A MODULAR EVENT-BASED ARCHITECTURE FOR WORKFLOW SYSTEMS

Mehran Najafi and Kamran Sartipi
Department of Computing and Software, McMaster University, Canada
{najafm, Sartipi}@mcmaster.ca

ABSTRACT
Real life work processes are dynamic and much richer in variation to be expressed by typical static workflow models. Two conflicting goals to be addressed include: flexibility to handle changing situations, and simplicity to design workflows that can be understood and implemented efficiently. This paper addresses these two issues by introducing a novel architecture for workflow systems. In this architecture, a workflow is defined in terms of modules and templates. Modularity provides simplicity and reusability for a workflow system. In our approach, problem independent modules are adopted for a special workflow system by placing them in problem dependent templates. Also different users in the system can interact with the workflow to make appropriate changes via events. Modules and templates have their own event handlers that allow them to be modified based on changes in user requests or situations. We illustrate these features in our architecture using two case studies, one in healthcare domain and another in a banking system.

KEY WORDS
Modelling; Flexible Workflow; Event-based Architecture; Modular Architecture; Intelligent Workflow.

1. Introduction

A workflow is a set of ordered steps consisting of tasks, data and resources (human or machine) in order to achieve a goal [1]. A workflow management system (WFMS) is a system that defines, creates and manages the execution of workflows on one or more workflow engines [2]. The system is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of other applications.
Workflows in a WFMS can be subject to change due to different reasons, such as unforeseen errors, faults and failures, new development, or administrative decisions (e.g., process optimization, incorporation of new resources, and modification of existing product lines). These reasons may cause different types of workflow changes, such as addition of a new task, deletion of an existing task, and modification in order of tasks. Workflow systems need mechanisms to make them flexible and deal with those deviations.
In a small system, it is reasonable to show a workflow by a dataflow diagram of single tasks. However, in a more complex system, modeling workflow by using only dataflow constructs can quickly lead to overly complex workflows that are hard to understand, reuse, reconfigure, maintain, and schedule. Modularization is a software design technique that divides a large system into a number of smaller and manageable parts, namely *modules* [3], [4]. Each module is designed individually where different modules can be combined to form a larger module. One of the most advantages of modularization is its reusability.
In this paper, reusability and flexibility in workflow systems are the main objectives. We address reusability by introducing modules and workflow templates. Modules are problem independent and can be used in different types of workflows. On the other hand, workflow templates are designed for individual processes and contain customized modules. In our architecture, we use *events* and *event handlers* to provide flexibility and dynamism for the workflows. Event handlers of modules and workflow templates are triggered by events and can make changes in the order or content of steps in a workflow.
Also, we introduce a consistency checking mechanism that prevents any changes that may invalidate the workflow. This consistency checking would become very important when we allow the users to design events and event handlers.
The organization of this paper is as follows. Related work is discussed in Section 2. The proposed architecture and its details are discussed in Section 3. Two case studies in the healthcare domain and banking system are explained in Section 4. Finally, the conclusion and future work are discussed in Section 5.

2. Related work

Current research in workflow systems concentrates primarily on making the workflows flexible with respect to the changes in processes.
The dynamism in workflows has been studied using exception handling techniques [9], [10] and [11]. In these approaches a workflow is designed in a regular style and any changes in a workflow system, presumed or un-presumed, is considered as an exception and is handled by using one of the exception handling methodologies. The main difference between our proposed architecture and these approaches in the case of change handling, can be addressed by differences between "events" and "exceptions". An exception is usually raised to signal exceptional situations like errors and failures, while in an event based system we concentrate on interaction between users and systems in all situations. In other words, our

event based architecture receives all possible and related events from the users, and the system evolves accordingly as the process progresses. Our architecture supports exceptions as a special type of events.

A mathematical model for dynamic structural change algorithm in workflow systems is introduced in [8]. In this method changes are divided into two categories: changing the structure of the workflow procedures, and changing the workflow in the middle of executing procedures. This model seems to have scalability problem and hence would not be applicable for complex workflows.

Dividing a workflow into independent parts (namely components) to make it easy to understand and reuse, has gained more attention recently. In [13] the problem of combining dataflow and control-flow is addressed for scientific workflows. Generic behavioral specifications (i.e., controlflow) in workflows are encapsulated in templates. Templates are separate components and thus can be easily reused in other workflows. Also templates can contain subcomponents. The notion of template in this approach is different from our definition of template. We consider two types of components: problem independent component as module and problem dependent component as workflow template. Workflow templates are designed for specific organizational activities while modules are reusable in different workflows.

Due to the variety of changes that can be applied on a workflow, correctness and verification of workflows are crucial. During the changes, a workflow must not enter an incorrect state after a predicated or unpredicted change. In [9] a foundation set of constraints for flexible workflow specifications are developed. The authors claimed that the constraints should optimally balance the flexibility and control requirements. However, this method and other methods that attempt to find process model for workflows [10] have introduced overheads since they need to import all modified nodes in the workflow. In our architecture, by equipping each module with simple constraints, the workflow will be guaranteed to be valid after authorized changes.

In the proposed architecture that will be discussed in the next section, we model each workflow based on its relevant events. Combination of events and modularization provides flexibility, reusability and consistency in workflows.

3. Proposed Architecture

In this section, we present an overview of our proposed architecture with focus on the relevant issues and the dynamism of modules and templates. Further, we discuss events and event handlers and different types of constraints in the architecture. Finally, we elaborate on workflow consistency.

3.1 Architecture Overview

Figure 1 illustrates different parts of the proposed workflow architecture.

- A *workflow* represents a set of modules with predefined orders among them. A workflow is modeled by a *workflow template* that is a collection of customized modules for a specific organizational operation. A workflow template (or simply a *template*) also establishes data flow and control flow connections among the template's modules.
- A *module* is a container for a set of tasks whose order of executions can be changed by receiving events. Modules are designed individually and are maintained in the *module repository* containing a set of reusable modules. A module can be placed either in a template as a part of the current workflow solution, or in a module repository to be reused. Each module has a collection of *shared data* that maintain the state of the module.
- An *event* is any information that is provided by the environment (user or system) and can change the current workflow solution.
- An *event handler* is a program that receives events from the environment and modifies the current tasks (or modules) within a module (or template) by consulting the state of the module (or template) accordingly. In general, a module or template contains several event handlers. An *event table* presents the relevant events in the architecture along with their corresponding event handlers.
- A *task* is an atomic entity that represents an indivisible functionality that is performed within a module. A task can have optional constraints in terms of *pre-conditions*, *post-conditions*, and *invariants* on its operation. The execution of a task may change the state of the module through module's shared data. This state is used by the event handlers to make changes in the execution order or the number of the module tasks.

A template uses its event handlers to customize its modules or import new modules from the repository to use for the modified workflow solution, provided that the imported modules preserve the template or module's consistencies. Similarly, the deleted modules from the template are returned back into the repository. The WFMS uses the event handlers of the modules and template to generate a flexible workflow. The operations of the WFMS for this architecture are as follows.

- Initialize the workflow template by using a number of modules in a predefined order (according to the default or general case of the workflow.)
- Receive events through interaction with the users and the environment.
- Change the current workflow by adding, removing, or reordering the modules in the template, using the template event handlers.
- If some changes are needed within a module, the template transfers the event to the proper module, where the module event handlers make appropriate changes on the internal workflow of the module. The module event handler can add or remove a task in a module, as well as reorder the tasks.

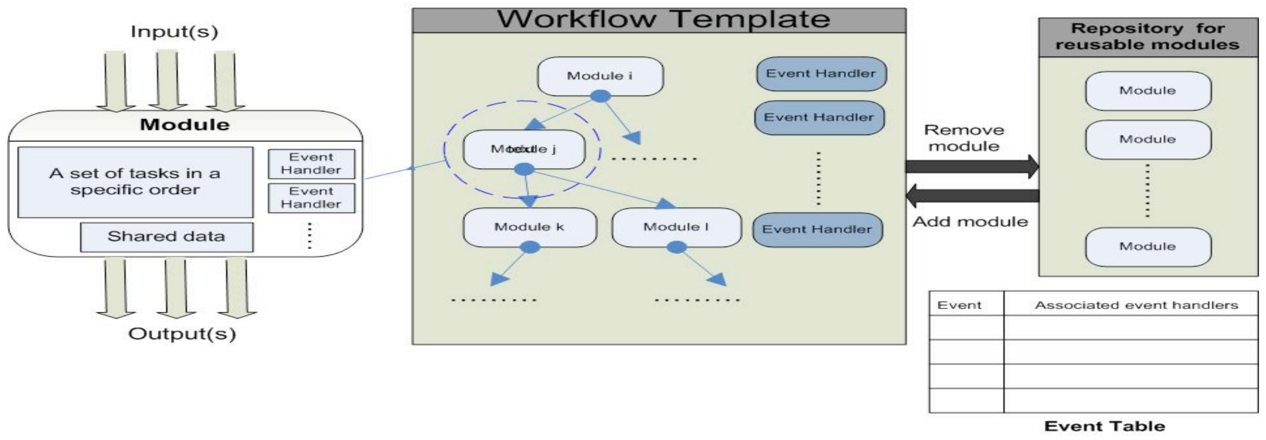

Figure 1: Different parts of the proposed event-based workflow architecture.

The major difference between event handlers corresponding to a module and to a template is the level of abstraction, where the former deals with details of an operation, and the latter adds or deletes major business functionality.

3.2 Workflow dynamic behavior

A template acts as a coordinator for the workflow modules by importing, re-ordering, and deleting modules from the workflow. Consequently, it provides flexibility and dynamism for the generated workflow. The workflow evolves through receiving events and adjusting itself to satisfy the requested operation by the events. This is achieved through a set of constrains for modules and tasks that allow the WFMS to modify the workflow ingredients and to keep the workflow consistency. These constraints are as follows.

1. **Precondition**: conditions that must be met before execution of a module or task.
2. **Postcondition**: conditions that are guaranteed to be met after the execution of a module or task.
3. **Invariant:** constraints that control the correct execution of a module or task.

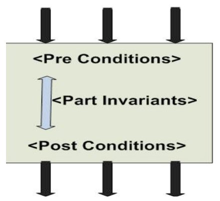

Figure 2: Constraints for tasks and modules.

A sequence of modules in a template (or tasks in a module) is called valid, if it preserves the constraints that are defined for each module (or task) based on their specification. Figure 2 presents the above constraints in a module. The case studies in Section 4 provide examples on the use of constraints.

Events are generated by the environment in the form of specific information. These events trigger the module (or template) event handlers, where the event handlers decide whether the required action should change the order of the modules within the template (i.e., high-level modification) or change the order of tasks within a module (i.e., low-level modification). In the first case, the event is consumed by the template event handler, whereas in the second case the event is forwarded to the corresponding module event handler to react upon it. The template events are categorized into two types:

- *Specific transaction,* where the transaction (as event) refers to a valid and high-level operation in the corresponding business process, e.g., a specific mortgage package request in a banking system.
- *Module-level*, which will be directed to the corresponding module as defined below.

Similarly, the module events are categorized into three types:

- *User interaction*, represents the general case where a user intends to change the workflow based on their choices and preferences. For example, when a patient could choose between a daily tablets and two monthly injections.
- *Exception*, in the form of unexpected input data to a module that causes the corresponding event handler to change the module workflow and use specific tasks to handle the situation. For example, when a patient gets sick by another type of disease in the course of treatment for his/her current disease.
- *Reflexive task*, refers to the event that is caused by the environment's reaction to the result of a task execution. For example, in the task of prescribing a medicine by a physician, each side effect of the medicine is considered as a reflexive event where the assigned event handler catches the event and modifies the treatment workflow.

In the proposed architecture most of events are of type 'user interaction' and hence they are suitable for domains that many factors can change the general workflow (e.g., healthcare domain as it is a dynamic environment).

The dynamic behavior of the proposed workflow mechanism is modeled in the form of ``*event [condition] / action*'' which allows the workflow to change its state and evolve. In Figure 3 ``event'' is a template event or a module event, as defined above. A *condition* is modeled as ``pre condition”, ``post condition” or ``invariants” and is evaluated by comparing the state of the module (or task) before and after the modification by the event handler.

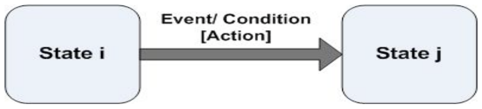

Figure 3: Event handling in the proposed architecture.

The *action* is the modification of the workflow that conforms with the module or task's constraints. In other words, if the condition is true, the change is accepted and workflow evolves to the new state, otherwise the change is ignored and the workflow remains in its current state. The possible changes to a workflow according to a ``template event” are as follows:

- *Adding* a module to the workflow from the repository of the reusable modules.
- *Removing* a module from the workflow (a copy of the module still exists in the repository).
- *Changing* the module interconnections (dataflow and control flow) with other modules in the workflow.
- *Delivering* the module events to the proper modules so that they can modify themselves accordingly.

Similarly, the possible changes to a module according to a ``module event” are as follows: i) introducing a new task to the module or deleting an existing task; ii) changing the execution order of tasks; iii) changing the description of tasks, and; iv) sending events to the workflow template to request modifications in other modules.

3.4 Consistency in the architecture

It is important to ensure a workflow remains valid after a change. Consistency for this architecture is defined as the correct positions of reusable modules in the template as well as preserving the main goals of the workflow through modifications. In this architecture, we confine changes in workflow to modifications that are done by event handlers. Each event handling is defined in terms of conditions and actions; i.e., if the condition is true for a modification it will be performed. For each type of possible modification, we define specific conditions. These conditions are in terms of architectural constraints in order to preserve the workflow consistency. Consistency checking consists of data flow and control flow checking for templates and modules. Data flow represents the flow of data in the workflow and if the connections between modules and template are complete data flow consistency is satisfied (i.e., other modules or the template provide all modules’ input data). If a part of input data for a module is eliminated by a modification (i.e., delete or change order) the event handler must provide its input via other sources in the template.

We assume workflow constraints are true before applying any changes; also adding reusable modules to an empty template can generate an initial workflow.

Control flow consistency checking follows the data flow consistency checking. Control flow conditions for templates that include adding a new module, deleting a module and changing the order between modules are illustrated in Figure 4. Sending and receiving events do not change the structure of workflow and because the workflow is valid before this action, the workflow will be valid after it too. Hence, no consistency checking is required. Control flow consistency checking within the modules (i.e., adding, removing, and changing the execution orders of tasks) are similar to those discussed for the workflow template. As a general rule, the module invariant must be satisfied after any changes.

4. Case studies

In this section, we present the application of our proposed approach on modeling two cases of healthcare and banking domains. Also some advantages of our architecture in comparison with previous ones are discussed in this section.

4.1 Case study 1: healthcare domain

The healthcare domain is an active area for developing and using workflows [6], [7]. A simple workflow for treatment of a muscle disease is shown in Figure 5 (top). This treatment consists of two parts: Step 1: *diagnosis and drug prescription*, in which the physician prescribes drug X for the patient. Step 2: *physiotherapy exercises*, as the follow up treatment after treatment 1. Each of these two treatments can be considered as a separate module. Their module specifications are illustrated in Table 1. Also, each task may have its own constraints. An example of task pre-condition is shown in the second task of the physiotherapy treatment. In Table 1, a number that is assigned to each event handler identifies their corresponding event.

The nature of each treatment imposes some obligations or considerations. For example, most drug prescriptions (i.e., task execution) have side effects on the patients and hence the physician may decide to change the treatment method by prescribing a different drug without those side effects.

Workflow consistency checking: According to Table 1 some of these consistency issues are as following:

- Since the pre-condition of module ``treatment by physiotherapy” is satisfied by post-condition of module ``treatment by drug X”, this combination is consistent.
- In module ``treatment by drug X” the event handler EH1 for event ``fever” is consistent if the dose of drug X prescribed by physician is between the LOWER and UPPER limits and also the ``new prescribed duration for drug X” is according to the pair (duration-of-applying-X, dose-D) in one of the tasks, otherwise this change will not be applied on the workflow.

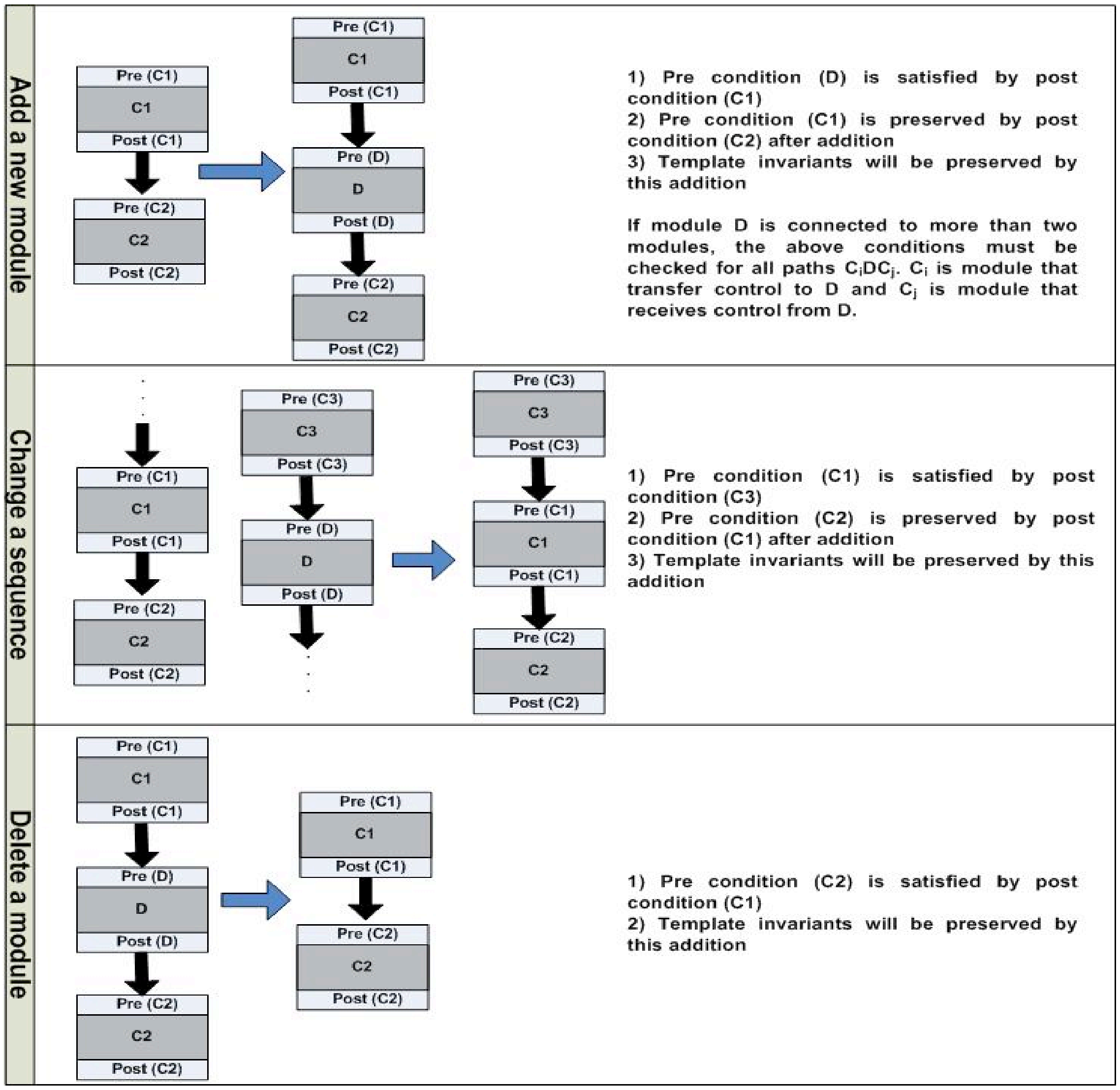
Add a new module
Pre (C1)
C1
Post (C1)
Pre (C2)
C2
Post (C2)
Pre (D)
D
Post (D)
1) Pre condition (D) is satisfied by post condition (C1)
2) Pre condition (C1) is preserved by post condition (C2) after addition
3) Template invariants will be preserved by this addition
If module D is connected to more than two modules, the above conditions must be checked for all paths C_iDC_j. C_i is module that transfer control to D and C_j is module that receives control from D.
Change a sequence
Pre (C3)
C3
Post (C3)
1) Pre condition (C1) is satisfied by post condition (C3)
2) Pre condition (C2) is preserved by post condition (C1) after addition
3) Template invariants will be preserved by this addition
Delete a module
1) Pre condition (C2) is satisfied by post condition (C1)
2) Template invariants will be preserved by this addition

Figure 4: Control flow constraints for workflow template

- For event ``strong headache" and its event handler EH2, because the pre-conditions of module CT-scan are satisfied, this module can be added to the workflow. Figure 5 (bottom) illustrates this modification.
- In module ``treatment by physiotherapy" the event handler EH1 for event ``high heartbeat rate" is consistent, because it decreases the time of vigorous workout and increases the time of the moderate workout, which is according to the module invariant.

Therefore, each modification in the workflow is consistent and the workflow remains consistent and valid.

4.2 Case study 2: banking domain

In the previous case study, we focused on "satisfying constraints" and the mechanism of applying changes on the workflow in order to keep it consistent. However, in the current case study, we present how templates and modules can collaborate through using events and event handlers, in the context of a complete workflow for the banking domain.

This workflow describes the process for applying, receiving and paying back of a house mortgage. The relevant modules to this workflow are defined in Table 2. These modules are general and can be used in any workflows in this domain or other domains. The template for this workflow is shown in Figure 6.

4.3 Advantageous of the proposed architecture

Although these two case studies can be modeled with other approaches, but the proposed architecture seems to be more appropriate. Some of its advantages are:

1) Where several factors exist that can change a workflow (e.g., healthcare domain), flexibility is inevitable. Design of a static workflow and apply possible changes through exceptions [5] can be one alternative to our architecture; but in the case of a large number of changing factors, (e.g., in a surgery operation) a lot of added exceptions complicate the workflow and reduce its understandability.

Table 1: Module specification for two treatment methods

Module name: Treatment by drug X	**Module name: Treatment by physiotherapy**
Pre condition: • Physician diagnoses diseases A or B or C for the patient **Tasks:** **1.** Prescribe drug X with dose D1 for N1 weeks (or) **2.** Prescribe drug X with dose D2 for N2 weeks (or) **3.** Prescribe drug X with dose D3 for N3 weeks **Post condition:** • The patient must not have pain in muscles **Module invariants:** • LOWER LIMIT < dose of X< UPPER LIMIT • The pair (duration-of-applying-X, dose-D) must match with one of the pairs defined in Tasks 1 to 3. **Events:** • Each side effect of drug X is considered as an event: o E1: fever o E2: strong headache **Event Handlers:** • EH1: weaken the dose of X and increase the duration of using it in the current task • EH2: ask for a CT-Scan via adding its module into the template	**Pre condition:** • The patient must not have pain in muscles **Tasks:** 1. Moderate workout for M1 minutes **2.** [Pre: the moderate step must be done] Vigorous workout for M2 minutes **Post condition:** • Null **Module Invariant:** • The duration of vigorous workout must be less than the moderate workout **Event:** • E1: high heartbeat rate **Event Handler:** • EH1: change the remaining time of the current vigorous workout to the moderate workout

Also some situations impose major modifications in a workflow (e.g., addition a new module in the first case study) according to definitions and applications of exception and event, the latter seems to be more reasonable.
2) In domains such as banking, where there are several dependent and independent workflows, the benefit of modularization is clear. These modules, as parts of a workflow, need a mechanism for interaction. Sending and receiving events under supervision of a workflow template that customizes independent modules and considers a workflow as a whole body, can improve traditional methods such as [13], which allow independent modules to communicate without any specific manager.
3) In the case of consistency checking, defining proper constraints for each task and module is simpler than dealing with complex equations, which describe the whole process model, as defined in [9].

5. Conclusions and Future Work

In this paper, we presented a new architecture for workflow evolution. The architecture takes advantage of the modularity concept in terms of modules and templates that generate reusable and understandable workflows. Also, it provides dynamism and flexibility in workflow systems by employing a mechanism based on event and event handling. In some cases, designing a complete workflow, which provides solutions for all possible operations, is almost impossible. With this architecture, a partial workflow is designed as a template, and the user is allowed to define and add new events, event handlers, and modules to the workflow as long as the WFMS can approve the consistency of the workflow after each modification. The future expansion of this architecture would provide an intelligent WFMS that predicts next states and possible events for each instance of the workflow in the execution time.

References

[1] W. Aalst & K. Hee, Workflow managment: models, methods, and systems: cooperative information systems, (The MIT Press, 2004).

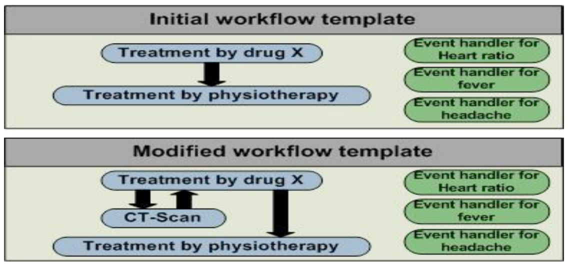
Initial workflow template
Treatment by drug X
Treatment by physiotherapy
Event handler for Heart ratio
Event handler for fever
Event handler for headache
Modified workflow template
Treatment by drug X
CT-Scan
Treatment by physiotherapy
Event handler for Heart ratio
Event handler for fever
Event handler for headache

Figure 5: Initial and modified workflow template for the treatment in case study 1.

[2] A. Cichocki, A. Helal, M. Rusinkiewicz, & D. Woelk, Workflow and process automation: concepts and technology (Kluwer Academic Publishers, 1997).
[3] M. Aquin, M. Sabou, & E. Motta, Modularization: a key for the dynamic selection of relevant knowledge components, ISWC 2006, Georgia, USA, 2006, 289-314.
[4] L. Cardelli, Program fragments, linking, and modularization, Symposium on Principles of programming languages, Paris, France,1997, 266–277.
[5] Z. Luo, A. Sheth, K. Kochut, & J. Miller. Exception handling in workflow systems, *Applied Intelligence*, *13* (2), 2000, 125-147.
[6] N. Krishnakumars & A. Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations, *Journal of Distributed and Parallel Database Systems*, *3*(2), 1995, 155-186.
[7] J. Eder & W. Liebhart, Contributions to exception handling in workflow systems, EDBT Workshop, Valencia, Spain, 1998, 399-412.
[8] W. Aalst, Generic workflow models: how to handle dynamic change and capture management information, IECIS, Washington, USA, 1999, 115–126
[9] S. Sadiq, M. Orlowska, & W. Sadiq. Specification and validation of process constraints for flexible workflows, *Information Systems*, *30* (5), 2005, 349–378.
[10] P. Mangan & S. Sadiq. On building workflow models for flexible processes, *Australian Computer Science Communications*, *5*,2002, 103–109.

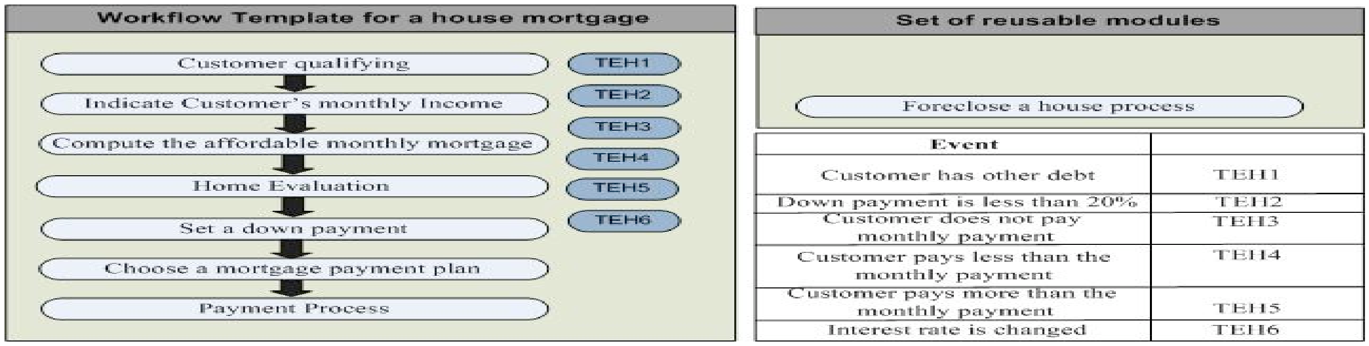
Workflow Template for a house mortgage
Customer qualifying
Indicate Customer's monthly Income
Compute the affordable monthly mortgage
Home Evaluation
Set a down payment
Choose a mortgage payment plan
Payment Process
TEH1
TEH2
TEH3
TEH4
TEH5
TEH6
Set of reusable modules
Foreclose a house process
Event
Customer has other debt
TEH1
Down payment is less than 20%
TEH2
Customer does not pay monthly payment
TEH3
Customer pays less than the monthly payment
TEH4
Customer pays more than the monthly payment
TEH5
Interest rate is changed
TEH6

Figure 6: Different parts of the proposed architecture for house mortgage

Table 2: Module specification for reusable modules in the banking domain

Module name : Customer qualifying	**Module name: Indicate Customer's Monthly Income (ICMI)**
Pre condition : • True **Tasks:** • Check customer credit **Post condition:** • The result of the qualification checking.	**Pre condition :** • True **Tasks:** • Add all customer's salaries for his/her part time and full time jobs **Post condition:** • The customer's monthly income is determined.
Module name : Compute the Affordable Monthly Mortgage (AMM)	**Module name: Payment Process**
Pre condition : • The customer's monthly income is determined. **Tasks:** 1) Multiply customer's gross income by C=0.28. 2) [Pre: Task 1] Subtract the monthly debt payments. 3) [Pre: Task 1] Subtract the monthly cost for property taxes and home owners insurance **Post condition:** • AMM is determined **Module invariants:** • C must be less than 0.36 **Events:** • E1: The customer has other debt. **Event Handlers:** • EH1: Set C=C+0.02 and start from Task 1.	**Pre condition :** • Down payment was determined • Monthly payment was determined • Interest rate was determined **Shared data:** • Remaining loan **Tasks:** • Reduce the paid amount by customer from the remaining loan variable. **Post condition:** • The loan is repaid **Events:** • E1: Customer does not pay any amount. • E2: Customer pays less than the monthly payment. • E3: Customer pays more than the monthly payment. **Event Handlers:** • EH1: According to the bank policy (E.g., increase the interest rate) • EH2: According to the bank policy • EH3: According to the bank policy
Module name: Home Evaluation	**Module name: Choose a Mortgage Payment Plan**
Pre condition : • True **Tasks:** 1) Evaluate the location 2) Evaluate the condition 3) [Pre: task 1 and 2] Evaluate the value **Post condition:** • The actual value of the house is determined.	**Pre condition :** • Down payment was determined. • AMM was determined. **Tasks:** 1) Determine loan duration. 2) [Pre: Task 1] Determine the annual interest rate. 3) [Pre: Task 2] Determine the monthly payment. **Post condition:** • Monthly payment is determined. **Module invariants:** • Monthly payment must be less than AMM.
Module name: Foreclose a house process	**Module name: Set a Down Payment**
Pre condition : • The customer has not paid the payment for N months. **Shared data:** • N **Tasks:** • According to the bank policy. **Events:** • E1: The same customer wants to buy the house from the bank again. **Event Handlers:** • EH1: According to the bank policy	**Pre condition :** • The customer was qualified. • The loan request amount was determined. **Tasks:** • The customer suggests an amount *A*. **Post condition:** • Down payment amount is determined. **Module invariants:** • *A* must be larger than 5 percent. **Events:** • E1: A is less than 20 percent. **Event Handlers:** • EH1: Ask for private mortgage insurance.

[11] K. Hee, H. Schonenberg, A. Serebrenik, N. Sidorova, & J. Werf. Lecture notes in computer science: adaptive workflows for healthcare information systems, (Springer Berlin / Heidelberg, 2008).
[12] H. Fu-Shiung. Context-aware workflow driven resource allocation for e-healthcare, International conference on e-Health networking, application and services, Taipei, Taiwan, 2007, 34–39.
[13] S. Bowers, B. Ludascher, A. Ngu, & T. Critchlow. Enabling scientific workflow reuse through structured composition of dataflow and control-flow. 22nd ICDE workshop, Washington DC, USA, 2006, 70–83.