Composite Application developers should not be constrained by the object technologies, development methodologies, applications types, or data sources that they can exploit. They should be able to gainfully mix-and-match, ‘any-to-any’, to achieve their desired goals.

A Composite Application White Paper
for NetManage by Anura Gurugé of IT In-Depth
**Composite Applications**
**With Zero Constraints**

by Anura Guruge, IT In-Depth
for NetManage

**Published**
February 2005

**About NetManage**
NetManage, Inc. (NASDAQ: NETM) specializes in leveraging the host systems that are core business assets to organizations worldwide. The NetManage Host Services Platform provides a full range of solutions that line-of-business and IT managers need when designing and implementing host-based business initiatives. By providing unprecedented flexibility and speed of deployment, the Host Services Platform lets customers easily maximize their host systems and leverage them with a service-oriented architecture across a wide variety of computing platforms. By extending mainframe data and business logic, NetManage delivers new efficiencies and higher returns for its customers in a Web-based business world.

For more information on NetManage, visit www.netmanage.com.

**About the Author**
Mr. Anura Gurugé, the Editor at Large for “IT In-Depth” [www.itindepth.com] is the author of “Web Services: Theory and Practice” (2004), “Corporate Portals Empowered with XML and Web Services” (2002) and three other books including “SNA: Theory and Practice” (1984). He is an ex-IBMer who worked on the IBM 3274, and was “Mr. SNA” for nearly two decades. He was also responsible for developing the IBM 2217 MultiProtocol Converter. He is an independent technology analyst, an educator and freelance writer.

For more information on the author visit www.guruge.com.
## Table of Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminating Constraints</td>
<td>4</td>
</tr>
<tr>
<td>The 3 Principal Phases</td>
<td>6</td>
</tr>
<tr>
<td>The Run-Time Execution Model</td>
<td>8</td>
</tr>
<tr>
<td>Snapshot Of What is Involved</td>
<td>10</td>
</tr>
<tr>
<td>Object Sourcing</td>
<td>11</td>
</tr>
<tr>
<td>Object Creation Mechanisms</td>
<td>15</td>
</tr>
<tr>
<td>The Bottom Line</td>
<td>20</td>
</tr>
</tbody>
</table>
Eliminating Constraints

Composite applications make use of functionality from other applications. *Given the technology that is already available, it is not necessary to add any caveats to limit the scope of this definition.* This White Paper is about composite applications that conform to this definition; i.e. composite applications with zero constraints. Access control and security required by these applications are not, however, considered to be constraints, but rather mandatory features.

With the enabling technology now deemed to be mature by all of the large analyst firms, expectations are running high that composite applications will expedite the development of the next crop of bespoke corporate applications – in particular those that exploit service-oriented architecture (SOA) principles. In order for this to happen, the developers tasked with creating such applications need to be given as much latitude as possible when it comes to the technologies and methodologies at their disposal, as well as the ‘source’ applications and data types that they can mix-and-match. Without such flexibility, we cannot claim to have made any real progress over what was theoretically possible even pre-Y2K with host integration and EAI.

One should not pigeon-hole composite applications as just pertaining to legacy applications, programmatic application access, or XML Web services. Their scope and potential is considerably greater than that – when one eliminates the constraints as to how a composite application may be realized. Having unnecessary, artificial limits just complicates matters, extends the development cycle and diminishes the potential ROI.

A composite application, at a minimum, should thus be able to:

- Amalgamate functionality, at will, from multiple, disparate applications, whether they be legacy [e.g. 1970 COBOL] or contemporary [e.g. SAP R/3 or Siebel Sales 7.7].
- Manipulate data from a variety of databases and flat files.
- Easily exploit all of today’s popular object technologies, i.e. Web services, EJBs, .NET assemblies, inclusively if need be, without restrictions.
- Use both API-based smart connector methodology [e.g. JCA adapters] and ‘screen-scraping’ techniques, *in parallel*, to capture the requisite transactions and data at the appropriate granularity.
- Access, in real-time, multiple applications and data sources, on different platforms, that are, furthermore, geographically dispersed, without its 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 it be via HTML, ‘thin-clients’, portlets or applets.
Successfully scale to handle large populations of interactive end-users with acceptable response times.

Figure 1 summarizes the key elements involved in creating a composite application with zero constrains. NetManage’s technology enables such applications to be developed and deployed, easily and with panache. The remainder of this White Paper describes the process for developing such applications, the issues that you should consider and the potential pitfalls that you should try to avoid.

**FIG. 1: THE KEY ELEMENTS INVOLVED IN CREATING A CONTEMPORARY COMPOSITE APPLICATION.**

Developers need to be given appropriate latitude in each of these elements (or ‘sectors’) so that they can deliver the best possible results, quickly and efficiently.
The 3 Principal Phases

There are three very distinct (and mandatory) major phases involved in creating a composite application:

1. **OBJECT SOURCING:** Locating and obtaining as much of the requisite business logic, business processes representations and data from existing applications and data sources.

2. **APPLICATION SYNTHESIS:** Creating the subject composite application by combining and orchestrating the various objects sourced in the previous phase with each other, as well as with newly created code to obtain the desired end result -- and where necessary also interfacing additional data sources for manipulation by the now expanded code base.

3. **APPLICATION DEPLOYMENT:** Ensuring that controlled and secure access is available, on an on-demand, run-time basis, between the subject composite application and all of the back-end applications and data sources that it will call upon.

While there is a natural sequential progression between these three phases, in practice there will be considerable amount of repetitive iterations to-and-fro among the phases as developers refine and validate the emerging composite application. This is not an issue; more of a hallmark and a strength of SOA-based composite application creation.

Each of these phases, as is to be expected, has its own set of issues, demands and disciplines. These will be discussed, in detail, starting on page 10 of this White Paper once the underlying principles involved with this 3 phase process have been established. In terms of these principles, the first one that needs to be addressed is the inherent, ‘cause-and-effect’ interdependency between the first and third phases. The coupling between these two phases is considerably tighter than most initially realize when embarking on SOA-based solutions. In reality, appreciating and exploiting this interdependency is the crux to developing effective composite applications; especially those with zero-constraints.

The inescapable interdependency between the first and third phases comes about because SOA-based composite applications rely on using a **run-time execution model** when it comes to invoking functionality from other applications. This 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 it gets even more convoluted than this since there will invariably be **intermediary pieces of code**, e.g. Web services, to facilitate the interactions between the composite application and the other applications. See Figure 2. The bottom line is that when it comes to composite applications, one has to deal with many different and autonomous
software processes that are running in parallel, most likely on different platforms, which, moreover, maybe, geographically dispersed.

**FIG. 2: SIMPLIFIED SCHEMATIC OF THE VARIOUS COMPUTING SYSTEMS THAT CAN BE INVOLVED IN A SOA-BASED COMPOSITE APPLICATION.**

The run-time execution model inherent with SOAs means that a composite application cannot be run in isolation. Composite Application Servers centralize and simplify the deployment and execution of composite applications.
The Run-Time Execution Model
The whole notion of SOA is contingent on the run-time invocation and execution of the application functionality being reused. This is what is meant by treating existing applications as services that deliver reusable functionality to newer applications; i.e. the composite applications. With SOA one does not borrow functionality by ‘cutting-and-pasting’ code segments from other applications into the new one. There is also no attempt made to convert existing application code (or even application subroutines) from their original ‘legacy’ programming languages [e.g. COBOL] to more strategic, platform-independent variants [e.g. Java].

Experience, particularly the Y2K conversion, has proved that trying to reuse application functionality (or recreate business processes) at the source-code level is often not possible or practical – primarily because of uncertainty as to the reliability and integrity, over time, of source-code maintenance and control. In addition, if functionality in the form of source-code is desired from 3rd party applications, as is invariably the case today, one can also be faced by intellectual property, copyright and royalty issues. Hence the growing popularity of reusing application functionality in the form of run-time services – which is what SOA, in the end, is all about.

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

Given this run-time execution mode inherent to SOA-based composite applications, one can quickly start to appreciate some of the key issues that have to be addressed; e.g. access control, overall connectivity, transaction orchestration etc. It also becomes clear as to why there has to be a tight-coupling between object sourcing phase and the composite application deployment phase, as mentioned in the previous section. One cannot just source objects to incorporate within a composite application on a ‘stand-alone’, one-time basis. One has to ensure that the required functionality, represented by the sourced objects, will be available, in real-time, when the new composite application is running. In practice developers will also need access to the source applications as they test and validate the new composite application.

The interdependencies implicit in the SOA run-time execution model dictates that one cannot go about creating composite applications without considerable foresight as to the objects that are going to be used – and, moreover, how and where they are going to be sourced. Hence the need for a structured, systematic and disciplined approach – ideally centered around a proven Composite Application Server solution, such as that offered by NetManage, that supports:

- Interactive object sourcing, using visual techniques, from multiple, disparate sources.
- Automated application connectivity with the 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.

**FIG. 3: THE PIVOTAL ROLE PLAYED BY A COMPOSITE APPLICATION SERVER DURING THE EXECUTION OF A COMPOSITE APPLICATION.**

Typically a full-spectrum Composite Application Server, such as NetManage’s OnWeb, will also be used in conjunction with a studio type IDE during the object sourcing phase to capture requisite application functionality and represent it as an object [e.g. EJB] or Web service.
**Snapshot Of What is Involved**

Now that the need for a run-time execution model for SOA-based composite applications has been established it is much easier to appreciate what is involved in creating contemporary composite applications, particularly those that impose zero constraints on the application developers and administrators. The requirements, concerns, methodologies, and technologies that are involved in the three principal phases of realizing such composite applications can be characterized as follows:

<table>
<thead>
<tr>
<th>Major Requirements</th>
<th>Application Synthesis</th>
<th>Application Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reusable functionality independent of source</td>
<td>Language, platform &amp; technology latitude</td>
<td>Platform flexibility</td>
</tr>
<tr>
<td>Flexibility in terms of object technologies</td>
<td>Ability to mix object technologies</td>
<td>Overall connectivity</td>
</tr>
<tr>
<td>Continued availability of the source applications</td>
<td>Interactive test tools</td>
<td>Scalability</td>
</tr>
<tr>
<td>Ability to ‘customize’ the functionality being extracted</td>
<td>Support for all major standards</td>
<td>Administration &amp; manageability</td>
</tr>
<tr>
<td></td>
<td>Simplified GUI creation</td>
<td>Platform stability</td>
</tr>
<tr>
<td>Achieving desired granularity</td>
<td>Delivering expected results</td>
<td>Security infrastructure [e.g. firewalls]</td>
</tr>
<tr>
<td>Ability to mix object technologies</td>
<td>Meeting schedules</td>
<td></td>
</tr>
<tr>
<td>2-way authentication, access control, security</td>
<td>Ensuring reliability and resilience</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relevant Methodologies</td>
<td>Application modeling</td>
<td>Authentication, access control security</td>
</tr>
<tr>
<td></td>
<td>Scripting</td>
<td>Real-time availability of the source applications</td>
</tr>
<tr>
<td></td>
<td>Transaction orchestration [e.g. BPEL4WS]</td>
<td>Meeting performance expectations</td>
</tr>
<tr>
<td>Smart Adapters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Screen Scrapping</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Studio type graphical IDEs</td>
<td></td>
</tr>
<tr>
<td>Pertinent Technologies</td>
<td>Application servers</td>
<td>Application access &amp; navigation</td>
</tr>
<tr>
<td>Web services, WSDL, UDDI</td>
<td>Scripting</td>
<td>Authentication, access control and security</td>
</tr>
<tr>
<td>Java: EJB, JCA, Java Beans</td>
<td></td>
<td>End user delivery facilitation</td>
</tr>
<tr>
<td>.NET assemblies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HTML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SOAP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HTTP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JSR 168</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Role of Composite Application Server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object creation</td>
<td>Presentation layer support &amp; tools</td>
<td></td>
</tr>
<tr>
<td>Object customization</td>
<td>Test &amp; debug tools</td>
<td></td>
</tr>
<tr>
<td>Object registration</td>
<td>Technology ‘bridging’</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authentication, access control and security</td>
</tr>
<tr>
<td></td>
<td></td>
<td>End user delivery facilitation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Central management</td>
</tr>
</tbody>
</table>
Object Sourcing
Gainfully reusing existing application functionality, to minimize software development and validation costs, is what composite applications are all about. Thus the entire endeavor of creating a composite application is predicated upon the developers being able to obtain the necessary functionality, in reusable object form. The SOA-based, run-time execution model for tapping into the pool of existing application functionality obviously complicates matters – despite this also being the undeniable strength of (and motivation for) this approach. With this in mind, the first thing that needs to be established, unequivocally, is how far and wide you are willing to go in order to obtain the required functionality. The options available, in this context, can be categorized as follows (with each category being additive to the preceding ones):

- **INTRANET ONLY**: functionality sourcing restricted just to the applications available on the corporate intranet; i.e. behind the corporate firewall.
- **EXTRANET EXTENSION**: functionality may be sourced from selected, authorized applications available across a secure Extranet from preferred business partners.
- **VPN MODEL**: variant of the Extranet scheme, above, but access to the required applications being run by preferred business partners is across the Internet (using secure protocols),
- **INTERNET/WEB SERVICES**: functionality acquired from a previously unknown, but since validated, service provider across the Internet per the original premise for Web services.

All of today’s popular object technologies can be used in all four of the above scenarios, totally independent of the underlying network infrastructure. Nonetheless, many of the choices related to object sourcing can be influenced, significantly, by which of the above categories is deemed to be needed in order to obtain all the necessary functionality. Security, though critical, is, however, only one of the factors that need to be considered: For a start, one may not need to use Web services if object sourcing is going to be restricted to the intranet or an Extranet. Instead, it might be more efficient just 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 happen 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 that they also do not fully support the creation of SOA-based solutions. This distinction is important because there can be many cases where one may (and should) use EJBs or .NET assemblies to create a Web service.

There is nothing mystical about Web services. Web services, from their inception, have always been about self-contained, software modules with standards-based [i.e. 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 is intended 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 Web service invocation [i.e. SOAP], and the *optional* service advertising mechanism [i.e. UDDI]. See figure 4.

There are no standards or even conventions as to how the actual body of a Web service, i.e. the business logic, should be implemented. But this is intentional. It is this flexibility vis-à-vis 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.

If a Web service consists of a .NET assembly or EJB at its core, one has to question what advantages one can derive from using the Web service as opposed to directly accessing the core object. The answer to this is very simple and somewhat anticlimactic! 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 thus boils down to the advantages of having XML-defined I/O. Obviously having XML-defined I/O, to avoid any and all ambiguity, is extremely attractive and desirable when one is trying to source functionality from a 3rd party – i.e. a service-provider.
This, in essence, should now put the potential role of Web services vis-à-vis composite applications into perspective. Going back to the four ‘topology’ categories listed in the previous page, it should now be clear that Web services are most relevant if one has to source functionality, over the Internet, from the previously foreign service provider. If on the other hand, your composite applications are all going to be based on functionality culled from applications already available on the corporate intranet, or on an Extranet with a preferred partner, you should retain the option of being able to use EJBs or .NET assemblies in addition to Web services. Forgoing that option is an unnecessary and potentially costly constraint. The NetManage Composite Application Server technology includes full support for Web services, .NET assemblies and EJBs so that developers have complete flexibility when it comes to object sourcing.

**Options for Obtaining the Necessary Objects**

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

1. **In-House Creation**: capture the necessary functionality from existing in-house applications and represent them as objects or Web services using proven, graphical tools such as NetManage’s OnWeb Object Builder.

2. **In-House Reuse**: locate pertinent objects that might have been created for prior projects.

3. **External Sourcing**: obtain the required functionality, typically in the form of a Web service, from a business partner or a service provider.

The key issues inherent in using external objects are relatively obvious, and, furthermore, have already been articulated in many public forums in an effort to justify the slower than predicted popularity of XML Web services. Given the run-time execution model, the primary concerns that one has to confront include:

- The whole gamut of “security” related issues spanning rogue service [e.g. that violate data privacy], services that have been surreptitiously ‘hi-jacked’, data transmission integrity etc.

- Reliability, responsiveness and scalability of the service so that 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, in theory, to get around the above concerns with external objects is to try and ‘acquire’ the rights to these 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 corporations. In many instances this approach of trying to bring in the functionality so that it can be behind the firewall proves to be impractical.

The main stumbling block being that the functionality required may not be in the form of a relatively compact, stand-alone piece of software. More than likely it is part of a bigger application. If the functionality is indeed a part of a bigger application, just acquiring a Web service that remotely invokes that functionality and running that 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.

A Web service, in many cases, may not contain any business logic per se! Instead, it might just be a XML-based calling mechanism to an application that contains the required functionality. This is alluded to in figures 2 and 3, and highlighted in figure 5. The notion of having Web services (or other objects) as just calling mechanisms is fundamental and pivotal to the successful realization of 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

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 such that being offered by NetManage. A composite application server, such as this, specializes 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. Security management, however, 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 [e.g. BEA’s WebLogic Server], an integration server [e.g TIBCO BusinessWorks], a host integration/host publishing server [e.g. NetManage OnWeb], or even the so called ‘proxy servers’ used by some of the ‘old-school’ EAI adapter vendors1. A composite application server is meant to provide developers, systems administrators and system operators with specialized, incisive functions and features to facilitate the development, deployment and execution of composite application.

1 Refer to “ADAPTERS CROSS THE RUBICON: New Pure Play Adapters With No Proprietary Layers Distance Themselves From The ‘Clunky’, Old-School Ones” White Paper (by this same author) available on NetManage’s Web site.
In particular, a true composite application server 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 was discussed earlier in the “The 3 Principal Phases” section. If such 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, to reflect the requirements of the objects embedded within the applications. Suffice to say that this could be complex, cumbersome, time-consuming and highly error-prone. Using objects created with screen-scraping can exacerbate these problems particularly when the objects invoke transactions with complex navigational paths within an application due to the use of I/O fields from multiple screens.

NetManage, with its long history of providing secure, automated application access for host publishing and host integration solutions, goes to great lengths in its composite application server technology to provide state-of-the-art automation to eliminate the need for such manual configurations and definitions. This greatly simplifies the entire development-deployment process, expedites application availability and eliminates object correlation related errors. Furthermore, NetManage understanding how automation techniques, in general, help developers be more productive (as well as creative) providing extensive scripting mechanisms as a part of their composite application server technology.

Developers thus have the option of using powerful scripting during the object creation phase, whether using programmatic [e.g. EAI adapters] or screen-scraping methodologies, knowing that all of the required correlation will be done for them, automatically, when they reach the deployment phase. This automation thus makes a huge difference when it comes to developing composite applications with zero constraints. The bottom line here is that *bona fide* composite application server technology, as that available from NetManage, is what enables composite applications with zero constraints to be a reality. If they are deprived of the services of a true composite application server, developers will encounter too many impediments and as such will be forced to develop composite applications that are far from optimum.

**Object Creation Mechanisms**

Objects to be invoked by a composite application have to be created (and tested) prior to the application being deployed for production use. Since the rationale for developing composite applications hinges on the reuse of existing software functionality, much of the objects of interest, need to be created to invoke processes running within other applications. There are just three ways to capture such processes from existing applications:

1. *programmatic* schemes, typically involving the use of application-specific adapters (such as the NetManage Librados adapters)

2. *screen-scraping*, either because there are no appropriate adapters for the applications in question, the object developer has prior experience with this approach, or the object
developer intends to exploit the capability to easily ‘combine, filter and skip’ I/O fields possible this approach.

3. extracting the relevant source code segments if one is confident that the application source code ahs been diligently maintained and is up-to-date

Objects [including Web services] that are to be used in composite applications do not have to be, and in most cases they will not be, self-contained units of software functionality. While it is indeed possible to have totally self-contained objects, such objects invariably fall into the ‘utility’ function category [e.g. create a new display window] as opposed to those that perform a complete business process. There is a very good reason for this. There is, unfortunately, nothing magical about objects – even Web services. They are but reusable pieces of software that confirm to certain standards.

**FIG. 5:** THE POWER AND FLEXIBILITY ASSOCIATED WITH COMPOSITE APPLICATIONS COMES FROM HAVING THE OPTIONS TO FREELY INVOKE FUNCTIONALITY WITH A CHOICE OF USING OBJECTS OR DIRECT ‘IN-LINE’ INVOCATION.
WHY THE SERVICE INVOCATION SCHEME OF SOA IS THE ONLY REAL OPTION

Overall, if one also includes the creation of brand new objects for subsequent reuse, there are still only three ways by which one can create object that will perform requisite business process vis-à-vis composite applications. Of these three options, only two are capable of generating true self-contained objects. The three options available for creating objects, categorized to show the two schemes that permit the creation of self-contained objects, can be summarized as follows

1. **CODE**: develop, test and validate the required 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 obtain the requisite functionality [with one having the option to create such objects using either programmatic access or screen-scraping as discussed above].

Note the obvious and striking parallels here when it comes to creating objects with the overall options available for creating composite applications themselves – which were discussed at the start of this White Paper. In essence object creation is a microcosm of what is involved in producing a composite application.

The cut-and-paste approach, as already discussed in the context of SOAs, is not often practical, whether it be for the creation of a single object or a whole new application. The issues and the costs related to software development, testing and validating end up being the same, whether one is talking about an object or an entire application. The only difference being that an object, being smaller, potentially just requiring less effort and resources when it comes to testing and validation.

This is the inescapable reason why object creation, as well as composite application development, both end up gravitating towards the service calling paradigm – particularly when Web services are used. Hence all the growing interest in SOA-based solutions. Figure 5 depicts the primary function invoking call mechanisms that may be used by a composite application [developed per a zero-constraint paradigm].

Note [using Figure 5 above] that EJB and .NET assemblies may be embedded within a composite application, 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 may be achieved directly or through an object. Furthermore, as already discussed above, a Web service may in turn make use of objects to obtain software functionality or to make the relevant calls to source applications. In some instances, especially when dealing with legacy applications, using screen-scraping methodology, an object or a Web service may end-up first calling a host integration “process” will assist in performing the necessary host access and application navigation functions.

The bottom line here is that in order for composite applications to truly excel, their developers need to have the freedom to use any valid function invocation mechanism – without any artificial
constraints as to the levels of nesting or abstraction. There is nothing wrong or strange with this approach of obtaining functionality, in the form of services, via a nested calling mechanism. It is in practice a proven, entrenched, and extremely sound computer science practice going back to the 1970s – the only difference being 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**

It is this application function abstraction capability, using objects, that in makes SOA-based composite applications viable – and attractive. Without such abstraction, using “calling” objects, it will often be impossible to obtain the necessary functionality from existing applications. It is always important to remember that the existing functionality being sought for reuse may never have been developed in a form conducive (or even accessible) for reuse. In many cases the required 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 in order to deliver that functionality. This is invariably the case with older applications, with the situation often further exacerbated with ‘on-the-fly’ fixes applied to these applications in the form of object-code patches that involve branching off to a new ‘patch routine’.

In the context of SOAs, application functionality, for example that 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 as well as 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 one can typically determine whether it completed its designated task successfully (or otherwise). Consequently, transactions [e.g. extract customer record, update quantity in stock for this item] are the basic, as well as 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.

It is, however, crucial to note that an object or Web service representing a transaction has to also contain all the necessary application access, user authentication and transaction location [i.e. navigation] data – in addition to the I/O fields used by the transaction. When dealing with host applications, NetManage’s screen-scraping technology, optimized and perfected over many years, will, in general, allow developers to capture “complex transactions” involving I/O fields from multiple screens – even spanning multiple, disparate applications.

With NetManage, developers also have the option to easily select which fields they want included as a part of the transaction, and what fields they wish to omit from all the screens involved – with no
restrictions when it comes to what developers refer to as ‘add, skip and delete’. Suffice to say that such “complex transactions” can involve extremely complicated and convoluted application access, authentication and I/O navigation linkages. However, as discussed in the previous section, NetManage’s composite application server technology will automate the tracking and maintenance of all the required linkages, and, moreover, do so between the object creation and application execution phases. Such automation, however, is contingent on enterprises using the NetManage composite application server technology not just for application deployment but also when objects are being created. The bottom line here is that SOA-based solutions, as discussed on page 6, always have a tight coupling between the object sourcing phase and the application execution phase. Genuine composite application servers, such as that being offered by NetManage, have been developed from ground-up to fully appreciate this inescapable coupling and to provide all the necessary automation to greatly simplify and expedite object creation, application synthesis and the subsequent application deployment. Using a true composite applications server is thus pivotal to the whole notion of developing and deploying composite application servers – with zero constraints.

**FIG. 6: USING NETMANAGE’S COMPOSITE APPLICATION SERVER TECHNOLOGY TO CREATE AN OBJECT OR WEB SERVICE REPRESENTING A ‘COMPLEX TRANSACTION’ INVOLVING I/O FIELDS FROM MULTIPLE HOST APPLICATION SCREENS.**
Composite Applications
With Zero Constraints

The Bottom Line
Composite applications make use of functionality from other applications. Trying to obtain the necessary software functionality at the source-code level is invariably impractical due to issues pertaining to source-code maintenance as well as intellectual rights. SOA-centric composite application methodology is thus contingent on the use of a run-time invocation model for obtaining the requisite functionality from the other applications. This can be achieved using objects or Web services – with Web services only having the edge if the required functionality is being obtained, across the Internet, from a third-party. In the case of ‘intranet’ scenarios, just using EJBs or .NET assemblies will typically be easier, and more efficient, than insisting on the functionality being available in Web services form.

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 for example EAI adapters) or screen-scraping can be used to capture transactions and create reusable objects or Web services. Such objects and Web services, however, need to contain all the necessary 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, as that offered by NetManage, greatly simplifies both object creation and application deployment – in addition to giving developers the option of creating “complex transactions” using powerful (but highly proven) screen-scraping techniques with ‘add, skip and delete’ capability. A true composite application server thus provides specialized functionality not available from conventional application servers.

At a minimum, a good composite application server will provide 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 [e.g. scripting], and supporting multiple object technologies. In practice, using a good composite application server, from the get-go, becomes the secret for developing and deploying composite applications that truly meet the ‘zero constraints’ criteria. Without a good composite application server at their disposal, developers will always be at a disadvantage and will be precluded from delivering the best solution – in the shortest possible time. Thus, trying to develop SOA-based composite applications without exploiting a true composite application server will, indubitably, prove to be a false economy! A true composite application server is a wise investment that will, without fail, generate a positive ROI in a relatively short time.