SOA-BASED COMPOSITE APPLICATIONS WITH ZERO CONSTRAINTS

Using Micro Focus OnWeb® & OnWeb® Connectors

Composite application developers are no longer constrained by object technologies, development methodologies, application types, or data sources.
Eliminating Constraints

Composite applications are applications that use functionalities from other applications. Since we have the technology to create composite applications today, we do not need to limit our definition in any way. This white paper is about composite applications – composite applications with zero constraints. (Necessary access control and security are not considered constraints in our definition, but rather as mandatory features.)

All of the large analyst firms today view the enabling technology of composite applications to be mature, and expectations are running high that composite application technology will speed development of the next crop of new corporate applications – in particular, ones that exploit service-oriented architecture (SOA) principles. For this to happen, developers that are tasked with creating these new composite applications need to be given as much latitude as possible when it comes to the technologies and methodologies at their disposal. They also need to be given latitude in terms of the source applications and data types they can mix and match. Without this flexibility, we really cannot make the claim that we have made any real progress over what was theoretically possible pre-Y2K with host integration and EAI.

We should not limit by definition composite applications as just having to do with core applications, programmatic application access, or XML Web services. The scope and potential of composite applications is considerably greater than that. When we eliminate constraints about how to actually realize composite applications, having unnecessary artificial limits just complicates matters, extends development cycles, and diminishes potential ROI.

At minimum, a composite application should be able to:

- Merge functionality easily from multiple, disparate applications, whether they are core (1970 COBOL) or contemporary (SAP R/3 or Siebel Sales 7.7)
- Manipulate data from a variety of databases and flat files
- Exploit all of today’s popular object technologies (Web services, EJBs, .NET assemblies) easily, inclusively if needed, and without restrictions
- Use both API-based smart connector methodology (JCA adapters) and screen scraping techniques in parallel to capture needed transactions and data with the right granularity
- Access in real time, multiple applications and data sources on different platforms (that might be geographically dispersed) without performance being limited by scalability criteria or throughput bottlenecks
- Impose stringent authentication and access controls to keep out unauthorized users
- Be developed using any of today’s major software development methodologies and IDEs
- Deliver end-user access in a variety of appropriate forms, whether via HTML thin-clients, portlets or applets
- Scale successfully to handle large populations of interactive end users with acceptable response times

The diagram below describes the key elements for creating a composite application with zero constraints. Micro Focus technology – specifically Micro Focus OnWeb and OnWeb Connectors – enables composite applications to be developed and deployed easily. This white paper describes the process for developing applications like this, potential issues, and pitfalls to avoid.
Three Principal Phases

Creating a composite application consists of three distinct (and mandatory) phases:

**Phase 1 – Object Sourcing**
Locating and obtaining as much of the needed business logic, business process representations and data from current applications and data sources.

**Phase 2 – Application Synthesis**
Creating the new composite application by combining and orchestrating the various objects obtained with each other – as well as with newly written code – to create the desired end result. (And when necessary, also interfacing additional data sources for manipulation by the now expanded code base.)

**Phase 3 – Application Deployment**
Ensuring that controlled and secure access is available in an on-demand, runtime basis between the new composite application and all of the back-end applications and data sources that it will call upon.

---

**Figure 1. The key elements for creating a contemporary composite application. Developers need to have appropriate freedom with each of the “key elements,” so they can deliver the best possible results quickly and efficiently.**
While a sequential progression exists among these three phases, in practice a considerable amount of repetitive iterations, back and forth among the phases will follow as developers refine and validate the emerging composite application. Rather than an issue, this is actually more of a strength in SOA-based composite application creation.

As expected, each of the three phases has its own set of issues, demands and disciplines. We will discuss these in detail later after we have covered the underlying principles of the three phases.

The first principle we’ll examine is the inherent cause-and-effect interdependency between the first and third phases. The coupling between these two phases is considerably tighter than most developers initially realize when they embark on building a SOA-based solution. In reality, exploiting this tight interdependency is at the crux of developing effective composite applications – especially with zero constraints.

This interdependency comes about because SOA-based composite applications rely on a runtime execution model when accessing functionality from other applications. This fact dictates that the composite application cannot run much of the time on its own. It has to run in conjunction (in real time) with the other applications it is relying on for functionality and data. In practice, this gets even more complicated since there will invariably be intermediary pieces of code (like Web services) to facilitate the interactions between the composite application and other applications. (See Figure 2.) With composite applications, you deal with many autonomous software processes that are running in parallel, most likely on different platforms, and which may be geographically dispersed.

![Figure 2. A simplified schematic of the various computing systems that can be part of a SOA-based composite application. The runtime execution model inherent with SOA means that a composite application cannot be run in isolation. Micro Focus OnWeb server or OnWeb Connectors working in concert with composite application servers centralize and simplify the deployment and execution of composite applications.](image-url)
Runtime Execution Model

The whole notion of a service-oriented architecture is based on the runtime execution of the application functionality being reused. This is what we mean by referring to current applications as services that deliver reusable functionality to newer applications (that is, composite applications). With SOA, you do not borrow functionality by cutting and pasting code segments from current applications into the new one. Also, you make no attempt to convert existing application code (or even application subroutines) from their original core programming languages (COBOL) to more strategic, platform-independent variants like Java.

During the era of Y2K conversions, we learned that trying to reuse application functionality (or to recreate business processes) at the source-code level is often not possible or practical. This is primarily because of the uncertainties as to the reliability and integrity over time of the source-code maintenance and control. In addition, if the application functionality you need is in source code from 3rd-party applications (as is invariably the case today), you can face intellectual property, copyright and royalty issues. Consequently, we see the growing popularity today of reusing application functionality in the form of runtime services, which is what SOA is all about.

With a service-oriented architecture approach, application functionality reuse is achieved through a standard function-call (or procedure/subroutine-call) model. The pivotal difference is that the software functions being invoked will not be part of the calling application. In most cases, the functions being invoked will not even be running on the same platform or even in the same data center. This is why XML Web services are often referred to as a platform- and language-independent form of remote procedure calls (RPCs).

With the runtime execution model inherent in SOA-based composite applications, you can quickly begin to see some of the issues that need to be addressed, including access control, overall connectivity, transaction orchestration, and others. It is clear why there must be a tight coupling between the object sourcing phase and the composite application deployment phase, as we mentioned in the previous section. You cannot just source objects for a composite application on a standalone, one-time basis. You have to make sure that the functionality you need – represented by the sourced objects – will be available in real time and when the new composite application is running. In practice, developers also need access to the source applications while they are testing and validating the new composite application.

The interdependencies inherent in a SOA runtime execution model dictate that you cannot create composite applications without careful planning about how the objects are going to be used. And moreover, how and where the objects are going to be sourced. Consequently, you need to have a structured and systematic approach. Ideally, an approach based on a proven composite application solution working in concert with the Micro Focus OnWeb server or OnWeb Connectors that supports:

- Interactive object sourcing (with a visual development environment) from multiple, disparate sources
- Automated application connectivity with necessary authentication and access control
- Application functionality access using Web services, Java objects (including JCA), .NET assemblies or XML
- Macros, scripts and event handling to coordinate object behavior
- Offline testing using previously captured data streams
Snapshot of What is Involved

So far, we have laid out the case for a runtime execution model for SOA-based composite applications. You should have a better idea of what is involved in creating contemporary composite applications, particularly those that impose zero constraints on application developers and administrators.

Figure 3. The pivotal role that a Micro Focus OnWeb server or OnWeb Connectors – in concert with a composite application server – play during execution of a composite application. Typically, a full-spectrum composite application server – like Micro Focus OnWeb – will be used with a studio-type IDE during object sourcing to capture needed application functionality and represent it as an object (EJB) or Web service.
<table>
<thead>
<tr>
<th>Major Requirements</th>
<th>Object Sourcing</th>
<th>Application Synthesis</th>
<th>Application Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reusable functionality independent of source</td>
<td>Language, platform and technology latitude</td>
<td>Platform flexibility</td>
</tr>
<tr>
<td></td>
<td>Flexibility in terms of object technologies</td>
<td>Ability to mix object technologies</td>
<td>Overall connectivity</td>
</tr>
<tr>
<td></td>
<td>Continued availability of the source applications</td>
<td>Interactive test tools</td>
<td>Scalability</td>
</tr>
<tr>
<td></td>
<td>Ability to “customize” the functionality being extracted</td>
<td>Support for all major standards</td>
<td>Administration and manageability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simplified GUI creation</td>
<td>Platform stability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Security infrastructure (firewalls)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Main Concerns</th>
<th>Object Sourcing</th>
<th>Application Synthesis</th>
<th>Application Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Achieving desired granularity</td>
<td>Delivering expected results</td>
<td>Authentication, access control security</td>
</tr>
<tr>
<td></td>
<td>Ability to mix object technologies</td>
<td>Meeting schedules</td>
<td>Real-time availability of the source applications</td>
</tr>
<tr>
<td></td>
<td>Two-way authentication, access control, security</td>
<td>Ensuring reliability and resilience</td>
<td>Meeting performance expectations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relevant Methodologies</th>
<th>Object Sourcing</th>
<th>Application Synthesis</th>
<th>Application Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Smart OnWeb Connectors</td>
<td>Application modeling</td>
<td>Application servers</td>
</tr>
<tr>
<td></td>
<td>Screen scraping</td>
<td>Scripting</td>
<td>Scripting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transaction orchestration (BPEL4WS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Studio-type graphical IDEs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applicable Technologies</th>
<th>Object Sourcing</th>
<th>Application Synthesis</th>
<th>Application Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Web services, WSDL, UDDI</td>
<td>Java, .NET, C# etc.</td>
<td>HTML</td>
</tr>
<tr>
<td></td>
<td>Java: EJB, JCA, Java Beans</td>
<td>Object technologies</td>
<td>SOAP</td>
</tr>
<tr>
<td></td>
<td>.NET assemblies</td>
<td>XML</td>
<td>HTTP</td>
</tr>
<tr>
<td></td>
<td>XML</td>
<td></td>
<td>SSL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>JSR 168</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Combined role of an Micro Focus OnWeb server/OnWeb Connectors plus a composite app server</th>
<th>Application access and navigation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Object creation</td>
<td>Authentication, access control and security</td>
</tr>
<tr>
<td></td>
<td>Object customization</td>
<td>End user delivery facilitation</td>
</tr>
<tr>
<td></td>
<td>Object registration</td>
<td>Central management</td>
</tr>
<tr>
<td></td>
<td>Presentation layer support and tools</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test and debug tools</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Technology ‘bridging’</td>
<td></td>
</tr>
</tbody>
</table>
Object Sourcing

Composite applications are all about reusing existing application functionality and minimizing software development and validation costs. So developers need to be able to get the application functionality they need in reusable object form. The first thing that you need to decide is how far you are willing to go to get the application functionality you need. We can categorize available options as follows (with each category additive to the preceding ones):

1. **Intranet only**: Application functionality sourcing restricted to applications available on the corporate intranet (behind the corporate firewall).

2. **Extranet extension**: Application functionality may be sourced from selected, authorized applications available across a secure extranet from preferred business partners.

3. **VPN model**: Variation of the extranet scheme above, but access to required applications being run by preferred business partners is across the Internet (using secure protocols).

4. **Internet/Web services**: Application functionality acquired from a previously unknown, but since validated, service provider across the Internet according to the original premise for Web services.

All of today’s popular object technologies can be used in all four scenarios, independent of the underlying network infrastructure. Nevertheless, which of the categories you need to access to get all the application functionality you want can significantly influence your object sourcing choices. As critical as security is, it is only one of the factors you have to consider. For starters, you may not need to use Web services if object sourcing is going to be restricted to an intranet or extranet. Instead, you might find it more efficient to use EJBs or .NET assemblies.

**Flexibility to Go Beyond Web Services**

SOA solutions do not always have to be based entirely on Web services. Web services just happens to be the newest (and most talked about) enabling technology for SOA. Just because .NET assemblies, EJBs and for that matter CORBA do not have the word services in their names, does not mean they do not also fully support the creation of SOA-based solutions. This distinction is important because many cases can exist where you may want to (and should) use EJBs or .NET assemblies to create a Web service.

There is nothing mystical about Web services. From its inception, Web services has always been about self-contained, software modules with standards-based (XML-defined) inputs and outputs. SOAP provides a way to remotely invoke these software modules, while WSDL is an XML-based text file that clearly defines what that Web service intends to do by specifying the format of the inputs and outputs. Note that all of the standards-related aspects of Web services pertain to the I/O definitions (via XML, WSDL), the preferred means for invoking a Web service (SOAP), and the optional service advertising mechanism (UDDI). (See Figure 4.)

There are no standards or even conventions for how the actual body of a Web service (the business logic) should be implemented. But this is intentional. It is this flexibility compared with the actual implementation that makes Web services so attractive and powerful. You can create a Web service using any programming methodology – even assembler, COBOL or FORTRAN. All that is mandatory is that its I/O requirements are XML-based and that it can be remotely invoked using a SOAP-like mechanism. In practice, with the current emphasis on object-oriented software development, most Web services are created using .NET assemblies or EJBs.

Fig. 4: Demystifying XML Web services. The essential architecture of a Web service showing the role played by the various standards (XML, WSDL, SOAP and UDDI) and how the body of the Web service can indeed consist of an EJB or .NET assembly.

If a Web service consists of a .NET assembly or EJB at its core, you have to question what advantages you can derive from using the Web service as opposed to directly accessing the core object. The answer to this is very simple. The Web service provides an XML-defined I/O mechanism that can be invoked using SOAP, whereas the native object scheme will use a less-rigorously defined I/O scheme. The issue boils down to the advantages of having XML-defined I/O. Obviously, having an XML-defined I/O is extremely desirable when you are trying to source functionality from a 3rd party (a service-provider).
This should put the potential role of Web services and composite applications into perspective. Going back to the four “topology” categories listed at the top of page 11, it should now be clear that Web services is most relevant if you have to source application functionality over the Internet from the previously 3rd-party service provider. On the other hand, if your composite applications are all going to be based on application functionality culled from applications already available on the corporate intranet or on an extranet with a preferred partner, you should keep the option open of being able to use EJBs or .NET assemblies in addition to Web services. Forgoing that option is an unnecessary and potentially costly constraint. Micro Focus OnWeb server and adapter technology includes full support for Web services, .NET assemblies and EJBs, so developers have complete flexibility when it comes to object sourcing.

Options for Obtaining the Necessary Objects

In reality, there are only three distinct ways you can obtain application functionality (in object form) for inclusion within a SOA-based composite application:

1. **In-house creation**: Capture the necessary application functionality from current in-house applications and represent as objects or Web services using proven, graphical tools like Micro Focus OnWeb Object Builder.

2. **In-house reuse**: Locate relevant objects that might have been created for previous projects.

3. **External sourcing**: Obtain the required application functionality – typically in the Extranet/VPN, form of a Web service – from a business partner or a service provider.

The key issues inherent with using external objects are relatively obvious and have been articulated on many public forums in an effort to explain the slower than predicted popularity of XML Web services. Given a runtime execution model, the main concerns include:

- The whole gamut of “security-related” issues, spanning rogue services (that violate data privacy), services that have surreptitiously “hi-jacked,” and data transmission integrity
- Reliability, responsiveness and scalability of the service so it does not compromise the performance of the composite application
- Actual cost of the service and usage monitoring, particularly if pricing is based on a usage model

➤ Change control enforcement (and monitoring) of the remote service, since any unexpected changes made to the service could adversely impact the composite application.

One way to get around these concerns with external objects is to try and “acquire” the rights to the objects so that they can be installed and executed behind the corporate firewall. In other words, stick to the intranet model even when the required functionality was originally not available within the corporation. In many instances, the approach of trying to bring in the functionality so it is behind the firewall can prove to be impractical.

The main stumbling block is that the required functionality may not be in the form of a relatively compact, standalone piece of software. More than likely, it is part of a bigger application. If the functionality is part of a bigger application, acquiring a Web service that remotely invokes that functionality and running the Web service behind the firewall does not solve the problem. The only way to truly solve the problem is to also bring the source application behind the firewall. And that can prove to be costly and time consuming.

This highlights a very fundamental notion that is easy to lose sight of when talking about Web services in the context of composite applications. In many cases, a Web service may not itself contain any business logic! Instead, it could just be an XML-based calling mechanism to an application that contains the required functionality. This is alluded to in Figures 2 and 3 and is also highlighted in Figure 5. The notion of using a Web service (or other object) as a calling mechanism is pivotal to successfully achieving composite applications. The principles involved in this are discussed in detail in the following section, particularly since this approach is the primary means of extracting application functionality from existing, in-house applications.

**The Role of a True Composite Application Server Combined with a Micro Focus OnWeb Server or OnWeb Connectors**

Another option for mitigating the concerns related to using external objects is to coordinate and channel all the interactions with the external objects through a powerful, purpose-built composite application server – combined with a Micro Focus OnWeb server or OnWeb Connectors. This would specialize in providing security, administration, monitoring and management functions, leveraging proven industry-standards à la LDAP and Kerberos. A composite application server can definitely help allay the concerns and potential dangers of using external application functionality by enforcing mechanisms, such as stringent two-way authentication and remote service monitoring. However, security management is not the only reason for wanting a composite application server when one is dealing with SOA-based composite applications.
A true composite application server is very different from an application server (BEA WebLogic Server), an integration server (TIBCO BusinessWorks), a host integration/host publishing server (Attachmate Synapta), or even the so-called “proxy servers” used by some of the old-school EAI adapter vendors. A composite application server – with a Micro Focus OnWeb server or OnWeb Connectors – provides developers, systems administrators and system operators with specialized, incisive functions and features to facilitate development, deployment and execution of composite applications. In particular, a true composite application server – with a Micro Focus OnWeb server or OnWeb Connectors – should provide all the necessary functionality to automatically and seamlessly handle the interdependencies that exist between the object sourcing and application deployment/execution phases that were discussed earlier. (See The Three Principal Phases section on page 4.)

If automated correlation is not available, developers will have to manually set up (and continually update and maintain) all the necessary links, file libraries and access control mechanisms for each application, reflecting the requirements of the objects embedded within the applications. Obviously, this could be a complex, cumbersome, time-consuming and highly error-prone process. Using objects created with screen scraping can make these problems worse, particularly when objects invoke transactions with complex navigational paths within an application because of I/O fields from multiple screens.

Micro Focus has a long history of providing secure, automated application access for host publishing and host integration solutions. With its composite application server technology, Micro Focus goes to great lengths to complement and augment composite server technology to provide state-of-the-art automation to eliminate the need for manual configurations and definitions. This greatly simplifies the entire development-deployment process, expedites application availability, and eliminates object correlation-related errors. Furthermore, Micro Focus understands how automation techniques generally help developers to be more productive (as well as creative) and so provides extensive scripting mechanisms as a part of its Micro Focus OnWeb server technology.

Developers have the option of using powerful scripting during the object-creation phase, whether using programmatic (EAI adapters) or screen scraping methodologies. They know that all of the required correlation will be done for them automatically when they reach the deployment phase. This automation makes a huge difference when it comes to developing composite applications with zero constraints. The end result is that bona fide composite application server technology – with a Micro Focus OnWeb server or OnWeb Connectors – is what makes composite applications with zero constraints a reality today. If developers are deprived of a true composite application server and the capabilities of a Micro Focus OnWeb server or OnWeb Connectors, they will encounter too many impediments and will be forced to develop composite applications that are far from optimum.

Object Creation Mechanisms

Objects that will be invoked by a composite application have to be created (and tested) before the application is deployed for production use. Since the rationale for developing composite applications hinges on reuse of existing software functionality, many of the new objects need to be created to invoke processes running within other applications. There are three ways to capture processes from existing applications:

1. Programmatic schemes typically involving the use of application-specific adapters (like Micro Focus OnWeb Connectors)

2. Screen scraping because there are not any appropriate adapters for the applications; the object developer does not have experience with this approach; or the object developer intends to exploit the capability to “combine, filter and skip” I/O fields with this approach.

3. Extracting the relevant source code segments, if one is confident that the application source code has been diligently maintained and is up-to-date Objects (including Web services) that will be used in composite applications do not have to be (and in most cases will not be) self-contained units of software functionality. While it is possible to have totally self-contained objects, these objects invariably fall into the “utility function” category (like creating a new display window) as opposed to objects that perform a complete business process. There is a very good reason for this. There is nothing magical about objects. Even Web services. They are simply reusable pieces of software that confirm to certain standards.
Why the Service Invocation Scheme of SOA is the Only Real Option

If you include creation of brand new objects for subsequent reuse, there are still only three ways to create objects for composite applications. Only two are capable of generating true self-contained objects. The three options can be summarized as follows:

1. **Code**: Develop, test and validate the needed functionality from scratch using standard software development methodology.

2. **Cut-and-Paste**: Synthesize an object using source code borrowed from existing applications.

3. **Call**: Make “calls” from within the object to one or more applications to get the needed functionality (with the option to create the objects using either programmatic access or screen scraping as discussed earlier).

The cut-and-paste approach is not often practical for the creation of a single object or a whole new application. The issues and costs related to software development, testing and validating end up being the same, whether you are talking about an object or an entire application. The only difference is that an object requires less effort and resources when it comes to testing and validation.

This is the reason why object creation, as well as composite application development, both end up gravitating towards the service-calling model – particularly when Web services are used. And this is the reason for the growing interest in SOA-based solutions. Figure 5 depicts the primary function-invoking call mechanisms that can be used by a composite application.

EJBs and .NET assemblies can be embedded within a composite application (see Figure 5), while a Web service by definition is an external routine that has to be invoked using its XML-specified I/O parameters. The invocation of a Web service can be achieved directly or through an object. Also, Web services may make use of objects to obtain software functionality or make the relevant calls to source applications. In some instances, especially when dealing with core applications (using screen scraping methodology), an object or a Web service may end up first calling a host integration “process” that will assist in performing needed host access and application navigation functions.

For composite applications to truly excel, developers need to have the freedom to use any valid function invocation mechanism – without any artificial constraints about the levels of nesting or abstraction. There is nothing wrong with obtaining functionality in the form of services via a nested calling mechanism. In practice, it is a proven and extremely sound computer science practice going back to the 1970s. The only difference is that there is no longer a need for all the called functions to co-reside alongside the calling application on the same machine.
Transactions as Representations of Business Processes

The capability to abstract application functionalities (using objects) is what makes SOA-based composite applications viable and attractive. Without this abstraction (using calling objects), it is often impossible to get the functionality you need from existing applications. It is always important to remember that the existing application functionality you are trying to get for reuse may never have been developed in a form that is conducive or even accessible for reuse. In many cases, the needed functionality will not even be implemented by a contiguous “block” of software within its parent application. Instead, there could be significant branching and linking within the parent application to deliver the functionality. This is invariably the case with older applications. The situation is often exacerbated with “on-the-fly” fixes applied to the applications in the form of object-code patches involving branching off to a new “patch routine.”

In the context of SOAs, application functionality (for example, that is associated with a particular business process) is invariably identified and extracted in terms of clearly defined and demarcated transactions. This is prudent and pragmatic. End users, line-of-business management, and programmers can relate to specific transactions performed by an application. Thus, there is no ambiguity when attempting to describe business processes (and the software functionality that performs them) in terms of transactions.

A transaction performs a predefined (and describable) process, has specific input/output characteristics, and you can typically determine whether it completed its designated task successfully or not. Consequently, transactions (for example, extracting a customer record, updating quantity in stock for an item) are the basic and the smallest units-of-work when it comes to SOA-based composite applications.

Hence the granularity of the functionality that can be extracted from a source application will be governed by the nature of the transactions it supports. Adapter-based programmatic access or screen scraping can be used to capture a transaction and represent it in the form of an object or Web service.

However, it is important to understand that an object or Web service representing a transaction also has to contain all the necessary application access, user authentication and transaction location (navigation) data, in addition to the I/O fields used by the transaction. When you are dealing with host applications, Micro Focus screen scraping technology, optimized and perfected over many years, will generally permit developers to capture “complex transactions” involving I/O fields from multiple screens – even spanning multiple, disparate applications.

With Micro Focus solutions, developers can also easily select which fields they want to include as a part of the transaction and which fields they want to omit from all the screens they are working with. (With no restrictions when it comes to what developers refer to as “add, skip and delete.”) Complex transactions can involve extremely complicated and convoluted application access, authentication and I/O navigation linkages. However, as we discussed in the previous section, Micro Focus OnWeb technology will automate tracking and maintenance of all required linkages. And moreover, do it between the object creation and application execution phases. (Automation like this is contingent on enterprises using the Micro Focus OnWeb server technology not just for application deployment, but also when the objects are being created.)

In summary, SOA-based solutions like the ones described on Page 6 (see Three Principal Phases section) always have a tight coupling between the object sourcing phase and the application execution phase. The technology and methodology being offered by Micro Focus in the form of Micro Focus OnWeb server and OnWeb Connectors have been developed from ground up to fully appreciate this inescapable coupling and to provide all the necessary automation to greatly simplify and speed object creation, application synthesis and the subsequent application deployment. Consequently, using a true composite application server – along with a Micro Focus OnWeb server or OnWeb Connectors – is pivotal to the whole notion of developing and deploying composite applications – with zero constraints.

Fig. 6: Using Micro Focus OnWeb server technology to create an object or Web service representing a “complex transaction” involving I/O fields from multiple host application screens.
Summary

Composite applications make use of functionality from other applications. Trying to get the needed software functionality at the source-code level is invariably impractical because of issues having to do with source-code maintenance and intellectual property rights. So SOA-centric composite application methodology is contingent on using a runtime invocation model to get needed functionality from other applications. You can achieve this using objects or Web services, with Web services having an edge only if the needed functionality is received across the Internet from a third-party. With intranet scenarios, using EJBs or .NET assemblies will typically be easier and more efficient than using a Web services form of the function.

Transactions tend to be the optimum unit-of-work when it comes to capturing existing application functionality for reuse within contemporary composite applications. Programmatic access (with EAI adapters, for example) or screen scraping can be used to capture transactions and create reusable objects or Web services. However, these objects and Web services must contain all the needed application access, user authorization, and transaction location “linkages,” in addition to the logic required to invoke the transaction with the appropriate I/O fields. Using a purpose-built, bona fide composite application server in conjunction with a Micro Focus OnWeb server or OnWeb Connectors greatly simplifies object creation and application deployment. In addition, it gives developers the option of creating “complex transactions” using powerful (and highly proven) screen scraping techniques with “add, skip and delete” capability. A true composite application server – with a Micro Focus OnWeb server or OnWeb Connectors – offers specialized functionality that is not available from conventional application servers.

At a minimum, a good composite application server – in concert with a Micro Focus OnWeb server or OnWeb Connectors – provides comprehensive and proven functionality for:

- automated application connectivity,
- stringent access control and user authentication,
- invoking “complex transactions” involving I/O fields from multiple screens,
- transaction coordination,
- process orchestration,
- process automation (scripting), and
- support of multiple object technologies.

In practice, you should use a good composite application server from the beginning. This is the secret of developing and deploying composite applications that truly meet “zero constraints” criteria. Without a good composite application server combined with Micro Focus OnWeb or OnWeb Connectors, developers will always be at a disadvantage and will be unable to deliver the best solution, in the shortest possible time. Attempting to develop SOA-based composite applications without using a true composite application server – along with a Micro Focus OnWeb server or OnWeb Connectors – will certainly prove to be a false economy. A composite application server – combined with a Micro Focus OnWeb server or OnWeb Connectors – is an intelligent investment that will certainly generate a positive ROI in a relatively short time.

About Micro Focus

Micro Focus, a member of the FTSE 250, provides innovative software that allows companies to dramatically improve the business value of their enterprise applications. Micro Focus Enterprise Application Modernization and Management software enables customers’ business applications to respond rapidly to market changes and embrace modern architectures with reduced cost and risk.