SELECT

BUSINESS SOLUTIONS

Select Business Solutions an Architected Approach for Microsoft .NET Development

White Paper

Version: 1.1

Disclaimer and Trademarks

For more information about Select Business Solutions Inc., or to download an electronic copy of this document please visit the Select Website: http://www.selectbs.com/

Table of Contents

Introduction

Computing is fast becoming distributed, fuelled largely by the Internet. Demands on computing from the Business require system integration in support of end-to-end processes, in a robust flexible manner. The challenge is to find better, quicker ways of developing such distributed systems. The key to success is the ability to define and create an architecture that will provide the flexibility and the capability to integrate many disparate systems.

Component-Based-Development (CBD) addresses this challenge; essential ingredients of CBD and the subsequent scope for this paper are: A proven development process and a technical platform on which to develop and deploy our component-based solution.

Select Perspective is a CBD process for creating business solutions in a component-based, service-oriented architecture.

Supply, Manage and Consume (SMaC) is the framework on which Select Perspective is based. It is a software development approach that represents a clear separation between the supply of services and the assembly of services into solutions. Select Perspective is structured using workflows; at the core of the process are two workflows that address the key principles of component architecture:

- Business Architecture – specifies the architecture of the application in terms of its *business components*;
- Technical Architecture Delivery – creates the common technical software environment ensuring components can co-operate and share a technical infrastructure.

Microsoft .NET provides a technical platform; the goal of which is to accelerate the shift towards Internet powered, distributed computing.

This white paper is intended for IT Manager, Architects and Developers who wish to fully leverage the Enterprise capabilities of .NET by using a proven CBD process and supporting toolset, such as Select Perspective and Select Component Factory. This paper explains the architectural emphasis of Select Perspective, in particular the Technical Architecture showing how architectural principles are applied to developing distributed systems for Microsoft .NET. The reader is assumed to have an interest in .NET

What is .NET

.NET is a combination of tools and platform technologies [1], targeted at the development and deployment of distributed systems. Microsoft recognizes Web Services as a key enabler to distributed computing, evidenced by the support for Web Services across the platform. For a more complete definition and description of Web Services – refer to the Select Business Solutions' white papers [2]

The Microsoft .NET platform consists of:

- Developer Tools;
- Servers;
- Foundation services;
- Device software.

Developer Tools

The .NET Framework and the Microsoft Visual Studio® .NET IDE make writing Web services as simple as possible. The .NET Framework is the way an application interacts with the overall .NET Platform. The common language runtime and class libraries (including Windows Forms, ADO.NET, and ASP.NET) combine to provide services and solutions.

Servers

.NET Servers can be categorized as either:

- "Platform" - Win 2000 (Server, Advanced Server, Datacenter Server) and Windows .NET Server;
- "Plumbing" - SQL, MIS, MOM, App Center Server;
- "Integration" - biztalk, exchange, HIS, sharepoint, commerce, Content Mgmt.

Building Block Services

.NET building block services such as security, benefit developer productivity.

Devices

.NET provide a range of device software to make the user experience simple and compelling, devices include: phones, PDA's.

Select Component Factory and Select Perspective

Select Component Factory is a suite of modeling tools, which supports UML, Business Process modeling, Database modeling and component management capabilities. Select Perspective provides a defined process for using these models in a supplier/consumers component and service based development process.

Select Component Factory includes:

- Select Component Architect – Business Process Modeling through UML and data modeling;
- Select Component Manager – Cataloguing, Searching, Configuration control of components, UML Components and Deployment modeling;
- Select Synchronizers (includes): C# Sync, C++ Sync, Java Sync, VB Sync and XML Sync;
- Reviewer for Select Component Architect.

About Select Business Solutions

For more than 10 years, Select Business Solutions has created a successful track record with tools and process solutions and is generally recognized as being one of the early adapters of Service and Component Based Development (CBD) worldwide. The technology behind the Select tools is continually recognized as being innovative, ensuring that customers' demands are met to the fullest satisfaction. It is this three-way focus on development, customers and emerging markets that makes Select Business Solutions a leader in its field.

Principles of Select Perspective

Select Perspective is structured using workflows that define: activities, roles, products, techniques etc.

The workflows collaborate, in the form of the SMaC process framework.

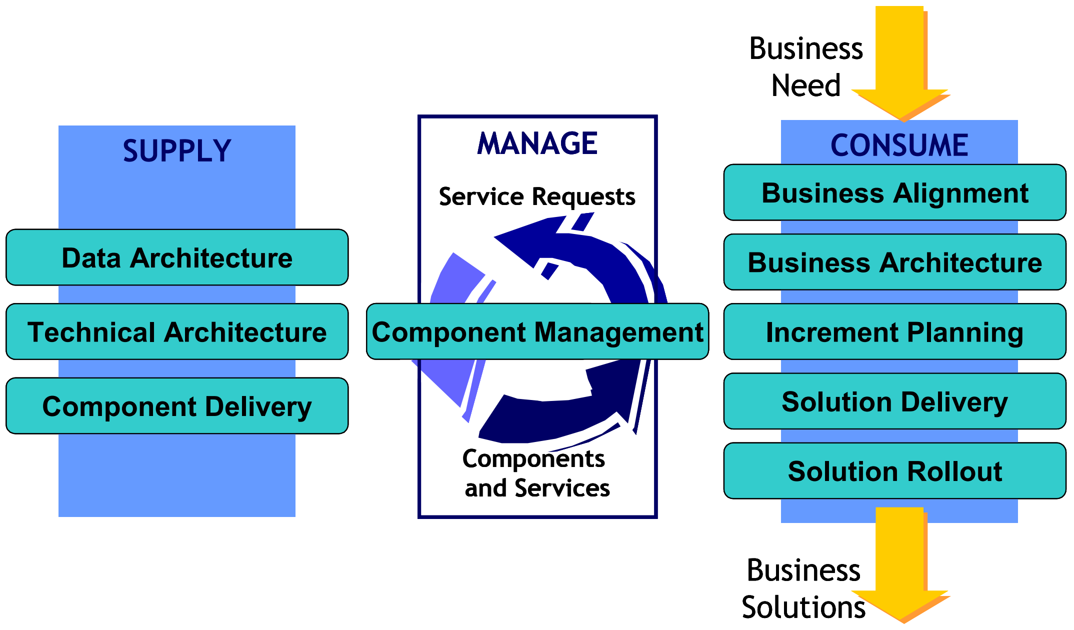


Figure 1 - Select Perspective workflow collaboration within SMaC

Workflow summary information provided as context for this paper:

- Business Alignment focuses on describing business processes and requirements;
- Based on the emerging requirements Business Architecture is the workflow for identifying and specifying the business components and the services they will offer;
- Technical Architecture defines a supporting software infrastructure, creating the common technical software environment for the solution;
- Component Management describes a set of activities for the management of the organizations component asset e.g. the broking of service requests, cataloging and configuration of components;
- Component Delivery focuses on the internal design and implementation of the component;
- Solution Delivery focuses on the User Interface design and integration of business components to form a solution.

The realization of components (Supply) and assembly of solutions (Consume) is enabled by a combination of the Business Architecture and Technical Architecture. This paper will focus on the architectural principles of Select Perspective, paying particular attention to the Technical Architecture process-driven use of the .NET platform.

Select Perspective Architectures

Business Architecture identifies and specifies the components and their services - aligned with the business processes and requirements; it is not concerned or necessarily constrained by technology choices. Its purpose is to guide the implementation of components towards meeting a required set of business functionality. However technical choices and constraints will impose themselves on the component implementation and final solution, it is the purpose of the Technical Architecture to make these technical choices. Ultimately the Component Delivery part of Select Perspective will create technology specific component implementations.

This approach bears an uncanny resemblance to the OMG's Model Driven Architecture (MDA). The MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. It defines a set of linked models. Linking and traceability are the cornerstones of MDA. The proposed layered models include;

- 'Computation Independent Business Models'.
- 'Platform Independent Component Views'.
- 'Platform Specific Component Views'

Using the models defined above Select Component Factory allows clear levels of abstraction, with viewpoints;

- Computation Independent Business Models are designed within the Business Process Modeling section of Select Component Architect. These relate only to the business domain and the business processes.
- Platform Independent Component Views are core to Select Component Architect. This unique strength allows you to design black-box components, their static relationships and procedural interactions. Separation from the Platform Specific Models is provided through Select Component Manager.
- Once the component specification contracts have been defined in the PIM, Platform Specific Models can be created for component-based, object-based and relational implementations. Select Component Factory employs a 'design by contract' paradigm for component specification, for example, Web Services or SOAP, followed by a platform specific implementation.

Technical Architecture Delivery for .NET

The Technical Architecture delivers a set of concrete artifacts that provide a solid framework within which components can be created and solutions developed. It is a key requirement for the successful delivery of component-based solutions. Components must guarantee that they can "inter-operate": co-exist and co-operate within a common technical environment. *Technical architecture Delivery* is the workflow that ensures a technical environment is: defined, communicated, reused, enhanced and maintained across multiple projects.

One key architectural decision that is usually taken up-front is the choice of platform. For the purposes of this white paper, the .NET Framework is the target platform. The choice of .NET could arise through organizational policy (corporate use of Microsoft platforms) or through an evaluation cycle based on the non-functional requirements of the system. Either way, the .NET Framework will form part of the technical architecture landscape.

Select Perspective encourages and enables the use of patterns to assist the designers and architects to speed development. When defining a technical architecture, patterns fulfill two distinct roles. In the first of these roles, the architect uses patterns to guide the design of the technical architecture. Some of these patterns may be platform-agnostic while others will use platform-specific techniques or address platform-specific issues (such as the J2EE Pattern catalog). The second role of patterns is to define and document the technical architecture itself, as discussed below.

The primary inputs for a technical architecture are the non-functional and functional requirements derived during Business Alignment, they will constrain the way in which .NET will be applied. It would be wrong to consider .NET as a technical architecture in its own right, it will not guarantee well crafted, architect systems that are resilient to change. It therefore needs to be considered as part of a planned architecture and its' usage conveyed. Select Perspective recommends defining the Technical Architecture i.e. the artifacts and their usage, in the following way:

- Technical Components and Services - provide technically focused functionality such as: security, logging, error handling and auditing;
- Library classes - categorized as common ***facilities*** that provide small-scale, but frequently needed functionality or ***base*** classes that offer general incomplete functionality from which other classes will be derived;
- Patterns - to describe the use of: classes, components, and .NET related elements.

These artifacts are described and delivered by:

- Technical Model – using Select Component Architect, UML class diagrams are used to specify the classes and technical components in terms of their attributes, operation, associations and dependencies. Interaction diagrams capture the Patterns. It is recommended when using a Pattern Language to fully capture the detail it's detail – Models within Select Component Architect are customized to accommodate the pattern description, this is simply achieved by extending a set of model template properties;
- Technical Component and infrastructure classes can be generated from Select Component Factory C# Sync;
- Component catalogue – using Select Component Manager, the technical component specifications and .NET assemblies are published for use;

- Supporting documentation such as a user guide, that provides context and detail for the common components, base and facility classes and patterns of use;
- Class library (C#/VB);
- Education and mentoring to ensure that the consumers of the technical architecture know its capabilities and how to use them.

Creating Scaleable Solutions with Select Perspective and .NET

Components provide distinct, cohesive units of functionality, they must be designed to separate responsibilities and avoid unnecessary coupling. The most effective method of ensuring separation of concerns is to establish a series of architectural layers into which various types of component will be placed. Solutions can be considered to have up to six distinct layers of functionality:

- Client. In .NET terms, this is either a Web browser or a rich client that uses the Windows interface and Windows Forms to interact with the user.
- Presentation. This governs the logical flow and navigation between screens in the application. For a rich client, this will be part of the Windows Form application. For a thin client this functionality will be implemented using Web Forms under ASP.NET.
- Business Process. Business process components provide sequenced and possibly transacted flow through business processes or parts of use cases. In all cases, these will be implemented as “standard” components under .NET based around DLL assemblies (see “Components and Services in a .NET World”).
- Business Logic. These components encapsulate the business rules surrounding domain data concepts, such as orders or customers. Again, these will be implemented as “standard” .NET components.
- Data Access. This layer decouples the Business Logic from the underlying data sources. This functionality may reside in separate “standard” .NET components or as one or more classes that form part of a Business Logic component.
- Data sources. This layer typically consists of the application database, such as SQL Server, together with other data held on mainframes or other non-Windows platforms.

Within a solution, each component will find its natural home in one of these logical layers.

For deployment, each of these logical layers will be assigned to a physical tier. This assignment will create distribution boundaries across which remote access is required. Although details may differ for some applications, typically distribution boundaries will occur between the presentation and business process layers, and the business logic and data access layers, as shown in Figure 2. The technical architecture should be designed to enforce these architectural separations.

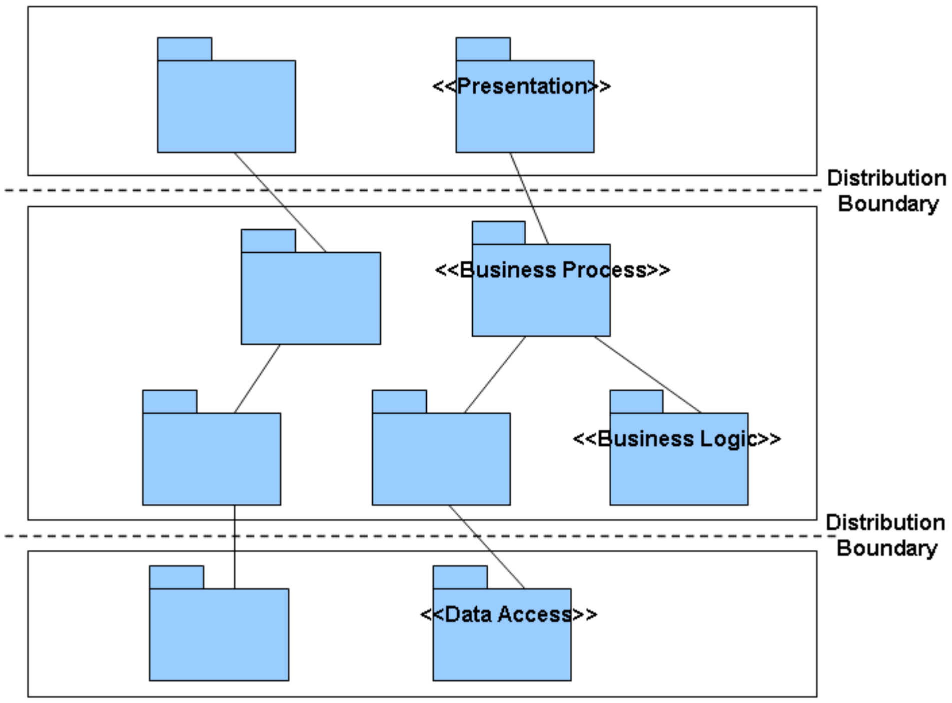


Figure 2 - Distribution boundaries in an enterprise application

Components and Services in a .NET World

Select Perspective targets the modeling and creation of components and services, whether those components and services are business-focused or technical in nature. The choice between implementing functionality as a service or as a component will depend on the long term strategy and infrastructure in which the solutions are implemented. However, .NET allows for the creation of both components and live services.

Components in .NET

Before .NET, Microsoft's component model was COM. However, COM (and even COM+) is now a legacy technology in .NET terms. The question then arises of what a component is in .NET terms. Put simply, .NET provides the constituent parts from which you can assemble components:

- Classes created in any .NET language can be instantiated and used by any .NET client. Hence, their methods can be used as part of the interface of a component. Similarly, .NET data types used as parameters can be used by any client;
- The interfaces defined on a component are realized as interfaces in any .NET language. Visual Basic .NET, C#, Managed C++ and J# all have interfaces as part of the language;
- Cohesive classes can reside in the same .NET DLL assembly. A DLL assembly acts as an external library that can be used from a .NET application or another .NET component;
- Service classes implement a component's interfaces. A service class is just a plain .NET class like any other, but service classes are the "public face" of the component and control the passage of data, messages and errors. In order to provide encapsulation for the internal component functionality, classes other than service classes can be marked as *internal* which will restrict their visibility to other classes within the same assembly. A single service class can implement all of the interfaces exposed by the component.

Therefore, in .NET terms a component is a set of classes contained in an assembly with one or more service classes exposing the component functionality to the outside world.

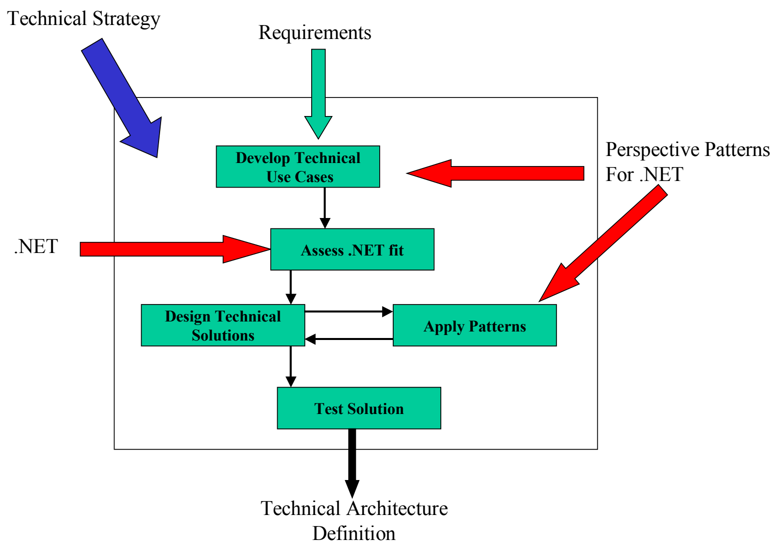
Services in .NET

.NET provides two distribution mechanisms for creating distributed components: .NET Remoting or ASP.NET Web Services. In either case, you create components as outlined previously and then build remoting façades. For .NET Remoting, this would be a remote object using HTTP or raw TCP as a transport, and XML or binary serialization for encoding. If you choose the combination of HTTP and XML, you can also treat this remote object as an RPC-style Web Service. Remoting servers, probably implemented as Windows services, must be running to handle requests for these remote objects. In the case of ASP.NET Web Services, the Web Service facades are dynamically invoked as required. In either case, the interface implemented by the façade can be an exact mapping of the service class. If the service class implements more than one interface, you may choose to expose these as separate remote interfaces.

A differentiation is made between the use of Web Service protocols and the overall concept of Web Services. To be a true, enterprise-wide service for any application, such services must be managed as part of the SMaC cycle so that they provide adaptable, scalable services for all their potential clients.

Technical Architecture Process

Having described the broad architectural basis for creating component-based solutions, we turn our


Technical Strategy
Requirements
Develop Technical Use Cases
Perspective Patterns For .NET
.NET
Assess .NET fit
Design Technical Solutions
Apply Patterns
Test Solution
Technical Architecture Definition

attention to defining the technical architecture and the process of delivery.

Figure 3 - The Select Perspective Technical Architecture Delivery Process

Develop Technical Use Cases

The technical architecture is designed to support the technical needs of the business solutions that are being delivered and will be delivered in the future. Any requirements are therefore derived from the requirements of the business solutions.

Non-functional requirements are catalogued within a SCA model, and their architectural impact described by a *technical use case*. These use cases are specialized from the 'normal' usage; stereotyping them as <<technical>> qualifies this. They describe an architectural interaction that will satisfy one or more requirements such as,*" application startup*", "user identification and access control" and so on. The use cases are clustered around a particular problem within an architectural layer or maybe across architectural boundaries, this will ensures that the technical architecture is focused on enforcing the layering and the principle of separation of concerns as illustrated in figure 2. The technical use cases also provide the means of prioritizing delivery of the technical architecture.

Assess .NET Fit

Based on the technical use cases and knowledge of existing patterns such as Select Perspective Patterns for .NET, the .NET framework is assessed to understand how it will be applied and what custom code (classes, components) will augment it.

Design Technical Solutions – Apply Patterns

From the technical use cases, interaction diagrams and class diagrams are drawn, as in Figures 4 and 5. These are expressed in terms of the stereotypical elements of a business solution rather than in terms of specific classes. Thus an interaction diagram may include classes representing generic UI forms or controls, a generic business component interface, and so on. Many of these generic elements will eventually be used to define the base classes. Some elements modeled in the interaction will be specific. These are the elements that will eventually form an implemented part of the technical architecture defined as service interfaces to the technical components and the facility classes, elements of the .NET framework will also appear on the interaction and class diagrams. Where existing patterns are known these may be used wholesale of customized as appropriate.

Test Solution

Because the real business solution will not be readily to hand, it is common practice to exercise the technical architecture by creating a *test harness.* Testing is based around the interactions required by the technical use case, and the model becomes the basis for the test planning and test case design. As with any project testing is a crucial exercise, this is especially true of the technical architecture, as it will be used by many components and solutions.

Using the Technical Architecture

Select Perspective considers a model-driven approach essential to the whole CBD lifecycle. The technical model is essential input to the detailed design conveyed by the business solution models; here are some examples of how the business solution designers can utilize the technical model:

- Patterns may be copied from technical architecture to business solution models and tailored as appropriate;
- Technical Components published into Select Component Manager can be imported into the business solution models
- Base and facility classes can be referenced from the business solution model – the technical architecture model acts as a library of classes.

From the model, we then move to implementation and the business solution developers will have recourse to various implementation artifacts that have been created alongside the technical architecture model.

Applying the Process

A sample set of typical architectural problems that will need to be addressed follows; a specific example will illustrate how the process can be applied in addressing one of these problems and defining the technical solution. For the purpose of this paper, assumptions on requirements and architectural decisions are mode. Of course, these will be specific, on a project-by-project basis.

Providing a User Interface

Many of the example applications provided for the .NET framework have Web-oriented interfaces. In this case, the user interface components will be implemented as ASP.NET pages. The ASP.NET pages and associated code-behind will use distribution proxies to access the components in the business tier. For example, in your ASP.NET project under Visual Studio .NET, you can add a Web Reference for a business Web Service. This generates a proxy class for that remote component on which the ASP.NET code can make method calls.

One thing that does need to be reflected in the modeling of both the solution and the technical architecture is the ASP.NET lifecycle. This is event-driven and there is no return path as control flows between pages. The management of client-specific state and the control of the flow through the pages are both very important. ASP.NET provides good state management capabilities through the HttpSession class. However, navigation through standard ASP.NET Web Form controls in Visual Studio .NET works on static URLs set at page creation time.

An example of the technical architecture definition follows: Requirement assumption - *Page navigation may be complex and pages should not refer to each other explicitly. It must also be possible to dynamically reconfigure the page sequence.*

A <<technical>> use case is created in the model; called ‘Manage Page Navigation’ this will be associated to the catalogued requirement, in Select Component Architect a requirement can be added to the dictionary and an association between it and the use case. To express the problem and design for the solution UML diagrams are created, in particular an object sequence diagram, this can be explicitly associated with the use case and so aid trace ability.

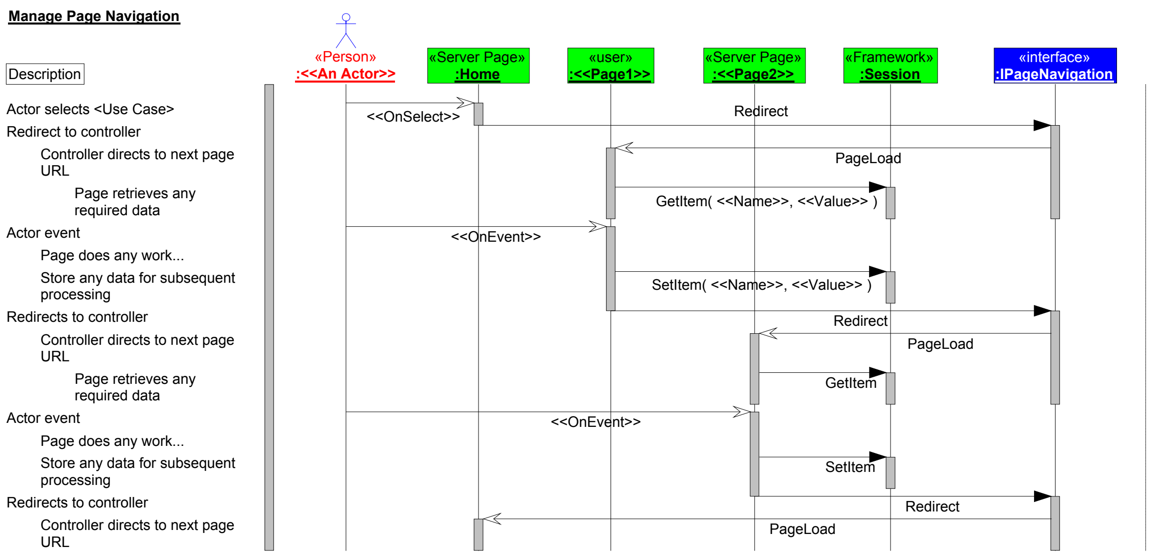


Figure 4 - Object Sequence Diagram for ASP.NET Page Navigation pattern

The architect will have available known design and architecture patterns [see Select Perspective Patterns] that may help design the technical solution; of course, the .NET framework will provide many facilities. The 'Mediator' pattern [3] is applied to the problem – it introduces the idea of a mediator that encapsulates how a set of objects interacts. The mediator promotes loose coupling by keeping the objects from referring to each other explicitly. The architect has to apply this general-purpose pattern to the specific problem at hand. It is decided that the mediator should enforce a standard protocol by which all pages will use, to that end a standard interface is introduced with a single service operation called Redirect (event, state). Of Course the actual decision making logic that determines the sequence of pages is specific to the business solution and its' use cases. So the implementation of the interface could be left to the specific business solution developers and the technical architecture provide only a base class and interface. However it has been decided that the component can be dynamically driven by data, this will address the requirement to dynamically reconfigure pages. As part of the technical architecture, a component 'PageNavigator' is, created that implements the interface.

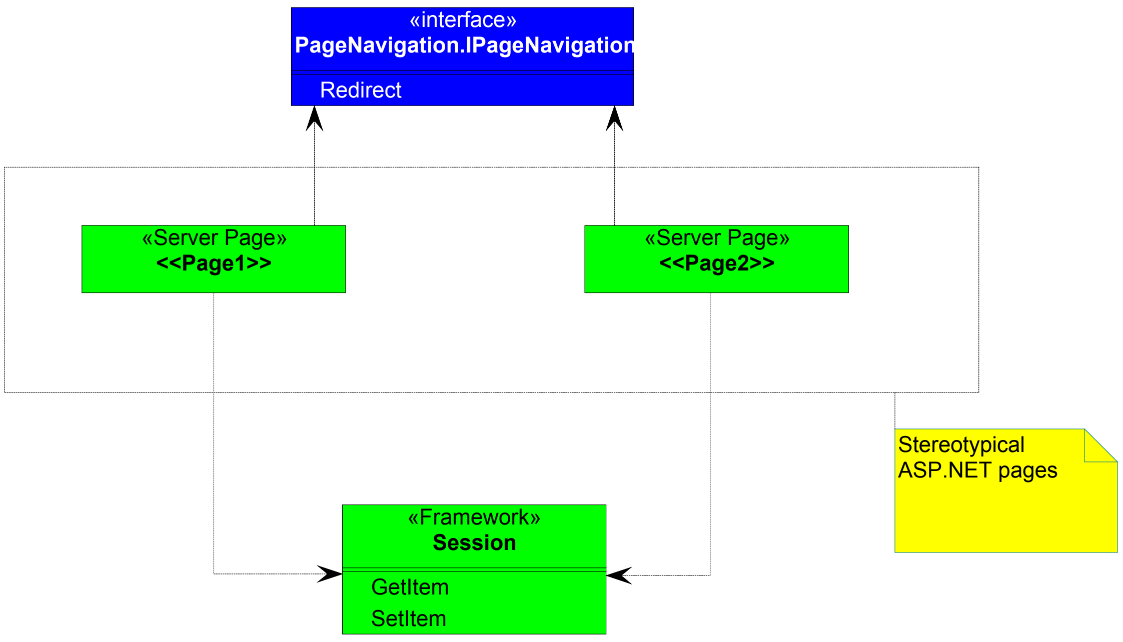


Figure 5 - Class diagram showing dependencies

Figures 4 and 5 show the final design, where stereotypical pages are introduced to show typical interactions.

Here is the succession of events by which a page triggers a redirect to another page, etc:

- On initialization of the PageNavigator a profile of state information is loaded in memory (navigation table);
- The ASP.NET page in response to a user gesture (such as a button click), will perform any required functionality, maybe calling a business service, stores any data required for further processing on the session object and then delegates to the Page Navigator by the Redirect (event) service operation;
- The PageNavigator, based on information supplied by the page will transfer the user to the next page in their current workflow. Navigation becomes flexible, since it is easily updated

through the navigation table, but simple, as the complexity is encapsulated within the PageNavigator.

Now we will look at the specific deliverables of the technical architecture:

- A pattern of usage that can be followed by the business solution designers to integrate their pages, showing how the Page Navigation component is used and also how the .NET session object is used;
- The interface specification to the Page navigation component;
- An implementation in VB.NET or C#.NET of the Page Navigation component;
- A code fragment for ASP.NET pages, with boilerplate functionality.

Providing Business Components

Business Process components and Business Logic components implemented in DLL assemblies will populate your Business Tier. Clients will probably access the Business Process components through Web Service interfaces. However, this is not the end of things. Frequently, the process flow through a Business Process component will be transactional in nature, requiring transaction support from the underlying platform. Currently, .NET does not have an explicit transactional component model such as those supplied by COM+ and EJB. However, using the capabilities defined in the System.EnterpriseServices you can still declaratively associate a transaction with a method on any of your .NET classes. These transactions are controlled by the Distributed Transaction Coordinator, and so will span multiple database connections on the same machine.

Another consideration for business-tier components is state management. The general .NET state model for the business tier is that all components are stateless. If client-specific data is to be retained, it should be persisted to an underlying data source through ADO.NET.

The architect will of course introduce technical use cases to reflect the requirements around transactional capabilities and state management and carry out the same process as described above.

Accessing the Data

Business Logic components work on underlying business data from persistent stores and legacy data sources. Access to this data will require specific knowledge of where and how it is stored. This knowledge, and the associated code, is irrelevant to the Business Logic component itself and so should be delegated to a separate Data Access layer as shown in Figure 2. Data Access Objects (DAO) are created that encapsulate the access to the business data, potentially merging data from multiple sources, to provide a single, consistent view on that data for the associated Business Logic components. This allows the Business Logic components to focus on the enforcement of business rules leading to a cleaner design and more flexibility and maintainability. As with the Navigator discussed previously, the DAO pattern can be provided as part of the technical architecture framework to guide the component designers in creating this layer.

Given that a solution will typically consist of the four layers shown in Figure 2, business data will be passed between these layers. The content and amount of data passed is governed by the functional requirements, but the format of the data should be optimized for the underlying platform. During Business Architecture, single instances or collections of domain data types, such as customer information, will be defined as inputs or outputs to service operations. Although you could continue this data format all the way down into the implementation, ADO.NET provides a more flexible format in the form of the DataSet.

Technical architecture patterns can be provided that define data passing signatures for Data Access Objects, Business Logic components, and Business Process components. These patterns guide the component designer to make the right choices for where and how to apply DataSets when creating the detailed component design from the solution model.

Select Perspective Patterns

At Select Business Solutions we have recognized and applied many recurring patterns across many projects, a selection of these are included as models, part of the standard Select Component Factory installation; they provide templates for a definition of a specific technical architecture. Models are:

- 'Select Perspective Patterns for IBM WebSphere'
- 'Select Perspective Patterns for. NET'.

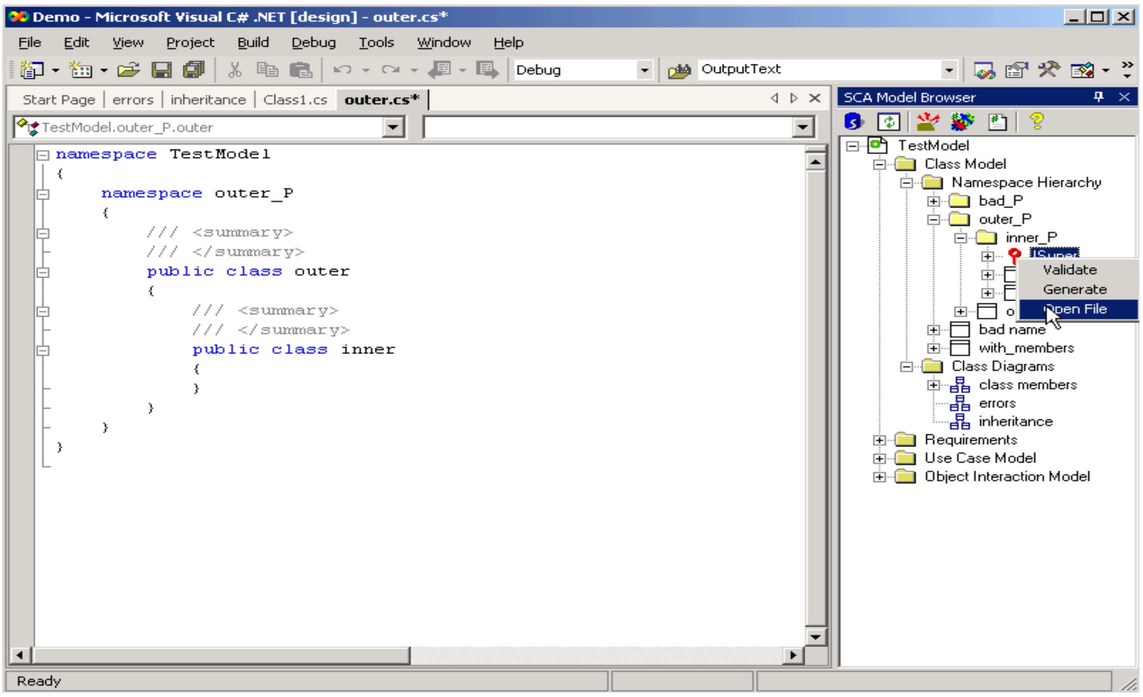
Select Component Factory for .NET

Select Component Factory provides lifecycle support from Business Process modeling through to deployment. Current and future .NET support includes:

- Browsing the SCA model dictionary, diagrams and relationships within the VisualStudio.NET IDE, Figure 6;
- Allowing users to generate .NET code (C#) per class/package from the integrated model browser into the current .NET project, having first validated the chosen item using Select Reviewer integration;
- Provide support for the import of .NET assemblies (components) into Component Manager;
- Provide models of the .NET Base Class Framework Libraries for easier modeling of .NET components;
- Provide for detailed information for .NET assemblies (e.g. security settings);
- Allow a user of VisualStudio.NET to reuse a component implementation into their current .NET project;
- Allow a user of VisualStudio.NET to publish a built component against a component specification in the SCM library;
- Support for the VB.NET.

In addition support will also be provide for Web Services, to include:

- Provide support for reversal of the definition of a Web Service as stored in its related WSDL;
- Provide support for generation of the definition of a Web Service as defined in WSDL;
- Provide support for the import of Web Service assemblies;
- Provide functionality so that SCM may be used as a UDDI server.



```
namespace TestModel
{
    namespace outer_P
    {
        /// <summary>
        /// </summary>
        public class outer
        {
            /// <summary>
            /// </summary>
            public class inner
            {
            }
        }
    }
}
```

Figure 6 Design through to Implementation – model integration into the .NET IDE

Conclusion

Developing component-based systems addresses many of the key challenges facing business and IT today. However to achieve success requires adherence to architectural principles; Select Perspective embodies those principles in its Business Architecture and Technical Architecture Delivery workflows. The technical architecture definition is applied in specific context to the technical platform, in this case Microsoft .NET. Select Component Factory provides essential tool support across the lifecycle with tight integration into the .NET development environment. Select Business Solutions provides process and tools to help deliver flexible, resilient .NET solutions – quickly. To find out more about Select Business Solutions and our support for Microsoft .NET visit our Web site [2].

References

[1] For detailed information about Microsoft .NET visit: http://www.microsoft.com/

[2] For all Select Business Solution's white papers and product datasheets, see http://www.selectbs.com/

[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J., "*Design Patterns: Elements of Reusable Object-Oriented Software"*, Addison Wesley, 1994