2008

# HP banqpro/ – Workflow in a WPF-based Smart Client

## Motivation & Patterns for Client-Workflows

**This whitepaper summarizes architectural concepts for building smart clients with Windows Presentation Foundation that require the integration of dynamically generated Workflows defined by power users. The concepts have been worked out in architecture design sessions between Microsoft and HP to support their current plans of developing a new client for their standardized banking software product "HP banqpro/". HP banqpro/ is deployed at several Enterprise Accounts in Austria and HP Austria is currently developing their new Consultation Client based on these concepts.**

**If you are interested in more detailed information about HP banqpro/ just take a look at the following web site: http://www.banqpro.at!**

**Author Information:**
Mario Szpuszta
Microsoft Austria
6/3/2008

**HP banqpro/ Architecture Team**
Branislav Grman
Dominik Harold
Heinz Kubis
Josip Sagaj
Markus Wolfsbauer

## Table of Contents

## Introduction

HP **banqpro/** is a very well-known and standardized banking-software in Austria which is used by several enterprise customers across the country. The product exists for more than a decade now and therefore is implemented based on a number of different technologies including C++ and UNIX on the backend and a proprietary UI framework called TFM on the frontend.

Currently HP Austria is in progress of developing a completely new front-end experience for the next generation of **banqpro/** and they decided to use Microsoft's Windows Presentation Foundation (WPF) on the client for providing next generation user experience in their banking product.

This whitepaper is a summary of core architectural concepts we have worked out together with the HP **banqpro/** team in Austria during several *Architecture Design Sessions*. While the foundational concepts of the architecture we've been working on are explained in this paper as well, the core focus captures details about the concepts for integrating workflows based on Windows Workflow Foundation into this smart client as well as their motivation for going this path.

## Setting the Expectations for this Whitepaper

This isn't easy material on the one-hand-side. But on the other hand this paper is not about coding and low-level implementation details. Yet it requires you to have a deep understanding of the following technologies because specifics of these technologies have influenced architectural decisions of the application design. Especially when it comes to the last section of Workflow integration design based on Windows Workflow Foundation, deep knowledge in this area is partially necessary to really understand the selected patterns. Therefore I recommend that you make yourself familiar with the following technologies before you start reading especially this last section (all the other sections do not have deep pre-requisites in terms of knowledge):

- ✓ Must: .NET Framework 2.0 know-how
- ✓ Must: basic Windows Presentation Foundation and XAML knowledge
- ✓ Optional: Microsoft Patterns & Practices Composite UI Application Block and Smart Client Software Factory knowledge
- ✓ Must for the last section: deep Windows Workflow Foundation know-how

**Also please read the conclusion! I have some very important statements about the approaches described in this paper!!!!**

## Understanding the Requirements

Before we start digging into architectural details we really should think about the core requirements the **banqpro/** team needs to address with subsequent releases of their enterprise banking software. And definitely the core question is: why the heck does someone need workflow on the client? Isn't that some kind of overkill? Well, for some scenarios client-based workflow definitely makes sense. To understand the core requirements that are driving this decision we need to take a look at the concept of products and product variations.

**Products and Product Variations**

Basically every bank is offering different kinds of products. These products include things such as saving accounts, credit accounts, loans and – in the meantime – much more. All of these products require some kind of information about the customer, details about a concrete "instance" of a product, contracts etc. While by default the software supports the full-blown version of products which require the banking-employee to enter all the necessary information (interest rates, duration of for example a loan, payment conditions etc.) the future version should also be able to handle product variations. For example a "loan for students" is a special version of the general product "loan" which has some fixed conditions such as higher but fixed interest rates, a minimum runtime which cannot be changed etc. These could be defined by a bank employee of the back office. Bank employees in the front office should then be able to use these products when being in progress of talking with a customer to close a new case. In that process the front office employee should not need to or even is allowed to enter and override information previously defined by the back office employee for a product. This scenario can involve single fields or even whole forms the employee needs to or does not need to fill out for the new case.

**Dynamic Definition of Product Variations**

As mentioned earlier, back office employees should be able to define product variations directly within the application. When defining product definitions, they would define the fields which need to be filled out and which don't need to be filled out, they define default values which are override able or not and they define which fields should be filled out on which forms contain which fields and are displayed when in the workflow of filling out information for submitting a new case with a customer to the backend application. The last part of this requirement sounds very much like a simple variation of a workflow for coordinating the order in which forms with different fields are displayed.

## Technology Decisions and Reasons

Based on the aforementioned requirements, we recommended and discussed the usage of several technologies and decided to build the application based on Windows Presentation Foundation without usage any additional UI frameworks such as Composite UI Application Block and we decided that for future, more complex client workflows in context of product variations, Windows Workflow Foundation will be used. Of course that brings up one very interesting question: why didn't we use the Composite UI Application Block in conjunction with Smart Client Software Factory?

Well, for me that question has several answers! First of all the time when we had to make the decision was very unlucky: CAB/SCSF was not available for Visual Studio 2008, yet, and the Patterns & Practices team started working on Prism as a composite smart client reference implementation with a set of building blocks to have a WPF-based successor for CAB/SCSF in place. CAB/SCSF was not VS 2008-ready, yet, Prism was not far enough yet. But of course that's not enough as a reason for not using a ready-to-use framework. And therefore we are getting to the second answer: the required level of skills to start working with CAB/SCSF is pretty high and requires a serious time for learning the infrastructure. That is especially hard to do when you are new to .NET development (what was the case for this team) and have to get started with.NET development on its own. And the number of features we would have used from CAB/SCSF (essentially the concepts of WorkItems and MVP/MVC) simply where not enough to justify the steep learning curve that would have been necessary to start productive development.

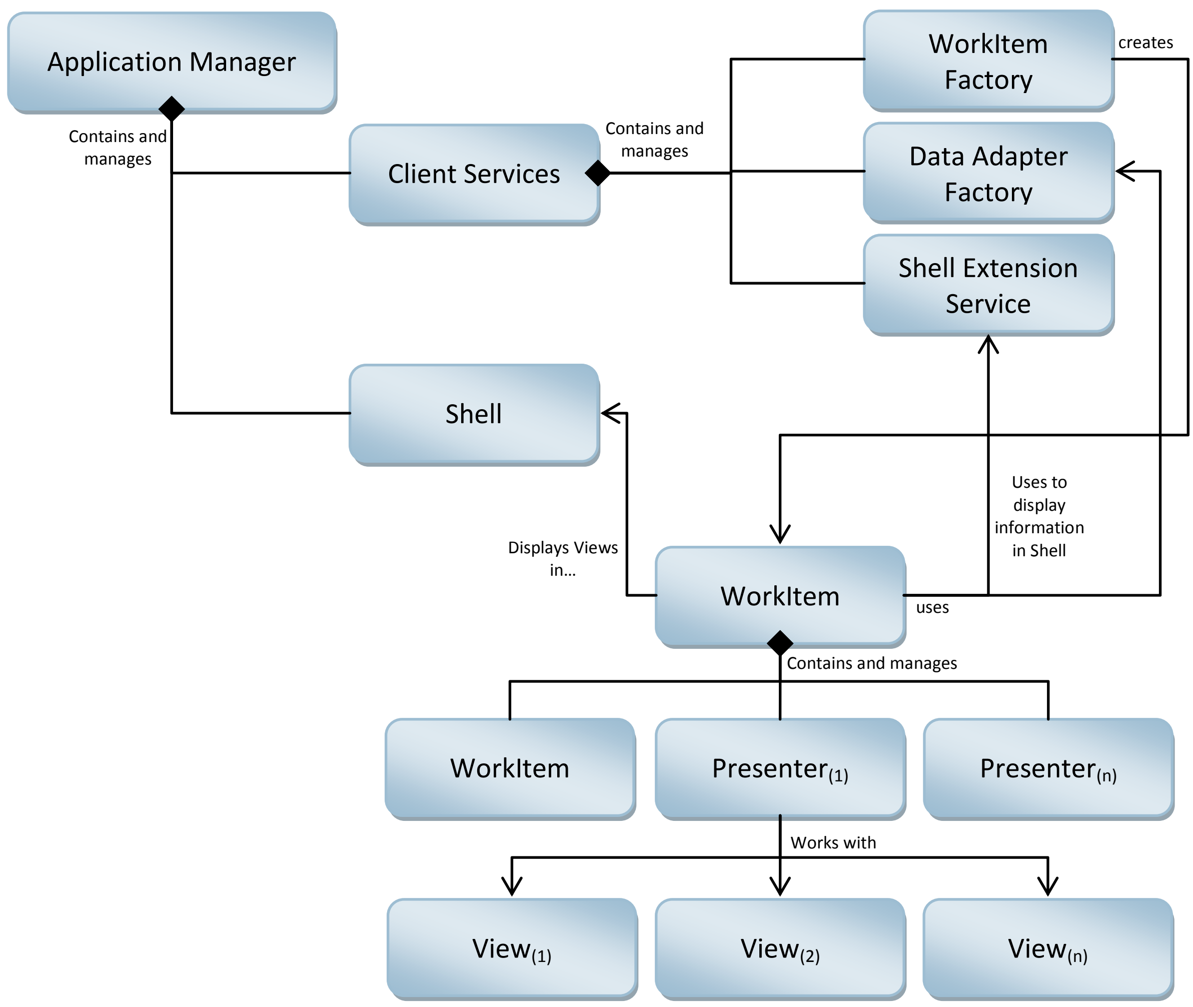
Therefore we decided to implement a much simpler framework based on WPF for the new **banqpro/** smart client as the core focus was a new, cool look & feel as well as the workflow integration. Nevertheless we've taken some of the concepts and patterns from CAB/SCSF as starting points for our work, therefore many parts of the framework developed are very familiar for CAB/SCSF developers. And as I cannot show source code from the real prototype in this paper I decided to use CAB/SCSF as a foundation for demonstrating parts of the concepts we've been working out in a demo application. Actually I built that demo application for a session at last year's TechEd Europe Developers and presented the concepts at a session there, as well. I've done this work with the **banqpro/** project in mind. To download the running demos from my presentation at TechEd just navigate to my blog by clicking here.

## The Architectural Foundation for the new Client

As mentioned earlier some of the core concepts of CAB/SCSF where a starting point for designing the architecture of the **banqpro/** client framework. We established the terms Application Manager, Shell, Client Services, WorkItem, Presenter and of course View. We used these terms in a very similar fashion how I suggested it in my whitepaper about CAB/SCSF for the RACON Linz project one and a half years ago. You can download my whitepaper about designing smart clients with CAB and SCSF here. But – as mentioned earlier – we simplified the architecture a little bit, and therefore my concepts described in this paper as well. In the following table you can see a summary of how we defined the concepts mentioned above for this project:

| Concept | Description |
|---|---|
| Application Manager | The application manager is implemented as a part of the WPF application class and is responsible for initializing the shell as well as a simple dictionary container for centralized access to client services. |
| Shell | The shell is the main window of the application. Again very much simplified compared to CAB/SCSF it just contains a navigation tool bar, a central search box, an information and a tree-navigation panel on the left-hand side as well as a main tab control where WorkItems can add content to. |
| Client Services | Client services are components which are providing central services of the client framework to all WorkItems and their parts. Examples of such client services are factories for instantiating and initializing WorkItems (WorkItem Factory) and Data Agents (Data Adapter Factory) or a service for extending the shell itself in a strongly typed and secure manner (ShellExtensionService). The Client Services Dictionary gives central access to such services for all other components in the client so that all these components can just use these services without the need of instantiating and initializing them again and again everywhere in the client application. |
| WorkItem | In our case a WorkItem encapsulates a user task. It is essential to understand that the user task really needs to be seen as a task in context of the end-user sitting in front of the machine. This is especially important for our next thoughts on workflows. Examples for WorkItems therefore would be: “Create new Customer”, “Edit a Customer”, “Edit an Address”, “Create new Loan” etc. These user tasks / WorkItems can be single tasks or compositions of multiple tasks (e.g. creating a new customer involves editing of base customer information as well as creation of addresses and creation of a new account). As the encapsulation of a user task a WorkItem therefore needs to orchestrate Views (see below) and child WorkItems to get the user task done. It also communicates with the backend through agents (called Data Adapters in this architecture) and performs client-side validations etc. |
| View | A simple user control used for displaying or modifying information about a business object. |
| Presenter | A presenter encapsulates the logic behind a view. In context of the dynamic product generations the separation of the actual control and its processing logic is essential because with different product configurations Views might be generated as XAML in the future as part of dynamic product configuration. This XAML then would need to work with a presenter instead of including the entire event-processing etc. on its own. |

The following figure shows, how the different bits and pieces explained in the previous table are working together:



Interesting is the special requirement for the flavor of the Model-View-Presenter Pattern in this scenario. The requirement of product variations mentioned in the previous chapter does not only include workflows – it also will include dynamic generation of forms used in workflows based on the field-selections the back office employee has made. A pragmatic approach for fulfilling this requirement can look as follows: To be able to re-use as much from the programmed logic as possible, all the logic for a view should be implemented in the presenter. WorkItems could launch presenters instead of Views directly and the presenter could decide which variation of a View to load from where based on the product configuration passed in by the WorkItem. While the presenter includes all the logic necessary for the most complete version of a View the View could also be a dynamically generated and compiled WPF-XAML with an extremely simple code-beside that just calls into the presenter through its interface definition because generating simple method calls based on code templates is much easier than generating complete code trees. And generating XAML – which essentially is XML – of course is much easier than generating anything else.

All the other concepts are essentially very similar to the approaches of CAB/SCSF and to my original whitepaper about designing CAB/SCSF-based smart clients – we just simplified the concepts a little bit for this scenario. With that knowledge in mind we can go one step further and talk about workflow.

## Integrating Dynamic Workflows

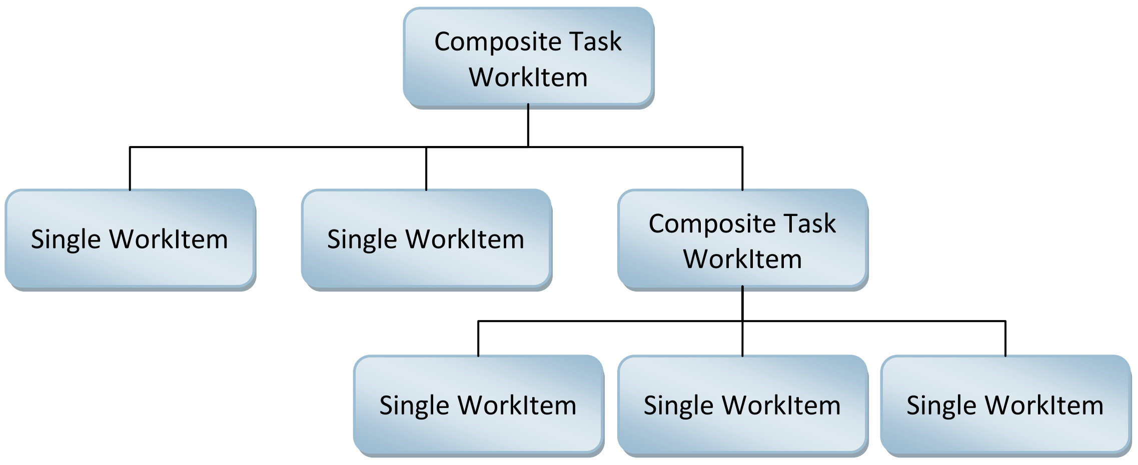
Workflows get relevant for the execution of the different steps for filling out information when creating new records of products or product variations as defined by the back office for further backend processing. For this whitepaper we focus on the client-side workflows because that's the part where Microsoft was involved into designing the architecture of the application.

For product variations, essentially the back office employee defines which steps need to be completed by the front office employee when creating a new instance of a product for a new case with a customer. These steps can include different types of views or other WorkItems. As mentioned earlier, the views can be statically defined (which is the case for the full-blown product variations) or dynamically generated (which is the case for the dynamically defined sub-sets of product variations). In any case the client-workflow is responsible for orchestrating these views and child WorkItems and the order in which they are launched when a front-office user creates a new instance of a product is defined dynamically by the back office employee. But that's a rather technical point-of-view. Before we start digging into the design we should rethink the situation from the end user's point-of-view to be able to ensure the best possible user experience.

### Design *for* Workflow-Support

Technically speaking we defined, that a workflow orchestrates views and child WorkItems. But the end user who is designing the product variations in the back office won't understand both terms. At the end of the day for the bank employee it's all about the completion of user tasks on the front-end and therefore we should stick with this metaphor for our architectural design of the application.

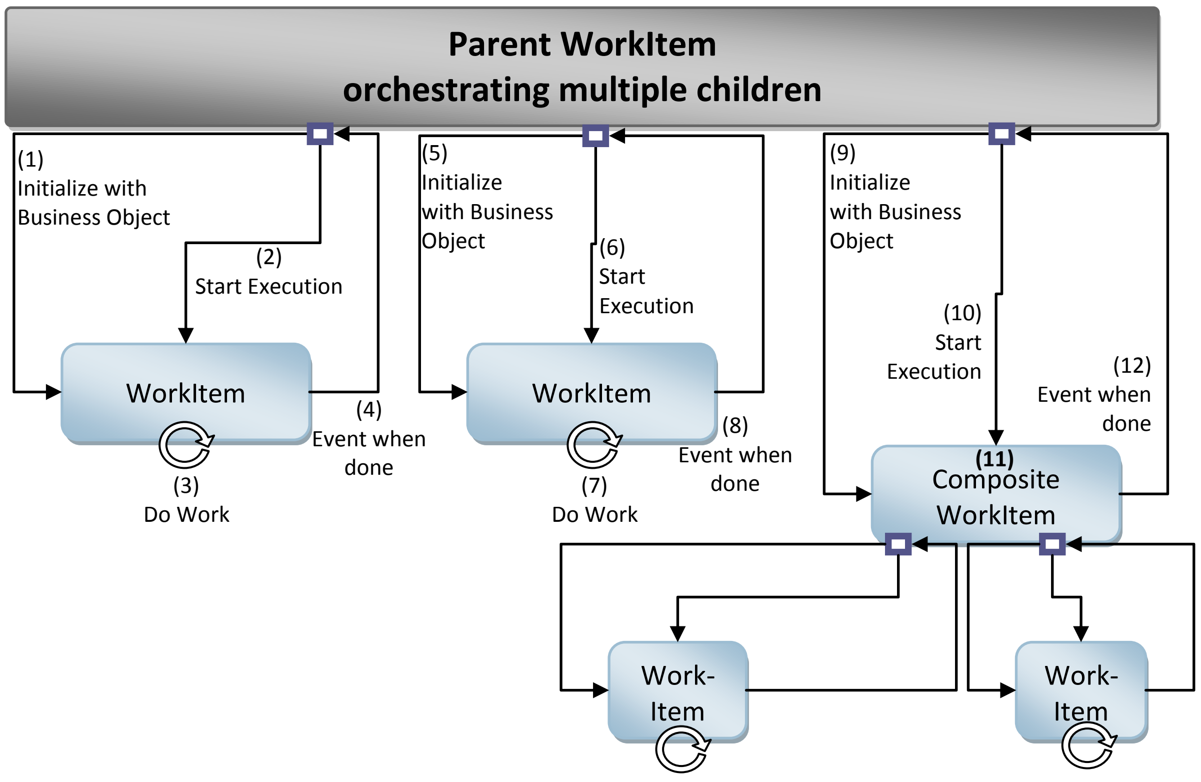
Taking a look back into the section "The Architectural Foundation for the New Client" of this paper we clearly defined the WorkItem as an encapsulation of a user task. As the back-office employee will work with the notion of user tasks this is the lowest level of granularity we should work with as atomic activity. Of course a user task can involve additional, other user tasks leading to a composition of user tasks where the root task is responsible for orchestrating child tasks. Having that said we can think of two types of user tasks – atomic tasks and composite tasks, but we won't use these terms in front of the user, of course. For the user, simply everything is a task. The following figure illustrates these thoughts:



Of course both types of WorkItems should inherit from the same base class so that they can be used in a generic fashion without having their parent needs to know about the fact that they are composite WorkItems. Again, for the user these are just tasks that he puts together for designing a new product variation in the product, dynamically.

In terms of the Views each and every WorkItem displays their own Views as usual – for our scenario we decided to not include orchestration of Views into the Workflow implementation as we think that this is not the level of granularity the user would design the workflows at. Therefore we were required to make sure; that "atomic" tasks from the user's point of view are encapsulated into a WorkItem which on its own displays several views either hardcoded or based on a simpler approach than a full-blown workflow (my simplified, pragmatic thoughts are going towards simple XML configurations that are loading dynamically generated WPF controls as views which are being created by a custom designer for WorkItem activities in a workflow).

Now we have clearly defined that our workflow in a composite WorkItem orchestrates WorkItems, only. We do not orchestrate Views as the end users which are going to design these workflows as part of product variations will work at the level user tasks, only! But what does orchestration of WorkItems technically mean? Which parts do we need to include into WorkItem interfaces to be able to support workflows in a generic fashion? Just take a look at the following figure that illustrates a simple WorkItem orchestration done by a parent WorkItem.



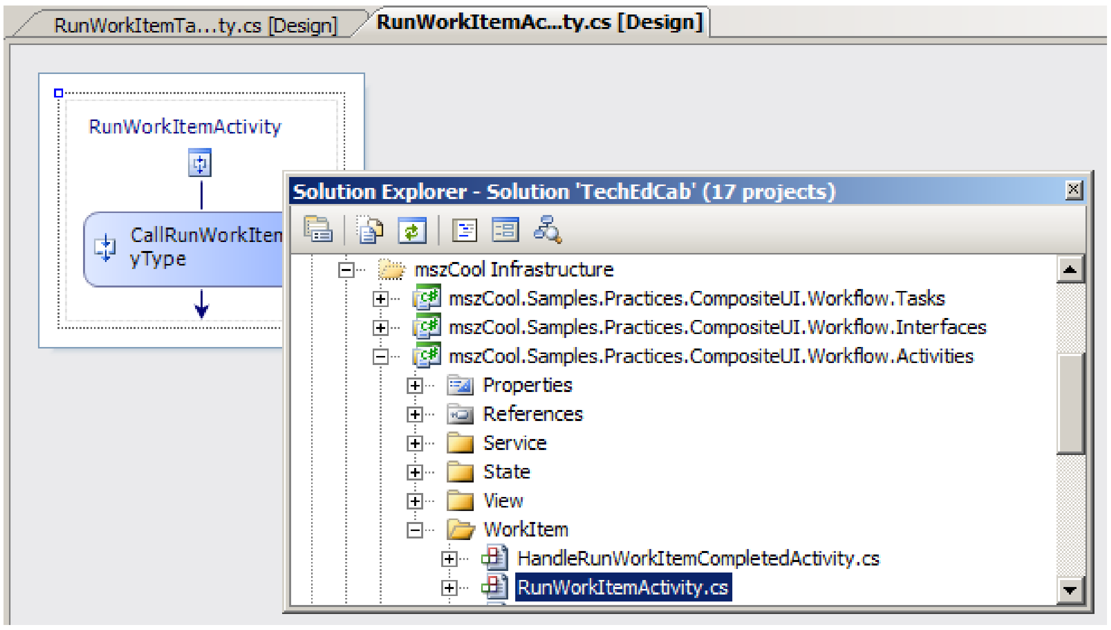
As you can see in the previous figure, every WorkItem needs to support at least three parts in its interfaces. When a parent WorkItem (which is a Composite WorkItem) starts its execution, it definitely will get some data – business objects – from the backend the WorkItem itself and its children will work with. When a child-WorkItem is launched it needs to be able to pass in the business object in a generic way. The possible way to implement that is through a generic State-bag dictionary (ala ChildWorkItem.State["Initialization"] = business_object). After the initialization is completed – what essentially means after the parent WorkItem has told the child WorkItem everything it needs to know – the parent can launch the execution of the child WorkItem. The child WorkItem can do its work and when it is finished it can raise an event to tell the parent that it has done its work. The parent can query state information from the WorkItem, dispose it and move on to the next WorkItem with the same steps: initialization through state, start execution and wait for a completed event. Of course the communication paths are the same for a Composite WorkItem which launches a workflow for orchestrating further WorkItems by itself.

With that we have clearly defined the requirements for a WorkItem interface and CAB/SCSF actually fulfills all these requirements out-of-the-box. On the other hand that does not mean we can forget about the thoughts – keep them in your mind for the next section, "Designing the Workflow Integration" itself.

## Designing the Workflow-Integration itself

Now we have laid the foundation for designing architecture of the Workflow integration into the smart client application based on Windows Workflow Foundation (WF).

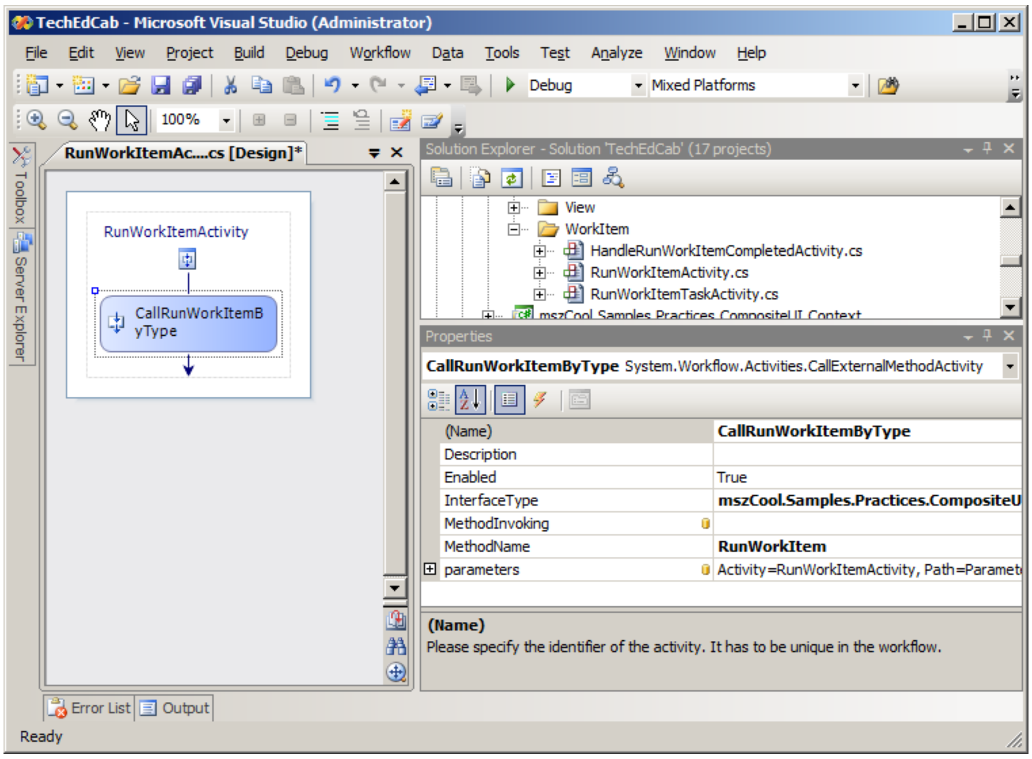
Again I would like to start my explanations from the end-users's perspective. When designing a workflow for data entry, the end user finally wants to drag a couple of artifacts onto a design surface that are associated with user tasks. These tasks could be custom Windows Workflow Foundation activities which – at the end of the day – are responsible for executing a single user task. In our case a single user task is nothing else than a WorkItem. Of course in the prototype I've developed for TechEd Europe last November to evaluate the concepts for the **banqpro/** project I didn't play that game to the end in terms of building end-user ready workflow activities and building a workflow designer. But I evaluated the concepts with custom workflow activities that a developer can use to design a workflow based on CAB WorkItems. So the first step is designing a custom activity library for Windows Workflow Foundation workflows that support the execution of WorkItems as you can see in the following figure.


RunWorkItemTa...ty.cs [Design]
RunWorkItemAc...ty.cs [Design]
RunWorkItemActivity
CallRunWorkIten
yType
Solution Explorer - Solution 'TechEdCab' (17 projects)

mszCool Infrastructure
mszCool.Samples.Practices.CompositeUI.Workflow.Tasks
mszCool.Samples.Practices.CompositeUI.Workflow.Interfaces
mszCool.Samples.Practices.CompositeUI.Workflow.Activities
Properties
References
Service
State
View
WorkItem
HandleRunWorkItemCompletedActivity.cs
RunWorkItemActivity.cs

When taking a close look at the previous figure you will notice that there is more than one activity for executing a WorkItem in my solution. You will find two activities in my solution – one for running the WorkItem and a second one that is a Handle Event activity. That is because the Workflow runs the WorkItem asynchronously, of course, and our code needs to call back into the workflow when it has completed its job.

Taking a closer look at the implementation of the RunWorkItemActivity will give us more details about the implementation. As you can see in the subsequent figure it shows that the activity itself does not execute the WorkItem itself as well:


TechEdCab - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Workflow Data Tools Test Analyze Window Help
Debug
Mixed Platforms
100%
Toolbox
Server Explorer
RunWorkItemAc....cs [Design]*
RunWorkItemActivity
CallRunWorkItemByType
Solution Explorer - Solution 'TechEdCab' (17 projects)
View
WorkItem
HandleRunWorkItemCompletedActivity.cs
RunWorkItemActivity.cs
RunWorkItemTaskActivity.cs
mszCool.Samples.Practices.CompositeUI.Context
Properties
Error List
Output
Ready

CallRunWorkItemByType System.Workflow.Activities.CallExternalMethodActivity

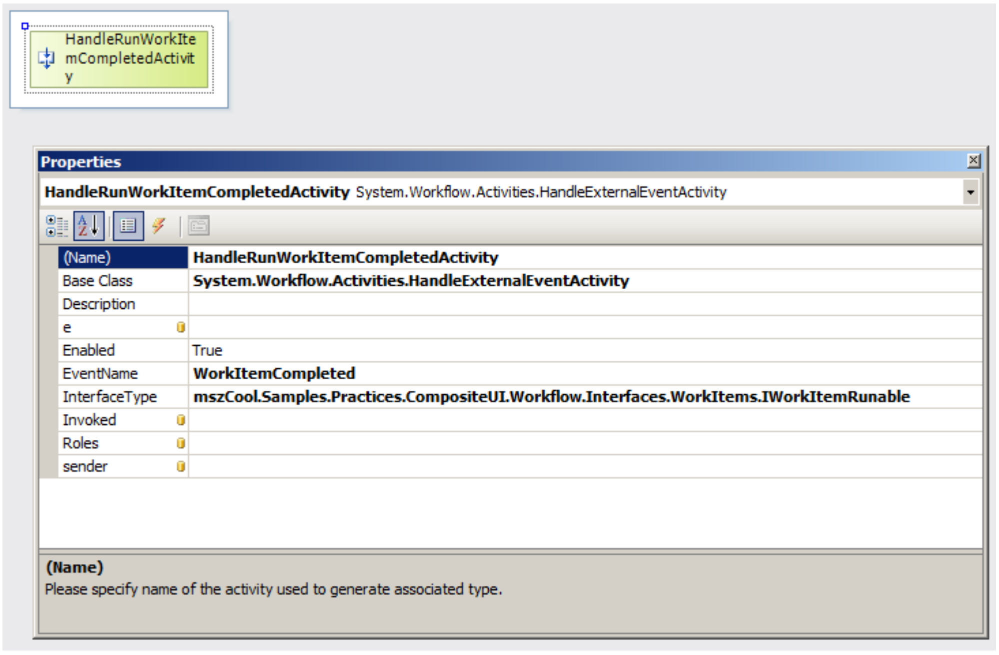
| | |
|---|---|
| (Name) | CallRunWorkItemByType |
| Description | |
| Enabled | True |
| InterfaceType | mszCool.Samples.Practices.CompositeU |
| MethodInvoking | |
| MethodName | RunWorkItem |
| parameters | Activity=RunWorkItemActivity, Path=Paramet |

**(Name)**
Please specify the identifier of the activity. It has to be unique in the workflow.

It calls its host through an interface called IWorkItemRunable which is part of our solution as well. Why is that the case? Well that has several reasons:

- ✓ First of all, architecturally the Workflow is not responsible for executing the WorkItem itself. He just responsible for orchestrating the steps within a composite WorkItem. The composite WorkItem then is responsible for instantiating, initializing and launching the WorkItem. But that has a second reason as well – one that is more technically oriented.
- ✓ The other reason why the Workflow is not able to execute the WorkItem directly has to do something with Threading. You have to know that every Workflow Instance is executed in a background thread. WorkItems on the other hand as part of the UI are executed in the UI thread of the application. They also access the UI thread by displaying views. As the Workflow runs in a separate thread it has no access to the UI thread. It even doesn't know the UI thread at all. Therefore it calls out the host application which in turn knows the UI thread and can switch over to it.

Before we can get one step further we need to take a look at the HandleEvent activity which gets notified by the hosting application when a WorkItem has completed its work:


HandleRunWorkItemCompletedActivity

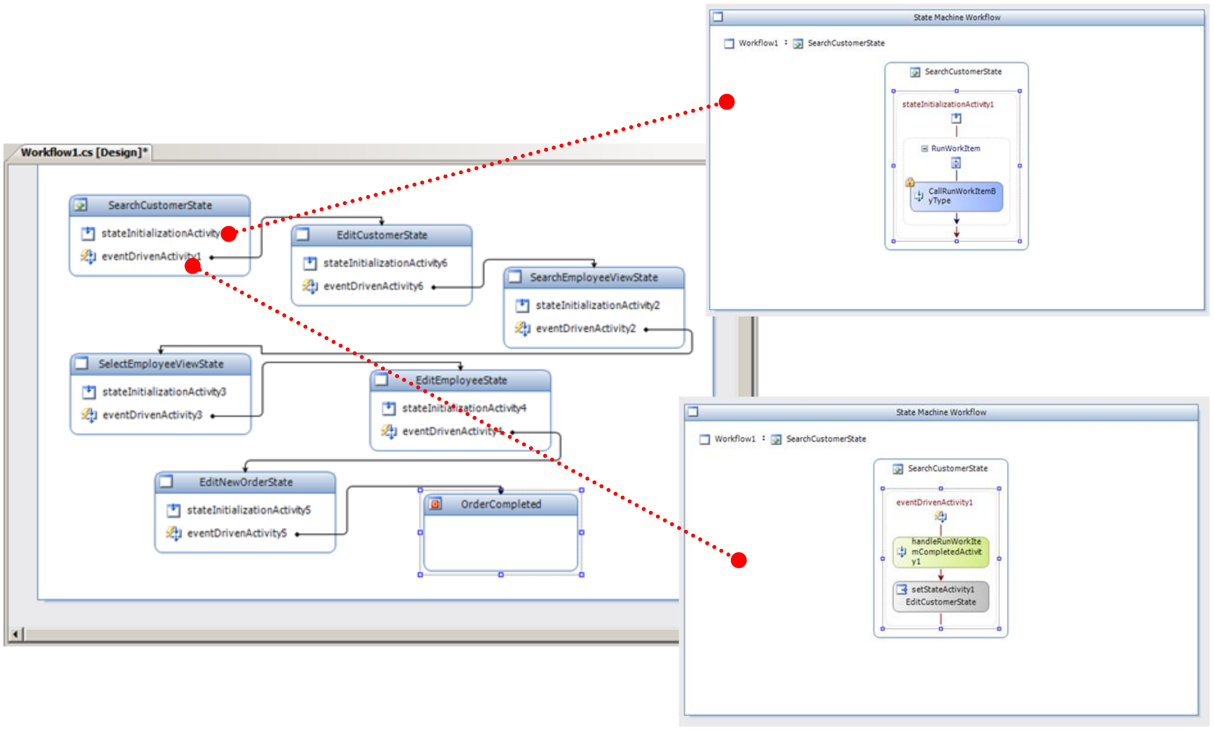
**Properties**

**HandleRunWorkItemCompletedActivity** System.Workflow.Activities.HandleExternalEventActivity

| | |
|---|---|
| (Name) | **HandleRunWorkItemCompletedActivity** |
| Base Class | **System.Workflow.Activities.HandleExternalEventActivity** |
| Description | |
| e | |
| Enabled | True |
| EventName | **WorkItemCompleted** |
| InterfaceType | **mszCool.Samples.Practices.CompositeUI.Workflow.Interfaces.WorkItems.IWorkItemRunable** |
| Invoked | |
| Roles | |
| sender | |

**(Name)**
Please specify name of the activity used to generate associated type.

This activity gets just notified by the application that hosts the Workflow through the WorkItemCompleted event defined in the IWorkItemRunable interface. It does not need to perform any specific actions in code as it just re-activates the workflow and lets it continue doing its work. These kinds of activities can then be used in a workflow to orchestrate WorkItems in your smart client – whereas you can use both, state machine based and sequential workflows. But I would recommend using sequential workflows for WorkItem orchestration as they allow more flexible combinations of transitions between WorkItems. The following figure illustrates a sample workflow:

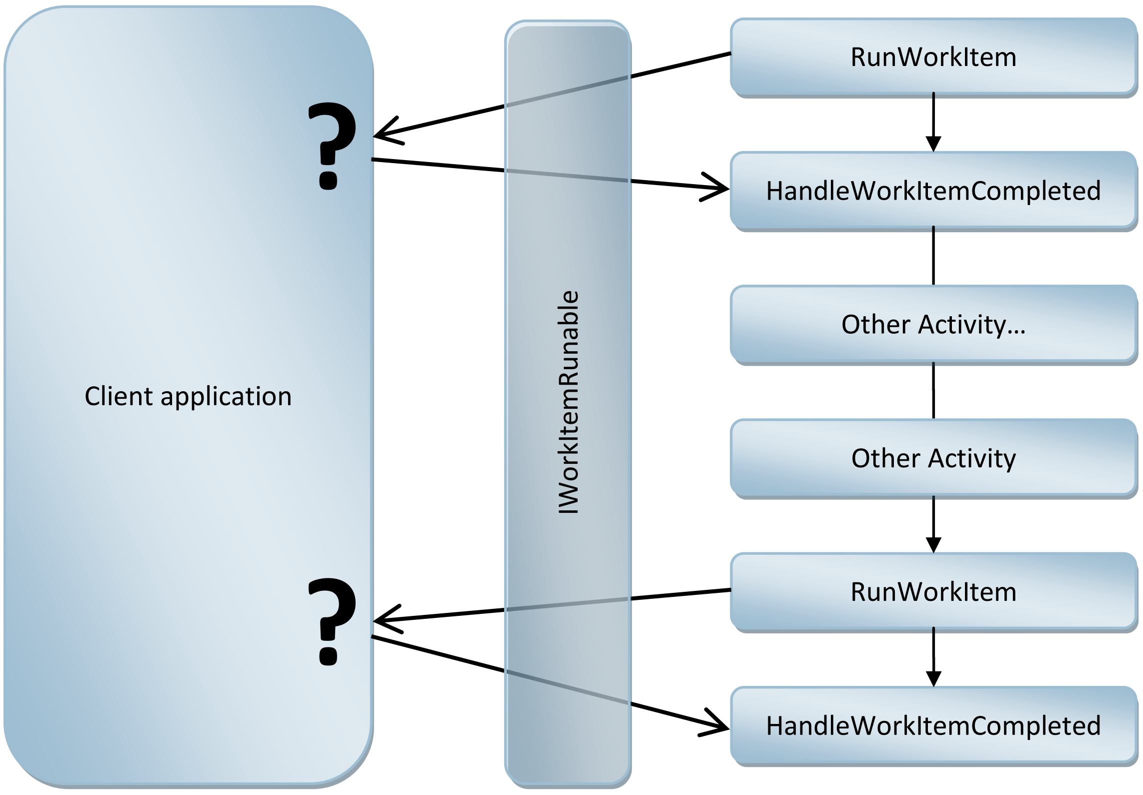


That allows quick creation of workflows for composite WorkItems without the need of digging into code all the time. But let's get to the next step. So far we have talked about activities that need to talk to the hosting environment in some way through an interface. As mentioned, that interface is the IWorkItemRunable interface as primary contract between the hosting application and the workflow. It's definition and involved event argument and parameter classes look as follows:

```
[ExternalDataExchange]
public interface IWorkItemRunable
{
    void RunWorkItem(RunWorkItemParameters parameters);
    event EventHandler<RunWorkItemCompletedEventArgs> WorkItemCompleted;
}

[Serializable]
public class RunWorkItemParameters
{
    private Guid WorkflowInstanceIdField;
    public Guid WorkflowInstanceId
    {
        get { return WorkflowInstanceIdField; }
        set { WorkflowInstanceIdField = value; }
    }

    private string WorkItemTypeNameField;
    public string WorkItemTypeName
    {
        get { return WorkItemTypeNameField; }
        set { WorkItemTypeNameField = value; }
    }

    private string WorkItemAssemblyNameField;
    public string WorkItemAssemblyName
    {
        get { return WorkItemAssemblyNameField; }
        set { WorkItemAssemblyNameField = value; }
    }
}

[Serializable]
public class RunWorkItemCompletedEventArgs : ExternalDataEventArgs
{
    private string WorkItemIdField;
    public string WorkItemId
    {
        get { return WorkItemIdField; }
    }

    public RunWorkItemCompletedEventArgs(Guid instanceId, string workItemId)
        : base(instanceId)
    {
        WorkItemIdField = workItemId;
    }
}
```

The interface includes the method RunWorkItem that allows the Workflow to tell the hosting application when that it should run a WorkItem now and it includes the Event Handler that is used by the hosting application to notify the Workflow whenever the execution of a WorkItem has been completed. To simplify the communication between the Workflow and the hosting application we packaged all parameters into classes and event arguments as you can see in the previous code snippet, as well.

Okay, let’s recap and think where we are now – we have designed an activity library with activities that can be used in a workflow. We agreed that the activities do not execute the WorkItems themselves because they are not running in the UI thread and because the Workflow is for orchestrating WorkItems (and eventually other things in the future) and not for knowing details about WorkItems in our framework. Therefore the activities communicate with the client application hosting the workflow through an interface called IWorkItemRunable. So what are the missing bits in the architecture? The following figure illustrates that!



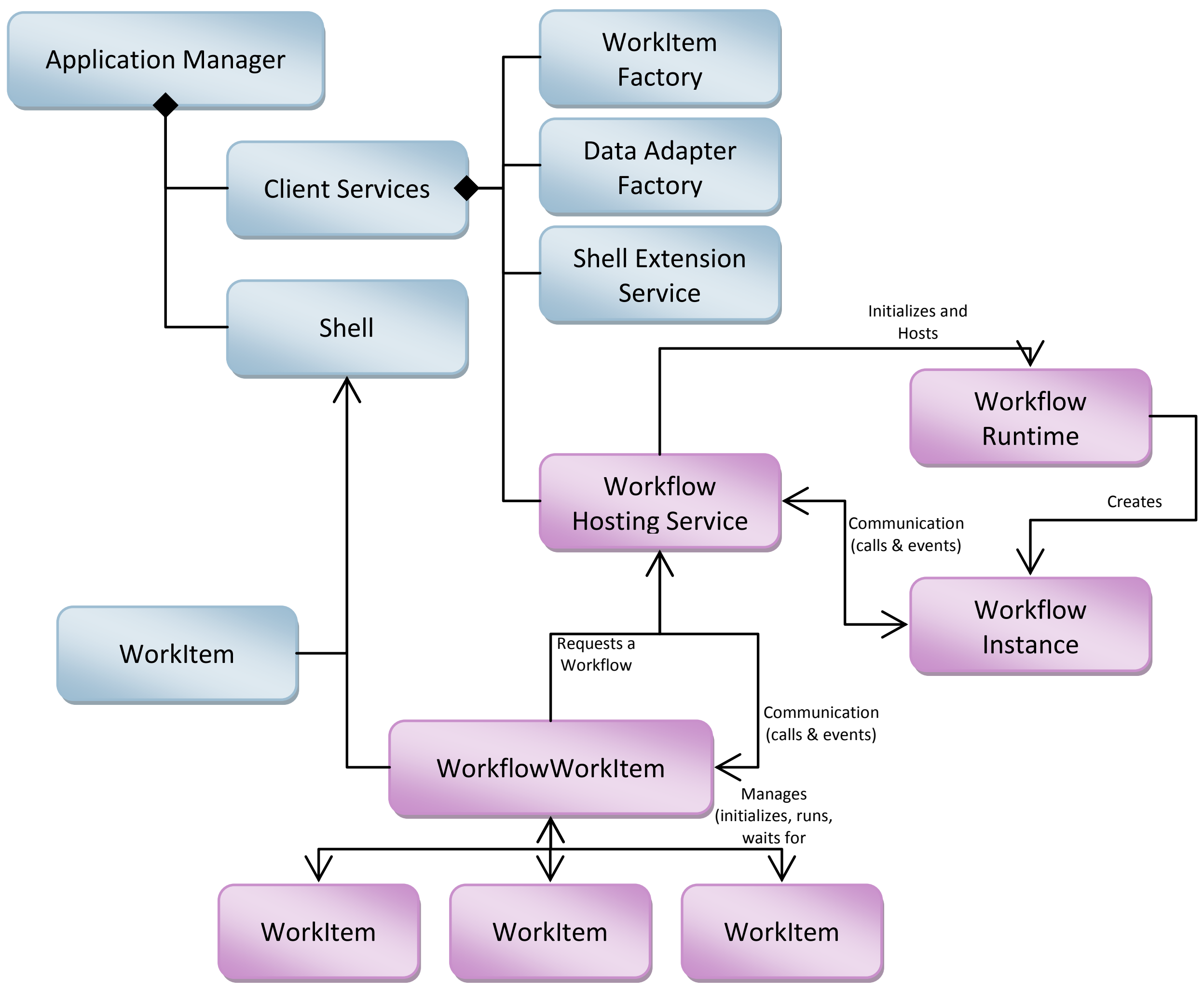
The previous figure clearly points out – our architecture talks about custom activities for a Workflow that is talking to the client application which is hosting the workflow through a defined interface. We haven't been talking about hosting the workflow and running the WorkItems in the client application, so far. That's actually the next step – we need to talk about the following details for our design:

- ✓ Who is hosting the Workflow in our client application?
- ✓ Who reacts to method calls from the Workflow and who tells the workflow when a WorkItem is completed?
- ✓ How can a parent WorkItem request a Workflow?

Well there are a couple of thoughts we should bear in mind:

- ✓ First of all hosting a workflow is something that should be done at a central location in our client application. This functionality shouldn't be implemented in each and every WorkItem that wants to leverage workflow functionality. **Conclusion:** we should create a client service called **WorkflowService** that is responsible for hosting the Workflow runtime and doing the communication between the runtime and the hosting client application. But – this service doesn't really know about WorkItem specifics and all that stuff. So while it is accepting method calls from Workflow instances it should actually not be able to really fulfill these method calls. They should be forwarded to another actor in the game – of course a WorkItem.
- ✓ Such a WorkItem is responsible for executing child-WorkItems based on what the Workflow requests through its method calls. Also this WorkItem needs to tell the Workflow when a child WorkItem has done its job and the Workflow can proceed with the next steps. The neat thing with it is that this WorkItem can and should do its job in a generic fashion. So you implement it one time and then you can create composite WorkItems which are orchestrated through a workflow. **Conclusion:** we build a generic **WorkflowWorkItem** that accepts notifications form the WorkflowService introduced before which in turn got those from the workflow from RunWorkItem activities. It also notifies the WorkflowService whenever a WorkItem completed its job. The WorkflowService in turn raises an event that notifies the Workflow instance which catches these events through its HandleWorkItemCompleted events that lead the workflow to continue its work.

Having that said we are coming to a complete design of our Workflow integration into the smart client that looks as follows (skipping the details about presenters and views for the sake of clarity:



As you can see, the Workflow Hosting Service is a client service that instantiates and and manages the workflow runtime. Whenever a WorkItem requests a workflow it uses the runtime to instantiate the Workflow Instance. The Workflow Hosting Service is doing also the whole communication between the Workflow instance and the client application. The WorkflowWorkItem is a generic implementation of a WorkItem that is able to accept requests sent from a Workflow Instance through the Workflow Service to the WorkflowWorkItem and it forwards event requests to the Workflow Instance through the Workflow Hosting Service (which is doing the communication). The WorkflowWorkItem is running child WorkItems (which are ordinary WorkItems) which the Workflow wants to run it. And that’s essentially it.

Of course there are many implementation details you need to be aware of, but I have published a sample implementation on my blog (http://blogs.msdn.com/mszcool) that demonstrates these concepts in action based on CAB/SCSF, WPF and Windows Workflow Foundation.

## Conclusion

In this paper you have seen how we have designed the architecture for the next generation smart clients of the HP **banqpro/** banking product. The basic patterns are very similar and simplified versions of many concepts from the Composite UI Application Block from the Microsoft Patterns & Practices team. These concepts are easy to understand and very easy to implement with WPF. The main reason why we have not selected CAB was, that the steep learning curve would have required too much learning effort and time compared to the number of features we would have finally used from CAB in the application (just WorkItems, Views, Presenters – none of the other features such as dynamic extensibility, the event broker or loosely coupled modules).

A special requirement that will find its way somehow in the future into the **banqpro/** client application is the integration of data gathering workflows which are defined by end users. In this paper I have demonstrated concepts that we have been thinking about during our architectural design sessions. I have to admin – the initial effort for integrating Workflow into clients is pretty high, especially if it is well aligned with your specific requirements. **I would never ever recommend doing that without a deep analysis whether that effort really pays off in your situation! It really depends on the requirements and if you analysis comes to the conclusion, the added-value of this pretty high effort of integrating workflows into the client does not bring enough value then KEEP your hands off of such approaches.** But if you have many different kinds of task flows on the client that need to be able to be designed in a comfortable and flexible way then integration of Workflow might make sense – always think twice before doing the decision.

Finally I wanted to say thank you to HP for the great co-operation and ideas for getting something like that put together. Also I would be happy about any feedback for this paper and if anyone has questions feel free getting in touch with me through my blog.

## Links to Resources

**Windows Presentation Foundation**

http://msdn.microsoft.com/en-us/library/ms754130(VS.85).aspx

http://msdn.microsoft.com/en-us/library/aa663364.aspx

http://msdn.microsoft.com/en-us/library/aa970268.aspx

http://msdn.microsoft.com/en-us/library/bb546194.aspx

http://www.microsoft.com/downloads/details.aspx?familyid=05755a9d-98fa-4f16-bfdc-023e3fd34763&displaylang=en

**Windows Workflow Foundation**

http://msdn2.microsoft.com/en-us/library/ms735967(vs.85).aspx

http://msdn.microsoft.com/en-us/netframework/aa663328.aspx

http://msdn.microsoft.com/en-us/library/aa663362.aspx

http://msdn2.microsoft.com/en-us/library/bb264459(VS.80).aspx

http://msdn.microsoft.com/en-us/library/ms741723(VS.85).aspx

http://msdn.microsoft.com/en-us/library/aa973808.aspx

http://msdn.microsoft.com/en-us/library/bb887609.aspx

http://msdn.microsoft.com/en-us/library/cc626077.aspx

**Composite UI Application Block & Smart Client Software Factory**

http://msdn.microsoft.com/en-us/practices/default.aspx

http://msdn.microsoft.com/en-us/library/aa480482.aspx

http://www.microsoft.com/downloads/details.aspx?FamilyID=5f9a8435-1651-4be2-956d-0446a89a7358&DisplayLang=en