ABSTRACT
The main goal of this paper is to describe the technical aspects of our generic approach for domain-specific model analysis. Our framework supports checks and information retrieval for a set of models which may be based on different notations and tools. Such a modelling scenario is typical for the analysis of clinical processes, where techniques of process modelling (e.g. Aris\(^1\)) are combined with techniques of IT-landscape modelling (e.g. using the 3LGM\(^2\)). All elements defined and used in the models under consideration are saved in a common repository based on a common meta model. The method supports the user during the modelling and the analysis phase. Support for the latter is achieved by providing aggregated information on model elements based on queries and views, which can be considered as model cross sections corresponding to given aspects. We present an architecture, a prototypical realization and the technologies used in our implementation. The architecture is modular, used technologies and software are open, thus the tool is easy to extend.

KEY WORDS
domain-specific model assessment, heterogeneous modelling environment, medical process evaluation support

1 Introduction
Nowadays there is a huge number of tools supporting model design in standardised notations such as UML\(^3\). Usually designed models cover specific aspects of a modelled system through different views (e.g. static structure and behavioural models). The whole set of models creates a complex model landscape interconnected by common elements used in different models. It is hardly possible to control the coherence and the validity of the models by manual means only, therefore a tool supporting model analysis is required. Existing methods and tools focus mainly on syntactical correctness independent of modelling domains, as well as on providing modelling metrics\(^4\) and design support\(^5\). Our approach on the other hand is focused on domain-specific analysis based on the conceptual model of a given domain (meta model) which allows us to define checks over an entire model landscape. Concerning the underlying meta model our method is slightly similar to the method developed by \(^1\) although our method does not only support consistency checks of models but also domain-specific checks. Moreover our framework is generic and can be easily adopted for modelling and analysis of another domain (a.o. by replacing the meta model with a new one).

A modelling scenario with many tools is typical for the analysis of clinical processes. Different techniques are used for processes modelling (e.g. Aris) and for IT-landscape modelling (e.g. using the 3LGM). Our approach has been developed within the MedFlow\(^4\) project.

In this paper we briefly describe the requirements for the tool supporting our approach (section 2), our methodology (section 3), and we present the architecture of the prototypic implementation followed by a usage scenario (section 4). The last section concludes with our experience gathered during the implementation phase of the MedFlow project.

2 Requirements
We consider two kinds of requirements, namely functional (section 2.1) and technical (section 2.2). The first are connected with the way the tool should work, the second with implementational aspects.

2.1 Functional Requirements
The main goal of our project is the analysis of clinical processes. Therefore our framework should provide a mechanism for aggregating information from all models and give the analyst some guidelines to further analysis (see requirement \(R_1\) in Figure 1). The aggregated information should be presented using different formats (\(R_2\)).

In practice modellers use different modelling notations and tools, thus the second goal of the project was to create a framework supporting consistency maintenance in

\(^1\)http://www.ids-scheer.de/
\(^2\)http://www.3lgm2.de/
\(^3\)Unified Modelling Language, http://www.uml.org/


519-116 56
heterogeneous modelling environments (R₃, R₄) with a centralised data storage (R₅).

R₆ The tool should enable the modeller to formulate and execute queries, checks and views expressed in a predicative language based on the structure of the meta model.

R₇ The tool should present results in graphical and textual form (modification of model representation, graphs, trees, tables, etc.).

R₈ The tool should provide interfaces to different modelling tools and notations.

R₉ The tool should guarantee (or at least support) the consistency of models.

R₁₀ The tool should store information about a complex enterprise model landscape, where all models correspond to a given meta model and are strongly connected with each other.

Figure 1. List of functional requirements.

2.2 Technical Requirements

To avoid reinventing the wheel we decided to extend existing tools and to use established development technologies (see requirement R₁ in Figure 2). This requirement entails the usage of open standards and open source tools, which can be extended easier (R₂). To increase the number of potential users of the tool the framework should preferably be platform–independent (R₃).

R₁₁ The tool should reuse and extend existing frameworks and tools.

R₁₂ The tool should be based on well–known, open standards and software.

R₁₃ The tool should be platform–independent as far as possible.

Figure 2. List of technical requirements.

3 Methodology

Our research study was initially inspired by modelling clinical processes (7, 8). The aim of the MedFlow project is the fusion of technical and socio–organisational views of processes. In the technical view we consider IT–support of processes and modelling IT–landscapes at different levels of abstraction (e.g. describing physical and logical tools defined in separate models). The quality of IT–support should be taken into consideration during the analysis of processes. For analysis purpose we provide the concepts of queries, checks and views defined on the meta model level and interpreted on the user model level (see subsequent sections). The developed method is based on the definition of modelled elements on the meta level and the connection of meta elements with elements used in the user level models through mapping procedures (section 3.1). The provided queries, checks and views support the modeller in the analysis of business process models (section 3.2). Due to space limitations we will not describe the whole methodology in details, for an exact description of our approach see (9).

3.1 Modelling on Meta and User Levels

Figure 3 depicts the basic concepts our methodology is based on. We consider two levels of abstraction, namely the meta model level and the user model level. The meta model defines the model elements of the application domain (called meta elements) and the relations between them. We call instances of meta elements elements. For example the≪information≫Referral is an instance of an≪information≫. The model type package defines the model types and their relationships. An instance of the model type package is a model landscape and the instances of model types are models. For example an instance of≪Business Process Model≫can be a≪BPM≫X–ray examination. Additionally we have an element–model–mapping (an EM–mapping for short), which interconnects model elements and model types. The last concept deals with the language over which quality criteria can be defined in form of queries, checks and views (see section 3.2).

3.2 Queries, Checks and Views

The goal of a query is to provide the modeller with information on the model landscape. The queries are defined over elements defined in the meta model. There are different types of queries regarding the type of the returned value: queries returning a boolean value (called checks, e.g. Is a given information saved in a given logical tool?); queries returning an integer value (e.g. Number of logical tools in the model landscape.); and finally queries returning a set of elements (e.g. The set of logical tools
an information is saved in). There are two special types of queries, namely predefined checks and complementary queries. Predefined checks are defined by the meta modeller (e.g. Each information used in Business Process Models has to be defined in the Information Model.) and are well–formed rules. Complementary queries are defined over the result of a view, whereas all other queries are defined over the model landscape.

To provide more complete and cross–sectional information we aggregate the results of queries in a view. Views provide information on sets of elements, unlike queries, which take single instances as arguments. Let us consider an example view in which we link information objects with logical tools. At the beginning we determine the set of all information objects defined in the model landscape (i.e. \( I = \{\text{Referral, Diagnostic Findings, Image,} \ldots\} \), compare Table 1 and the set of all logical tools (i.e. \( LT = \{\text{KIS, PACS, PaterNoster,} \ldots\} \)). Then for each possible pair of the Cartesian product of the sets above (i.e. \( I \times LT \)) we evaluate the query: Is a given information saved in a given logical tool? The complete result of the view is depicted in Table 1.

Table 1. Exemplary result of a view linking information objects with logical tools.

<table>
<thead>
<tr>
<th>Information \ Logical Tool</th>
<th>KIS</th>
<th>PACS</th>
<th>PaterNoster</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Referral</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>...</td>
</tr>
<tr>
<td>Diagnostic Findings</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>...</td>
</tr>
<tr>
<td>Image</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

4 Prototypic Architecture

In the following section we describe the underlying standards, tools, technologies and the prototypic architecture based on them. On the topmost level we can distinguish between parts connected with meta modelling, user–level modelling, data repository and analysis (see Figure 4). In the following paragraphs those requirements motivating our design decisions are listed in parentheses.

Repository To be able to analyse information on all model elements defined in different modelling tools we decided to use one central repository. The central data storage is realised by means of technologies supported by AndroMDA\(^4\), which is an open source implementation of the MDA\(^5\) framework (\( R_1, R_2, R_3 \)). Using AndroMDA we generated Java classes based on a meta model XMI file. The meta model itself should be MOF\(^6\) conform. On the next layer we utilise Hibernate\(^7\), an object/relational persistence framework, to save the data in a relational database. The MDA framework allows us to postpone the decision concerning the relational data base management system (RDBMS) to the last phase of the implementation. It makes our tool more portable. We can either use a local or a server data base. The two remaining parts of the repository, namely the generic repository access layer and the generator, are currently being implemented. The generic repository access layer is based on Java reflection mechanism and allows us to create instances of Java classes during run time based on information from adapters and the EM–mapping XML file. The generator will translate queries from query languages to the repository internal representation.

Modelling Environment In the prototypic implementation we support three model description notations: UML, XMI and XML (\( R_4 \)). UML is used in modelling tools and transformed to the repository notation with the help of adapters (separately provided for each tool). Initially we will support two modelling tools: MS Visio and MagicDraw (\( R_5 \)). MS Visio is not a strict UML modelling tool, but is widely–used in practice. MagicDraw is a platform–independent UML modelling tool (\( R_6 \)). An additional adapter supports model transformations from XMI notation to the corresponding repository representation (\( R_7 \)). This allows our framework to support all modelling tools which can save models in XMI. The second tool–independent adapter will support models described using XML (\( R_8 \)) in a format defined as an XML Schema. The format description in XML Schema and the UML profile should be meta model conform. All adapters interpret the XML file containing the EM–mapping.

Analysis Tool Within the query language modellers can construct their own queries and views (\( R_9 \)). Both HTML pages and images illustrating the results graphically (\( R_{10} \)).
will be generated. As our tool is based on J2EE technologies, we are able to use an integrated server as well as a web–server.

**Usage scenario** Our framework is a generic approach and after appropriate initialisation it can be used for analysis of models from any domain. In the following the basic steps to configure the application are listed.

1. Definition of the meta model through an UML Model stored in XMI.
2. Definition of the EM–mapping in an XML file that is conform to the XML Schema file we provide.
3. Definition of an UML profile for modelling tools corresponding to the meta model.

After the configuration phase modelling and analysis can be done iteratively. To be able to map the information contained in created models to the common repository it is necessary to either use modelling tools supported by our framework or tools able to save models in XMI format. The models can also be saved in XML format if XML Schemas (meta model conform and described in EM–mapping) were created before. The queries and views can be activated in modelling tools or in the analysis tool.

5 Conclusion

This paper briefly presents a framework for a domain-specific model assessment. Our methodology is based on a common meta model describing model elements and their relationships. Due to the meta model it is possible to use a common repository. Using queries, checks and views the information saved in the repository can be analysed. We showed how to implement our methodology and integrate it with heterogeneous modelling environments. There are some limitations in the prototypic version of our tool, such as a limited number of adapters or hard coded queries. Since the queries will be hard coded in the first version of our tool, the generator implementation is postponed. In further work we plan to implement a generic analysis tool and to increase the number of supported modelling tools also for non–UML (but MOF conform) notations.

**Acknowledgements**

We would like to thank two members of our research group, Bernd Tilg and Frank Innerhofer–Oberperfler, who designed the architecture of the tool with us. Additionally we would like to express our thanks to all other members of the MedFlow project for discussion.

---

References


