Configurable Execution Environments for Medical Processes

S. Jablonski*, M. Faerber*, M. Götz*, B. Volz*, S. Dornstauder*, S. Müller**

*) Chair for Databases and Information Systems, University of Bayreuth
Universitätsstrasse 30, 95447 Bayreuth, Germany
**) Chair for Database Systems, University of Erlangen-Nuremberg
Martensstraße 3, 91058 Erlangen, Germany

Abstract. We present the process modeling and execution tools iPM and iPE that can dynamically be adjusted to domain specific requirements. While many publications about iPM are focusing on its modeling flexibility, this paper concentrates on its capability to execute such customized process models. The iPM execution environment (called iPE) is based upon a template driven generation approach: for each domain specific process modeling language, templates are provided which are compiled into a specialized execution environment.

1. Introduction

Gathering and organizing data is a major problem for many, especially complex application domains like the clinical / medical application domain. Here, data from different medical devices have to be collected and prepared for various analyzing steps such as diagnosis and reporting. We investigate such a clinical application currently in a research project funded by the German Research Society [1] at the ophthalmic department of the University of Erlangen-Nuremberg. Our task is to integrate various diagnostic data into a common database (the so called Glaucoma Register) and to provide physicians and researchers with data and tools to effectively and efficiently perform research in that medical realm.

Due to the dual purpose of the application – especially of the medical devices – as medical research infrastructure on one hand side and as infrastructure for the normal clinical consultation on the other hand side, the application scenarios for clinical devices become very difficult and complex. Thus, clinical processes are chosen to describe and illustrate these scenarios. In another DFG funded research project we found out that clinical processes are a powerful means to model complex application systems [2]. Some reasons foster this approach: first, (clinical) processes are recognized to illustrate complex scenarios in a clear and concise way such that a user can easily comprehend their content. Secondly, changes can easily be performed on process models – we will see that changes are a challenge in our application.

In a first step we used standard process modeling tools (e.g. ARIS, VISIO) to describe the clinical processes. However, we were experiencing two major problems:

- The semantics to be described was so complex that the resulting process models were hard to comprehend (Figure 1a, a VISIO example). The physicians could not understand – and thus could not review – this complex process model. As a conse-

quence from this observation, we created application specific process modeling elements that describe application semantics in a compact and comprehensible way which is a first contribution of this paper. Figure 1b depicts the resulting process model. The grey process step comprises the same semantics as the process of Figure 1a. By the way, for the purpose of this paper the content of the process depicted in Figure 1 it is not of interest; the comparison of the two representations in Figure 1a and Figure 1b should rather illustrate that tailored, compact modeling constructs facilitate the readability and comprehension of process models.

- Another major challenge of the clinical domain is its constant change; this feature is primarily due to the research character of the application. To meet this requirement we are also fostering process modeling to describe the application: thereby changes in the application can be reconstructed quickly in the process model. A challenge here is that such a change must not only be reflected in a process model but must also be reflected in the process execution environment. A second major contribution of this paper is to present the design and implementation of a process execution infrastructure that perfectly matches this requirement.

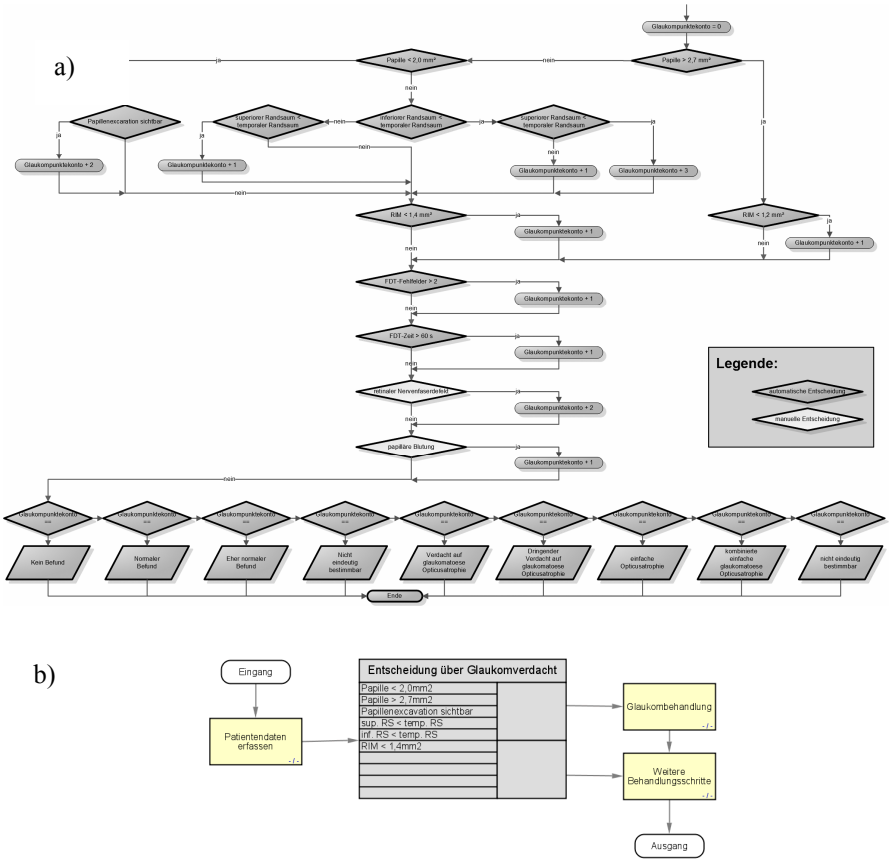


Figure 1: **The advantage of domain specific modeling**

From the above discussion two components of our solution approach can be derived:

- The required use of specialized process modeling constructs needs a domain specific modeling language (Section 2).
- The implementation of clinical process requires a flexible process execution infrastructure. This must be extremely adaptable, since it firstly must be able to interpret domain specific constructs introduced "on the fly", and secondly to interpret, i.e. to execute them in an efficient way. This is a demanding feature since adaptation and flexibility are – in principle – mutually contradictory (Section 3).

If these two requirements can be met, we are able to provide a powerful process modeling and execution environment for flexible process models.

2. Domain Specific Process Modeling

Domain-Specific Modeling is an approach that facilitates communication between domain experts and software developers, as it uses modeling concepts directly connected to the domain the software will be applied in [7]. This way errors caused by misunderstandings between modeling experts and domain experts can be avoided or can be detected earlier [6]. Consequently, for each domain a Domain Specific Language (DSL) must be constructed that contains domain specific modeling elements. However, in our case having a DSL is just the first step. Since modeled processes must be executed subsequently, a domain, i.e. a DSL specific process execution facility must also be provided. The modeling elements contained in a DSL for Process Modeling can be partitioned in two subcategories:

- Elements that are needed in every process model, independent of the current application domain (common process modeling elements)
- Elements that are especially introduced for one application domain (specific process modeling elements)

At the end of this section we will present some examples for common and for specific process modeling elements, respectively.

This partition and the flexibility demanded in Section 1 require a specific modeling concept. We have developed an approach that is called perspective oriented process modeling [2, 4]; this concept is implemented by our process modeling tool iPM (integrated ProcessManager) that will be demonstrated. Its main idea is the following: When a process model is needed that has to be adapted to new requirements frequently, it must be divided into small “pieces” in order to better cope with such changes. This is nothing but the well known principle of "divide and conquer", a synonym for the term "modularization" in software engineering.

The only firm and static part of our perspective oriented process model is referred to as the modeling construct skeleton. This skeleton knows that so-called perspectives are defined; together these perspectives establish a (process) modeling construct. The typical perspectives of a modeling construct for processes are [4]:

- The functional perspective describes the tasks of a process.
- The organizational perspective identifies the agents (e.g. physicians) responsible to perform a task.
- The operational aspect specifies applications needed to execute a process.

- The control flow perspective defines the execution order between workflows.
- The data perspective defines both the in and out parameters of workflows and how data flows between workflow steps.

In order to create a new modeling construct, each perspective must be customized. Therefore, for each perspective a meta model is provided that facilitates customization. Having the perspective oriented process model, (almost) arbitrary modeling languages can be constructed. Hereby, we use the following interpretation: a modeling language is constituted through the modeling constructs it comprises. Thus, for each application domain, a domain specific process modeling language can be provided.

It is out of the scope of this paper to present a complete set of modeling constructs for process modeling, separated into common and specific constructs. However, some examples should illustrate the meaning and purpose. The two major common modeling elements for the functional perspective are elementary and composite process steps. Elementary process steps cannot be refined any more, while composite process steps consist of elementary or other composite process steps. One of the major common modeling elements for the control flow perspective is the choice construct with two or more exits. In the application depicted in Figure 1a, a medical decision has to be made that determines whether a suspicion of Glaucoma must be confirmed or can be abandoned. The whole process consists of two sorts of elements: in the choices (diamonds) certain criteria are interrogated. Depending on the outcome of these questions, certain variables are increased (or decreased). Finally, a decision is taken. We kind of summarize this whole decision into the compact, domain specific construct of Figure 1b: the main criteria for this decision are depicted by the grey modeling construct and can therefore be read and interpreted by a physician who immediately understands which criteria are relevant in the decision process. In order to get more detailed information about the medical algorithm being executed, the physician can zoom into the modeling construct.

The example of Figure 1 shows that compact, customized modeling constructs increase the readability of process models: while the contents of Figure 1a and Figure 1b are the same, the physicians much easier and faster comprehend the meaning of the process in Figure 1b and therefore accept process modeling much more. As we already stated in Section 1 the main challenge is to also provide an execution environment for such flexible modeling constructs. The next section introduces iPE, the execution environment for process models developed with the iPM process modeling tool.

3. Executing Domain Specific Processes: iPE

3.1. Architecture of iPE

For each process model (defined in a domain specific process modeling language) a so-called iPE (“integrated Process Execution”) application is generated by the iPE compiler (Figure 2). The iPE application is a domain specific execution environment for processes modeled in a domain specific process modeling language. There are two main inputs for the iPE compiler: first, the process model that has to be executed, and

second, a series of so called iPE templates. Among these iPE templates there is one template defining the domain specific process language used to describe the process model under consideration. Some other templates are needed to construct the components of the generated iPE application as they are also shown in Figure 2 (e.g. Runtime Persistence Service, Dynamic Navigation Service).

The generation of the iPE application starts with a transformation of the corresponding process model into a language neutral representation which can be seen as a superset of the modeling languages iPM can support principally. Secondly, the components of the iPE application are generated from the iPE templates. By exchanging the iPE templates different components can be created. For example, the Runtime Persistence Service can be based on a database system or on a file system, respectively.

The iPE application is based on a three tier architecture which separates the frontend, the data manipulation and the data storage layer from each other by well defined interfaces. The frontend of the iPE application implements the interface for the process participants. Typically, the process users have to input data and interact with the iPE application (e.g. determine the next steps to be executed). This input is then sent to the process control layer. The process control layer forwards the incoming calls to the attached services. According to the use of different iPE templates, different services can be deployed. For instance, the frontend (i.e. the user interface) could be implemented as a web application (running in a standard browser) or a stand-alone application. The following list gives a glimpse on the function of each service shown in Figure 2:

- Runtime Persistence Service: Stores the data of the process during execution in such a way that the path along a process can be easily tracked back afterwards.
- Offline Persistence Service: According to strict regulations in the medical application field, it is necessary to store some parts of the data into special databases for a longer period of time. This service synchronizes such databases by transferring the important data of a process from the Runtime Persistence Service to these storages
- Knowledge Management Service: This service supports the execution of one particular process by providing additional information from a knowledge management system. In the clinical application field one can think of providing information about medical treatment or diseases using this service.
- External Application Execution Service: Often it is necessary to execute external applications like a word processor or retrieving data from a medical device (e.g. an X-ray image, blood pressure).
- Dynamic Navigation Service: The path along a process is not fixed but will be determined during runtime of the process depending on the data available in the process and the process execution history.

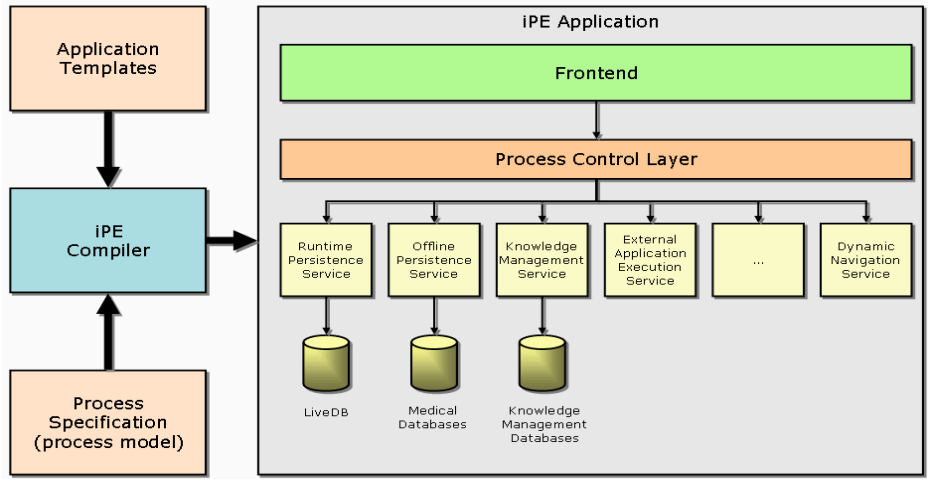


Figure 2: The iPE compiler generates the iPE application

We implemented prototypic templates for the frontend using JavaServer Faces [8]. The iPE compiler (implemented in Java 1.5) then generates a WAR file which can be deployed to every Servlet Container.

3.2. Executing Processes

This section provides a short introduction into the execution of processes by an iPE application. Therefore, we take a short process as example (Figure 3) and explain how the components of the iPE application are used to execute this sample process.

In the step “Anamnesis” patient data (e.g. name and age) are entered. These data are needed in the next step “Examination” as the tests are set up individually for patients of different age. In the last step, the finding of an examination is composed. Consequently, all data used in the process up to now, are needed.

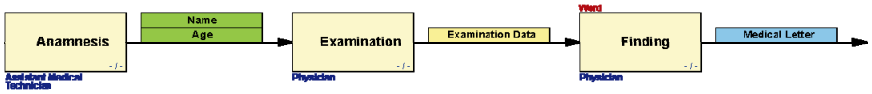


Figure 3: A sample process

To execute a process, e.g. the sample process of (Figure 3), a user must log into the iPE application using the frontend of the iPE application. In this example, we provide a default frontend which runs in a standard web browser. In the step “Anamnesis” some input fields like "name" and "age" are shown which have to be filled by the process user. Another default frontend to enter examination data is used in the process step "Examination". However, the third process step "Finding" is different: here, a word processor (e.g. MS Word) must be called in order to fill out a template for findings. In the following, we briefly show how the components of the iPE application are interacting in order to execute the sample process.

All data entered by the process users are collected by the Process Control Layer (Figure 2) and subsequently forwarded to the underlying services of the iPE application like Runtime Persistence Service (RPS), Knowledge Management Service (KMS), and External Application Execution Service (EAES). After a process step was executed, e.g. the step "Anamnesis", the data collected by the Process Control Layer are stored in a database by the RPS. After having entered such data, the KMS could be called to search for relevant information in the scope of interest (e.g. the glaucoma disease). This search is depending on the data collected at the corresponding process step. To call the KMS is optional whereas calling some other services like the RPS is mandatory. It must be specified in the iPE templates whether service calls are optional or mandatory. In the third step of our sample process model ("Finding"), the EAES needs to be called. It is responsible to start the MS Word text processing system in order to write the diagnostic findings. Hereby, a MS Word template for findings is used, which is specified in the process model. The finding will then be stored in the database (as "Medical Letter"). In order to accomplish this, the RPS is called. If in the course of process execution, this finding must be revisited again, the two services EAES and RPS are again responsible to present this finding to the process user.

The iPE application also assigns user or roles to process steps. For example, while the first step in the process (Figure 3) must be executed by an Assistant Medical Technician, the second and the third step must be performed by a physician.

According to the scenario presented above, the various services are called by the Process Control Layer in order to execute process models. However, even this small scenario depicts how flexible process execution can be organized. Consequently process models (i.e. the process languages) and iPE templates can be customized in such a way that they ideally fit the requirements of specific process execution semantics.

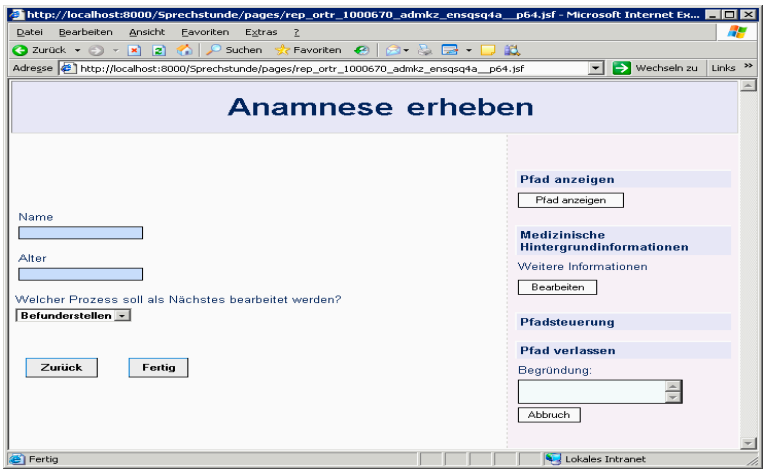
4. Demonstration: Presented Application Areas

While this paper mainly focuses on the medical domain, the iPE execution environment can also be used for other application scenarios. This is due to its Meta modeling approach and its modular architecture. To support another application domain means to provide an alternative domain specific process modeling language. While the process modeling language of our medical domain is called iPM4MED, we are also able to provide the process modeling language iPM4QM for the quality management domain. This process language is compliant to the reference models of the ISO 15504 quality standards [5]. In our demonstration we will present both modeling domains, the medical domain (iPM4MED, cf. [3]) and the quality management domain (iPM4QM). For both domains we will demonstrate

- how to specify new modeling constructs,
- how to define and modify new process models, and
- how to execute such process models for different application domains.

Through these three demonstrations we can show how to extend a modeling language and how to execute processes that are defined according to this customized modeling language. That way, the whole flexibility of our two tools iPM (for process modeling) and iPE (for process execution) will be shown. Below, you'll find a screenshot of the iPE frontend. On the left hand side, the fields to input data for the process step "An-

amneses" can be seen. On the right hand side, context information provided by the KMS can be shown.



References

1. Special Research Area 539: „Glaukome einschließlich Pseudoexfoliationssyndrom“ Homepage. http://www.sfb539.forschung.uni-erlangen.de/. Retrieved 2006-05-30.
2. Jablonski, S.; Lay, R.; Meiler, C.; Müller, S.; Hümmer, W.: Data Logistics as a Means of Integration in Healthcare Applications. Proceedings of the 2005 ACM Symposium on Applied Computing (SAC) - Special Track on Computer Appplications in Health Care, Santa Fe, New Mexico, 2005.
3. Jablonski, S.: Process Based Data Logistics: Data Integration for Healthcare Applications. ECEH 2006 (1st European Conference on eHealth Fribourg, Switzerland), 2006.
4. Jablonski S., Bußler C., 1996. Workflow management - modeling concepts, architecture and implementation. London. International Thomson Computer Press, 1996.
5. Jablonski, S.; Faerber, M.; Schlundt, J.; Bridging the gap between SPICE Reference Processes and OU Processes: An iterative business process modelling approach. SPiCE 2006 (Proceedings of the 6th International SPICE Conference on Process Assessment and Improvement (ISO/IEC 15504 standard)), 2006.
6. Cook, S.: Domain-Specific Modeling an Model Driven Architecture. MDA Journal, January 2004.
7. Luoma, J.; Kelly, S.; Tolvanen, J.: Defining Domain-Specific Modeling Languages: Collected Experiences. Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling (DSM’04), Vancouver, British Columbia, Canada, Oct 2004, Computer Science and Information System Reports, Technical Reports, TR-33, University of Jyväskylä, Finland, 2004
8. Sun Developer Network: JavaServer Faces Technology, http://java.sun.com/javaee/javaserverfaces/, retrieved 2006-07-11