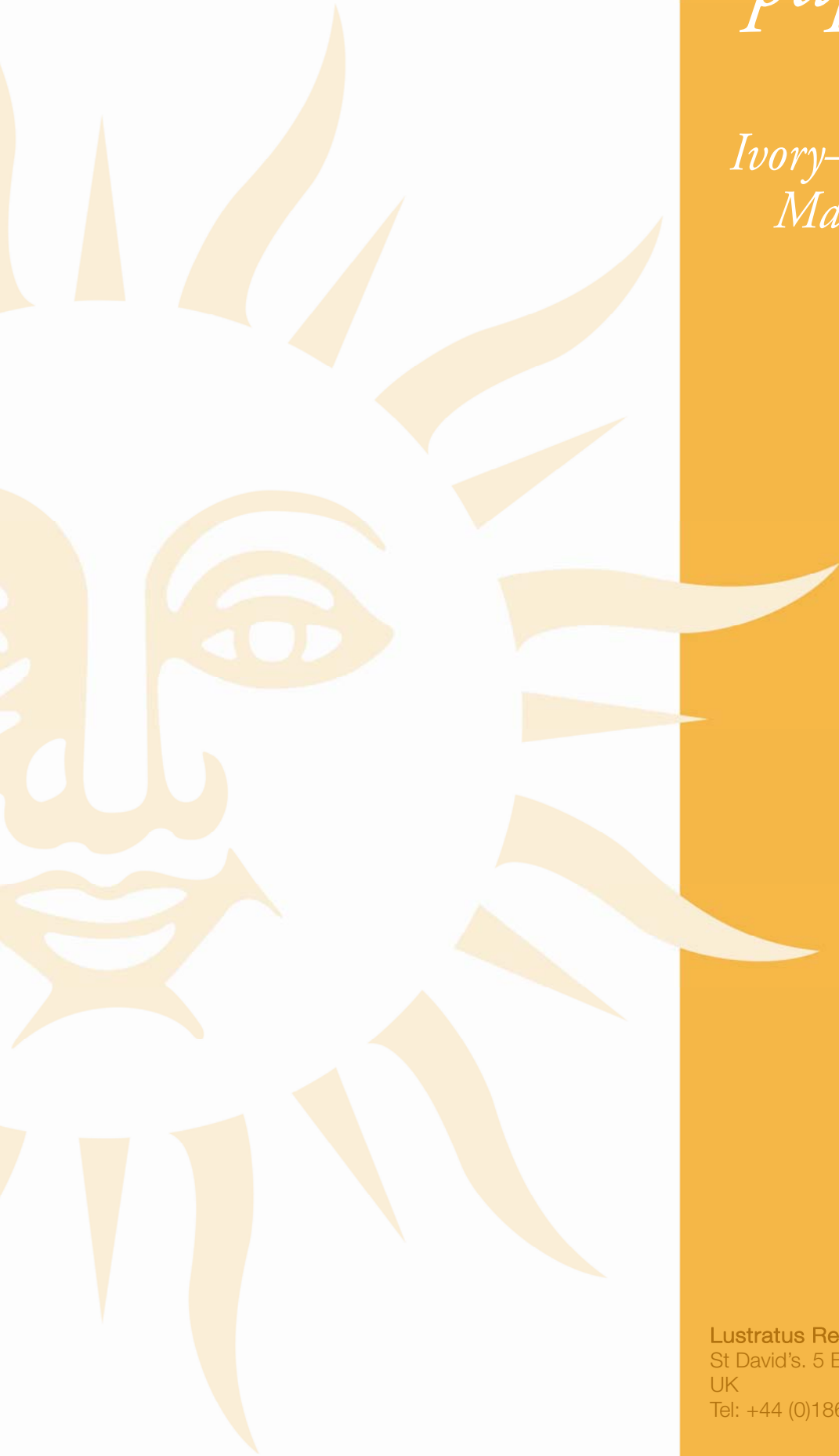




lustratus

*white
paper*

*Ivory—GT Software's
Mainframe SOA
solution*



Lustratus Research Limited
St David's, 5 Elsfeld Way, Oxford OX2 8EW,
UK
Tel: +44 (0)1865 559040



Iustratus

Table of Contents

I.	Executive summary	3
II.	Introduction	4
III.	About GT Software	4
IV.	Ivory Service Architect	4
	i. Overview	5
	ii. Ivory Studio	5
	iii. Ivory Server	9
V.	Ivory Data Access	10
VI.	Assessing the Ivory mainframe SOA solution	10
	i. Basic functionality	10
	ii. Development and deployment	11
	iii. Execution	12
VI.	Summary	14
A.	Appendix—Four user interviews	15

About Lustratus Research

Lustratus Research Limited, founded in 2006, aims to deliver independent and unbiased analysis of global software technology trends for senior IT and business unit management, shedding light on the latest developments and best practices and interpreting them into business value and impact. Lustratus analysts include some of the top thought leaders in market segments such as service-oriented architecture (SOA) and business integration.

Lustratus offers a unique structure of materials, consisting of three categories—Insights, Reports and Research. The Insight offers concise analysis and opinion, while the Report offers more comprehensive breadth and depth. Research documents provide the results of practical investigations and experiences. Lustratus prides itself on bringing the technical and business aspects of technology and best practices together, in order to clearly address the business impacts. Each Lustratus document is graded based on its technical or business orientation, as a guide to readers.

Terms and Conditions

© 2007—Lustratus Research Ltd.

Customers who have purchased this report individually or as part of a general access agreement, can freely copy and print this document for their internal use. Customers can also excerpt material from this document provided that they label the document as Proprietary and Confidential and add the following notice in the document: “Copyright © 2006 Lustratus Research. Used with the permission of the copyright holder”. Additional reproduction of this publication in any form without prior written permission is forbidden. For information on reproduction rights and allowed usage, email info@Lustratus.com.

While the information is based on best available resources, Lustratus Research Ltd disclaims all warranties as to the accuracy, completeness or adequacy of such information. Lustratus Research Ltd shall have no liability for errors, omissions or in adequacies in the information contained herein or for interpretations thereof. Opinions reflect judgment at the time and are subject to change. All trademarks appearing in this report are trademarks of their respective owners.

Executive Summary

Service-oriented architecture is definitely one of the key initiatives for many companies, and likely to remain so for some time, as they try to improve the efficiency, effectiveness and flexibility of their IT systems while improving the alignment between IT and business goals. But, as often with new technology developments, the mainframe environment has been rather neglected. To many SOA tools suppliers, the mainframe is at most a source of services to the wider SOA network, and the main aim seems to be to get in and out with minimal exposure to the arcane world of the mainframe. However, this is just not acceptable to many mainframe users. The mainframe remains a key asset, with substantial investment and value tied up in it, and rather than just try to find a way to leverage this value externally, a far more attractive option is to combine the power of investment on all platforms to deliver maximum return to the business.

SOA can be the key to achieving this return on investments, but in order to optimize the benefits it is essential to find a way for the mainframe to participate in SOA as an equal player. However, since SOA has primarily been driven in the distributed world, new tools are required to bring the mainframe into the fold. These mainframe SOA tools need to bridge the two, sometimes alien, worlds of mainframe and distributed platforms to make both operate effectively within the service-oriented architecture.

This paper looks at one such solution, GT Software's Ivory product suite. It reviews the functionality in Ivory, and then assesses the Ivory solution against market needs, using as its reference point the Lustratus report, "Best of Breed Mainframe SOA Tools". In order to get some additional validation of the conclusions, the Appendix reflects discussions held between Lustratus and four Ivory users.

The conclusion is that Ivory offers a viable mainframe SOA solution, although naturally there are some areas for improvement. Ivory requires little in the way of education, covers the full range of mainframe application types, and unlike most other offerings it makes the mainframe an equal partner in the SOA deployment, ensuring all corporate assets can be leveraged. The table below summarizes the results of the assessment.

What is SOA?

SOA stands for service-oriented architecture, where IT programs and resources are encapsulated and made available as business services, each enacting a discrete business function such as 'Get Customer Details'. The services can then be used and reused as building blocks, assembled together into other business functions and processes as required.

Introduction

With service-oriented architecture (SOA) continuing to be the focus for users around the world, it seems as if nothing can stop this juggernaut. However, one thing that does stop many users in their tracks is the problem of how to bring the huge quantity of mainframe applications and resources into the modern, standards-based SOA framework. While SOA is fine for new application environments, it is still true that globally around 75% of programs reside in legacy systems, and IBM mainframes are by far the dominant player in this segment, and therefore SOA cannot deliver its real value until these environments can be brought into the SOA fold.

Fortunately, tools are available from a number of vendors that enable users to leverage these extensive mainframe investments in the SOA environment. The Lustratus report, "Best of breed Mainframe Integration Tools", discussed the characteristics that are required to deliver the necessary functionality, providing a useful framework against which to evaluate possible mainframe integration solutions. This paper assesses one such toolset—GT Software's Ivory mainframe SOA solution.

GT Software

The company has been around for more than 20 years, with a stated mandate to help mainframe users get the best out of their investments. Over the years, the company has produced a range of tools for the mainframe environment, working closely with IBM and developing a broad customer base worldwide. These tools are used to do things like generate CICS BMS (Basic Mapping Service) screens, make mainframe applications available over the Internet and put web-style interfaces on them, as well as providing SQL access to mainframe-hosted data. However, with its Ivory solution, GT Software has taken a 'complete mainframe' approach to solving the problem of SOA participation, addressing the needs of mainframe transaction processors, databases and even batch programs.

Ivory

Ivory is GT Software's answer to mainframe integration needs. On the SOA front, its intention is to make the mainframe an active and equal participant in SOA, unlocking investments in the mainframe and leveraging SOA investments across all environments. Ivory consists of:-

- Ivory Service Architect—SOA for the mainframe
- Ivory Data Access—SQL-based data access to mainframe datastores
- Ivory VisualConnect—Re-facing tool for mainframe applications
- Ivory Emulator—Emulation for mainframe 'green screens'
- Ivory BMS/TS—Screen generator for CICS BMS applications

While Ivory Service Architect will be the main focus for this paper, Ivory Data Access is also relevant to the SOA discussion since it provides distributed access to mainframe data, rather than applications.

Ivory Service Architect

In simple terms, Ivory Service Architect allows mainframe applications and programs to be turned into web services that can be consumed by any SOA application, and also makes it possible for mainframe applications to themselves consume SOA services. In other words, the mainframe becomes an equal

SOA benefits

Using SOA transforms IT assets into reusable, business oriented components. This results in a range of business benefits. New projects can be developed more quickly, productively and reliably because of the higher level of reuse. IT systems become more closely aligned to business activities, enhancing operational visibility and improving business effectiveness. In short, businesses become more agile and efficient, with minimal risk.

member of the overall SOA deployment.

Ivory Service Architect consists of two products—one, Ivory Studio, providing the development environment, and the other, Ivory Server, delivering the run-time execution engine. In essence, Ivory Studio provides the tools necessary to turn mainframe programs into web services for use in the wider SOA deployment, offer an interface to mainframe applications that allows them to invoke other services and describe any orchestrations required to implement the desired service flows. Ivory Server processes the information created by Ivory Studio and implements it in the run-time environment.

Before looking at these two products in more detail, it is worth summarizing the basic environments supported by Ivory Service Architect.

Overview

The first point to make is that Ivory Service Architect supports the full range of mainframe technologies and environments, including

- CICS
- IMS
- Batch
- IDMS
- Natural

Mainframe applications running in any of these environments can be brought into the SOA arena, and the support is bi-directional; that is, applications in these environments are able to invoke services anywhere across the SOA deployment.

The second observation is that Ivory Service Architect also supports orchestration, allowing business service flows to be specified and implemented. This is particularly useful when dealing with a mixed environment of mainframes and other platforms, because orchestration can often provide a way to optimize communications between platforms, for example by carrying out multiple mainframe activities in an orchestrated flow to avoid the need for a mainframe round trip for every action. It also enables composite applications support.

The third aspect is that Ivory Service Architect does not generate code. Instead, specifications and other artefacts are held in XML format, to be processed by Ivory Server as required. In fact, the only time any coding is needed is when building the ability to call a web service into a mainframe application, where the calling application must define an I/O area before making the service invocation.

Now, before the formal assessment, it is necessary to go into the product functionality in more detail.

Ivory Studio

Ivory Studio runs in a Windows environment, and is graphical and intuitive, with a .NET look and feel. It is used to create services, deploy them and build orchestrated flows to implement multi-layered or composite services. Importantly, services created are Ivory services, which can then be deployed for use by any environment across the SOA network. In Ivory terms, this deployment could be as a web service or a 'callable service'. Web service deployment makes the service available as a standard web service, with the WSDL being generated by Ivory on deployment. Callable service deployment makes the service available to mainframe applications that do not support web service calls. To emphasize the decoupling of service creation from service deployment, notice that if a mainframe application is turned into an Ivory service, it can now be called via a WSDL/SOAP/XML approach, or by another mainframe application through the callable

Key SOA functions

An SOA offers a number of basic services:

- A common way to describe the interface to components
- A registry of available components, their characteristics, locations etc
- Support for standard communication protocols such as SOAP and JMS
- Support for XML to provide standard, self-defining data formats

service interface. The service is unchanged—it is just the means of invocation that is different.

Turning mainframe applications into Ivory services

Users can rely to a great extent on the Ivory Studio Project Wizard to quickly create Ivory mainframe services and flows, augmented by a graphical, drag-and-drop style drawing board for manual activity. There are variations of the wizard for CICS, IMS, 3270, Link3270 and Data Access usage. This is an important point—Ivory includes components that support a wide range of integration and SOA needs on the mainframe. As well as the most commonly assumed support for CICS and IMS applications with programmable interfaces, Ivory can handle applications where the only route in is through a 3270 screen, or where access is through Link3270. In these cases, the same type of wizard-based approach is used to get an understanding of the application flow and input/output requirements. The Data Access aspect of Ivory is discussed in a later section.

With general mainframe applications, the first step is to import the copybooks that describe the input/output area for the desired application—in CICS terms, for example, this is the COMMAREA. Ivory imports these directly from the mainframe libraries where they reside, either as a whole or just the required records. These copybooks can be in COBOL or PL/1 forms. The inputs and outputs can now be mapped as needed, the target application is specified and Ivory now generates the flow. The flow is stored as an XML file—no code is generated. Now this flow can be accessed in flowchart form from Ivory Studio, and can be further manipulated as required.

Supported CICS applications are those accessible through COMMAREAs, the 3270 linkable bridge or MQSeries (WebSphereMQ). IMS applications can be leveraged using 3270, OTMA and IMS Connect, in both conversational and non-conversational modes. Ivory also supports batch.

Making services available as web services

Once the service is created, it can be deployed for use as a web service. When this is done, Ivory generates the required WSDL to be used for invocation, and can publish this to any UDDI-compliant registry. All SOAP message and XML handling is carried out by the Ivory Server component at runtime. When an application elsewhere in the SOA implementation wants to invoke the mainframe service, the WSDL provides the necessary information to define the input and output areas and method required for making the call.

Making services available as callable services

When a service is deployed as a callable service in Ivory terms, the basic approach is very similar to that just described for turning mainframe applications into services. However, in this case there is no pre-defined input/output area for the service. Therefore, one has to be created. Ivory therefore generates copybooks to specify an input and an output area, that can now be embedded in any application wanting to call the service. In addition, Ivory generates sample code to illustrate how the calling program invokes the service, which can be used as a template by the calling application. In this way, Ivory provides all the necessary information that in the web services case is found as part of the WSDL.

Ivory callable services can be called from just about any mainframe environment. Not only does this cover transaction processors such as IMS and CICS, but it also includes batch and even database stored procedures. One point worth noting here is that when an Ivory service is invoked as a callable service, this is not the same thing as a web service in execution terms. There is no requirement for XML, or even SOAP messaging, as there would be in the case of a web service call. This can be extremely useful when working in a mainframe environment, because switching formats from non-XML and back and dealing with SOAP

Service granularity

Service granularity is a major issue for most SOA users, and needs to be considered carefully in the light of several factors. For example, to get reuse, the service cannot be too big or it will be too specialized. But a large number of small services will quickly become unmanageable, and 'call / return' overheads build rapidly.

The trick is to plan carefully, and to use orchestration to increase efficiency and reduce the overheads.

messages can introduce quite an overhead.

Top-down vs bottom-up development

As already described, Ivory Studio can be used to import copybooks and build a service related to a particular mainframe application. This is sometimes referred to as 'bottom-up' development, because you start at the bottom, turn the applications into services and then make them available for external use or join them up in orchestrated flows. Although a natural way for IT programmers to think, this approach can be the source of problems if insufficient thought is given to which services to build and at what level of granularity. The right decision may not be to turn every application into a web service, for example, but instead to link some of the applications as a single orchestrated service under a single WSDL layer.

An alternative method for SOA development is 'top-down'. In the top-down mode, the business requirements are considered first, and once the needs are understood then the required services can be defined, perhaps creating WSDL as a product of a modelling tool. This WSDL can now be imported into Ivory and a service flow can be built accordingly.

Neither approach is right or wrong—each has advantages and disadvantages. However, Ivory allows the user to choose which makes the most sense, and switch between them as required. In general, though, best practices would suggest that the bottom-up approach is best limited to addressing tactical integration needs, while wider SOA initiatives will benefit most from the top-down approach.

Orchestration and composite services

When an Ivory service is defined, its structure is mapped onto a graphical workbench where it can be further manipulated and extended.

The service flow can be modified with the use of a range of nodes that can be dragged in from the toolbox palette, such as

- Decisions—changing the flow based on a set of conditions
- Functions—for example, string manipulations or mathematical calculations
- Loop—to implement a controlled loop in the flow
- Actions—eg authentication, commit, rollback, execute a transaction

Database calls can be made, either directly or using the Ivory Data Access component, and other external web services can be added, making it possible to build composite services consisting of multiple services linked together. And all of this can be carried out below the Ivory service invocation layer. So, for example, an Ivory service consisting of a string of mainframe operations and applications can be deployed as a web service with a single WSDL-defined invocation, with all the implementation details of the flow residing below the WSDL level. This can be very useful to optimize performance and fully decouple the calling environment from having to understand the mainframe implementation details.

One special usage case of this orchestration is the support for IMS conversational applications. Here, Ivory makes use of the scratch pad area (SPA) to share information required by the different application steps, and carries out all the activity within a conversation in an orchestrated flow under a single web service request to ensure that the IMS conversation is not left hanging.

Isolating the application

Since one aim of SOA is to bring application components together, it may seem odd to look at application isolation as another. However, this seeming contradiction is easily resolved. In order to achieve the flexibility and reuse promised by SOA, it is critical to ensure that the application is not tied to the underlying services. Otherwise changes are likely to ripple through the system, affecting all the tied components.

So, the isolation required is not cutting the application off from any connectivity, but is all about encapsulating the application so it is protected from needing to know any of the implementation details of the underlying services.

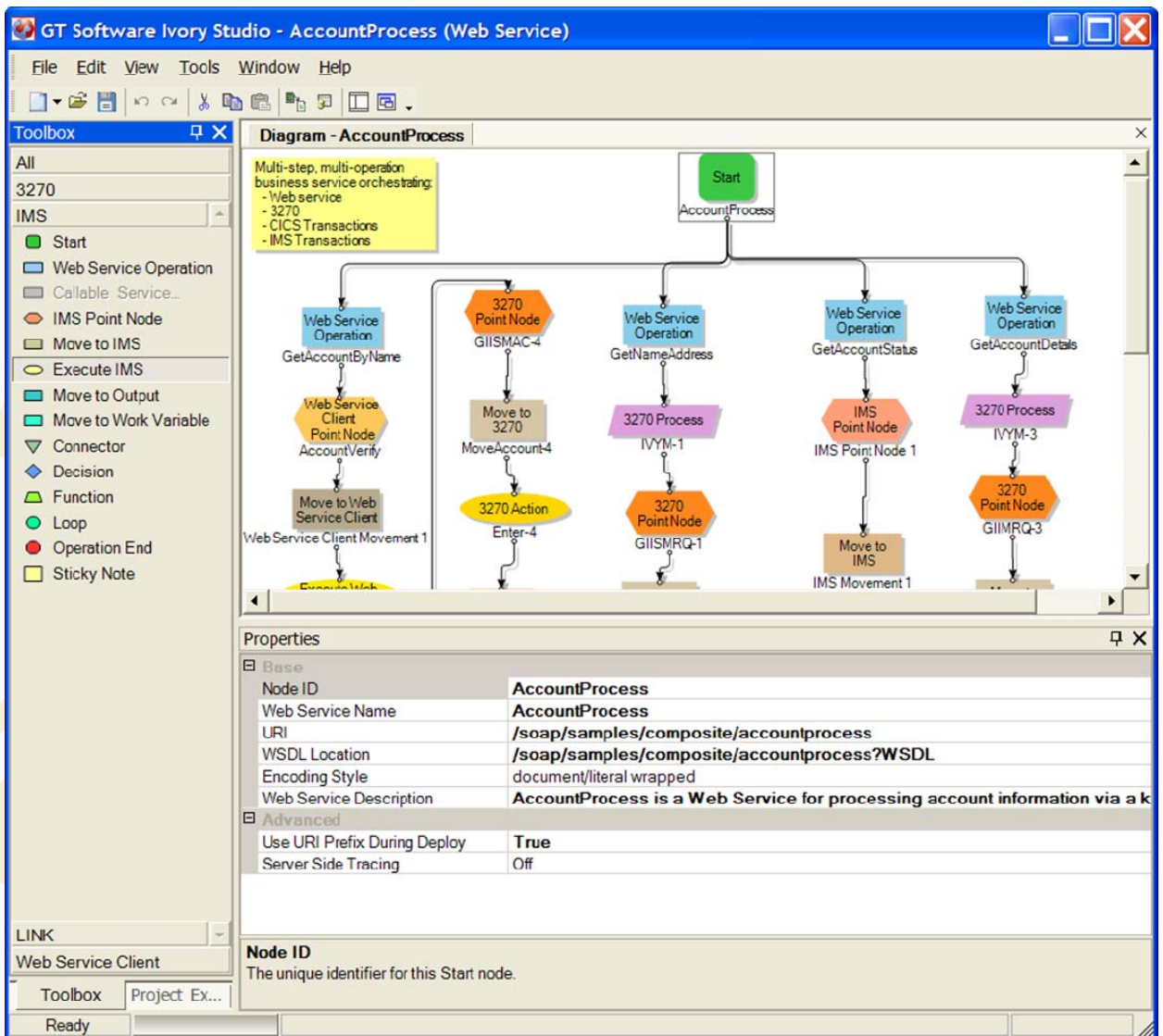


Figure 2:- Screenshot of Ivory Studio orchestration

Testing and management

One feature of Ivory is the ability to quickly unit-test a service flow. A mechanism is provided to issue a SOAP request to trigger the service, allowing the results to be monitored. These test messages can be saved for subsequent reuse. Also, Ivory can import WSDL from external repositories—indeed, there is a WSDL discovery tool that can be used to browse registries and automatically update Ivory with external services and associated WSDL. As mentioned earlier, the WSDL for Ivory services deployed as web services can be exported to any UDDI-compliant registry, and during all of the handling of COBOL copybooks and WSDL, Ivory will handle the marshalling of field names.

Other schemas can also be imported to Ivory as required, such as XML Schemas (XSDs). Usage of XSDs is important, as industry standards such as ACORD, and also proprietary standards, can be easily enforced using XSDs to ensure consistency of data definitions. In terms of maintaining integrity, Ivory automatically validates any WSDL to be used and can police the use of XSDs to ensure that these remain unchanged.

Ivory Server

Ivory server provides the run-time support to handle the execution of Ivory services and orchestrations. The first point to understand is that Ivory server can be deployed in a number of different ways. It can run on or off the mainframe, depending on user needs. Outside of the mainframe, supported environments include Linux and other UNIX systems. When running on the mainframe, Ivory can either run natively in its own address space, or under CICS as a CICS application. In addition Ivory supports z/Linux, offering increased deployment flexibility. The choice of deployment will probably be dictated by factors such as performance and available capacity. However, importantly this is not an either/or decision. Because Ivory Studio does not generate code, it can operate quite easily in a federated fashion, with copies running on both the mainframe and other servers.

The following diagram illustrates these deployment options.

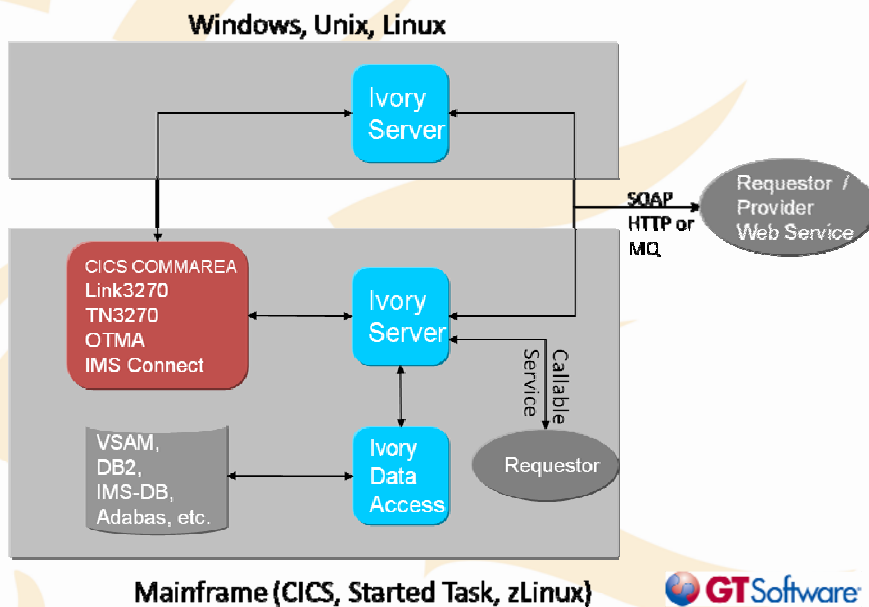


Figure 3: Ivory Server deployment options

Performance considerations

The question of Ivory server deployment leads to the performance characteristics of Ivory. Deployment is the first consideration—for example, if CICS is a major target for Ivory, then the optimal performance will usually be found when running the Ivory server as a CICS application. However, if IMS is the key system, it may be necessary to run the server in native mode.

The ability to run the Ivory server outside of the mainframe environment reduces the mainframe overhead almost to zero, but of course the trade-off is that if multiple mainframe actions are required as part of an orchestrated flow, each new action will require a trip to the mainframe and back since there is no mainframe-based orchestration engine, thereby consuming additional network bandwidth.

Special consideration has been given to Ivory operations with batch applications, however. Here, the Ivory server can run within the batch partition, giving high performance access from batch to other SOA services or vice versa. But batch applications tend to be different in nature to online mainframe applications such as IMS and CICS, in that they tend to execute a long stream of activities rather than short, sharp transactions. Because of this, Ivory has a performance optimization for batch where the communications pipe is left open after each

Speciality engines

Users have been pressing platform vendors like IBM for some time to lower costs, and one response has been the speciality engine— a dedicated processor that can be used to support a specific workload. In fact, it is usually not a full-power processor—it is more a slave to the general processors, and consequently a lot cheaper.

Now, the dedicated workload will be restricted to the speciality engine, and won't consume more expensive general processor power, thereby helping to control costs.

IBM's zAAP is an example of a speciality engine, running Java and XML operations, and it has an additional cost advantage in that IBM does not charge for software running on a zAAP. Similarly, zLIP runs DB2 operations and IFL focuses on z/Linux.

service execution, instead of being closed and then reopened between requests.

Management

In the area of management, mainframe facilities are leveraged where possible. Security is provided at a basic authentication level, while recovery can be addressed with explicit commit and rollback steps in the service flow. If the Ivory server is running in CICS, then Ivory services benefit from using the CICS recovery facilities. Audit facilities are provided through the use of either the CICS or the MVS logger, while the Ivory trace facility can be used to follow service flows and identify problems.

Ivory can also raise SOAP faults, to inform the caller of the service of any problems and send back error codes and related information. In fact, Ivory even supports interaction directly with the SOAP header, for example to get at the non-functional elements of the call such as the userid of the caller.

Ivory Data Access

Reference has already been made to Ivory Data Access, another member of the Ivory product suite. This has a key part to play in enabling the mainframe to participate as an equal partner in SOA, because it makes mainframe databases and files available through a standard SQL interface. In fact, more than this, it makes it possible to create a logical database view made up of different contributions from multiple types of data stores. This enables Ivory Service Architect service flows to address any data needs, easily and safely, by making calls to Ivory Data Access as part of the service flow created within Ivory Studio.

As with Ivory Service Architect, a key characteristic of Ivory Data Access is that no coding is required. Using the graphical tools provided, different parts of data stores can be joined together in a single view for processing, whether the stores are in relational, non-relational or distributed format. Filters ensure that only the data required is brought in, and security is addressed through mainframe and database security procedures. The access provided is not just read-only—updates are also supported, fully protected through such means as two-phase commit processing.

Assessing the Ivory mainframe SOA solution

The assessment will be carried out using the Lustratus report, “Best-of-breed Mainframe Integration Solutions”, as a backdrop. In this report, mainframe integration functionality is considered in three main areas: basic functionality, development and deployment and execution, and this arrangement fits nicely with the assessment of Ivory. One point is worth stressing, however—this assessment is focused on Ivory as an enabler of mainframe SOA, rather than mainframe integration as a whole. Therefore, little attention will be paid to products such as Ivory VisualConnect, Ivory BMS/TS and Ivory Emulator.

Basic functionality

The first point to observe in the area of basic functionality is that Ivory Service Architect supports bi-directional SOA connectivity rather than just one way. That is, SOA services can be created and driven anywhere by any application in the SOA deployment. Often, SOA integration options only look at the mainframe as a source of SOA services, rather than also a user of them. This unbalanced approach may allow mainframe assets to be leveraged, but it does not place the

IBM TP systems

IBM mainframes typically run one of two leading transaction processing (TP) solutions, CICS or IMS-DC. These were introduced many years ago, offering an online environment to supplement traditional batch processing.

Today, virtually every Fortune 500 company runs either CICS or IMS. Although IT options have expanded rapidly, the continued success of these products is based both on continued vitality and power, and perhaps more importantly the fact that there are estimated to be many trillion lines of COBOL code designed for these systems.

As a result, no company can afford to simply walk away from such substantial investments.

mainframe as an equal player in SOA, and hence it limits flexibility and opportunity to create more value from SOA investments. With Ivory Service Architect, these restrictions are lifted.

The next key point is that Ivory provides support for a comprehensive range of mainframe environments and programming languages. In environment terms, most mainframe SOA solutions provide the ability to turn CICS applications into SOA services, but Ivory Service Architect extends that to IMS, other transaction environments like IDMS, batch and even access from database stored procedures. What is more, the support is available for just about any version of these products. Use of Ivory Data Access extends connectivity to most database and file types too. On the language side, by supporting COBOL, PL/1 and Natural, Ivory has all the major bases covered. Admittedly RPG and Assembler are missing from the list, but these are relatively rare. So on balance, this area is a strong plus for Ivory. In particular, the inclusion of logical data access across data sources is a differentiator from most other mainframe SOA tools on the market.

Finally, Ivory provides an orchestration capability. The major reasons this is such an important requirement are that it allows SOA services to be developed with the most appropriate level of granularity, and enables the creation of composite services built of a combination of other services.

Service granularity is one of the most critical issues for successful SOA implementation, and it is even more important when mainframes are involved. The first consideration is to ensure that the caller of the service does not need to be exposed to the underlying implementation details. It may be, for example, that a number of applications need to be brought together to execute a particular operation. While each of these could be turned into services, a caller wanting to execute the operation would have to understand how the different applications interact, making the caller susceptible to future changes. However, if the service is delivered at the level of the required operation, with the multiple applications executed in an orchestrated flow 'under the covers', then this problem is removed. In addition, in a mainframe scenario, if the caller is external and the applications all reside on the mainframe then too low a level of granularity would also require unnecessary trips to and from the mainframe, a not inconsiderable overhead. Not only is there the transport costs, but also there is the problem of data formats. Typically, mainframe formats are non-XML, while it is quite likely that in the wider SOA XML is the norm. Therefore, trips to and from the mainframe usually need transformations to map from XML to mainframe-specific formats and back.

Composite services support is a key way of leveraging investment in an SOA. The idea is that as a pool of SOA services is created, it becomes possible to deliver new business functionality by combining services together in different ways. This maximizes reuse, speeds project delivery, reduces cost and can improve quality of service.

Development and deployment

Having touched on some of the basic functions of the Ivory mainframe SOA solution, the next area to review is the general development and deployment functionality. The first observation is that Ivory Studio is lightweight with an intuitive look and feel, providing wizards and a drag-and-drop interface, and as such the tool is easy to install, easy to use and doesn't take up much space or resource. This contrasts to Eclipse-based tools which can seem a bit cumbersome in comparison, although of course they have the advantage that Eclipse is a standard interface. In fact, the simplicity and ease of use of Ivory Studio is a theme throughout this review, offering quick results with very little education required.

Another important point is the fact that once a service has been created in Ivory,

Bridging the skills chasm

Programmers are extremely loyal to their roots, and tend to quickly take entrenched views on any aspect of technology. One of the big issues for many companies today is that mainframe and distributed programmers tend to live in their own worlds, with as little as possible communication between the two parties. Personnel with in-depth familiarity of both worlds are like gold dust.

But SOA may offer a way to bridge this skills chasm. With a service-oriented approach, it becomes possible to get programmers from each world to extend their horizons at least as far as creating SOA services of their components. This does not break any great taboos, such as asking a programmer from one world to learn about the other – instead, it transforms both types of programmers into service developers.

it can now be deployed as both a web service, invoked through the generated WSDL, and a callable service that can be executed from a mainframe application through the use of the skeleton sample code and associated input/output descriptors. This flexibility means that development skills can be focused on building services, as opposed to building web services or callable services—the design and construction of the service is decoupled from the deployment. The reason this is of value is that typically, programmers belong to either the mainframe or distributed community, with each viewing the other as somewhat alien. It is asking a lot for a distributed programmer to have to understand mainframe quirks, and vice versa. But with the Ivory approach, the mainframe or distributed programmer becomes a service developer, without needing to know about the deployment environment. This reduces skills problems substantially.

The wizards provided by Ivory make it pretty easy to create services from, and make services available to, mainframe applications. One neat feature of Ivory here is that when copybooks are imported from the MVS libraries, they can be selectively imported at the record level. Since some of these copybooks might be very large, this makes the whole process more manageable and efficient. In terms of the applications that can be handled by Ivory, it is worth also noting that through the use of other Ivory products such as Ivory VisualConnect expand the list of candidate applications to include those that can only be driven through a 3270 interface, something that is not always provided by other mainframe SOA tools vendors.

A major plus of the Ivory approach to mainframe SOA is the fact that no code generation is required. The reason why this freedom from code generation is important is that whenever code is required, then it has to be managed. It has to be stored and maintained, and is susceptible to being affected by future changes. Instead, Ivory puts the intelligence built up with the Studio tool into XML files which can then be uploaded to Ivory. One point to note here is that Ivory definitions, configurations and these XML files are all stored in a proprietary Ivory repository. This could be viewed as a bit of a drawback, since IT installations often want to gather system information together in one standard, enterprise-wide repository shared by all.

The Ivory orchestration facilities seem fairly easy to use, with a high enough level of functionality to satisfy most needs. In particular, the ability to include data activities in the flow through the use of the Ivory Data Access tool is a differentiator—while some local database calls might be supported by other solutions, the feature of being able to bring together data from a wide range of sources in a single logical image for interrogation and action offers far more options.

The IMS support offered by Ivory deserves special mention. Although there are other tools that can provide external access to IMS applications, through IMS Connect or OTMA for instance, the ability to make web services calls from within IMS applications is very rare, and the fact that conversational as well as non-conversational transactions are supported strengthens the proposition even more. Companies who use IMS are often very interested in SOA, both as a way to leverage their investments and perhaps to help manage their reliance on a skill-set that is becoming more and more difficult to find. Therefore, full SOA support for IMS which is bi-directional is highly desirable.

As one might expect, Ivory allows WSDL to be published externally, and also provides the means to import it. The WSDL discovery tool is a useful addition to this task. However, of more weight is the ability of Ivory to support industry standards such as ACCORD, allowing the relevant external XSDs to be imported as needed. The ability to validate and police these schemas helps to ensure that integrity is protected.

Execution

The choice of deployment for the Ivory server(s) offers a lot of flexibility. While many companies may choose to run it under CICS, thereby benefiting from CICS facilities such as recovery, the fact that it can run natively, either under z/OS or z/Linux, offers an attractive option for companies either without CICS or with no free CICS capacity. It can even run in a batch partition, alongside the batch application, which offers an important high-performance option. However, there will be companies that have put in place a moratorium on adding any extra load to the mainframe, and here the option of running the Ivory server outside of the mainframe, for example on a Linux server, will be attractive. As usual with these types of decisions, there are always trade-offs, but the flexibility offered by Ivory provides the widest range of options.

The Ivory batch support deserves particular consideration. Batch is an area that many integration solutions ignore, but for many mainframe companies batch is an essential part of operations. The nature of batch applications is that they typically carry out a large amount of repetitive work, and pressure for more and more online hours has forced the batch window, when batch applications can run, tighter and tighter. Therefore, performance is a major concern for batch. Fortunately, the Ivory solution for batch seems to have been designed with performance very much in mind. Not only can the server run in the batch partition, but also the software has been optimized for batch—for example, the feature of being able to leave the Ivory session in place between service calls to avoid the need to keep recreating the environment.

One very interesting feature of Ivory is the fact that the callable service support does not require the use of XML and SOAP for communications. It can use them, for instance when the target service is a web service, but it does not rely on them. This has an important optimization implication which could be another significant differentiator for Ivory. From an architectural point of view, it makes sense for mainframe applications to be turned into Ivory services regardless of where they will be used from. It would be debilitating to have to think about whether the service invocation was actually driving a mainframe-based or distributed service and change the implementation accordingly. But with Ivory, if a mainframe application invokes a service which is another mainframe application, communications can be carried out without the use of XML and SOAP. Because mainframe applications do not tend to use XML and SOAP natively, this avoids a significant overhead, making it feasible to adopt a clean architectural approach to deploying SOA on the mainframe.

In terms of general management functionality, Ivory leverages what is already in place. So, for example, when running in CICS Ivory can make use of CICS facilities to maintain integrity and handle recovery and other failures. Otherwise, recovery options are fairly limited, being more or less restricted to implementing commits and roll-backs in the service flows. Security support is pretty basic too. Statistics and reporting mechanisms are also very limited, although auditing can be done through use of the CICS or MVS logger.

On the testing/problem determination front, the SOAP test facility offered in Ivory Studio is a very useful unit test tool, and the ability to store the test messages provides the means to develop a regression test bucket for future use. Tracing is available to follow Ivory service orchestrations and flow to facilitate debugging.

Summary

Mainframes still form a critical element of IT investments for many companies, representing considerable corporate value. SOA provides the means to unlock that value, making it accessible across the enterprise and from all disciplines. But what is sometimes forgotten is that SOA also offers a way to enhance the value generated by these mainframe assets, by incorporating investments that have been made in the wider, distributed enterprise. The key to gaining optimal advantage is to make the mainframe an equal partner in the SOA, as opposed to a supplier of services to others.

This goal of SOA where the mainframe is on an equal footing with other platforms is the cornerstone of the Ivory approach. But on top of this, Ivory has benefited from the fact that GT Software has had many years of experience with the mainframe platform. It is well known that mainframes are particularly arcane and inscrutable from the outside, and this can cause problems with some suppliers of mainframe SOA tools because they only provide access without understanding the nuances of the environment. In contrast, Ivory has wide-ranging support for just about all the main mainframe environments—not just CICS and IMS, but batch and other mainframe subsystems, as well as the various different types of mainframe data stores, whether relational, hierarchical or flat files.

The fact that Ivory can leverage this mainframe skill base is evident throughout the product suite. Other examples include the specific performance measures for batch, the ability to optimize mainframe to mainframe service calls and support for applications only accessible through a 3270 interface.

However, having stressed GT Software's mainframe credibility, the other point that comes across strongly is that this does not mean the tools are mainframe-centric in terms of skills. The intuitive look and feel coupled with the extensive wizards and the 'no code generation' approach make Ivory easy to install and use by people with either mainframe or distributed skills. In other words, it provides an environment where the developer can concentrate on building services without knowledge of the underlying technologies.

The general area of manageability does leave something to be desired, however. Particularly when Ivory is not running under CICS where it can benefit from native CICS features, functions such as recovery, security and reporting are fairly basic. However, this is not unusual in the early life of products, and it is to be expected that these areas may receive more consideration in future release.

But the final point to note is that by providing orchestration facilities, Ivory has really lifted the overall value of the tool. Orchestration is an absolutely critical feature in mainframe SOA, because it enables services to be built with optimal granularity from both an architectural and efficiency perspective. Because mainframes do not generally deal in standards such as XML and SOAP, any communication to the mainframe and back will incur transformation penalties, and need to be minimized where possible. In addition, true SOA loose coupling is impossible without orchestration. In addition, orchestration is what enables Ivory's composite application support, another key element of the SOA armoury.

Ivory provides an easy-to-use, efficient and effective approach to mainframe SOA in its fullest sense. It seems to manage to cleverly span the mainframe and distributed worlds, providing a high degree of affinity to the mainframe environment while at the same time offering the distributed world an easily understood interface. Most important of all, Ivory offers the chance for all mainframe and distributed investments to be brought together in an SOA of equal members, providing the best possible return across the whole enterprise.

Appendix

In order to validate the assessments made in this review, Lustratus talked to four financial services companies who are using GT Software's Ivory product suite. By way of a caveat, the four interviews were arranged by GT Software, but the vendor did not participate in the calls in any way, providing Lustratus with direct and independent access. Although none of the four companies were prepared to be named in the report, largely because of the daunting challenge of gaining the necessary internal corporate approvals, they were happy to share their identities with Lustratus on a confidential basis, so Lustratus can vouch for their authenticity.

In the following examples, all points made came directly from the customer.

Customer examples—Company A

Background

This European banking user has a wide spread of applications running on different architectures. Basically, the core banking systems run on an IBM mainframe running IMS-DC, being made up of about 1000 COBOL IMS programs. Data is held in IMS-DB or DB2. In contrast, front-end systems are built in UNIX, WebSphere and Java.

The main driver for a mainframe integration solution was to satisfy the need for new front-end applications, preferably browser-based or at least graphical, without having to migrate applications off the mainframe. The bank is very happy with the mainframe as a platform, particularly from the viewpoint of stability and availability, and has no intention of changing. The major issue that had to be addressed in any solution was that the mainframe programmers were finding it difficult to break out from many years of IMS-DC MFS screen thinking.

Product selection

Realizing that a new approach was needed, the bank started with the premise that it would try to use standards as much as possible. Given the need to maintain and leverage the IMS-DC investment, the only options the bank identified initially were IBM MQseries or IMS-Connect, but both of these options led to the bank needing to build proprietary integrations which it wanted to avoid. In the end, the decision was that a web services approach would be best, given the widespread industry backing from companies.

The bank identified two possible suppliers, IBM and GT Software, but initial discussions with IBM resulted only in confusion around what products were needed to deliver the desired function, so the decision was made to investigate GT Software's Ivory further. GT Software was invited on site for one week to carry out a proof of concept. The vendor worked with bank staff to get the product installed and staff educated, which was all completed on the first day. By the end of the third day, staff were able not only to develop new services themselves, but also to understand how to build them most effectively when taking into account the special needs of the IMS-DC environment.

Usage experiences

The new services are owned and maintained by the mainframe team. This has caused some problems, since the developer of the front-end application often has specific needs and tends to want a service designed as an exact fit, but in the end the approach seems to be the best to ensure that the services created can be used by as many front-end developers as possible.

The bank is running the Ivory server on the mainframe as a z/OS started task. Administration is carried out by the mainframe systems programming team, and

they have not experienced any difficulties in this. General performance has been fine, with the software being very stable. Initially there were some difficulties on the security front, but these were addressed by an update to the product to allow a user-id and password to be passed in the SOAP header for user-level authentication. One of the applications delivered through Ivory is to support internet banking, and therefore the bank had some stringent scalability requirements. Load tests were carried out, confirming that the server met these requirements comfortably.

An important by-product of the move to use web services to handle the needs of modernising the application front-ends has been that the department responsible for building these solutions has been able to greatly increase its value, both actual and perceived, to the bank as a whole. Previously, the department was seen as simply enabling the need to change front-ends, but now it is building services that have potential uses and advantages on a much wider level. For example, the department can contribute to initiatives such as improving straight-through processing rates, or whatever other initiatives the bank may have.

Wish-list and final comments

The most pressing needs that the bank had, such as those in the areas of security and scalability, have already been addressed in the product. The only area mentioned where the bank would like to see more support is in integrating better with change and configuration management processes and procedures. As far as the vendor itself is concerned, the bank praised GT Software, particularly from the viewpoint of a European company dealing with a US-based vendor—support has been excellent.

Customer examples—Company B

Background

Company B is a US-based financial services firm, active in a number of different markets. Historically, it is an IBM mainframe shop with IBM CICS as its main transaction processing environment. It also has batch applications and uses DB2, VSAM and a third-party database for data stores. Outside of the mainframe, the environment is Windows desktops. The company makes use of some Java in a client server environment, with WebSphere for zSeries as the server.

The main drivers for Company B's interest in mainframe integration are to expose the legacy assets to Visual Basic desktop clients and Java browser applications, and to find a way to leverage its extensive COBOL skill set. The company is also discovering the need for a lot of composite applications, so an environment to support this is required.

Product selection

A number of suppliers were considered, including IBM with its SOAP Server for CICS and Clientsoft, now part of Progress Software. In the end, company B decided on GT Software's Ivory product suite, giving the following main reasons for the decision:-

- Wanted a solution that could be put in the hands of COBOL programmers
- Needed to call multiple COBOL programs and combine information within a single exposed service (orchestration)
- Required bi-directional support (host to workstation as well as workstation to host) since the mainframe is still central to future plans and goals
- Functional details met requirements
 - For example, the company wanted to be able to manipulate WSDL

Usage experiences

The company found Ivory easy to install and configure, with the observation that 'it works'. Operationally, Ivory has been a success in the projects where it has been deployed so far. Initially, the company had GT Software on-site for one week, at the end of which company employees were building their own services and orchestrations, and administrators were able to carry out their tasks.

While programmers were quick to be able to use Ivory, additional education was needed to help them understand what a service should be and how it should be designed and implemented. Company B ended up developing a 12 hour class for new users of Ivory, both to introduce them to the tool but also to provide the ancillary knowledge needed about service and SOA concepts and best practices. A key point made by company A is that the services and orchestrations are owned by the mainframe COBOL programming team. When looking at other solutions, the concern was that the services created needed to be owned by the workstation-based service consumers rather than the mainframe resources because of the skill set required. Ivory has proved to be something that mainframe COBOL programmers can handle comfortably.

Company B has stringent performance requirements, so this area is a key focus. The company has found response times to be absolutely fine, with the Ivory server adding only a few hundredths of a second overhead. However, CPU utilization has increased on the mainframe due to the XML transformations and SOAP request formatting required.

Wish-list and final comments

Ivory is working successfully today, but at the moment company B does not have any monitoring or version control in place. These are areas where it would like to see more support in Ivory. The company also commented that security is too coarse at the moment, being just at the user authentication level, and that it would like to see security implemented at the service/operation level.

The biggest wish, however, was for additional flexibility over where the Ivory server runs. Company B would like to avoid the CPU utilization impact mentioned above, but cannot move the Ivory server off the mainframe currently due to the fact that the data sizes exceed Ivory's limits for this configuration. One wish expressed in this area was to be able to run the Ivory server, or at least the XML and SOAP manipulations, in a zAAP processor.

Overall, company B said it was very happy with its experiences with Ivory as a product suite, and GT Software as a supplier. The company specifically wanted to mention the excellent level of ongoing support it has had from the vendor.

Customer examples—Company C

Background

Company C is a US-based Insurance company, running primarily CICS/VSAM on the mainframe, with some DB2 and batch processing as well. There are also other applications running on the mainframe outside of CICS. The rest of the systems are mostly Java on UNIX. The core claims systems run on the mainframe, while enrolment applications typically run on the UNIX servers. The company has also implemented TIBCO solutions to provide an enterprise service bus (ESB) capability.

The driving force behind the Insurance company's interest in mainframe SOA is the need to deliver increased agility, building flexible and disposable solutions based on the overall move to consumerism. Cycle times are moving from 12 months to 6 months then 3 months. On a more immediate and practical front, the company is also seeing its mainframe skills base decreasing, and therefore

wants to leverage its investments in the mainframe by exposing its assets to a wider audience.

Product selection

Company C decided on an SOA strategy to address the agility, flexibility and time-to-market issues, as well as to mitigate the risk of decreasing mainframe skills while continuing to deliver an ongoing return on mainframe investments. The solution had to support bi-directional operations because the company wanted to add value to the mainframe as well as use its assets externally. However, the company also wanted to leverage its investment in the TIBCO ESB, and it was this point that really closed out IBM as a supplier, since IBM was proposing to replace the TIBCO ESB with its own. What the insurance company really wanted was an adapter from the TIBCO ESB to and from mainframe applications. The only other alternative considered was to use the MQ bridge, but the company felt this resulted in too many one-off custom solutions.

In the end, the shortlist was GT Software and NEON Systems (now part of Progress Software), and both were invited to a bake-off. However, NEON could not complete the bake-off, whereas GT Software did. In addition, company C's mainframe audience felt greater affinity to the GT Software solution. Hence, Ivory was selected.

Usage experiences

With regard to using Ivory, the company said that it has not been unsuccessful in anything it has tried. However, the view is that Ivory is not a zero-training tool. The first problem is that getting mainframe staff to make the mental mind-shift to SOA has not been easy. For example, initially programmers were building WSDL containing Name1, Name2 and Filler fields. Also, GT Software had to be called in a few times to help staff understand best practice patterns. In general, skills transfer was most successful when GT Software was directly involved.

In terms of time to market, company C observed that what would have taken four to five weeks with the MQ bridge took only a week with Ivory. In addition, the company found Ivory easy to administer, and that the administration could comfortably be done by mainframe staff. Also, mainframe programmers were able to own the services and orchestrations comfortably. The only problems occurred due to company C's configuration, where it has two mainframes with Ivory installed on one and linked to the other. This caused a number of security set-up problems.

Wish-list and final comments

Company C found that Ivory did not support security at the WS Basic Profile, although apparently there has been work done in this area recently. It was also felt that the lack of active monitoring was an issue. At the moment, the company is having to create some innovative solutions to implement its security and monitoring needs, and it would love Ivory to do more in this area, perhaps by supplying some agents for different tools and environments.

However, to finish off, the company is satisfied with Ivory and GT Software—indeed, the spokesperson for company C stated that “GT Software is the most responsive vendor I have ever dealt within the last 20 years”.

Customer examples—Company D

Background

Company D is a US-based bank, offering a range of financial services products covering retail, commercial and investment banking needs. The company has a strong IBM mainframe investment spanning many years. The mainframe runs in a Sysplex environment, supporting products such as CICS, DB2 and MQSeries

together with a range of historic batch applications. Outside of the mainframe, there is a fairly wide mix of hardware and software including Windows, SUN and Oracle. One of the reasons for this diversity is that the bank has been following a vendor application-driven approach, selecting the right application packages for the business and then deploying whatever technology is required, although there is a general direction to try to go with WebSphere Application Server-based environments if possible.

One of the main reasons why the bank wanted a mainframe integration solution was that it has a strategic imperative to leverage existing mainframe investments as much as possible. However since most new application packages, typically either Java or .NET based, are likely to need to interface to core systems such as deposit and loan, which reside on the mainframe, this is another key reason why mainframe integration is required. The company had been using MQSeries to achieve the necessary integration, but this has resulted in a rapidly increasing number of one-to-one connections that have become unmanageable. Creating MQ connections to solve integration needs was also taking too long.

At the same time, the bank had decided on a service-oriented strategy to deliver the agility, flexibility and responsiveness demanded by the financial services market today. However, a key need was to be able to implement this without having to rewrite mainframe applications—the solution needed to be as non-intrusive as possible.

Product selection


In order to address the service-oriented strategy, the bank developed a reference architecture and roadmap based around the concept of CICS service enablement. It then produced an RFI (request for information) package which it shipped out to a range of potential solution providers, and through this process it was able to narrow the shortlist down to IBM WebSphere Designer for zSeries, Seagull Legasuite and GT Software Ivory.

The next step of the process was to bring each candidate in for one week for a proof of concept. The idea was that the bank set some challenges requiring the creation of some mainframe services, and although the vendor could bring in whatever resources it needed, the bank's own mainframe programmers should do the actual work. The only vendor to achieve the goals in the time available was GT Software. Specific differentiators referenced were the ease of installation, ease of use and extent of match with functional requirements.

Usage experiences

The Ivory server runs on CICS, and usage is primarily to provide access to mainframe applications from outside. The bank anticipates limited service usage in the other direction, such as the ability for the mainframe application to call an external service to send out an email alert, but focus is definitely on exposing mainframe services. Having said that, the other usage mode that is important to the bank is the ability for batch applications to drive CICS services.

So far, two projects have been completed. The bank says it has not experienced any bugs so far, and that the services have been extremely quick to build. In fact, it has assessed the length of time to build the services with Ivory against the time to carry out the same integration operation through use of MQSeries, as done previously, and it claims that using Ivory is a 75% improvement. The bank offers its own one-day training class, during which attendees end up developing three web services, and following this training it has found that staff are able to then be productive. It also reports that the examples and help facilities provided with Ivory were really helpful in helping the mainframe programmers to pick up the tool quickly.

A decorative background graphic on the left side of the page, featuring a stylized, golden-yellow face with large, expressive eyes and a wide, smiling mouth. The face is composed of various geometric and organic shapes. To the right of the face, several long, curved, golden-yellow rays emanate outwards, resembling a sun or a stylized flame. The overall aesthetic is clean and modern, with a warm color palette.

On the performance front, response times and scalability were fine, other than the CPU required to perform the XML/SOAP manipulations on the mainframe. The bank is keen to see the ability to put this part of the load into a zAAP processor, but is also currently moving towards the Ivory callable service approach rather than web services, to bypass the XML/SOAP processing. The bank is OK with basic authentication for security at the moment, but if it moves more heavily to web services it would like to see closer support of the WS standards around security.

Wish-list and final comments

The bank was generally happy with Ivory functionality overall. The only areas mentioned where it would like to see some improvements were lifecycle management and support, and better integration across the Ivory family. This last point reflects the fact that the bank is having a few problems when dealing with non-BMS 3270 applications. Although one of the Ivory products, Ivory VisualConnect, provides this support, the comment was the user found it annoying that this had to be handled outside Ivory Visual Studio, although the resultant script could then be brought into the Studio tool once it had been created.

Apart from this, the bank commented that all other issues have been handled by GT Software quickly, and that in general they have found GT Software to be an 'extremely responsive' vendor.



lustratus

Lustratus Research Limited
St David's, 5 Elsfeld Way, Oxford OX2 8EW,
UK
Tel: +44 (0)1865 559040

Ref SC/LR/89955603/V1.0