The Workflow Management System Panta Rhei *

Johann Eder, Herbert Groiss, Walter Liebhart

Department of Informatics, University of Klagenfurt, Austria
email: {hans, herb, walter}@ifi.uni-klu.ac.at

Abstract. We present the prototypical workflow management system Panta Rhei which was designed to support various types of workflows in an uniform and elegant way. The main characteristics of this system are its flexibility, the lean architecture, the integration in the Web, the transactional features and its support of process management. We discuss the basic concepts of the system, its architecture and its implementation.

1 Introduction

With hundred workflow management systems on the market [Law97] and even more workflow prototypes in research labs the question arises: Why design and implement another one? If we have a closer look on workflow systems, we discover that this term is used for quite different concepts ranging from documented email to scripting languages and workflow management systems offer very different functionalities.

A popular classification ([GHS95]) distinguishes three types of workflows:

- *ad-hoc workflows*
  Workflows are controlled by users at runtime. Users can react to situations not considered at build-time.
- *administrative workflows*
  Predictable and repeatable workflows described in simple description languages. Activities are mainly performed by humans.
- *production workflow*
  Predictable and repeatable complex workflows which are predefined completely and in great detail using complex information structures and involve application programs and automatic activities.

Although this classification seems to be very helpful at first glance and suggest a need of different workflow management systems we have some objections against this conclusion. The first observation is that in most organizations we found different types of workflows. Some are ad-hoc, some administrative, some of production type. And we observe that persons usually are involved in different types of workflows. We regard it as disadvantage,

* Heraklit: "Everything flows"

if they had to use different systems. The other observation is that the classification does not hold in practice. Take for example an highly structured and often-repeated production workflow. It is simply impossible to preplan all exceptions that could arise since workflows are models of real world processes. Folklore in workflow systems says that exceptions are not exceptional. Therefore, ad-hoc intervention in production workflows has to be possible too to avoid workflow system to become bureaucratic nuisances.

These observations and considerations as well as our experiences with an earlier prototype which was based on form flow metaphor and active databases [EGN94] led us to design a new system. We claim that it is possible to support all these different types of workflows with diverse and seemingly contradicting requirements with a single workflow management system. And with our workflow system Panta Rhei we show that this can be achieved without an inflation of features. Our main contributions are:

- The design of a set of concepts which fit well together and allow the support of different types of workflows.
- A set of features for improving the management of workflows that work well together with the basic concepts, in particular workflow transaction, time management and process management.
- An open architecture supporting user interaction as well as collaboration of workflow systems of different organizations.
- A lean architecture and a comparably simple implementation.

The remainder of this paper is organized as follows: in the next section we present and discuss the basic concepts of Panta Rhei and highlight some of the additional features. In section 3 we present our workflow definition language WDL. Section 4 contains the architecture and implementation aspects of Panta Rhei. Section 5 concludes this paper by giving a summary and presenting some ideas of further research.

2 The concepts of Panta Rhei

2.1 Classification of Workflows

We consider workflow systems as man-machine systems with the following ingredients: Agents perform activities which means among other possibilities the manipulation of data. Processes are partial orderings of activities. The workflow systems organizes who performs which activity with which data at which time.

We can classify workflow management systems along several dimensions:

- *initiative*
  In user centered systems the initiative is with the user. He decides on the continuation of the process, who is next, etc. In the extreme the workflow management system is a rather passive delivery and bookkeeping device.

The initiative could also be with the system. In the extreme the users cannot influence the process and merely fulfill tasks they receive from the system.
- *structure of data*
  Data can be structured or unstructured. Structured data can be accessed and interpreted by the system, while unstructured data (like text documents) cannot.
- *data classes*
  We distinguish between case data (e.g. client name in an insurance claim process) and process data (e.g. termination state of an activity).
- *activity types*
  Manual activities are performed by users without further support from the system. Automatic activities are carried out by IT-systems without human intervention. For semiautomatic activities humans use specific interactive programs for performing an activity.
- *process specification*
  Here we distinguish how processes are specified and when. They could be completely ad-hoc which means the process is defined at run time step by step by the agents. The other extreme is a fully specified process where agents cannot make decisions about the continuation of the process at all. We distinguish also according to the expressiveness of the process specification language, in particular, whether the process can be defined dependent on case or process data.
- *agents*
  Agents are the ones performing activities. Agents can be humans or IT-systems, or abstractions thereof, like roles or organizational units.

It is easy to see that these dimensions are all but orthogonal. For an example, if the system should make decisions based on case data it is mandatory that case data are available to the system in form of structured data. Another example shows the dependencies between the dimensions initiative and process specification: if workflows are defined undeterministically then the user has to take the initiative.

2.2 Basic Concepts

Our aim was to develop a small set of basic concepts which allow to express all workflows which are possible according to the classification above in an uniform way. The basic decisions for achieving this goal were the following:

- All process information is mapped to the data space. Process schema as well as process instance data are stored in the database. Thus processes can be manipulated as data which offers enormous flexibility.
- Case data and process data are represented in a uniform way. We use the form metaphor for representing data. From a programming point of view a form is a data type. From an implementation perspective, a form is a view on the database.

- Processes are defined in a highly generic workflow definition language with the usual control structures. The only variables in this language are of type form, such that all data used in processes is stored in the database.
- We rigorously tried to abstract and parameterize concepts. All concepts are first class citizens in our workflow language. Agents, forms, activities and even process definitions and control structures can be taken from form items.

These decisions brought several advantages. Above all we achieved great flexibility and expressiveness with a comparably small set of concepts. We do not have to distinguish between ad-hoc processes and predefined processes at this level. For an example, the name of an agent and the name of the next activity and the forms it should manipulate can all be taken from a form manipulated by an agent in a previous activity. With this concept we tightly integrate ad-hoc and production processes. The workflow engine is kept small and generic. It can be seen as an automaton operating on workflow specification data, process data and case data representing the state space of the workflow system.

2.3 Additional Features

Panta Rhei offers several possibilities in order to guarantee a consistent and reliable execution of business processes, i.e. the handling of failures and exceptions (see [EL95,EL96,EL97]). Therefore it is necessary to distinguish between possible failure and exception classes, like system failures (e.g. a client breakdown), semantic failures (if an activity terminates unsuccessfully) or unexpected exceptions at runtime, like changing the order of activities within a workflow. The *advanced transaction* system of Panta Rhei tries to automate the recovery from failures and exceptions as much as possible. The transaction system is both, a runtime feature and a build time feature. As build-time feature it offers additional possibilities for a workflow designer to specify valid processes. The workflow schema can be enriched by additional information which is used in case of an exceptional situation. As runtime feature it (automatically) recovers from (semantical) failures, compensates activities and triggers alternative executions. Many of the underlying concepts are borrowed and adapted from advanced transaction models [Elm92,AAA+96].

Although most workflow management systems offer sophisticated modeling tools to specify and analyze workflows, they suffer from an important shortcoming: they are not capable to handle the concept of time adequately [PEL97,JZ96,ELP97]. *Time* appears in Panta Rhei in two notions. Structural time aspects express the (estimated) duration of activities and structural time dependencies between activities. Explicit time aspects are used to formulate time constraints of the real world (e.g. an activity d has to be finished 5 days after activity a has finished). For the integration of these time aspects a new method called ePERT, based on the optimization method PERT (Program Evaluation and Review Technique) was developed [PEL97]. The time information which becomes available by this approach can be used to find the critical path within a workflow, to determine which alternative executions are still valid and what are the latest start time for activities in order to meet some deadline. The system is capable of monitoring deadlines and generates time errors in the case that deadlines are missed. Another aspect is the possibility to delay tasks for a certain amount of time or to a specified data.

Modern businesses build the whole organization around their key business processes. Important for the success of process centered organizations is that each process has a responsible manager. Similar we see the management of workflows. A *workflow manager* is assigned to each workflow. He is in charge of monitoring workflows, handle exceptions and failures which cannot be resolved automatically and has several possibilities to manipulate the execution of workflows. In particular he should be able to quickly analyze the current status of a workflow, stop and resume a workflow, abort a workflow, change a workflow specification, map a workflow to a different one, start an ad-hoc process, make decisions about priorities, etc. Of course, all these manual interventions should not violate the consistency of the overall workflow and therefore they are based on the systems internal advanced transaction mechanism. To enable workflow managers to achieve all this, a special workflow client application is needed which offers all this functionality. Furthermore, the time information computed with the ePERT method can be used to calculate priorities within worklists of users and thus it contributes to avoid time errors beforehand. It allows to discover potential timing problems early such that a process manager can take steps to avoid them.

2.4 Open Architecture

One design decision of Panta Rhei was to make the participation in a workflow as easy as possible. First, the interaction between the workflow system and the outside world (users, applications, other workflow systems at the same site or at remote sites) is solely realized by exchange of forms. This has two important advantages: The interface of a user to Panta Rhei is completely integrated in a web browser. This makes the system open such that anybody with access to the WWW can participate in a workflow. This feature is important, if one realizes that workflows are frequently started as a reaction to some request from outside of an organization (e.g. an order is received). Second, the communication between the workflow system and other systems is realized through the exchange of forms. Panta Rhei does not make any assumptions on client applications and remote systems but that they are able to receive forms and return or send forms. All process specific information is represented in forms. So there is no need for complex suite of APIs to make workflow systems collaborate.

3 Workflow Description Language

The formal representation of processes within Panta Rhei is based on the **W**orkflow **D**efinition **L**anguage WDL. This language was explicitly designed for the system Panta Rhei and its "look and feel" is similar to that of traditional programming languages. Since workflow designers are not necessarily programmers, Panta Rhei also offers an easy to use graphical interface to specify workflows. Such a specification can be mapped to WDL and vice versa. WDL itself consists of 5 basic units: the workflow specification part, the definition of activities, roles, organization structures and inter-process communication.

3.1 Workflow Specification

The workflow specification part of WDL enables the formal description of a workflow (process). Primarily this means to specify the control and data flow of a workflow. Additionally, transactional and time relevant aspects are specified within this part.

Structure of a workflow specification. The structure of a typical workflow specification is similar to the structure of ordinary procedural programs:

- *Header:* Every workflow has a name and any number of optional arguments. Arguments are forms which are passed into the workflow and/or produced by the workflow.
- *Declaration part:* The declaration part consists of two units: workflow declaration and data declaration. The *workflow declaration part* serves for the specification of general process information. This type of information is equal for all processes and comprises information of the process owner, a subject text, a short description of the current process, the specification of the maximal execution time and the specification of some action (e.g. the execution of a special time activity) in case of a time failure.
  Forms which are produced within a workflow have to be declared in the *data declaration* part. This is simply be done by assigning form-types to form variables. Fields within the form can be accessed via the dot notation. The scope of a data declaration is the whole workflow.
- *Body:* This is the main part of a workflow specification. Included between the keywords `begin` and `end` the description of the control and data flow of the process is expressed. Within the workflow context this means to answer the following W-questions: *who has to do what and when and with which data*. Based on this paradigm, typical statement sequences in WDL are of the following simple structure (see also Fig. 1):
  `agent_name activity_name (form_names);`
  The semantics of this is: The workflow agent `user_name` performs the activity `activity_name` with the forms defined by the arguments `form_names`.

Assignment of agents to activities. The assignment of agents (human or machines) to activities is a fundamental concept within workflow management systems. At run time, flexible and dynamic assignment resolution techniques are necessary to react adequately on organizational needs and changes. WDL fulfills this requirement by the following assignment techniques:

– The agent is defined directly by the user name of the responsible person.
– The agent is defined indirectly via a role concept. Any user who belongs to such a role is allowed to execute the activity. Of course, roles within different organization structures (e.g. departments) may have the same name. Therefore, in WDL the workflow designer can extend the role name by a department name in order to avoid such conflicts.
– Often there are different activities within a process which should be executed by one and the same agent. In most cases, it is not known at process definition time, which agent this will be. WDL offers a simple concept to handle this requirement.
– The highest degree of flexibility is achieved by assigning agents dynamically during run time (especially for weak structured workflows). Since forms are first class citizens within WDL, it is easy to support this by introducing an extra field within the form into which the name of the next agent can be written or executed out of some other information at run time. Consider the following example: If a request for a credit is accepted then the applicant should be informed by the person who always takes care of this applicant. At run time the actual name of this person is stored in the field `responsible_agent` within the form `credit_form`. The WDL command is:
  `credit_form.responsible_agent accept(credit_form);`

Control flow specification. As explained earlier, the basic statement sequence in WDL is the assignment of agents to activities and the data specification in the form of `agent activity(form1, form2)`. Let's call such an assignment a "workflow step". WDL now offers a variety of control structures to specify the execution order of workflow steps. The most relevant control structures are:

– *sequence:* A sequential execution order of workflow steps is specified by listing the corresponding workflow steps one after another, each separated by a semicolon.
– *alternative:* For the definition of alternatives the well known `if - then - else` construct is available. For the specification of expressions, WDL offers almost all the concepts which are known from procedural languages. Above all, the usage of form fields within expressions is a very convenient feature to construct expressions. An example of an expressions within a control structure is:

```
IF credit_form.sum > 1000 and credit_form.judge = ''True''
THEN boss approve(credit_form);
ELSE ... ENDIF
```

– *loop:* Two types of loops are realized in WDL: while-loops and repeat-until-loops. Within a loop, workflow steps are repeated as long as a certain condition holds.
– *parallelism:* Parallelism helps to reduce workflow execution time and is therefore an important control flow construct. In WDL two types of parallelism are available: and-parallelism (`andpar`) and or-parallelism (`orpar`). The difference between these two types is the synchronization of parallel paths. In the first case, synchronization means that all parallel executing paths have to be finished in order to continue with workflow execution after the synchronization point. In the second case, workflow execution can continue as soon as one parallel execution path finishes.
– *nesting:* In order to reduce complexity and to reuse already defined workflows, Panta Rhei supports the nesting of activities. Activities within a workflow can themselves again be a complex workflow.

Transaction support. WDL allows the specification of workflow transactions [EL95,EL96,EL97] which are very helpful to support reliable and consistent workflow executions. The main idea is to compute workflows within relaxed ACID transactions in order to guarantee the consistency of the business process.The following concepts are necessary to express workflow transactions and are hence fully integrated within WDL:

– *transaction specific control structures:* Besides the already mentioned control structures there are two transaction specific control structures: `free choice` and `ranked choice`. These constructs enable the modeling of contingency paths which are executed in case the previous execution path cannot be executed successfully. A contingency path is intended to accomplish a similar goal as the original path. The difference between the two choice constructs is the selection - fixed or dynamically - of the different paths at run time.
– *vitality*: Typically, workflows have a hierarchical structure. Within such a structure, one (parent) process may exist of several other (child) processes or elementary activities. Whether a parent process achieves its goal or not, mainly depends on its child components. The relation between parent and child components can be specified as vital or non-vital, meaning, that vital connected childs have to terminate successfully in order to enable a successful termination of the parent. Non-vital connected components do not directly influence the result of the parent component.
– *compensation*: Since workflows are long-running applications it is necessary to split the execution into multiple transactions. Doing this, the benefits of single ACID-transactions, isolation (i.e., serializability) and atomicity (i.e., all-or-nothing behavior) is lost. Therefore the concept of

compensation must be introduced in order to cope with the problem of partially executed workflows. WDL supports the specification of workflows with compensation activities. During activity definition, the workflow designer can assign a corresponding compensation activity which semantically undoes the effects of the original activity. Unfortunately, there are different levels of compensation. Therefore in WDL the compensation type must be specified with the special keyword `stornotype`. Four different stornotypes are possibilities: Type none (1) means, that the committed activity does not need to be compensated because it is not necessary from an application point of view. Type undoable (2) means that a committed activity can be undone by a corresponding compensation activity without any side-effects, in the sense of an inverse operation. Type compensatable (3) means that the compensation of an activity leads to some side-effects (e.g. money transfer and back transfer with transfer fees). Type critical (4) means that an activity which has already terminated cannot be undone or compensated afterwards because its effects are irreversible within the current context (e.g., drilling a hole, mailing a sensitive information). The following example shows a cutout of an activity definition with compensation in WDL:

```
activity money_transfer (f1) stornotype compensatable
money_back_transfer(f1);
```

In Panta Rhei *concurrency control* is supported on the level of forms. For every form and all fields within a form various access rights can be specified. At runtime the system monitors the compliance of these rights.

As an example we briefly describe a simple process for a credit application (see Fig. 1). After the header some properties of the process are defined (owner, description, maxtime, subject). The data container for this process is a form of type `c_appl` (credit application form), declared as `appl`. The process is initiated by a `salesperson` performing the task `insert` with the form `appl`. If the field `judge` of this form is set to `'refuse'`, a secretary will perform the step `make_refusal`. Otherwise the else-block is executed in the same fashion.

Definition of Activities, Roles and Organization Structures. The definition of activities consists of a header part and a declaration part. In the header the name of the activity and the forms which are passed to or from the activity are specified. The declaration part contains the following information:

- *procedure*: If the activity uses external applications then this must be specified at this position.
- *postcondition*: This area allows the definition of a postcondition (i.e., any complex expression) which is evaluated when the activity is finished. The activity will terminate successfully if this condition is satisfied.

```
process credit
owner herb;
description 'simple treatment of credit application';
maxtime 10 days;
forms appl c_appl;
subject f.applicant;

begin
 salesperson insert(appl);
 if appl.judge = 'refuse' then
        secr make_refusal(appl);
    else
        clerk check(f);
        if appl.c_sum > 1000000 then
                boss check(appl);
        end;
        if appl.judge = 'refuse' then
                secr make_refusal(appl);
          else
                secr accept(appl);
        end;
end;
```

Fig. 1. Credit application workflow

- *description*: Short description of the activity.
- *maxtime*: Allows the definition of the maximum execution time of the activity.
- *formprops*: This attribute allows the specification of access rights on the forms which are manipulated within the activity. The granularity of access rights are form fields.
- *costfunction*: For run-time optimization purpose, cost-information (e.g., how expensive is an activity, what is the success probability of an activity) is necessary. This information is either specified by a numeric constant or by a program which computes the cost of the activity at run time.

Example: Definition of the activity order_entry:

```
ACTIVITY order_entry(f order-form)
DESCRIPTION 'Fill in the order form';
PROCEDURE start_order_application;
POSTCONDITION f.judge = 'OK';
MAXTIME 1 DAY;
FORMPROPS f.customer_data write, f.order_number read,
f.security invisible;
```

The assignment of users to roles is in WDL realized by the construct:

```
agent <user> has <role> in <dept>
```

Additionally, there are two special roles: `all` and `system`. Every user automatically belongs to the role `all`. The role `system` means that the role is not occupied by a human being but by a system. The *definition of organization structures* is possible via the construct:

```
department <dept> contains <dept>
```

Inter-process communication. Panta Rhei allows two types of distribution of process execution. First, a process can be triggered by sending a form to the workflow server. This can be done from another workflow system or an arbitrary application. The WDL-construct for defining the behavior when receiving a form is:

```
on receive <form-type> start <process> in <dept>
```

The second kind of inter-process communication is the remote execution of an activity or a subprocess. It is performed when the agent of the activity is an URL, i.e. a pointer to another workflow system. In this case, all process relevant data are sent to the other workflow system, the results are sent back after execution.

3.2 Execution of WDL

A process description defined in WDL is transformed to an execution graph, Fig. 2 shows the graph of the credit process. Every activity or call of a subprocess is translated to a node. Additional special nodes are defined for handling of control structures, `if`-nodes for conditional statements and loops, `andjoin` and `orjoin` for the *anpar* and *orpar* constructs. For the workflow engine these control structures are another type of activities, the only difference to other activities is that they are directly executed in the engine. Three types of edges connect the graph: unconditional edges, true-edges followed if the expression of the preceeding node evaluates to true, and false-edges followed in the other case.

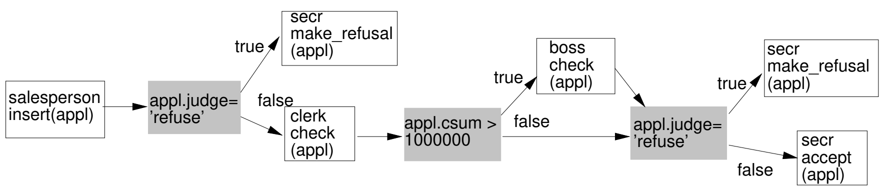

Fig. 2. Execution graph of process credit

Fig. 3 shows the two procedures, which make up the interpreter of this graph. When a workflow is initiated, the procedure `start_activity` is called. It selects the initial activity of the process and calls the procedure recursively. If the activity is not a special one (like if), the following steps are performed: the (optional) procedure defined for this activity is executed, the user is assigned. If the agent is *system*, we are finished and call the procedure `finish_activity`. Otherwise the activity is now visible in the worklist(s) of the assigned agent(s) and the (manual) execution of the activity can begin.

If a user finishes an activity, the procedure `finish_activity` is called, it checks the post-condition and starts the successor activities, if such exists. When the execution of the postcondition fails, the exception handling procedure is called.

This architecture allows the modification of processes at run-time. If a user wants to change the execution of the current process, he is able to change the execution graph of the process instance. This possibility allows the definition of ad-hoc processes and process versions and is an advantage over our first approach, where process descriptions have been compiled into active database rules [EGN94].

4 Architecture and Implementation

The architecture of Panta Rhei is based on the Web technologies. Three concepts make up the World-Wide Web (WWW): uniform addressing of information in the Internet via the Uniform Resource Locator (URL), presentation of information in the Hypertext Markup Language (HTML), and transmission of data using the Hypertext Transfer Protocol HTTP.

The HTML format [BLC95] allows the integration of different media type into a document. So-called hyper-links enable the integration and connection to other documents or media types. Important for using the WWW for workflow systems is the feature of fill-in forms in HTML, which allows a form based interaction between the user and a program.

HTTP [BLFF95] is a simple protocol for transmitting information over the net. The client (browser) requests a document from a server, by opening a socket connection and sending the URL of the document to the server. The server sends back the content of this document together with some status information. If the URL points to an executable program the server executes this programs and sends the output back to the client. The HTTP-Server calls this programs using the Common Gateway Interface (CGI). Moreover, the HTTP protocol provides a mechanism for user authorization allowing to restrict access to a group of users or hosts.

Recently, the mechanism of mobile code got significance. Small programs, so-called Applets are integrated in HTML pages and executed at the client machine. These programs enhance the functionality of the browsers and allow the implementation of arbitrary complex user interfaces. The programming

```
procedure start_activity(act)
   case type_of(act)
      process:
         start_activity(init_activity(act));
      activity:
         execute_procedure(act);
         assign_agent(act);
         if agent = system then
            finish_activity(act,0);
         end if;
      if:
         branch := execute_expression(act);
         finish_activity(act, branch);
      ...

   end case;
end;

procedure finish_activity(act, b)
   if act = null then
      return;
   end if;
   status := execute_postcondition(act)
   if status ='success' then
      if no successors of act then
         finish_activity(parent(act));
      else
         for all successors succ of act in branch b do
            start_activity(act);
         end do;
      end if;
    else if status = 'fail' then
       handle_exception(act, status);
    end if;
end;
```

Fig. 3. Execution of workflows

language Java [Mic95] is used for this purposes, most browsers allow the execution of Java Code.

Fig. 4 shows the components of the Panta Rhei system. The architecture differs from the well-known reference model of the Workflow Management Coalition [Hol95]. Here, the workflow engine is a relatively small component containing the process interpreter. All other components are directly connected to the database and make modifications in it. The workflow management system comprises the following components:

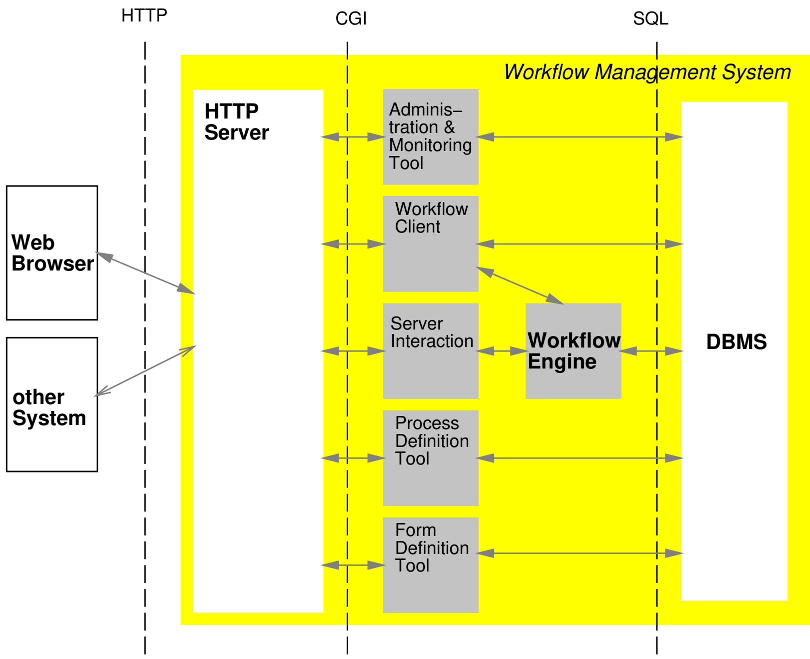

Fig. 4.

- *database:* The database contains all data relevant for process execution, process definition, organizational hierarchy, roles, as well as the dynamic data of the process instances.
- *workflow engine:* This component contains the interpreter for the defined processes, it is called whenever a process is started or an activity is finished through the user interface. Additionally, the engine comprises the advanced transaction facility and a special time component for monitoring structural and explicit time information.

  Due to the usage of the database for storing the process descriptions as well as the status information of process instances we gain several advantages: every process state is persistent, process and case data are under control of the transaction system of the DBMS, and dynamic changes of workflow specifications are possible during execution.
- *administration and monitoring tool:* It contains functions for creating, modifying and deleting users, roles, and departments. It also allows the inspection and modification of running processes, like terminating instances, reassigning steps, etc. Like the other components communicating with the HTTP Server, the interactions with the user are done by creating and receiving HTML pages and forms.

- *workflow client:* It generates the HTML pages and forms used for interaction with the user. The main page is the user worklist (see Fig. 5), which contains links to the other relevant information, i.e. the forms, process descriptions, history, etc.
  This component calls the workflow engine if the user wants to start a process or finishes an activity. Other interactions, for example viewing the worklist, access the database directly.
- *server interaction:* Activities or subprocesses can be executed on another workflow server. The system sends the required forms to the other server and the results are sent back. Both interactions are performed using the HTTP protocol.
- *process definition:* Two interfaces are available for process definition: Workflows defined as WDL scripts can be compiled and loaded into the system. Additionally, a graphical process designer has been implemented. Both components are accessible using a Web browser. The WDL description can be uploaded using an input field of type file in the HTML form. The process designer is written as a Java Applet, and therefore accessible from the browser (see left window of Fig. 5).
- *form definition:* Forms are created using a standard HTML editor. A parser extracts all input fields from the form and presents the user with a suggestion for the definition of the corresponding database table. The user can alter the data-types and creates the form table. The HTML form is stored in the database.
- *HTTP Server:* The HTTP server is the interface between the Web and the workflow system. It translates the requests from the users to calls of the corresponding procedures of the workflow system using the CGI Interface.
- *Browser:* Every interaction with the system is done by a Web browser. This allows wide availability and platform independence and made system implementation easier.

The current version is implemented in Java connecting the database using JDBC (Java Database Connection). We also use the NCSA HTTP-Server and an ORACLE 7 database. However, by using standard interfaces between the components (HTTP, CGI, JDBC) the implementation is fairly independent of a specific HTTP server or database.

5 Conclusions

We achieved our goal to find a small set of basic concepts to support a wide range of different workflow types. We validated our ideas by actually building a workflow management system. We also found it quite easy to extend the core system with some additional features like time management, transactions, and support for workflow administration and management.

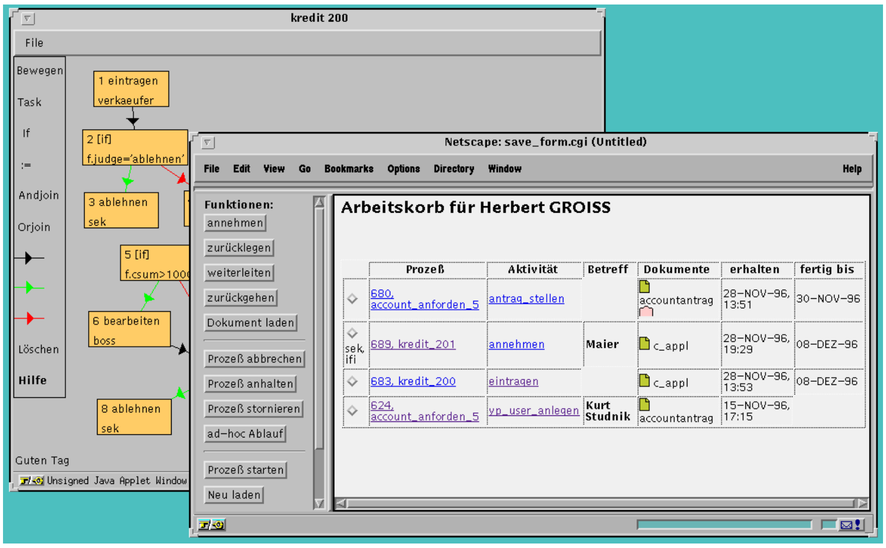

Fig. 5. Worklist and process designer of Panta Rhei

Ongoing research, therefore, concentrates on providing additional features to facilitate the application of the workflow management system in organizations. We plan to complement workflow management with resource management. By resources in this project we mean business resources like budgets, machines, work-time, etc. The goal is to manage, monitor and control the assignment of resources to processes to better support business transactions and the management of workflows. Furthermore, we intend to design a more sophisticated workflow client. It should support the user in planning his/her work and making decisions about priorities. Such a system serves as a personal productivity tool which assists the empowerment of employees and contrasts the fear that workflow systems install virtual conveyor belts through offices.

To summarize, the basic concepts for a workflow management system developed in our Panta Rhei project proved to be very useful. They led to the design of a rather clear and lean architecture and implementation. The Panta Rhei prototype system was implemented with the effort of about two person-years. It is operational and runs fairly stable. Currently it is evaluated by third parties for potential use in industrial applications.

References

[AAA+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced Transaction Models in Workflow Contexts. In

Proc. of 12th IEEE Intl. Conference on Data Engineering, New Orleans, LA, 1996.

[BLC95] T. Berners-Lee and D. Connolly. Hypertext Markup Language 2.0. http://www.w3.org/hypertext/WWW/MarkUp/html-spec/html-spec_toc.html, June 1995. RFC 1866.

[BLFF95] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol - HTTP/1.0. http://www.w3.org/WWW/Protocols, August 1995. RFC 1945.

[EGN94] J. Eder, H. Groiss, and H. Nekvasil. A Workflow System Based on Active Databases. In G. Chroust and A. Benczur, editors, *CON 94: Workflow Management: Challenges, Paradigms and Products*, pages 249–265. Oldenburg, Linz, Austria, 1994.

[EL95] J. Eder and W. Liebhart. The Workflow Activity Model WAMO. In *Proc. of the 3rd Int. Conference on Cooperative Information Systems*, Vienna, Austria, May 1995.

[EL96] J. Eder and W. Liebhart. Workflow Recovery. 1st IFCIS Int. Conference on Cooperative Information Systems, IEEE Computer Society Press, June 1996.

[EL97] Johann Eder and Walter Liebhart. Workflow Transactions. In P. Lawrence, editor, *Workflow Handbook 1997*. Wiley & Sons, 1997.

[Elm92] A.K. Elmagarmid. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, 1992.

[ELP97] J. Eder, W. Liebhart, and H. Pozewaunig. Time Issues of Workflow Management Systems. submitted for publication, May 1997.

[GE95] Herbert Groiss and Johann Eder. Interoperability with World Wide Workflows. In *1st World Conference on Integrated Design & Process Technology*, Austin, TX, December 1995.

[GE96] Herbert Groiss and Johann Eder. Integrating Workflow Systems and the World Wide Web. In P. Lawrence, editor, *Workflow Handbook 1997*. Wiley & Sons, 1996.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation. In A. Elmagarmid, editor, *Distributed and Parallel Databases*, volume 3, pages 119–153. Kluwer Academic Publishers, Boston, 1995.

[Hol95] D. Hollingsworth. The Workflow Reference Model. Draft 1.1 TC00-1003, Workflow Management Coalition, July 1995.

[JZ96] H. Jasper and O. Zukunft. Zeitaspekte bei der Modellierung und Ausführung von Workflows. In S. Jablonski, H. Groiss, R. Kaschek, and W. Liebhart, editors, *Geschäftsprozeßmodellierung und Workflowsysteme*, volume 2 of *Proceedings Reihe der Informatik '96*, pages 109 – 119, Escherweg 2, 26121 Oldenburg, 1996.

[Law97] P. Lawrence, editor. *Workflow Handbook 1997*. John Wiley, 1997.

[Mic95] Sun Microsystems. *The Java Language Specification*. Sun Microsystems, 1995. http://java.sun.com/doc/language_specification.html.

[PEL97] H. Pozewaunig, J. Eder, and W. Liebhart. ePERT: Extending PERT for Workflow Management Systems. In *First East-European Symposium on Advances in Database and Information Systems ADBIS '97 (to appear)*, St. Petersburg, Russia, Sept. 1997.