A Workflow-driven Design of Web Information Systems

Peter Barna[1]
Technische Universiteit Eindhoven[1]
Department of Mathematics and Computer Science
5600 MB, Eindhoven, the Netherlands
p.barna;g.j.houben@tue.nl

Flavius Frasincar[2]
Erasmus University of Rotterdam[2]
Faculty of Economics
PO Box 1738
3000 DR, Rotterdam, the Netherlands
frasincar@few.eur.nl

Geert-Jan Houben[1,3]
Vrije Universiteit Brussel[3]
Computer Science - WISE
Pleinlaan 2
1050 Brussels, Belgium
Geert-Jan.Houben@vub.ac.be

ABSTRACT
The World Wide Web is a rapidly growing business environment hosting a large number of business transactions. Methods for designing Web Information Systems (WIS) have adopted process models, typically as extensions of the navigation models they are based on. We observe that the structure of business processes in WIS goes beyond the scope of the navigation structure and deserves to be a more prominent aspect of the design, abstracted from the navigation specification. This paper explains the design of WIS driven by the specification of business processes of WIS that is used for the automatic generation of models describing the application logic including the navigation structure. The specification of processes is explained on a conference review and submission system and the generation process is demonstrated on the Hera application model.

Categories and Subject Descriptors
H.1.m [**Models and Principles**]: Miscellaneous; D.2.10 [**Software Engineering**]: Design; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Web-based interaction*

General Terms
Design

Keywords
Process-driven design, Web-based Information Systems, Navigation

1. INTRODUCTION
The World Wide Web has become not only the most important source of information and the largest communication platform, but also a rapidly growing business environment hosting a large number of business transactions. One of consequences is that the web interface is becoming the key gateway for participants involved in complex business processes. The traditional data and navigation-driven models are not suitable for capturing these processes, because they often go beyond the scope of (only) navigation structure. Web interfaces used by a WIS specified in a navigation model reflect the participation of users in the business process from their point of view, but do not explicitly capture the whole business process itself. For example, in a submission system tasks of reviewers and authors are different but both can be specified in an appropriate navigation model. Nevertheless, this model does not explicitly define the overall submission processing, but only sequences of actions taken by author, reviewer, etc. Missing explicit specification of the business process makes the navigation-based design and maintenance of a WIS with a complex application logic difficult. Consider, for example, a requirement changing the submission process in the existing submission system based on a navigation model. The redesign of this system would be cumbersome because the aspect that needs to be changed can be only indirectly implied from the navigation structure.

Existing methods for designing Web Information Systems (WIS) are typically data and navigation-driven, but they adopt process models in some way. Let us briefly discuss how the process models are adopted in few representative methods for WIS design. The approach described in [1] uses Business Process Modelling Notation (BPMN) graphical language for expressing process models and it is adapted to be used in WebML [2]. The process modeling phase is one of the first design phases and the navigation model is built afterwards based on the process model. The process and navigation design should be integrated according to [5] by authors of UWE and OO-H object-oriented methods. A skeleton of the navigation structure can be semi-automatically generated from the process model. Another object-oriented process modeling technique described in [6] introduces new modeling primitives (activity nodes associated with navigation classes) in the context and navigation models of OOHDM method [7]. WSDM [8] is an audience-driven method, where in the first design phase the task hierarchy is built using concurrent task trees and in the second phase the skeleton of navigation structure is generated.

As it follows from the previous text, some authors consider computer-aided generation of navigation models from process models. In this paper we propose a specification method and a procedure for automatic generation of a target model from a workflow model. The workflow model describes the business process in the system and the collaboration with users and external systems. A concrete target model defines the application logic of WIS including its basic navigation structure derivable from the workflow model. For the purpose of workflow modelling we have adopted UML use case and activity diagrams and extended them with a language for detailed specification of activity states. Our approach is explained using a conference submission system example and it is demonstrated on the Hera [4] application model as the target model. The details about the workflow modeling are explained in Section 2, and the principles of its transformation to Hera application model are described in Section 3. Section 4 concludes the paper and suggests future work.

2. PROCESS-DRIVEN DESIGN

UML offers a variety of diagrams aimed to capture dynamic properties of the designed system. Use case diagrams contain high-level description of tasks and their relationships with stakeholders (actors) of the system. Actors represent different toes of users or external systems. Activity diagrams (AD) describe the business process of a system collaborating with actors, where activity states can be distributed to tasks of actors. AD use a small set of modeling constructs and allow the description of distributed and parallel processes. Their inherit some good properties from theoretically well-founded models (Petri Nets) and combine them with easy understanding and popularity. Besides this, with small restrictions the semantics of AD is formalized in [3]. For AD aimed for WIS design we require the following:

- Every task of all actors defined in use cases (task model) must be specified in workflow(s) - it must have a swim-lane in the AD describing the WIS. Tasks of actors express the interaction of the system and its users (or external systems).
- Every AD has at least one swim-lane representing the task of the system. This task represents all process synchronization and internal data processing, the application logic hidden to users.
- Activity states in a task of an actor cannot have transitions directly to activities of another actor, they can have only transitions to system activity states. This system-centric communication assures the full control of the system over the business process.

We propose a non-graphical representation of workflows that facilitates the detailed specification of activity states and their sequencing using an abstract syntax. This language allows defining ECA rules that describe workflows and the skeleton of the workflow specification in this language can be generated from an AD and refined manually. Where AD is a good graphical representation, the abstract syntax defines the details that cannot be captured in the diagram. Such detailed specification of a workflow allows automatic generation of a target model defining WIS. Although we explain the generation process using the Hera application model (Section 3) as the target model, we believe that general principles of the described generation process can be applied universally. The phases of the process-driven WIS design are:

- Identification of actors/roles representing user groups or external systems and their tasks by means of task model (TM) depicted as a use case diagram.
- Specification of the application domain data structure in the domain model (DM).
- Specification of overall workflow of the system (or set of workflows) partitioned to tasks in the workflow model (WM).
- Automatic transformation of WM and TM into a target model describing the internal data processing and (basic) navigation structure of the designed WIS.

2.1 Conference Submission System Example

The workflow model and design concepts are demonstrated on a conference submission system example. The system should register authors, receive a submission from authors, and provide submission evaluation consisting of assigning reviewers by PC members, receiving reviews, and receiving final evaluations from PC members. We assume that reviewers and PC members are already registered in the system. The goal of the system we model is to provide every submission with proper evaluation process.

2.2 Task Model

The functional specification of the system is determined by (business) goals that must be met. These goals can be decomposed to tasks that are performed by actor's in co-operation with the system. The system has also reserved task(s). We consider the system-centric business process model, where tasks of actors contain only activities providing interaction with the system process. The mapping of tasks and actors is stored in the task model (TM) graphically represented as a use case diagram. shown for the submission system example in Figure 1.

DEFINITION 2.1 (TASK MODEL). *A task model of the system S is the tuple* $T_M(S) =< A, T, \alpha, t_S >$*, where:*

- *A is a set of roles (actors) interacting with the system,*
- *T is a set of task belonging to the roles from A,*
- $\alpha \subseteq A \times T$ *is a relation determining the mapping of actors to task specifications, and*
- t_S *is a the system task,*

The use case diagram with aforementioned restrictions can be easily mapped into TM from Definition 2.1.

2.3 Domain Model

The domain contains concepts representing submissions, evaluations, actors, and conference details. The main concept is *Paper*, users (actors) are divided to (*Author*, *Reviewer*, and *PCMember*) concepts, the evaluation part is represented by *Review* and *Evaluation*, and the general conference information is expressed by *Conference* and *Track* concepts. Figure 2 shows UML representation of the domain model.

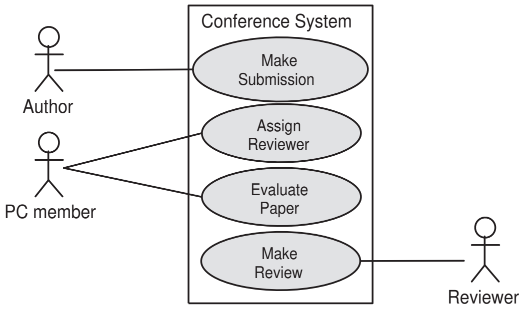


Figure 1: Task model

2.4 Workflow Model

In this section we explain the specification of processes in AD and in the abstract syntax language using the ECA rules paradigm. The business process(es) are described in the workflow model (WM) that expresses the collaboration of actors with the system in order to achieve the systems's business goals. It is a sequence of activities divided into interactive activities in the actor's tasks, and data maintenance activities and process synchronization in the system task(s). All tasks (actors' tasks and the system task) are represented by AD swim-lanes. The workflow model is AD with Labelled Transition System (LTS) semantics often used for describing reactive systems (systems that react to stimuli provided by the environment modelled by actors). The notion of workflow in Hera contains also mapping of activity states to tasks.

Definition 2.2 (Workflow). *The workflow F is a LTS represented by the tuple* $W =< S, E, G, \rightarrow, S_0, S_t, T_s, T, \tau >$, *where:*

- S *is a set of all activity states in* W,
- E *is a set of transition labels (events) in* W,
- G *is a set of boolean expressions (guards),*
- $\rightarrow \subseteq S \times S \times E \times G$ *is a set of transition relations triggered by events from* E *if guarding statements from* G *(set of boolean expressions) are satisfied,*
- $S_0, S_t \subseteq S$ *are sets of initial, respectively terminal states,*
- T_s *is a (set of) system tasks(s),*
- T *is a set of non-system (actor's) tasks, and*
- $\tau \subseteq S \times (T_s \cup T)$ *is a set of relations mapping states to tasks.*

This definition determines the structure of workflow models composed from states, transitions labelled by events, and guarding expressions allowing to model decision blocks. It includes also mapping of states to tasks. Workflows conforming Definition 2.2 can be graphically represented using AD, as the submission system workflow shown in Figure 3. This workflow represents (a part of) the submission life span

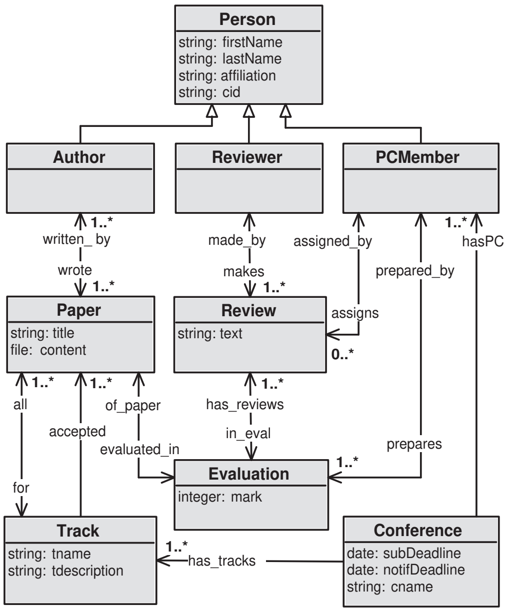


Figure 2: Domain model

from the moment of it submitting until its evaluation. We consider only the following constructs in AD: *initial* and *terminal* states depicted as circles, *activity* states depicted as ovals, *wait* blocks depicted as ovals and the label "wait", *forks* and *joins* depicted as black rectangles, and *decisions* and *merges* depicted as diamonds. Lines with arrows denote transitions and labelled rectangles represent data passed as transition parameters. Sequences of states placed between fork-join blocks represent parallel execution branches, and state sequences placed between decision-merge blocks represent optional execution branches selected in runtime based on the evaluation of guarding conditions. The workflow describing the submission processing is modelled by the AD in Figure 3. It is also possible to have composed activity states that are represented by separate workflows (not in the example). In this case the activity can be unfolded by replacing the activity state with its detailed workflow. WM can be naturally represented by sets of ECA rules in the form $e[g] \rightarrow a$, where the action a is triggered by the event e of the guarding condition g is satisfied. For the abstract syntax we use the rule representation, where the action part describes activity states, and the event and condition determine the synchronization and sequencing of activity states. For the further explanation the following shortcut notation is used: $\overleftarrow{a}$ denotes accomplishing of activity a, and $\overrightarrow{a}$ denotes start of activity a. For two subsequent activity states a and b we can write then $\overleftarrow{a} = \overrightarrow{b}$. Moreover, transition from one activity to another will be denoted as $a \xrightarrow{e[g]} b$, for two subsequent activity states it will be $a \xrightarrow{\overleftarrow{a}[true]} b$ what can be simply written as $a \rightarrow b$. The rules are in our extended AD expressed using an abstract syntax. For every rule the event, guarding condition, and a single action is defined. Besides the default events $\overleftarrow{a}$ and $\overrightarrow{a}$ denoting regular transitions between activity states external events can be considered using the *wait* states (they are also necessary for correct synchronization for joins, where all execution branches must

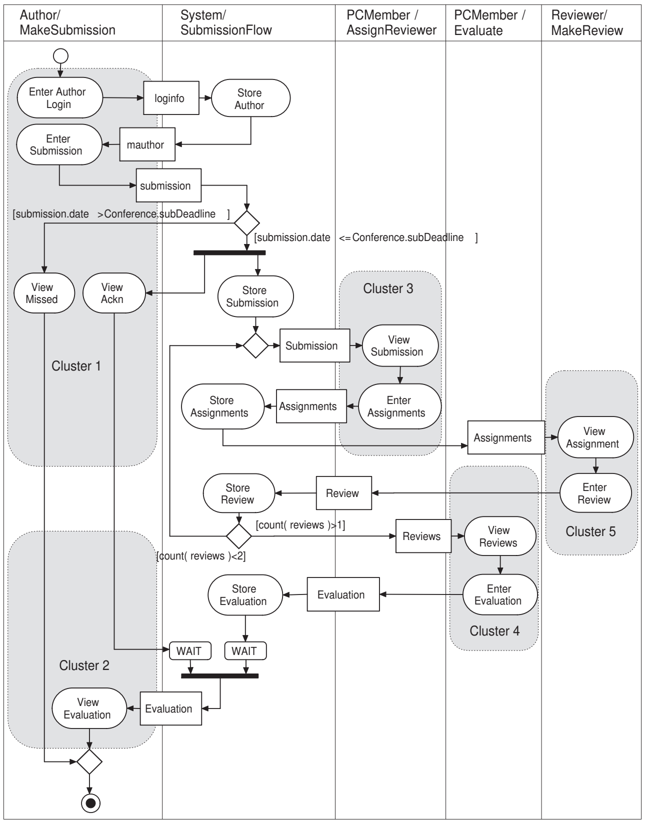


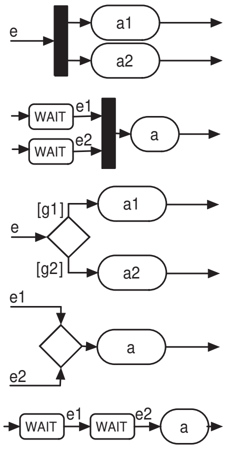
Figure 3: Workflow model

wait until parallel ones finish). For expressing ECA rules we need to introduce composed events that are expressions containing events that can occur in the system of the following types:

- *Joined event* constructed using the join operator (,) are of the form $e_1, e_2, ..., e_n$ and take place after all events $e_1, ..., e_n$ take place regardless order.
- *Sequenced event* constructed using the sequence operator (;) are of the form $e_1; e_2, ...; e_n$ and take place when all events $e_1, ..., e_n$ take place in the order their are written.
- *Merged event* constructed using the merge operator (|) are of the form $e_1|e_2|...|e_n$ and take place when at least one of events $e_1, ..., e_n$ takes place.

For correct expressing of the workflow structure by ECA rules every particular event must be in the composed event accompanied with the event guard (if different then $true$). Figure 4 shows how the basic workflow elements are expressed using ECA rules. Guards of events allow to describe decisions, and composed events allow to describe merges and joins. The action part of rules contains a detailed specification of activity states (what is happening). The following types of activity states are the default part of the WM:

- *Information observation*, wherein a particular information is served to actors by the system, so the observation activity states should be included in the tasks of actors.
- *Information entry*, wherein an information required by the system is entered by actors, so the information entry activity states should be included in the tasks of actors.
- *Information updates*, wherein the information content defining the domain and the system state is updated. The information updates activity states should be included in the system task. The information updates are further divided into:
  - *Insert operations* that provide addition of new resources (concept instances) to the existing data.
  - *Update operations* that provide updates of attributes of existing resources given by the operation path expressions.
  - *Remove operations* that remove instances of existing resources.



$$e[true] \rightarrow a_1$$
$$e[true] \rightarrow a_2$$
$$e_1[true], e_2[true] \rightarrow a$$
$$e[g_1] \rightarrow a_1$$
$$e[g_2] \rightarrow a_2$$
$$e_1[true]|e_2[true] \rightarrow a$$
$$e_1[true]; e_2[true] \rightarrow a$$

Figure 4: Expressing basic workflow elements with ECA rules

Events can carry parameters that provide information exchange between activity states. The following examples demonstrate the detailed specification of selected activity states and a rule. The first example represents the specification of the information entry activity state $EnterSubmission$ with the parameter l carrying information about registered (first) author. For the information entry states the structure of required information is described as a set of inputs (input types - selection of a set of values or text input, predefined values possibly retrieved from domain data, and input structure), and the parameters of the $\overleftarrow{EnterSubmission}$ event (what data is transferred), typically it is the complete content of the input form. The `RETRIEVE` keyword defines a query retrieving predefined values for inputs, in this case for a selection of one pre-retrieved items from many. The `RETRIEVE` keyword denotes a definition of an activity state, `FORM` denotes an information entry activity state, `TEXT`, `SELECT1` denote the input types, and `RECORD` denotes a structured item. Note that syntax of path expressions used in the activity state descriptions is based on the syntax of RDFS query language SeRQL. In our opinion the path expressions are suitable for retrieving (and updating) data structured in graph-based data models (not only RDF/S)

```
ACTIVITY: EnterSubmission(form:loginfo l)= {
    FORM submission={
        ptitle TEXT{ xsd:string },
        track SELECT1 {
            RETRIEVE {
                PATH{ {dm:Track AS T}<dm:tname>{N} },
                WHERE{ true },
                DISPLAY{ N },
                SELECT{ T }
            }
        },
        author RECORD {
            aname = l.aname TEXT{ xsd:string },
            asurname = l.asurname TEXT{ xsd:string },
        ...
        }
        *coauthors RECORD {
            aname TEXT{ xsd:string },
            asurname TEXT{ xsd:string },
            ...
        }
    }
    TRANSFERS { submission }
}
```

The following example demonstrates the observation activity *ViewAckn* notifying a user about the reception of a submission. This is decided during the transformation to a target model.

```
ACTIVITY: ViewAckn(String aname)= {
    OBSERVE {
        CONST("Dear " + aname + ",
            your submission has been registered.")
    }
}
```

Another example demonstrates the information update activity state *StoreSubmission*:. This insert operation adds a new instance of the paper and coauthors of the paper. The author data is stored earlier in the workflow by the *StoreAuthor* activity states.

```
ACTIVITY: StoreSubmission(form:submission s)= {
    INSERT    {
        {#new(dm:Paper AS P)}
             <dm:written_by>{#new(dm:Author AS CO)},
        {CO}<dm:wrote>{P},
        {A}<dm:wrote>{P},
        {P}<dm:written_by>{A}
        {P}<dm:for>{dm:Track AS T},
        {T}<dm:all>{P},
        {P}<dm:title>{title};
           <dm:content>{file},
        {CO}<dm:firstName>{cofname},
           <dm:lastName>{colname},
           <dm:affiliation>{coaffil},
        {A}<dm:firstName>{afname},
           <dm:lastName>{alname},
           <dm:affiliation>{aaffil}
        WHERE    {
            title = s.ptitle AND
            file  = s.file AND
            FOR{ coauthor IN s.coauthors AS }
            {
                cofname = coauthor.aname,
                colname = coauthor.asurname,
                coaffil = coauthor.affil
            } AND
            afname = s.author.aname  AND
            alname = s.author.asurname AND
            aaffil = s.author.affil AND
            T = s.track
        }
    }
}
```

The complete rule encapsulating the *ViewSubmission* activity state is triggered by a composed event. Note the composed event and a condition that uses the `RETRIEVE` block to retrieve data needed for the guarding condition evaluation. The operator `finished(activity)` has the same meaning as $\overleftarrow{activity}$. The `PARAMETERS` section defines how possibly different parameters of the merged event are mapped into the parameter that is used in the action.

```
RULE Rule_ViewSubmission = {
   ON { finished(StoreSubmission) AS s} IF { true } |
   ON { finished(StoreReview) AS r}
   IF { c > 1
     RETRIEVE {
        PATH { {REV}<dm:in_eval>{dm:Evaluation}
           <dm:has_reviews>{dm:Review AS REVS}}
        WHERE{ REV = r.review }
        SELECT{ count(REVS) AS c }
     }
   }
   PARAMETERS { p = s.submission | r.submission }
   ACTION {  ViewSubmission( p )  }
}
```

3. GENERATING A TARGET MODEL

In this section we explain the generation of a target model from WM. As example target model we consider the Hera application model (further only AM). Note that a single instance of a target model (AM) presents multiple instances of a workflow. For example, considering the submission system - a PC member within the activity state *ViewSubmission* realized as an instance of a navigation unit would rather see all new submissions (all new instances of the workflow) than only one. The difference between instances of workflow and appropriate navigation structure requires additional data processing facilities realizing message queries explained in the next section. The generation of target models consists from the following steps:

- Activity states contained in the system task in WM constitute the data processing aspect of a target model(s). The appropriate rules are transformed to data manipulation queries and appearance conditions in AM.
- The data processing specification in the target model is extended with service facilities providing user identification and presenting of multiple workflow instances in the web interface. In AM these facilities are implemented as data retrieval and manipulation queries.
- Actor tasks are partitioned to clusters in such a way that a single cluster represents a set of activities that can be performed by a concrete user without waiting for another user (or human/external actor).

- Information observation and entry activity states are transformed into the navigation structure in a target model. Using the activity clusters from the previous step the structure of navigation units (NU) (slices) is generated in AM.

3.1 Generating Data Processing Facilities

The data processing includes all data manipulation operations, decisions, and activity synchronization. In WM it is represented by a set of information update ECA rules that communicate with user activities through events.

3.1.1 Context Data and Asynchronous Messages

Let us call messages those events that are sent by the system to the actors tasks (labelling the transitions from system activity states to actors activity states) and back. It is an appropriate name, because these messages are exchanged between the system and its environment, and they often carry information. This information is presented to users or it is entered by users. Most of users tasks (or their regions) represent the transformation of incoming messages to outgoing ones, where some additional information (a value in business terms) is added. For example, a PC member receives a submission information, adds assignments of reviewers to it, and sends it back to the system. Note that these messages are asynchronous, because (human or external system) recipients might not be ready (logged to the system) at the time messages are sent. Furthermore, since there are possibly multiple instances of the workflow running at the same time (e.g. multiple submissions are processed), also multiple instances of the same type of messages can be sent. This classical problem of asynchronous communication is obviously solved by using message queues automatically maintained by the workflow system (Hera web server according to specifications in generated AM). An example of the queues is shown in Figure 5. The context data model (CDM) determines the structure of information needed for user identification, workflow instance identification, and message queueing. The context data is a persistent and updatable. The simplified structure of CDM is in Figure 6. There are multiple $Message$ and $MQueue$ classes generated according to the workflow structure - for every message one. The detailed structure of $Parameter$ is also based on the structure of event parameters from the workflow specification. $Queue$ classes are singletons generated initially when an AM is deployed and $Message$ instances are determined by AM instances maintained at runtime. The instances of a workflow are determined by instances of a concept from DM, in the case of the submission system it is the $Paper$ concept.

For explain facilities added to the target model (AM) the following abstract operations working with users, messages, and message queues are used:

- $boolean$: $find(user : U)$ finds out if the user U is registered. The user can be identified based on is name password, etc.
- $register(user : U, role : R)$ registers a new user
- msg : $get(queue : Q, user : U)$ returns a first appropriate (matching a concrete user U) message from a $queue$ and at the same time removes the event from the queue
- $put(queue : Q, msg : In)$ inserts a message In to the $queue$
- $boolean$: $empty(queue : Q, user : U)$ check if the queue Q contains some messages for the user U

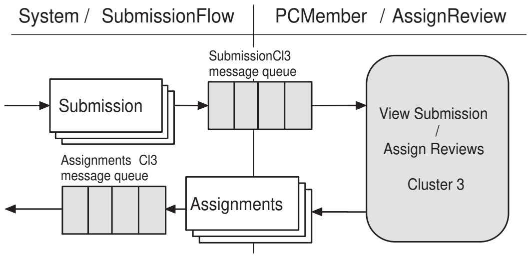


Figure 5: Message queues for multiple workflow instances

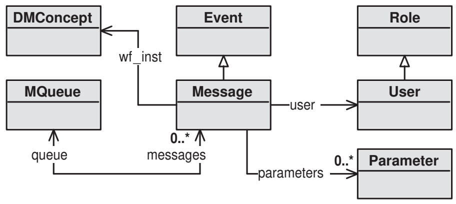


Figure 6: Context Data Model

The operations are transformed to CDM data retrieval and manipulation facilities (in Hera AM to SeRQL queries). The following queries are added for handling the following events to Hera AM:

- The system sends a message M to a user (his AM instance). A query is added to AM every queue providing the operation $put(Q, M)$. The identification of the user is part of it (the message carries information about the target user).
- A user logs in. Two user identification queries are added, one for detecting if the user is registered using $find(U)$, another for registering using $register(U, R)$. All queues belonging to the user and his tasks are also checked if they contain waiting messages using $empty(Q, U)$. This information is used for adapting AM to provide access of the user to his unfinished tasks (or activity clusters). The waiting messages are also retrieved from input queues by $get(Q, U)$.
- A user logs out. In this case two sets of queries are added to AM. The first set is generated for every incoming (from the point of view of users) message queue that puts unfinished activity clusters by $put(Q_{In}, M_{In})$ back to the queues. The second set of queries is generated for all outgoing queues representing finished activity clusters that are put into an appropriate outgoing queue $put(Q_{Out}, M_{Out})$.

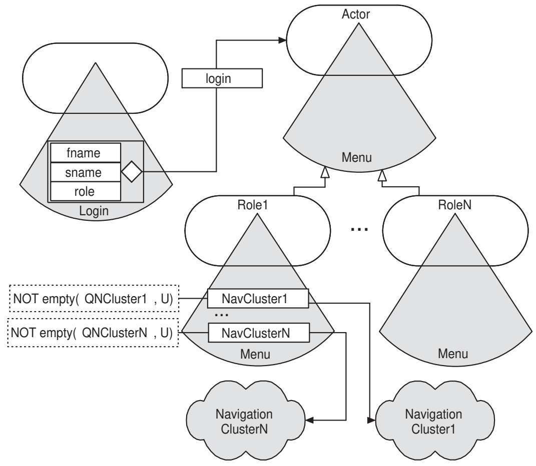


Figure 7: The default navigation structure generated from a workflow

As it follows from the previous explanation the state of every workflow instance (implemented as an AM instance in Hera) is given by the content of queues where the system puts its messages. If all queues where the system puts messages are empty, the system is not waiting for any message from users and then there is no running workflow instance (or it is only under system control). The exceptions are user messages that initiate a new workflow instance, in the case of the submission system it is the $loginfo$ messages from the $Author$ role.

3.2 Generating Navigation Structure

All tasks of actors are transformed into portions of the overall navigation structure. Tasks are divided into regions called activity clusters (AC) that can be performed in a single session, and the state of which is determined by the input message queues. The universal structure of generated AM contains the login unit (slice), a menu allowing personalized access to AC for concrete users, and a set of navigation structures (navigation clusters) representing AC itself. An AC is a group of activities that does not depend on activities of another users, so called blocking activities (because aforementioned asynchronous communication). Clustering of tasks is in Figure 3 shown as gray ovals in swim-lanes, and it is explained in Section 3.2.1. Clusters are used for building the first iteration of the navigation topology. AC are mapped into navigation clusters (groups of navigation units (slices)) explained in details in Section 3.2.2. Figure 7 shows an example of the default navigation structure generated from a workflow specification. Access to navigation clusters representing AC is determined by conditions attached to menu items corresponding to particular clusters. The conditions check the message queues if they contain any new messages representing new particular tasks as it has been explained previously.

3.2.1 Partitioning Tasks

For partitioning we consider only tasks assigned to actors, not the system task(s). It is important to realize the actors are humans, or external systems in which we do not have any information about synchronicity (in this case promptness) of their responses. A response in this case corresponds to the time in which a particular message is sent to the system from an actor. For example, this is the time between $\overrightarrow{ViewReviews}$ and $\overleftarrow{EnterEvaluation}$. For defining task partitioning we need to introduce the event ordering.

Definition 3.1 (Event Order). *Let F be a workflow as it is specified in Definition 2.2 with the set of events E. The relation $\trianglelefteq : E \times E$ determines the order of events in the course of execution. For two subsequent events $e_{pre}, e_{succ} \in E$ holds: $e_{pre} \trianglelefteq e_{succ} \leftrightarrow \exists 1 \leq i < n, a_i \in S, e_i \in E : a_i \xrightarrow{e_i} a_{i+1}$ where $e_1 = e_{pre}, e_{n-1} = e_{succ}$*

It can be shown that the relation $\trianglelefteq$ is a partial order of the set E (it is reflexive, antisymmetric, and transitive). If for a particular workflow model two events are in the order $e_1 \trianglelefteq e_2$ then in any workflow instance of this model e_1 occurs before e_2. Note that not all pairs of events in a workflow must be in this relation, for instance events labelling transitions in parallel execution branches can not have this relation. Partitioning actor's tasks to AC that are independent on interaction of another (slow) actors with the system is the first step of reconstitution of tasks towards the navigation structure. The clusters are then transformed to primary NU that can be later reconfigured.

Definition 3.2 (Activity Cluster). *Let $T_A(a) = \{s | \exists t : (s,t) \in \tau \wedge (a,t) \in \alpha\}$ be a set of activity states in tasks of a single actor a for a task model $< A, T, \alpha, t_S >$ and a task workflow $< S, E, G, \rightarrow, S_0, S_t, T_s, T, \tau >$. Activity cluster is a set of activity states $C \subseteq S$ such that $\forall a, b, c \in C, \overrightarrow{a} \trianglelefteq \overleftarrow{b} \trianglelefteq \overrightarrow{c} \longrightarrow a, b, c \in \{T_A(a) \cup T_A(system)\}$.*

Definition 3.2 says that an AC is a set of activity states, where any three states in the cluster that must be performed in an order must be performed by the same actor, or the system. This property ensures that no activity state in a cluster of an actor follow after an activity state of another actor. This means that performing activities in a cluster is independent on collaboration with another actors. The clustering procedure traverses the workflow structure for activities for all tasks and tries to detect maximal (largest possible satisfying the property from Definition 3.2) AC. The resulting clusters for the submission system workflow are shown in Figure 3 as grey ovals.

3.2.2 Generating Navigation Clusters

Navigation clusters (NC) are parts of AM that represent the navigation structure realizing AC. The internal structure of NC mimics the sequences of activity states in corresponding AC. However there are possibly multiple (heuristic) strategies to achieve reasonable results for automated generation of NC. One of the strategies (the default one presently used) is further partitioning of AC to units. These units then are transformed to navigation units using a one-to-one mapping. Concretely, the strategy is based on the following integration rules:

- Integration of subsequent information observation and information entry activities to one unit (slice) based on the common fact that information entry usually follows after information observation (user first must have some information to decide). It can be done for $a_1 \in A_O, a_2 \in A_E \wedge a_1 \rightarrow a_2$, where A_O is a set of all information observation activity states within the same task, and A_E is a set of all information entry activity states within the same task, and

- Integration of subsequent information entry states to one unit (slice) can be done for $\forall a_i \in A_E, a_i \rightarrow a_{i+1}$ and can be restricted by a maximal number of inputs.

These two rules represent a very simple but useful strategy for generation of navigation units (slices). The first integration rule would be used for clusters 3,4, and 5 from Figure 3. Because of the lack of space we show only AC "Cluster 3" transformed to the navigation unit $Cluster3$ depicted in Figure 8 using the Hera AM graphical notation. The $ViewSubmission$ activity is transformed to the $Cluster3$ slice with the $Paper.AssignRevForm$ slice presenting all new papers (submissions). They are stored as parameters of messages in the queue $SubmissionCl3$ (see Figure 5). The abstract operator $get()$ for retrieving messages from queues is implemented as the retrieval query of the slice $Paper.AssignRev$ and removal query $RemoveSubCl3$ triggered when the $Cluster3$ slice is instantiated (it removes retrieved messages from the queue). The paper title is used for retrieving the paper and author information from the domain database. The activity $EnterAssignments$ is transformed into the $Reviewer$ form allowing selection of reviewers. $Assignmetns$ is a service query that puts all papers in the list that have not yet assigned reviewers back to the $SubmissionCl3$ queue and puts the $Assignment$ message with the filled form instance as a parameter into the $AssignmentsCl3$ message queue (see Figure 5). It also triggers another processing represented by subsequent queries ($StoreAssignemnt$, etc.). The $Nav_{P}anel$ form contains a button for logging of triggering the $logoff\_Cluster3$ query is a service query described in Section 3.1.1 for logging off, shortly it puts all retrieved submissions (papers) back to the $SubmissionCl3$ queue (no assignment was created). The following SeRQL query is generated as the retrieval query of the $Paper.AssignRev$ slice:

```
SELECT P FROM {SubmissionCl3}<cdm:messages>{M},
              {M}<cdm:user>{U},
              {M}<cdm:wf_inst>{P},
              {U}<cdm:id>{uid}
WHERE uid=$user
```

The $\$user$ is a session parameter representing the current (logged in) user, and the P variable stores the paper instance that is the message parameter.

4. CONCLUSION

The contribution of this paper is twofold: it describes a language for detailed specification of workflows modelling business processes of WIS, and it explains the automated generation of a target model describing the application logic of the WIS. Although we briefly show this generation process for a Hera-compliant AM, we believe that it is possible to build programs implementing the generation procedure for arbitrary modeling methods, or directly generate source code for any (programming) language implementing web applications. In our current work we particular considered the extension of other forms of communication with actors than the web interface, including communicating with external systems using web services. Moreover, we think it would be beneficial to build a visual tool facilitating the construction of WM and steering the generation process, possibly for different target models.

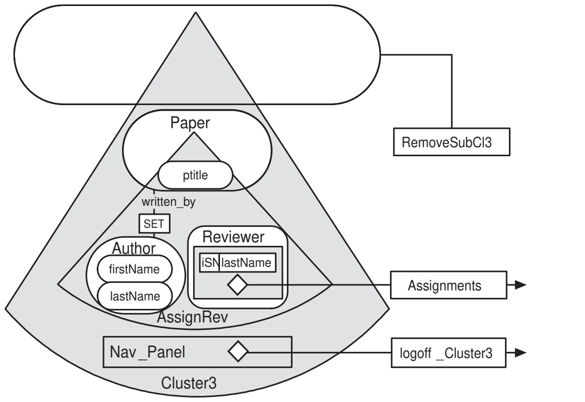


Figure 8: The example NU representing an AC ("Cluster 3")

5. REFERENCES

[1] M. Brambilla, S. Ceri, and P. Fraternali. Process modeling in web applications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2006.

[2] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (webml): a modeling language for designing web sites. In *9th International Conference on the World Wide Web (WWW9), Amsterdam*, 2000.

[3] H. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling, PhD thesis.* CTIT PhD Thesis series, 2002.

[4] G.-J. Houben, F. Frasincar, P. Barna, and R. Vdovjak. Modeling user input and hypermedia dynamics in hera. In *ICWE 2004, International Conference on Web Engineering*, 2004.

[5] N. Koch, A. Kraus, C. Cachero, and S. Melia. Integration of business processes in web application models. *Journal of Web Engineering*, 3:22–49, 2004.

[6] G. Rossi, H. Schmid, and F. Lyardet. Engineering business processes in web applications: Modeling and navigation issues. In *Third International Workshop on Web-Oriented Software Technology*, 2003.

[7] D. Schwabe, G. Rossi, and S. D. J. Barbosa. Systematic hypermedia application design with OOHDM. In *Hypertext '96, The Seventh ACM Conference on Hypertext, Washington DC*, 1996.

[8] O. Troyer and S. Casteleyn. Modeling complex processes for web applications using wsdm. In *Third International Workshop on Web-Oriented Software Technology*, 2003.