A concept of a Software Development Process Guidance Platform

Quang Minh Tran (Student, Fakultät der Informatik, Hochschule Furtwangen)
Prof. Dr. Friedbert Kaspar (Professor, Fakultät der Informatik, Hochschule Furtwangen)
Diplom-Ingenieur (FH) Martin Zähringer (Entwicklungsleiter M&M Software GmbH)

Abstract
This paper describes a concept for a Software Development Process Guidance Platform that can give guidance through the steps needed to finish a specific workflow by considering the concrete project context. A technology-neutral implementation - code name ProcessAssist is introduced.

1 Introduction

Professional software development means conducting software development after a well-defined software development process, shortly called as *process* in this paper. Today, a process is mostly described in form of plain text, for example, on a company's Intranet. A process usually dictates that certain tasks or activities must be done during the development of a project depending in which context a project currently resides. Such a task is a well-defined sequence of steps that is called a *workflow*.

An example of workflow is creating a monthly status report for a project. The process might specify that during a running project, the project manager is responsible for creating a status report at the end of month. After being created, his direct senior project manager is authorized to review the status report. Only when the status report is approved, it is sent to the customer.

The sequenced steps might look like this in the process description:

- Project Manager (PM) updates states and prognosis for all work items of the project.
- PM makes an overall assessment of project progress. For example, whether the planned milestones can be met, whether the project quality is on the good level.
- PM updates the project milestones.
- PM updates project risks list.
- PM specifies the actions required by customer if applicable.
- All information above flows into the status report, for example, in Word format.
- Senior Project Manager (Senior PM) reviews the status report. If not ok, PM needs to improve the status report. Otherwise, if ok, he sends it to the customer in PDF format.
- Workflow ends.

A workflow usually requires the collaboration of multiple persons. Each person plays a certain role during the workflow execution.

In contemporary time, data is essential. As a result, software companies have their own strategies to manage their data. Besides master data such as the list of customers, the list of employees etc. of the company, development-related data such as budget, milestones and working time etc. are centrally persisted somewhere to track the development within the company. These development-related data evolve on a frequent basic. For example, the milestones might be regularly updated to reflect the latest state of the project regarding delivery deadlines. Some milestones might be finalized due to successful delivery while others are adjusted. New milestones might be added as well.

The most widely used persistence storage today is undoubtedly relational database. Such a central database provides a great source, from which useful information can be retrieved, both for employees and management. For example, a project manager has insight into the project's progress through a UI client that display project related data. Management can gain an overview over the overall performance in the development e.g. whether the budget limit is kept to or whether there is a delay in delivery etc. Reports or diagrams can be even directly generated from the database if needed. The more comprehensive and sophisticated the data in the database is, the more (hopefully) useful information can be extracted from it.

The maintenance and enhancement of such a database are often done via various UI clients, often being independent from each other. For instance, a project management client application might be used to interact with project related data, for example, to organize the work break-down structure and to track the project state etc. Additionally, a time accounting client application provides a user interface to employees to account their daily working time according to work items. That means, various tools are used independently but maintain the same central database.

During the life time of a project, a *project context* is referred to as the boundary consisting of all states and artifacts belonging to the project at the given time. It is not difficult to understand that process and project context cannot live without one another. Without a project context, a software development process is no more than a meaningless description. On the other hand, a project context at a given time is often the result of applying the process on a specific project (if this is not the case, the company has a big problem!). Unfortunately, despite this close relationship, today process and project context mostly live completely separately.

The separation of software development process and project context represents the following drawbacks:

- **Process is violated accidently**
  Process is often not correctly followed because employees are lacked of the process context at their fingertips. Process descriptions are plain texts distributed in various places and it is hard for employees to find the right action at a given time.

- **Process is static plain text, not a model**
  The process description is just pure text and is not understandable for computers. Adjusting an existing workflow is pure changes of plain text. Similarly, adopting a new workflow is a matter of editing plain text. These changes in workflow and new adopted workflows are not immediately visible to employees until they are told to read the documents describing these changes.

This paper describes a concept for a Software Development Process Guidance Platform – code name ProcessAssist that can give guidance through the steps needed to finish a specific workflow by considering the concrete project context. That means, ProcessAssist is aware of both process and project context and can therefore solve the drawbacks mentioned above. It is to emphasize that ProcessAssist is not aimed at modeling the whole software development process but rather supports workflows that make sense.

2 Usage scenario

In order to get an impression how ProcessAssist might look like in action, the Figure 1 shows a screenshot of the prototype developed by the authors using .NET.

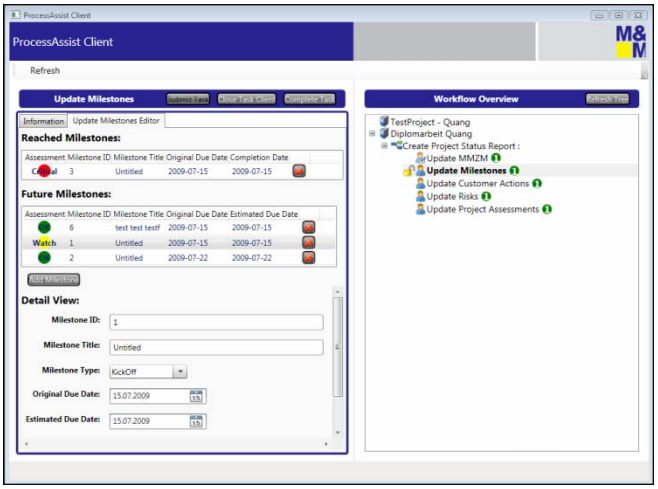


Figure 1 ProcessAssist screenshot

The right area displays the overview of all running workflows relevant to the current user. Each workflow belongs to a project. In order to ease the navigation, the workflow overview is structured as a tree. The first tree level contains projects that are relevant to the user. Each project might contain more than one child workflows. Each workflow in turns shows all belonging tasks - both complete and in progress.

While navigating in the tree, important information such as the project name, the workflow name, the task name depending on the type of the current tree node is displayed. If the current tree node is a human task, the user may work on it by loading a human task client. This human task client is used to work with that human task.

For example, if the human task is updating the project milestones of a certain project, the human task client provides an editor for updating, removing, adding project milestones. In case such a task client is not necessary, for example such tasks that cannot be done using software but require a physical activity instead, enough information is given to the user so he knows what must be done to complete the task. When a human task client is open, the corresponding task is locked so that other persons cannot work on it anymore. The human task is unlocked again when the human task client is closed. The user may also signal the completion of a human task as soon as he completes it using a complete task button.

3 Terminologies

Before continuing, often used terminologies are described here to avoid confusion.

Workflow: describes a sequence of steps needed to achieve a goal. A workflow defined as a part of a software development process. Create status report mentioned above is an example of workflow.

Human task: a step within a workflow that is done by a human. For example, approval of a status report is a human task.

Automatic activity: a step within a workflow that is executed automatically. For example, an activity that calls a Web service operation is an automatic activity.

Human task client: a UI client used to work on a specific human task.

4 Requirements

This chapter defines the requirements to be met by ProcessAssist.

4.1 Functional requirements

Functional requirements describe the features that the system shall provide. The features provided by ProcessAssist are briefly described here:

Start workflow

1. User chooses the project.
2. User starts a specific workflow (e.g. create status report).
3. ProcessAssist shows the newly started workflow with the initial human tasks.

Open human task client

1. User chooses the human task to open.
2. Human task client is displayed with all necessary information.
3. Human task is marked as locked by the current user (other persons cannot work on the task until it is unlocked again).

Close human task client

1. User chooses the human task to close.
2. Human task is unlocked (other persons can lock it and work on it).
3. Human task client is closed.

Submit changes in human task client

1. User chooses the human task to submit.
2. Unsaved information on the human task client is saved into persistence storage.

Complete human task

1. User chooses the human task to complete.
2. Human task is marked as completed.
3. ProcessAssist shows the next human tasks based on workflow definition.

4.2 Non-functional requirements

Unlike functional requirements that describe the “what” of the system, non-functional requirements refer to all characteristics a system should have and therefore describe the “how”. In following, we list the most important non-functional requirements ProcessAssist shall meet:

- **Workflows are centrally defined in a machine-readable form:** workflows to be supported are modeled in machine-readable form. The process of modeling workflows is done using a visual designer.
- **An executing workflow is bound to a specific context:** once executed, a workflow is bound to a concrete context, for example to a specific project. All necessary context information is set to workflow at the time it is started. For instance, in case of the create status report workflow, once executed it is bound to a specific project which is provided as a project ID.

- **Workflow can contain both automatic activities and human tasks**: a workflow shall be able to contain both these kinds of workflow in its model.
- **Multi-person support:** multiple persons can collaborate on the same workflow. Each person plays a certain role during the workflow execution. For example, in case of the create status report workflow, the PM edits the status report while the senior PM reviews it.

5 Architectural Design

Having discussed the requirements, this section gives an overview over the architectural design. The architecture described here is technology neutral and can theoretically be implemented by any technology that supports all aspects mentioned.

Basically, ProcessAssist is developed based on the 3-layer architecture:

- **Presentation Layer:** called ProcessAssist-Client, is realized as a UI client that provides a user interface to user to see information related to workflows as well as interact with them e.g. start a workflow, complete a human task etc. Multiple ProcessAssist-Clients can be used by multiple persons. Section 5.2 describes ProcessAssist-Client.
- **Application Logic Layer:** called ProcessAssist–Server**,** is realized as a server that implements the workflow logic. This layer is the most important one and is described detailed in section 5.1.1.
- **Persistence Layer,** called ProcessAssist-Database, is a relational database. It contains all workflow information. Section 5.3 describes ProcessAssist-Database.

5.1 ProcessAssist-Server

ProcessAssist-Server is divided into 3 layers:

- **Service Layer:** provides a set of Web service operations used by ProcessAssist-Client to interact with the server. ProcessAssist-Clients send their requests to this layer to interact with ProcessAssist-Server. Service implementation, however, does not contain any business logic but just delegate client requests to the Business Layer.
- **Business Layer:** contains all business logic implementation of ProcessAssist-Server. The most significant components in this layer are the **workflow runtime** that executes and manages workflows, the **activities component** that comprises all activities used to construct workflows and the **workflows component** that contain all workflow definitions supported by ProcessAssist.
- **Data Access Layer:** This layer is responsible for writing data into / retrieving data from ProcessAssist-Database. It abstracts database access in an object-oriented way.

5.1.1 Service Layer

The following describes the significant Web service operations provided by Service Layer. These operations can be categorized into Workflow-related and HumanTask-related operations:

Workflow-related operations

- **StartWorkflow(string workflowType, Map<string, object> contextualInformation):** starts a new workflow of type specified by the workflow type. The contextualInformation map contains workflow contextual information that provides the workflow an execution context. For example, when a create status report workflow is started, this map contains the ID of the project, for which a status report should be created.
- **CancelWorkflow(int workflowID):** cancels a running workflow identified by workflowID. All database entries related to this workflow are removed from the database.

Besides these operations, there are operations used to retrieve workflows from the database such as GetWorkflowOfProject and GetWorkflowByID to retrieve workflows of a certain project or a workflow by its ID, respectively.

HumanTask-related operations

- **CompleteHumanTask(int humanTaskID, Map<string, object> humanTaskReply):** This operation signals the workflow that the human task specified by humanTaskID is completed. If the human task produces outputs that affects the workflow flow (for example a decision whether a status report is approved or not), it is sent in the humanTaskReply map.
- **LockHumanTask(int humanTaskID):** locks the human task specified by the humanTaskID to reserve the human task for the current person. This prevents other persons from working on this human task.
- **UnlockHumanTask(int humanTaskID):** is the reverse operation of LockHumanTask.

Besides these operations, there are operations used to retrieve human tasks from the database such as GetHumanTaskOfWorkflow and GetHumanTaskByID to retrieve human tasks of a certain workflow or a human task by its ID, respectively.

5.1.2 Business Layer

Business Layer is the place where all business logic of ProcessAssist-Server is implemented. The following components are significant:

- **Activities component:** described in section 5.1.2.1.
- **Workflows component:** described in section 5.1.2.2.
- **Workflow runtime:** described in section 5.1.2.3.

5.1.2.1 Activities

The activities component contains all activities used to construct workflows including human tasks and automatic activities. The Figure 2 is the class diagram showing the activity classes.

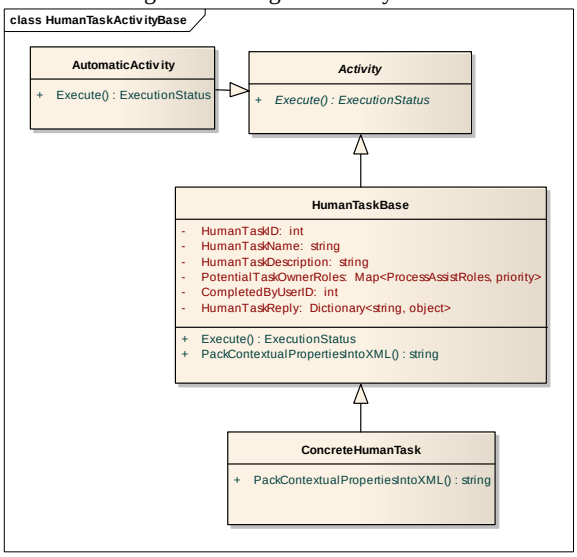


Figure 2 Class diagram of activity classes

Activity is a base abstract class that all activities and workflows (a workflow is a special activity) shall derive from. It has an Execute method that represents the execution logic of an activity at runtime.

Automatic activities derive from the base class Activity and implement their execution logic. HumanTaskBase is the base class that all human task activities shall implement. HumanTaskBase extends the base Activity class and implements the custom behavior to simulate a human task.

While an automatic activity often takes a relatively short time to execute (e.g. calling an external Web service operation might last some seconds), a human task characterizes itself by the fact that its completion time is unpredictable. Its running time is even measured in days or weeks.

One way to realize a human task is modeling it as a waiting node within a workflow. Whenever the workflow execution arrives in a human task, it transforms into a waiting state and does nothing but wait. The completion of a human task is signaled by sending an event back to the waiting workflow. Upon receiving this event, the workflow leaves the human task and advances the execution to the next step. [1] has a nice discussion about modeling a manual activity in Microsoft ® Workflow Foundation

This is exactly what the base class HumanTaskBase does. Whenever the execution reaches the human task, HumanTaskBase's Execute method is called. The method registers to a wake-up event and transits the activity into a waiting state.

Furthermore, HumanTaskBase defines all common properties shared by all human tasks. The following describes the important attributes and methods defined in HumanTaskBase: The reader can find more inspirations about human task in [2].

Important Attributes

- **HumanTaskID:** the ID that uniquely identifies the human task.
- **HumanTaskName:** the human-readable name of the human task e.g. Update Project Milestones
- **HumanTaskDescription:** a human-readable text that briefly describes the purpose of the human task
- **PotentialTaskOwnerRoles:** the list that contains all roles together with a priority who have the right to work on the human task. When a person has the 1.priority for a certain human task that means that he is the favorite one to work one that human task.
- **CompletedByUserID:** the ID of the person who completed the human task.
- **HumanTaskReply:** the replies of the human task when it is completed. This includes replies that have impact on the workflow flow for example a decision made by person.

Important Methods

- **Execute(): ExecutionStatus** is inherited from the Activity class. When the workflow runtime reaches the human task, it calls this method. It writes a human task entry into the database (a ProcessAssist-Client makes use of this entry to display a human task). Furthermore, it registers to an external wake-up event before transforming into a waiting state.
- **PackContextualPropertiesIntoXML(): string:** is a virtual method. Concrete human task classes shall override it to implement custom behaviors. A concrete human task class has its own set of contextual properties beside the common properties mentioned above. Once executed, these properties have concrete values that describe the execution context of the human task. The PackContextualPropertiesIntoXML method implementation of a concrete human task packs all its own properties into an XML and returns it. This XML string is stored as a column within the human task entry in the database. Whenever the corresponding human task client is opened, it parses this XML string to get the context information to display its content.

- **PackHumanTaskReplyIntoXML(): string:** is also a virtual method that concrete human task classes shall override. When the human task is complete, this shall pack all replies sent by the human task client an XML. It is also in this method that some contextual properties might need to be updated based on the replies. This XML string is also stored as a column within the human task entry in the database as well. In addition to the contextual information as XML, the corresponding human task client, when opened, parses this XML string to get the context information to display its content.

5.1.2.2 Workflows

The workflows component contains all workflows supported by ProcessAssist. Each workflow corresponds to a real-world workflow defined in a software development process.
The
Figure 3 is the class diagram showing the workflow classes.

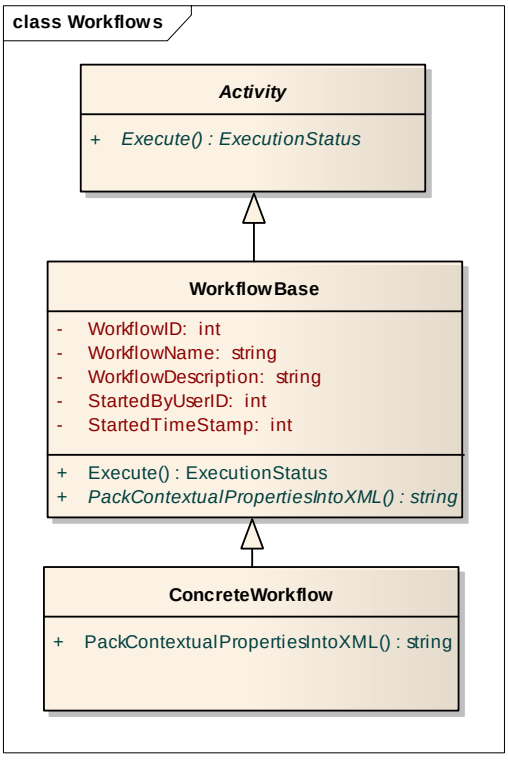


Figure 3 Workflows class diagram

This component contains the base class WorkflowBase that all workflows shall implement. Like HumanTaskBase, WorkflowBase implement all generic behaviors applied to all workflows. Concrete workflows derive from it and implement their concrete behaviors. The following describes the important attributes and methods defined in WorkflowBase:

Important Attributes

- **WorkflowID:** the ID used to uniquely identify the workflow.
- **WorkflowName:** the human-readable name of the workflow e.g. Create Project Status Report.
- **WorkflowDescription:** a human-readable text that briefly describes the purpose of the workflow e.g. Create Project Status Report.
- **StartedByUserID:** the ID of the person who started the workflow.

Important Methods

- **Execute(): ExecutionStatus:** is called by the workflow runtime when the workflow is started for the first time. It writes a workflow entry into the database (a ProcessAssist-Client the information in this entry to display a workflow)
- **PackContextualPropertiesIntoXML(): string**

This method is similar to the one with the same name defined in HumanTaskBase. Similar to human task, a concrete workflow class has its own set of contextual properties beside the common properties shared by all workflows. Once executed, these properties have concrete values that describe workflow's execution context. The PackContextualPropertiesIntoXML implementation of a concrete workflow packs all its own properties set into an XML. This XML string is also stored as a column in the database within a workflow entry. This is necessary to capture workflow's initial execution context.

5.1.2.3 Workflow Runtime

Workflow runtime is probably the most important component. It provides an execution environment for workflows to execute.
The Figure 4 shows the class diagram of workflow runtime.

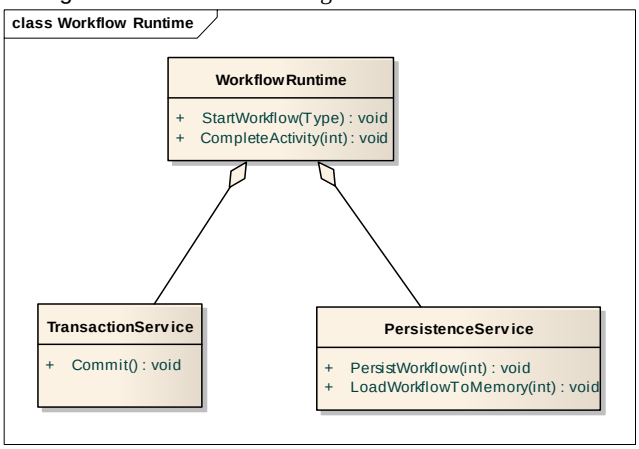


Figure 4 Workflow runtime class diagram

A workflow runtime uses persistence service to persist workflows into ProcessAssist-Database. The reason for the need of a persistence service is as follows.
Once a human task is executed and transforms into a waiting state, the workflow is in the waiting state as well. Keeping a workflow alive in memory is not practical since it consumes memory resources without doing anything. The better approach is to persist it to a durable store such as a relational database or files. Upon receiving a completion signal, the workflow is reloaded into memory again to continue its execution. Another reason why a mechanism to persist workflows is necessary is the fact that in-memory workflows cannot survive a system crash.

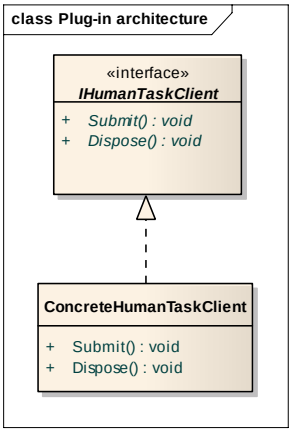
Furthermore, the workflow runtime needs a transaction service that assure that when a workflow entry and all associated human task entries (e.g. when a workflow is started and several human tasks are executed) are written into the database in the same transaction in order to keep the database consistent.

5.1.3 Data Layer

The Data Layer abstracts the database access to the underlying relational ProcessAssist-Database. The Business Layer uses Data Layer when writing data into or retrieve data from ProcessAssist-Database. The way to realize this layer normally depends on the specific technology used. For example, with .NET technology LINQ to SQL can be leveraged to implement Data Layer.

5.2 ProcessAssist-Client

ProcessAssist-Client provides a main application that shows the workflow information and the human tasks in a tree. Each human task client for a human task activity type can be developed and plugged into the main application. The communication between the main application and the human task clients take place via a well-defined interface.
The Figure 5 is the class diagram showing the plug-in-architecture for human task client.



- **Submit(): void:** This method shall be called by the main application to submit the data entered by user on the human task client.
- **Dispose(): void:** This method shall be called by the main application to dispose any resources occupied by the main application when the human task client is closed.

When a human task client is loaded, it receives all context information it needs in the ContextualInformationXML and possibly HumanTaskReplyXML (if the human task is completed and the completion sent replies back to the workflow) as described in chapter 5.1.2.1

5.3 ProcessAssist-Database

The Figure 5 shows the database schema of ProcessAssist-Database.

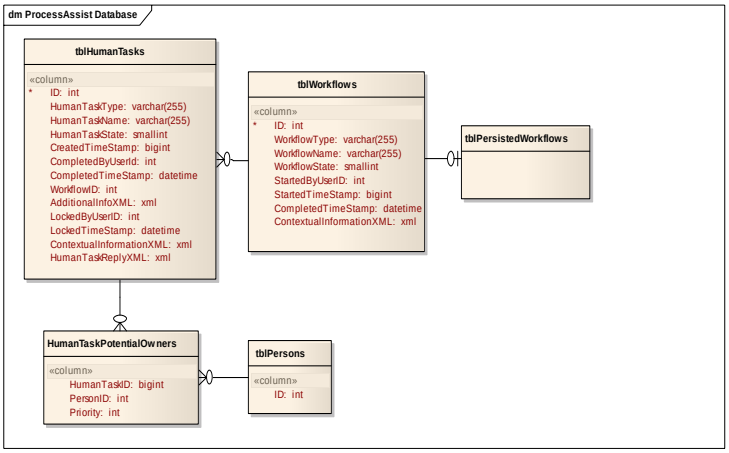


Figure 5 ProcessAssist's database schema

Here is a brief explanation to the tables found in the diagram:

- **tblWorkflows:** stores workflow-related information. It has columns that correspond to workflow's generic information such as workflow name, workflow type and the user ID who started the workflow etc. ContextualInformationXML contains workflow's contextual information when started as described in section 5.1.2.2.
- **tblHumanTasks:** stores human task's information. Similar to tblWorkflows, it has columns such as human task name, human task state etc. to store human task's generic information. ContextualInformationXML contains human task's contextual information when it is executed as described in section 5.1.2.1.
- **tblHumanTaskPotentialOwners:** stores potential users that can own a certain human task. Each entry contains a person ID, a human task ID and a priority number and describes that the person specified by ID can work the human task specified by the human task ID. The priority determines with which priority a person should work on the human task. The priority 1 is the highest priority.

6 Implementation

The architecture introduced above is intentionally kept technology neutral. The authors already successfully implemented a prototype using the .NET framework.

The following technologies were used:

- **Workflow Foundation (WF):** is used to implement workflow support. WF includes a visual designer integrated in Visual Studio used to model workflows. Further WF provides a workflow runtime that manages the execution of workflow. Transaction and persistence are also provided.
- **Windows Presentation Foundation (WPF):** is used to implement ProcessAssist-Client.
- **Windows Communication Foundation (WCF):** is the technology for communication between ProcessAssist-Client and ProcessAssist-Server.
- **ProcessAssist-Database:** is a relational Microsoft ® SQL Server database.

Even though, the prototype shows that such a tool is feasible, it turns out that the current WF 3.5 has still many limitations. The two types of built-in workflow Sequential Workflow and State Machine provided in WF are not powerful enough to make the workflow logic for ProcessAssist as flexible as the authors expected. If a more powerful and flexible workflow framework is available, this can be used to drive the workflow logic for ProcessAssist.

Furthermore, ProcessAssist can be implemented with other technologies such as Java as well.

7 Conclusion and Outlook

The goal of the paper is to introduce a concept for a Software Development Process Guidance Platform namely ProcessAssist that provides guidance to workflows defined as parts of a software development process.

The paper begins with pointing to the fact that today software development process and project context are separated from each other. The disadvantages are then illustrated. ProcessAssist is introduced as a tool to solve that problem by having knowledge of both process (workflow definitions) and project context.

The requirements ProcessAssist shall meet are identified and an architectural solution is introduced how ProcessAssist can be implemented.

The authors are of the opinion that such a tool that has knowledge of both process and project context could be very powerful that could assist the employees at a software company in their daily work.

8 References

[1] Lublinsky, B. (2008, Mai 05). InfoQ. Retrieved March 28, 2009, from http://www.infoq.com/articles/lublinksy-workqueue-mgr

[2] BPEL4PEOPLE Specification. (2007, June). Retrieved March 28, 2009, from http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf