# The Web Service Sample

## Catapulse Pacitic Bank

The Rational Unified Process is a roadmap for engineering a piece of software. It is flexible and scalable enough to be applied to projects of varying sizes. Moreover, it is fairly independent of the type of software being developed. You can apply RUP to standalone applications, client/server applications and web applications with equal facility. In this sample, we will walk you through the application of RUP to develop a set of software services offered by a bank.

Consider the case of Catapulse Pacific, a bank that built its business on the basis of its currency exchange services in the 80s and early part of the 90s. Customers carried out currency exchange over the phone or on a walk-in basis in the bank. Soon, the popularity of the service prompted Catapulse Pacific to open counters at banks, malls and international tourist attractions.

When the Internet emerged as a powerful medium of conducting business, Catapulse Pacific seized on the opportunity to put its currency exchange services online. The benefits would be enormous. By making the service globally accessible and on 24-7, customers could get past the inconvenience of finding a location, waiting in line and making sure they arrived during business hours. Catapulse Pacific would save on the operational costs of running counters at multiple locations.

This sample describes the process used by Catapulse Pacific to migrate their services on the web. We'll describe the key actors, activities and artifacts on the way. Where relevant, we'll describe briefly the key technologies used to develop the project. If you are new to RUP, you will find this sample useful as a starting point. Readers familiar with RUP will be able to observe RUP in action as applied to relatively new technology.

## Inception

RUP is not a sequential but rather an iterative process. Nevertheless it is broken up roughly into four phases, each of which can be cycled through multiple times: inception, elaboration, construction and transition. We'll take our project through the first three phases and present artifacts to mark key milestones reached during these phases.

At the very beginning of the project the business process analyst plays a key role. This person can be a team of people as well. The business process analysts' job is to capture the business requirements of the project. The analyst also captures key requirements usually in the form of use cases. These use cases are focused around business entities. In other words, the analyst looks at the use cases the way a customer would.

A key artifact to emerge from her efforts is the vision document. The vision document describes the market, the reason behind the effort and captures the idea behind the product being developed. The vision document defines the problem, explains the product concept and positions the product vis-à-vis the company's priorities. It then outlines the types of stakeholder and customers that are important to the product. A stakeholder is defined as an entity with a vested interest in the end result of the system, for example clients, business partners and investors. It then goes on to provide thumbnail sketches of the stakeholders and customers. Part of each sketch is a description, type of involvement in the system, why they are interested in the end result and what deliverables are they expecting. This is important information to document in the inception phase of the development life cycle. It allows the subsequent actors involved in the project to identify the type of people they should be collecting input from and what needs to be delivered to them at various points in the project.

| Phase | Artifact | Actors | Rational program used | Location |
|---|---|---|---|---|
| Inception | Business vision document | Business process analyst | RequisitePro | .\Project\Projects.rqs |

While the vision document is being written, another artifact is already taking shape. This is the glossary, a document that keeps track of terms and acronyms used to describe things in the project. This functions as a project dictionary of sorts and is constantly updated throughout the life of the project. This brings us to a key point: artifacts generated in RUP are not static but rather dynamic pieces of information. They can be modified during an iteration of the phase that they belong to.

| Phase | Artifact | Actors | Rational program used | Location |
|---|---|---|---|---|
| All | Glossary | Business process analyst, System analyst | RequisitePro | .\Project\Projects.rqs |

These documents are Microsoft Word documents. Since projects can produce copious output in terms of documents, Rational provides an application that manages requirements and also acts as a repository for all the documents generated during any phase of the system. The application is called Rational RequisitePro or ReqPro for short. Before creating the first document in ReqPro, the different types of documents that the project will produce must be created. This is done by clicking on the *File | Properties* menu and going to the *Document Types* tab. Once in this tab, a new document type can be created. ReqPro provides the entire set of documents that can be created in RUP as templates. When creating a new document, you can simply inherit from this template and a sample document will automatically be generated with the relevant sections and brief explanation of what each section must contain. This can provide a jump-start because a thorough understanding of RUP is not required to get going.

All document types don't have to be defined at the beginning of a project. You can come back and set up new document types later. A good practice is to decide on the minimal set of documents that your project requires them and create document types for them up front. By specifying these types, you are specifying the minimum set of documentation that your project must create as part of its life cycle. Then create document types only as need evolves.

The Business process analyst also captures some use cases during the inception phase. Use-cases and other Unified Modeling Language (UML) diagrams are best captured in Rational Rose. If you start Rose and load the sample file Webservice.mdl, you can take a look at these use cases. When you first load Rose, you are taken to the main class diagram. One of the main areas of confusion when using Rose is that the program does not impose any kind of sequence in the way the system is modeled. While this is a good thing, there tends to be no "starting point" in reading a Rose file. For our project, the first diagrams you should look at are the ones in the *Use Case View* folder and the *Business Use-Case Model* subfolder in the top left pane. The use case *Global View* breaks down the higher level features offered by the system. The use case *Account Manager* captures the functions performed by a business entity of Catapulse Pacific, namely, the creation and maintenance of customer accounts.

| Phase | Artifact | Actors | Rational program used | Location | Section |
|---|---|---|---|---|---|
| Inception | Business use case model | Business process analyst | Rational Rose | .\Project\Webservice.mdl | Use Case View -> Business use-case model |

## Elaboration

Once the business vision has been laid out and the glossary has been initiated, we are ready to collect requirements. We introduce two key actors who shepherd the project through this aspect of its life cycle.

The system analyst is responsible for the task of gathering requirements and capturing them using models like use cases. He captures the needs of the customers and shapes the system. Part of his responsibility is to also capture interactions between use cases. A system analyst doesn't necessarily do all the work: it simply remains his responsibility to make sure it gets done.

The system analyst contributes to two preceding documents: the vision document and the glossary. He also produces two key documents: the requirements management plan and the stakeholder requests document. These documents are essential to set the stage before going out to elicit and capture requirements. Lets take a quick look at what goes on in each of these artifacts.

The Requirements management plan (RMP) is a document that describes what type of requirements must be collected. It is a sort of a "meta-requirements" document. In the sample, the System analyst decides to collect three types of requirements: features, needs and change requests. Each requirement can be further described by properties such as importance, cost, benefit, etc. These properties are captured as attributes and the RMP must list and describe each attribute for every type of requirement that will be collected. Once these requirement types have been established, they must be entered into ReqPro. The first step is to add the requirement types themselves. This is done by clicking on *File | Properties* and then selecting the *Requirements* tab. Once the requirement types have been entered, their attributes can be added by clicking on the *Attributes* tab. It would be nice if this information could be used by ReqPro to automatically generate the RMP. However, this not being the case, you will have to make sure that the information about requirements is consistent in both places: the RMP document and ReqPro.

| Phase | Artifact | Actors | Rational program used | Location |
|---|---|---|---|---|
| Elaboration | Requirements management plan | System analyst | RequisitePro | .\Project\Projects.rqs |

Along with the RMP, the System analyst also generates the *Stakeholder requests* document during the elaboration phase. This document is used to collect the actual requirements. The concept behind this document is simple: establish a set of questions that should be asked during a stakeholder interview that will yield all the information needed to characterize a set of requirements. The reason this document is so important is lends structure to the process of requirements collection and analysis. By collecting information uniformly, the requirements can be categorized, prioritized and documented consistently.

| Phase | Artifact | Actors | Rational program used | Location |
|---|---|---|---|---|
| Elaboration | Stakeholder requests | System analyst | RequisitePro | .\Project\Projects.rqs |

Armed with this information, the team can then go about collecting the requirements. A second iteration of the inception phase can occur at this time in order to collect high-level requirements. During the elaboration phase, the requirements collected must have sufficient detail to enable the System architect and designer to construct the system. Once collected, requirements are created in ReqPro. The attributes of each requirement can be assigned values at this stage itself. These requirements can be used to generate the Software requirements plan (SRP). In the sample, we show two SRP documents, one generated during the inception phase and another during the elaboration phase. These two documents show two iterations of the SRP.

Once requirements are collected the Business object model can be completed. In the Rose file, these models are captured in the *Logical View* of the system under *Business object model*. In the sample, we use collaboration diagrams to model the key functionality to be provided by the system. In our case, these functions are *advertise currency*, *buy currency* and *sell currency*.

## Construction

Once a sufficient set of requirements have been captured, the design and implementation of the system may begin. We now enter the construction phase of the development life cycle. During the design, we match up our customers' requirements with existing technology to see how we can best construct the system. In our sample, we use Microsoft .NET to produce a Web Service. Web services are components that run on a server and can be invoked using an XML based, operating system independent protocol called Simple Object Access Protocol (SOAP). Microsoft .NET offers an implementation of SOAP over HTML (hence the name *web* service). Results from the Web service are returned in XML. As part of the system, Catapulse Pacific will move account management online and provide this web service that allow online currency exchange.

A third party bank or financial institution will most likely use the web service and offer it to their clients as part of their services. In such cases, the client that consumes the Web service may be a script that is embedded in a vendor's web page. Each customer of the system must have set up an account previously. Setting up an account involves creating a debit and credit account. These accounts can be the same but are internally tracked as two separate accounts. When a currency exchange is made, the money is moved from the debit account, converted to the desired currency and moved into the credit account.

Clients of the Web service can browse currencies and query the exchange rate of each currency. When the actual exchange takes place, the Web service returns a transaction ID. Receipt of the transaction ID does not guarantee a transfer of funds. The actual transaction may occur later but no later than in 3 hours.

When designing and implementing the system, we use Rose heavily. Take a look at the *Design Model* in the top left pane of Rose. It has four subfolders. Each subfolder is a layer in the system.

| Layer | Description | Diagrams used in Rose |
|---|---|---|
| System software | Contains components such as operating systems, databases, interfaces to specific hardware, etc. | Class diagram |
| Middleware | Contains components such as GUI-builders, interfaces to database management systems, platform-independent operating system services, and OLE-components such as spreadsheets and diagram editors. | Class diagram |
| Business-specific | Contains business specific components, used in several applications. | None for our sample |
| Application | Contains the application specific services and value-added software developed by the organization. | Sequence diagram |

Rose allows you to pick the best UML diagram that illustrates what you are attempting to convey. In our sample, we show you how to use a few important ones. Separating the design model into layers conveys an excellent breakdown of the system architecture.

A critical design artifact is the class diagram for the Web service. This can be found in the *Web service* subsystem folder under the *Application* layer of the *Design Model* in the Rose file. The class diagram shows the Web service interface as defined in the business model and then breaks down the various classes and their methods and properties. Elements can be copied in Rose between different folders, i.e. phases of implementation. When this is done, Rose will cross-reference the element by tagging with its origin. You can see this with the *Web Service* interface.

In the class diagram we also show the various system modules that the Web service uses. This allows us to map dependencies between our modules and system services. These dependencies intimately affect development and deployment.

Lets take a quick look at the implementation of the system. Our implementation is written in C# in Microsoft .NET or Visual Studio 7. We used beta 1 of the product so if you are using a later beta or the production version of the code, you may encounter some differences in the implementation.

| Phase | Artifact | Actors | Program used | Location |
|---|---|---|---|---|
| Construction | Source code | Developers, Testers, Managers | Microsoft Visual Studio .NET beta 1 | .\Source\ CatapulsePacificExchange\ CatapulsePacificExchange.csproj |

The actors mentioned here are really categories of actors. The RUP specifies the different roles within each category. Loading the csproj file into Visual Studio.NET will show the implementation of the Web service. Explaining how a Web service is written and executed is beyond the scope of this narrative, but you can follow some of the links provided in order to gain a thorough understanding.

The implementation details of the system are captured in the *Component View* in the left hand pane in Rose. The component view in Rose is used to set up a description of the subsystems used in the implementation. Here, Rose is more restrictive in terms of the diagrams that can be used to describe the system. The only one allowed is a component diagram because it is the only one that makes sense in this context. The *Currency Exchange Service* diagram maps out the implementation of the sample.

At this point, since you are probably getting familiar with Rose, lets explore an important feature. Rose allows a lot of documentation about the design to be embedded in the models themselves. Each element in Rose has associated documentation. How does one view the documentation? First, you need to locate the element used in the diagram. Each element is listed in the same view in which the diagram shows up. For example, in the *Implementation View*, under the component diagram, you will see each of the elements in the form of packages and components used in the diagram. Double-clicking any of these elements will bring up the element's specification. The specification contains additional documentation for each element. You can also invoke this dialog by double-clicking on the element from within the diagram itself.

Well, if the documentation is embedded all over the Rose diagrams, does one have to navigate to each element to view all of it? It would seem you would have to but here the Rational Development Suite comes to the rescue with a report generation utility called SoDA. SoDA allows the contents of the Rose file to be selectively saved to a Microsoft Word document. SoDA also works with ReqPro, which means that you can select the type of information you want from ReqPro and use SoDA to save it to a Word file. Some such sample reports are part of the sample.

| Artifact | Location | Contents |
|---|---|---|
| SoDA Report – design.doc | .\Project | Analysis and design view from Rose. |
| SoDA Report – Requirements Summary.doc | .\Project | Description field of each requirement in ReqPro. |
| SoDA Report – Requirements.doc | .\Project | Requirements in ReqPro with description and attributes. |
| SoDA Report – Use Case Model Survey.doc | .\Project | Use-case diagrams and documentation from Rose. |