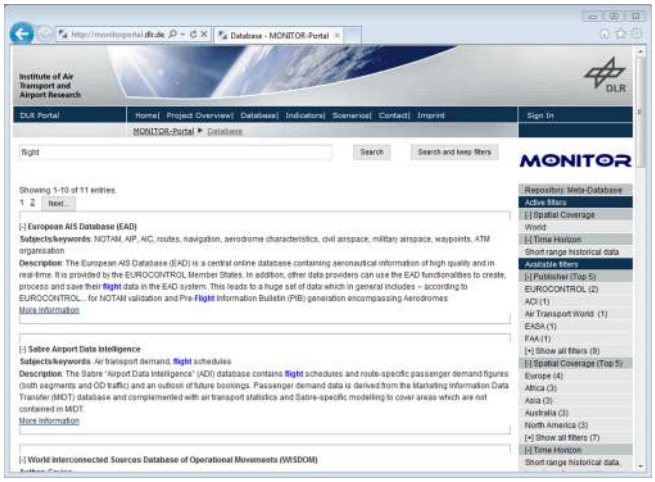


Figure 3. Screenshot of the XPS web client in the context of the Monitor project

We expect a tremendous decrease in time and complexity by integrating LAMBDA and generating code parts that depend on the metadata model.

## IV. Conclusions and Outlook

The presented framework provides a high degree of flexibility in the domain of knowledge management tools. It has a simple approach to provide powerful metadata capabilities with dynamic and smart behavior while the user enters data in the final application. LAMBDA offers a single point of information with its DSL-based metadata description from which metadata-related program code is generated for the target application. This allows even non-programmers to change the metadata model. With the code generators in place, the consistent change of all relevant sections in the program code is ensured.

Bringing the concept of domain specific languages in the knowledge management field has become convenient just for a couple of years since language frameworks such as Xtext and Xtend are available. With Xtext, only a simple grammar has to be developed to get a large toolset with a full-featured text editor tailored to the defined language.

The output of the code generators of LAMBDA is not limited to a particular programming language, so the framework can be integrated in a large variety of KM tools.

LAMBDA is constantly improved and extended to provide the metadata infrastructure for the presented projects SimMoLib and XPS for CFD. In the future, we want to integrate LAMBDA into other applications like the Python-based tool DataFinder [11].

One major open issue exists: conversion of metadata models. LAMBDA makes it easy to change the metadata model during development. Nevertheless, when the software has been used and data has been entered into the system, this causes a change of the metadata model for a new release and old data needs to be converted to the new version of the metadata. Currently, there is no automatic solution to generate metadata-converters. This issue must currently be addressed by manually implementing converters. Future research will include studies to evaluate if it is possible to generate those converters also with LAMBDA at least partially.

## Acknowledgment

The authors thank Jens Rühmkorf, Marcel Rehfeld, and Christian Kerl for their earlier work and ideas, which has eventually led to the development of LAMBDA.

## References

[1] J. Liebowitz, "A look at NASA Goddard Space Flight Center's knowledge management initiatives," IEEE Software, vol. 19, no. 3, pp. 40–42, May 2002

[2] T. Kotnour, C. Orr, J. Spaulding, J. Guidi, "Determining the benefit of knowledge management activities," IEEE Int. Conf. of Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, vol.1, pp.94–99, October 1997

[3] Knowledge Management at NASA, http://www.km.nasa.gov (accessed 29-May-2012).

[4] Proceedings of the International Workshop on Lessons Learned in Program/Project Management, ESTEC Noordwijk, March 2012

[5] Yang Xu, A. Bernard, N. Perry, Lian Lian, "Managing knowledge management tools: a systematic classification and comparison," Int. Conf. of Management and Service Science (MASS), pp.1–4, August 2011

[6] M. Fowler, R. J. Parsons, Domain Specific Languages. Addison Wesley, 2010.

[7] D. Crockford, "JSON: the fat-free alternative to XML," Proc. of XML 2006, Boston, USA, December 2006. http://www.json.org/fatfree.html

[8] S. Efftinge, M. Völter, "oAW xText: A framework for textual DSLs," Workshop on Modeling Symposium at Eclipse Summit, 2006.

[9] S. Efftinge, S. Zarnekow, "Extending Java – Xtend: a new language for Java developers," PragPub, The Pragmatic Bookshelf, no. 30, pp. 5–11, December 2011.

[10] D. Lüdtke, J. Ardaens, M. Deshmukh, R.P. Lopez, A. Braukhane, I. Pelivan, S. Theil, A. Gerndt, "Collaborative development and cataloging of simulation and calculation models for space systems," 3rd IEEE Track on Collaborative Modeling and Simulation (CoMetS), Toulouse, France, June 2012, in press.

[11] A. Schreiber, J. Rühmkorf, D. Seider, "Scientific data and knowledge management in aerospace engineering," 3rd Int. Conf. of Advanced Engineering Computing and Applications in Sciences (ADVCOMP '09), October 2009, pp. 111–116.

[12] MONITOR, Monitoring System of the development of global aviation. http://www.monitor-project.eu/ (accessed 29-May-2012).