SC-BAT Delivery Phases

We are conducting the SC-BAT project in two complimentary phases. In the first phase we created an initial Guidance Package, reference implementation, and documentation. In the second phase we will incorporate feedback to improve upon and expand the Guidance Package and documentation. The primary driving force is a set of smart client features and capabilities illustrated by the second reference implementation.

We have started development of the second reference implementation, the Bank Branch Workbench. We are relying heavily on feedback (from the community site, customer queries, the workshop, etc.) to identify the features and guidance to illustrate with the second reference implementation. **This is a great opportunity to influence what we do in the second phase of the project!**

What is the best way to influence the development? Submit your requirements/feature requests to the SC-BAT community site, and help us to understand/refine the requirements.

From the requirements we've gathered so far (see **Customer Requirements**), we have identified a set of technical objectives (see **Technical Objectives**) to demonstrate selected features/capabilities and design challenges. Given these objectives, we selected a business and application scenario (see **Business Scenario**). We chose one that should be rich enough to provide the opportunity to demonstrate all kinds of features and challenges. From the business scenario we created a set of use cases for the application. (The help file, SC-BAT.CHM, contains the use case documentation.) The use cases are the input to the development stories that we select and implement in our weekly iterations.

Below is a summary of what was delivered for the first reference implementation, followed by a description of the requirements, objective, and business scenario for second reference implementation.

We look forward to your feedback!

Reference Implementation 1 Summary

The following categories describe the elements of the first reference implementation.

- Modularity
    - The application has one module loaded based on configuration
    - The application provides shared services as well as services provided by the module
    - The application components communicate via events
- User eXperience
    - Shell provides two workspaces used by the work items
    - View navigation
    - Customization of the shell form and use of application commands
    - Status bar describing the state of service connectivity
    - Communication of the progress of user operations (downloading files)

- Deployment and updates
    - ClickOnce deployment
    - User notifications about new versions
- Web service communication
    - Abstraction of the service into the service agent
    - Asynchronous service invocation with timeout
    - Download/upload of multiple files over HTTP channel
    - Resuming download/upload after server communication has been interrupted and then reestablished
- Local storage
    - The application stores business entities (appraisal data) and files (attachments) in the local file system
- Security
    - The application uses Windows authentication to identify user
- Manageability
    - The application uses the Enterprise Library Exception Handling Application Block to process exceptions
    - The application uses Enterprise Library Logging Application Block to keep an audit log of the interactions with the server.
    - CAB configuration

Reference Implementation 2 – The Bank Branch Workbench

Customer Requirements

Below is a summary of our current understanding of the most common community requirements.

- Modularity
    - A standardized shell that is developed independently of the pluggable application modules that can be hosted in that shell
    - Make it possible to add and remove modules
    - Change the look and feel of the client without redeploying the shell
    - Load modules on demand, not just on startup
    - Provide a way to develop modules in a simple and predictable fashion. Provide a module development environment (a SC Module Software Factory)
- User eXperience
    - Change the user experience based upon the user's role

    - The application should be operational (possibly with limited capabilities) when there is no connectivity to some services
- .Deployment and updates
    - Manage module profilecatalog in a central location. Retrieve profilecatalog from client machine on application startup.
    - Deployment of new modules and module versions.
- Web service communication
    - Integrate legacy components/systems
    - Communicate with multiple services that use different authentication and authorization schemas and subsequently require different log-on steps and credentials
- Local storage
    - Show use of cached reference data
    - Show on demand caching of data
- Security
    - Encrypt data and show how to detect data tampering
    - Support different user roles. Expose different application capabilities to different roles.
    - Load only the modules required by a role.
- Manageability

  Note: we did not identify new manageability requirements

Technical Objectives

In this phase we want to separate the concerns of the shell developers and the module developers.

A partial list of the shell developer questions includes:

- What is the recommended structure of the “foundational” modules that provide shared services to other modules?
- How do I control module loading sequence?
- How do I control what modules are loaded depending on user role?
- How do I recognize when new module versions are available?
- How do I load modules on demand?
- What services should be provided in the shell and what services should be provided in foundational modules?
- How do I enable or disable client capabilities depending on the user’s role?
- What “hosting” services should be provided to the application modules so they can
    - Register themselves with the shell
    - Recognize the shared services (possibly from a foundational module)

        - Recognize what other modules are in the shell
        - Communicate with other modules
        - Analyze the shared application state (also referred to as the client context)
        - Identify the workspace allocated to the module
        - Identify and modify the extensibility sites
    - How do I implement a flexible client layout (look and feel) that can be modified without impacting the shell and other modules?
    - How do I make the host resilient to failing (or malicious) modules? For example, blacklist such modules and prevent them from loading again?
    - What is the skeleton of the managed-code application module?
    - What interfaces should an application module implement and what is the required module initialization (registration) sequence and rules?
    - What parts of application module development can be automated?
    - What is the wrapper-module for Web Client Applications and how does it support two-way communication between the Web pages and the client components?
    - How do I manage multiple user credentials when communicating with different services?
    - How do I support collaboration between different views in different modules so that they can refer to the same data?

Many of the module developer concerns are the flipsides of the concerns of the shell developer. For example:

- How do I initialize the module in the shell host?
- How do I discover what workspace I should be working in?
- How do I discover what service and other modules are already in the shell?
- How do I recognize that a new work item and service is required and that a corresponding module must be loaded?
- How to I communicate what I am doing to the other modules?
- How do I find out what other modules are doing?

The phase should also address the tasks of creating a module development environment (a Module Software Factory) for developing modules hosted in our shell architecture. This implies that we should:

- Have a module development shell solution
- Create different starting points (project templates) for different module developments. For example:
    - Managed code module
        - Application module
        - Foundational module
    - Unmanaged code module
    - Web module

- Reference the shell dll from the module solutions (i.e. do not include the shell code in the template)
- Allow the module developer to select the foundational modules they need to develop their module
- Write how-tos that describe how to create modules for that specific shell architecture
- Automate the common tasks of writing such modules

Business Scenario

The reference implementation represents an application developed for Global Bank, a midsize, traditional bank. The reference implementation provides business logic that integrates back-end services to support different roles within a bank branch. The bank's back-end systems consist of different hosts with different authentication systems. Access to the host systems is enabled through Web services. You can see an expanded business scenario and description in the SC-BAT.CHM help file.