

Composing CICS® Services

A HostBridge® White Book

By Russ Teubner, CEO



866-965-2427

info@HostBridge.com

www.HostBridge.com

Copyright Notice

Copyright © 2008-2010 by HostBridge Technology. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, optical, chemical, manual, or otherwise, without prior written permission. You have limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry this copyright notice. No other rights under copyright are granted without prior written permission. The document is not intended for production and is furnished “as is” without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Revision date: 1/11/2010

Trademarks

HostBridge and the HostBridge logo are trademarked by HostBridge Technology.

Composing CICS® Services

A HostBridge® White Book

Preface

More than a year ago I set about writing an “anti-white paper” – more relaxed in style, hopefully less boring than most, including a few I’ve authored, and honest about its marketing message. It was titled *The Role of HostBridge® in the CICS® Services Architecture*; it remains a work in progress.

The “white book” you are reading now represents the culmination of the first phase of this effort. It synthesizes versions of sections and chapters we published during 2009.

More importantly, it provides you, the reader, with a thorough analysis of a topic that is only gaining in importance – composing services to integrate CICS with other systems.

Many of the largest organizations in the world rely on CICS Transaction Server to run critical operations. Now more than ever, they are compelled to do more, do it better, and spend less. Practically speaking, they – and probably you – need to make processes more efficient, make people more productive, cut costs, and squeeze more revenue out of System z mainframes and software.

Composing CICS-based services is one of the most effective ways to accomplish these objectives. Organizations are using services to extend CICS applications and mainframe data to the entire workforce; “modernize” CICS applications with Web interfaces for today’s users; orchestrate and automate transaction processes; and integrate CICS with BPM, CRM, ERP, or any other distributed system.

The Point

In this white book, I have a point to make:

That HostBridge, CICS Transaction Server, and IBM’s Service Flow Feature, when used together, meet the broadest possible range of CICS application integration requirements – and in the most intelligent manner possible.¹

My hope is to make this point with technical accuracy, passion, humor, and a bit of irreverence. But there are a number of problems.

First, the point is amazingly difficult to make! Why? Because it relies on a clear understanding of what HostBridge, CICS TS, and SFF do individually and how they can work in concert.

¹ The phrase “application integration” is meant in the most generic way, to describe the incorporation of CICS applications into architectures such as SOA, WOA, ESB, or any other yet-to-be-discovered architecture with three-letter acronym. I do not intend any linkage to Enterprise Application Integration (i.e., EAI).

The fact is, CICS integration isn't simple, and neither is its story. To really understand CICS integration, you have to understand a range of interlocking business and technical issues, and you have to understand how IBM products and complementary products like HostBridge can be used together to address those issues. An accurate understanding of such a nuanced story is always more difficult. But the thorough approach pays rich dividends.

Second, there's you – the technology, architecture, or business integration professional. If you're like many of your peers, you feel overwhelmed. Overwhelmed by the myriad business requirements and projects fighting for attention. Overwhelmed by the diversity of systems and applications in your organization. Overwhelmed by the fact that the technology innovation rate always seems to outpace the technology implementation rate (and the larger your organization, the more true this is). Whatever you do, it seems you can't keep up or catch up. Technology professionals who support mainframe systems are acutely aware of this dynamic.

Because you're overwhelmed, it's often too easy to embrace simple, all-encompassing marketing messages from software vendors. But simple, all-encompassing messages rarely tell the whole story. Unfortunately, *you* are usually left on your own to figure this out! Too many customers, including many of ours, have bought into simple messages and then bought a "do it all" CICS integration product before understanding the story in all of its depth and complexity. After their projects fail, they are usually very well informed about what products do or don't do. Unfortunately, they've wasted a lot of time and money on what amounts to an expensive learning experience.

Hence, my goals in writing this white book: provide usable insight into CICS integration and save you time, expense, and agony in the bargain.

Outline of Sections

Here, section by section, is what you'll be reading:

- CICS Services and Integration: Key Terms and Concepts
- Services Technologies: CICS, WSA, SFF, and HostBridge
- What We Compose: Visual and Non-Visual CICS Applications
- Why We Compose: Business Realities
- Where to Compose: Inside CICS
- How to Compose: Modeling *and* Scripting
- Who Composes
- SFM and HostBridge – Together at Last
- Making CICS Services Decisions

So let's peel back the onion and savor the nuance of the CICS services story.

1. CICS Services and Integration: Key Terms and Concepts

Well, wait a second. Before we swan dive into the world of Web services, specifically as implemented in the CICS world, we need to define some key concepts. Getting these things nailed down early should help us decode what IBM, HostBridge, and other vendors have to say about these topics – and help us see through any coded or cryptic messages regarding application integration to the underlying truth. In essence, we’re building a CICS Web services “secret decoder ring.”²

Web Services vs. Business Services

In these pages, the terms “business service” and “Web service” will be used liberally. While similar, they are not synonymous.

It has become common practice in our industry to use the term “business service” to denote a programmatic entity that accomplishes a specific function relevant to a business. The function may be high level (e.g., obtain details on all pending orders for a customer) or low level (e.g., change the billing ZIP Code for a customer). When we say “business service” we are highlighting the fact that a programmatic service exists. What we are trying to avoid are any assumptions as to HOW the business service is invoked or how data is exchanged. Specifically, we are often trying to avoid the baggage and imprecision associated with the term “Web service.”

The term “Web service” has become part of popular culture and vocabulary. Many people use “Web service” to describe any service that is accessible via a Web browser – e.g., looking up the value of a used car on Edmunds.com. This usage is perfectly understandable. (The only problem arises when some IT vendors purposely exploit the imprecision of the term, which drives me crazy.)

However, in the IT community the term “Web service” means something a bit more specific – or at least it should. Per Wikipedia:

A Web service is defined by the W3C as “a software system designed to support interoperable Machine-to-Machine interaction over a network.” Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate using XML messages that follow the SOAP standard. Common in both the field and the terminology is the assumption that there is also a machine readable description of the operations supported by the server written in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP

² For readers not acquainted with American popular culture, a secret decoder was an inexpensive toy popular among children beginning in the 1930s. Decoder badges or pins were included as a toy prize in boxes of breakfast cereal and Cracker Jack. Among the most famous decoders were the ones given to kids by the makers of Ovaltine. Ovaltine started cryptological premiums for the *Little Orphan Annie* radio program and continued with the *Captain Midnight* radio and TV programs. Decoder rings were introduced in the 1960s by the PF Shoes company and others. (Wikipedia: Secret Decoder Ring, accessed 6/21/08)

endpoint, but it is a prerequisite for automated client-side code generation in many Java and .NET SOAP frameworks. Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a Web service. (Wikipedia: Web service, accessed 6/21/08)

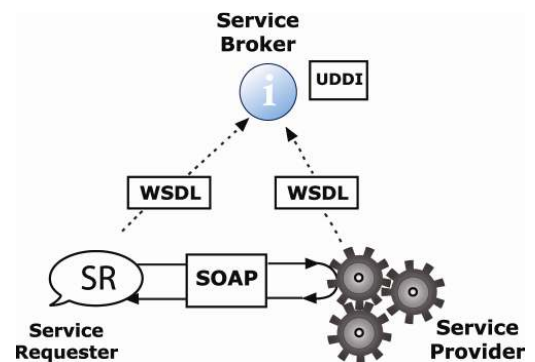
That's a lot of alphabet soup, but we can glean two observations from this definition. First, the term "Web service" simply implies a program-to-program interaction over a network, and this network can take many different shapes and forms. The de facto assumption is usually that the network is implemented using TCP/IP, and the programs exchange requests and responses using HTTP. But this is just an assumption, as many alternatives exist (e.g., message brokers such as WebSphere MQ or TIBCO). Second, there are a number of ways to implement this program-to-program interaction (i.e., how requests are described and data exchanged between clients and servers). This distinction creates the need to define two other terms.

Formal vs. Informal Web Services

The definition above reflects the fact that there are a number of ways to implement program-to-program interactions and still be considered as Web services. For example, the Web service can be implemented as a "Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services." Or the Web service can be implemented according to a more formal approach: "clients and servers that communicate using XML messages that follow the SOAP standard." This distinction is very important. But we need some words to describe the difference. In this paper I'll use the words "informal" and "formal" to denote two basic approaches to implementation of Web services.

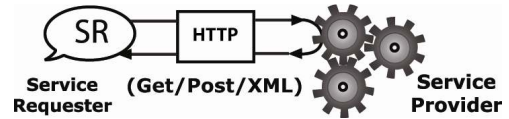
At this point in history, when someone uses the phrase "Web services" or "Services Oriented Architecture" (SOA) they often assume the formal approach (use of SOAP, WSDL, etc.). But the practical reality is that a formal approach to Web services requires more human planning, as well as machine processing, on both sides of the connection. Thus, deploying a formal Web services approach may be more appropriate when:

- You don't own/control both the requesting and responding system (e.g., inter-company connections)
- The client and server are in different security zones (e.g., outside and inside the firewall, respectively)
- Volume is low to medium
- The business relationship between the two parties is "loosely coupled."



Alternatively, deploying an informal Web services approach may be appropriate when:

- You do own/control both the requesting and responding system (e.g., intra-company connections).
- The client and server are in the same security zone.
- Volume is medium to high.



The distinction between informal and formal Web services is very important. In fact, we are starting to see glimpses of support for informal Web services built into CICS TS (e.g., support for RESTful services and protocols like ATOM).³

HostBridge customers have always been at the leading edge of implementing CICS-based Web services. And while some use HostBridge as part of a formal Web services approach, the majority use HostBridge to implement informal Web services. Specifically, the most common methodology is using HTTP GET or POST requests, whereby the input parameters are specified using a URL query string or POST data, and the output data is an XML document (using a customer-specific or service-specific XML vocabulary). Why do customers do this? Because it's very simple and efficient.⁴

Summary

I may slip up here or there, but these are my working definitions in this paper:

- Business service – a programmatic entity that accomplishes a specific function relevant to the business; it might be deployed as a Web service, but there is no such assumption made.
- Web service – a particular approach to making a business service broadly available in a networked, machine-to-machine environment:
 - Formal Web service – an approach whereby clients and servers communicate using XML messages that follow the SOAP standard; WSDL is used to describe the operations supported by the server.
 - Informal Web service – any approach to implementing a Web service that does not abide by the SOAP and/or WSDL specifications. HTTP and XML are used but the formality of SOAP and WSDL are dispensed with.

Terms in hand, we're ready to delve the deeper depths of CICS application integration.

³ Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. As such, it is not strictly a method for building what are sometimes called "Web services." The terms "representational state transfer" and "REST" were introduced in 2000 in the doctoral dissertation of Roy Fielding, one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification. The terms have since come into widespread use in the networking community. REST strictly refers to a collection of network architecture principles that outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface that transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies. These two meanings can conflict as well as overlap. (Wikipedia)

⁴ For more on formal and informal services, read our white paper, "SOAP and REST: Choosing Formal and Informal Web Services for CICS Integration" – www.hostbridge.com/index.php/library/whitepapers.

2. Services Technologies: CICS TS, WSA, SFF, and HostBridge

Everyone encounters situations where they think, “If I had a dollar for every time I (*fill in the blank*), I’d be rich.” For me, I’d fill in the blank with “every time I explain what CICS TS and Service Flow Feature do or don’t do.” Understanding the capabilities of CICS TS and SFF is critical to understanding the value of HostBridge.

In CICS TS v3, IBM added a number of significant features under the heading of “Web services enablement.” For example, CICS TS v3 added an elegant and robust transport management architecture built around new CICS components (i.e., pipelines, channels, and containers). Furthermore, support for HTTP 1.1 was added and support for SOAP was formalized. Layered on top of these capabilities were new features to implement formalized Web services. These features were implemented via new resource definitions (e.g., WEBSERVICE), EXEC CICS commands (e.g., EXEC CICS WEBSERVICE), and the CICS Web Services Assistant utility program. Collectively, these infrastructure improvements laid the foundation for CICS TS functioning as a first class citizen in the Web services world.

I used the word “infrastructure” deliberately, because it describes the correct relationship between HostBridge and CICS TS: HostBridge *builds upon* the Web services infrastructure provided by CICS TS. Our focus has always been on the layers *above* Web services infrastructure. Our focus is *application integration* – not SOA infrastructure.

In fact, the explosion of new features in CICS TS v3 became the foundation of numerous HostBridge enhancements, thus providing additional value to HostBridge customers. For example, HTTP 1.1 support improved HostBridge throughput; channels and containers provided new means to handle large blocks of data within HostBridge; HTTP and SOAP enhancements made it easier to issue outbound service requests from HostBridge scripts; and the additional transport layer security options gave HostBridge customers many new options.

However, for all its new features and capabilities, CICS TS is not a “full weapons platform” when it comes to CICS application integration. To illustrate this fact, let’s pose a question. Without modifying your existing applications, what kinds of Web services can you build using CICS TS? The answer is simple. A Web service that invokes a single COMMAREA program. That’s good, but that’s it!

CICS TS Web Services Assistant

The component of CICS TS that facilitates deployment of a single COMMAREA program as a Web service is the Web Services Assistant (WSA). WSA provides two utility programs: one that generates a WSDL from a language structure (e.g., a COBOL copybook) and one that generates a language structure from a WSDL. Additionally, these utility programs assist with the generation of the CICS artifacts/definitions that permit deployment of the Web service (e.g., the WSBIND file).

WSA is a great tool for those organizations (a) that have lots of COMMAREA programs and (b) whose individual COMMAREA programs fully implement the required business

service. Just remember, WSA does *not* allow you to build a Web service from multiple COMMAREA programs or from terminal-oriented applications. So how do you do that? Enter tools like Service Flow Feature and HostBridge.

Service Flow Feature

On November 22, 2005, IBM announced the CICS Service Flow Feature (SFF). The title of the announcement is important: “IBM CICS Service Flow Feature enables composition of CICS applications to create CICS business services.” Note two key concepts: “composition” and “business services” (we’ve already discussed “business services”; we’ll delve into “composition” momentarily). The title reflects SFF’s primary role as service composition, *without* dictating how the resulting service is deployed. To quote the announcement letter:

CICS Service Flow Feature, comprising a tooling component and a run-time component:

- Is a business service integration adapter for CICS applications.
- Enables integration developers to create CICS business services by composing a sequence of CICS application interactions. These can be integrated in Service-Oriented Architectures (SOA) or business process collaborations.
- Delivers a graphical integrated development environment for composing the CICS application interactions, a generation facility for creating a run-time application, and a run-time component that exploits CICS interfaces to support the service flow.

The driving idea behind Service Flow Feature is that of composing a “coarse grained” service from existing “fine grained” CICS applications (we’ll explore granularity later).

The tooling component of SFF is Service Flow Modeler (SFM); the run-time component is Service Flow Run-Time (SFR). I won’t say much about SFR. If you understand what SFM is and does, you can pretty much take SFR for granted (well, you can, but I can’t).

When IBM announced CICS TS v3, they also defined their commitment to Rational Developer for System z (RDz) as a strategic tool for CICS application development.⁵ Consistent with this, SFM layers on top of the facilities and services of RDz.

Service Flow Modeler

Here’s the IBM explanation of SFM: “The CICS Service Flow Modeler delivers the flow modeling and mapping capabilities required to transform the behavior of the application components to form a business service without changing the underlying implementation.” At first blush, it sounds like SFM is some sort of a really big magic wand – it can transform the behavior of an existing CICS application without having to

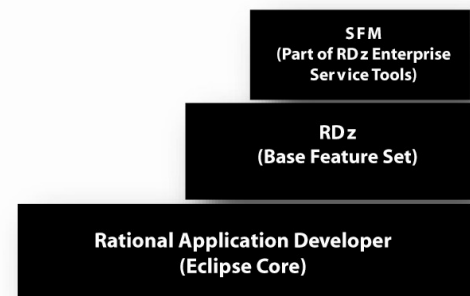
⁵ At the time CICS TS v3 was announced, RDz was known as WDz (WebSphere Developer for System z). Before that, WDz was known as WSED (WebSphere Enterprise Edition). And before that... well, you get the idea. It’s now called RDz.

change the actual application. Cool! Sign me up! I want two of those puppies!... Back to reality. SFM doesn't auto-magically transform anything in the sense that your existing CICS applications behave or operate any differently. The magic (and I'm not opposed to calling it that) is in orchestrating the execution of existing CICS applications according to a specified flow.

Repeat after me.... "The magic is in the flow." (Before it's all over, we're going to talk a lot about flows – and not just flows, but micro flows and macro flows.)

That's why another description of SFM sounds more correct: "a graphical modeling integrated development environment that enables the creation of CICS business services by composing a flow of CICS application interactions." But what does "integrated development environment" mean? It means that SFM layers on top of RDz. RDz, in turn, is built on top of Rational Developer (RD). And RD is IBM's proprietary superset of Eclipse. A diagram of the SFM environment would look like this...

But what happens after the flow has been "modeled"? Well, here's where reality sets in. If you are new to the world of modeling, you need to understand that there is usually another notion that goes hand-in-hand with it: code generation. Commensurate with this, once a CICS business service has been modeled, SFM generates a set of COBOL programs to implement the service flow. This is what IBM means when they say that SFM delivers "A generation capability that transforms the composed flow of CICS application interactions to form a run-time application, highly optimized for the CICS environment...." Let that all soak in for just a moment. It's not bad, just different.⁶



These COBOL programs will orchestrate invocation of your existing CICS applications to implement the functionality described by the model. The generated COBOL programs are then uploaded to the mainframe for compilation (under TSO) and testing (under CICS). This process (model, generate, upload, compile, test) is repeated during the development cycle or whenever changes are required. This approach also relies upon downloading various source code artifacts to the RDz environment (e.g., COBOL copybooks and BMS maps), but that's something of a distraction at this point.

HostBridge and CICS

As mentioned at the outset, my point is to show how CICS and HostBridge, working in concert, meet the broadest possible range of CICS integration requirements. With that in mind, it is fitting to spend just a few moments on CICS and HostBridge now.

⁶ Actually, in some cases it is bad. One weakness of the modeling/code generation approach is that the functionality implemented by the generated code is inherently constrained by the operations allowed via the modeling tool. Furthermore, generated code is difficult or impossible for developers to hand-edit. If they do hand-edit the code, they will lose the changes when they regenerate the code using the modeling tool. So it's never a good idea to change the generated code. Don't even think about it.

HostBridge started as a “clean sheet” design to solve a specific CICS integration problem. In late 1999, on a sunny day in San Francisco, at a meeting with Scott Glenn and I (HostBridge co-founders), Marshall Gordon (then Chief IT Architect at Delta Dental of California), stated, “We will no longer deploy any CICS integration solutions that rely on ‘screen scraping’ techniques.”

Marshall didn’t need to elaborate. Having worked in the industry for years, Scott and I understood. Screen scraping was the fundamental flaw in many integration solutions.

The essence of screen scraping is its dependence on row/column coordinates to specify the location of data on a screen and create a relationship between two programs. Imagine a CICS transaction that displays data as part of a screen image, and a distributed program that masquerades as a terminal so it can capture the screen image in memory and copy (or “scrape”) the data off it. By using row/column coordinates to reference the screen data, the distributed program creates a static, binding relationship between itself and the presentation logic output by CICS. If the CICS transaction is changed such that it alters the screen format in any way, then the binding will likely be incorrect and the distributed program will likely fail.

Because of this limitation, screen scraping led to what could best be described as “application rigor mortis,” whereby applications could not evolve over time because too many things broke when they did! Not to mention that the performance of such solutions, especially in high-volume environments, was usually very poor.

During our meeting, we discussed two exciting new approaches to solving these problems – the use of XML to exchange information between an existing CICS transaction and a distributed system (remember, XML was still big news in 1999) and a new “bridging” feature that IBM introduced in CICS Transaction Server v1.2 and formalized in CICS TS v1.3. Using this feature in conjunction with a supported API, a program could intercept CICS Basic Mapping Support (BMS) SEND and RECEIVE commands *before* the terminal data stream, with its presentation logic, was generated as output or expected as input.

The implication for integration of terminal-oriented CICS applications was huge. Interaction with CICS by distributed systems would no longer be rigidly dependent on screen geometry – i.e., screen-scraping.

Later that evening, Scott and I completed the first HostBridge block diagram. The initial version of HostBridge would be designed to exploit the Start Bridge interface within CICS TS v1.3 for the purpose of XML-enabling CICS BMS applications.⁷

⁷ **The HostBridge Patent:** On December 27, 2005, HostBridge was awarded US Patent No. 6,981,257 B2. In the wonderfully arcane language of “patent-speak,” the HostBridge patent describes: “A system, method and apparatus to facilitate the invocation of existing CICS BMS transactions and deliver the executed transaction output to a requesting application as a standardized XML document.” The broad objectives of the patent are to: allow enterprises to integrate CICS transactions into scalable e-business applications; facilitate communications between CICS and non-CICS platforms and applications without screen scraping; provide for XML-based communications between CICS transactions and client applications without CICS application, transaction, or system modification.

Guiding Principles

This brief story illustrates the principles that have guided HostBridge the company, HostBridge the product, and HostBridge the partner since the beginning:

- Start with a full understanding of what *real* customers do with *real* CICS applications
- Exploit the best of what CICS TS has to offer natively
- Assess the latest trends in Web-based application development
- Adhere to relevant integration standards (HTTP, XML, SOAP, REST, etc.)
- Retain our obsession with “high-fidelity” integration.

HostBridge Process Automation

Customers were pleased with the value proposition of HostBridge – XML-enabling CICS applications and data – and its technical merits. As more customers implemented the product, however, needs and issues continued to emerge, particularly involving orchestration of the “micro flows” intrinsic to CICS transactions and sequences of HTTP/XML requests and responses.

Terminal-oriented applications were designed for use by human operators. As a result, a single interaction with such an application rarely embodies the full scope of work that must be performed to accomplish a business process. The business process usually requires the execution of multiple transactions as part of a conversation or “pseudo-conversation” (a CICS technical concept). Thus, a detailed sequence of steps must usually be performed. These steps are often referred to as the “micro flows” necessary to accomplish a business service.

In the early days of HostBridge, customers would typically write a server-based process, such as a Java application running under WebSphere, to automate the exchange of a series of HTTP/XML requests and responses with HostBridge. However, when a distributed program was used to automate the micro flows required to implement a business service, there were certain ramifications. First, if the process was complicated, involving many steps, it would require numerous HTTP requests/responses, and the aggregate latency associated with execution of the overall business service could become excessive. Second, CPU consumption associated with the processing of individual HTTP requests/responses on the z/OS system could be high.

In response, we introduced a CICS-based Process Automation Engine as part of HostBridge version 4 in 2004. This component (initially referred to as HostBridge Extended) allows a customer to describe/codify a sequence of interactions using ECMAScript (a.k.a., JavaScript).⁸

⁸ The name “ECMAScript” was a compromise between the organizations involved in standardizing the language, especially Netscape and Microsoft. Brendan Eich, the creator of JavaScript, is on record as saying that “ECMAScript was always an unwanted trade name that sounds like a skin disease.”

High-Fidelity CICS Integration

Using the HostBridge Process Automation Engine, our customers are able to achieve the highest fidelity CICS integration. I realize that in the context of CICS integration, the phrase “high fidelity” may be new to some people, so a brief explanation is in order.

“High-fidelity,” when applied to video or audio, implies the reproduction of sights or sounds with minimal distortion and maximum resolution. The concepts of minimal distortion and maximum resolution are directly applicable to CICS integration as well.

Integration technologies can “reproduce” CICS applications with either a high degree of clarity and resolution – i.e., fidelity – or with lower resolution and more distortion. Screen scraping injects a high degree of distortion and/or loss of resolution with respect to originating CICS applications and integration. Other integration techniques can diminish fidelity as well.

High-fidelity CICS integration has been our sole focus since day one. Read on, and we’re confident you’ll understand what high fidelity integration is and how we got there.⁹

On to Composition

Since Service Flow Feature and HostBridge are all about “composition,” it’s time to turn to this key concept. The very word “composition” begs certain essential questions:

- What are we composing the service from?
- Why are we composing the service?
- Where are we deploying the service?
- How do we compose the service?
- Who composes the service?

These questions will serve as the lens through which we explore CICS application integration in general, along with SFF and HostBridge in particular. When all the dust settles, I hope to have laid the foundation for understanding why SFF and HostBridge are *very* complementary, and how to leverage their respective strengths...together.

⁹ You’ll find more on high-fidelity CICS integration in our aptly titled white paper, “High-Fidelity CICS Integration: HostBridge and IBM, A Brief History” – www.hostbridge.com/index.php/library/whitepapers.

3. What We Compose: Visual and Non-Visual CICS Applications

In music, we form a composition from notes that have a particular pitch and duration. (Get used to the music analogies; integration technology borrows heavily from the concepts and vocabulary of music.) The primary “notes” from which a CICS business service is composed are existing CICS applications. Just as musical notes have unique characteristics, so do CICS resources. Broadly speaking, existing CICS application resources fall into two broad categories. Various words or phrases can be used to describe these categories. I tend to use “visual” and “non-visual.”

“Visual” vs. “Non-Visual” CICS Applications

A “visual” application is one that expresses a presentation interface to an end-user at a terminal. You can also refer to a visual application as a “terminal-oriented” application. In contrast, “non-visual” applications do not interact with an end-user. Instead, they are designed to be invoked by another program. Non-visual programs are also referred to as “COMMAREA” programs because the input/output parameters are passed to/from the program using an area of storage referred to as the communication area, or COMMAREA.¹⁰

In describing existing CICS applications, the SFF announcement letter reflects this dichotomy:

Traditional CICS applications were designed to be accessed from a 3270 terminal. The applications exploit CICS-provided interfaces to generate and interpret the screen content and to manage the control and interaction with the 3270 terminal. Typically such applications were designed to deliver a sub-second response to a terminal user by exchanging small amounts of information in quick succession; in other words, to support a “fine grained” interaction pattern.

In some cases, well-structured CICS applications implement a separation of the 3270 presentation logic and the application business logic. They introduce the option of exposing a callable component interface typically implemented as a CICS COMMAREA [program]....”

The distinction between visual (terminal-oriented) and non-visual (COMMAREA) applications is huge. Why? Because from a composition and hence an integration perspective, they are radically different.

¹⁰ With the advent of CICS TS v3, IBM created the concept of “channels” and “containers” that can be used to pass data between programs. These architectures overcome the historic limitation that a CICS COMMAREA could be no more than 32,500 bytes. Question: if a LINKable CICS program expects to receive/return data via the channel/container architecture, should we still refer to it as a “COMMAREA program”? Technically, probably not. We should probably refer to it more generically as a “LINKable program.” However, the phrase “COMMAREA program” is so ingrained in the collective CICS consciousness (and synonymous with “LINKable program”) that I’ll continue to use it until someone objects or it causes too much confusion.

“Fine Grained” vs. “Coarse Grained” CICS Applications

One way to describe the difference between visual and non-visual CICS applications is in terms of their granularity.¹¹ I (along with IBM) frequently use the terms “fine grained” and “coarse grained” to describe the differences between them.

Terminal-oriented applications are often described as fine-grained because they are designed to deliver a sub-second response to a terminal user by executing a very specific processing step and exchanging small amounts of information in quick succession. By comparison, COMMAREA programs are usually considered more coarse-grained (or at least not fine-grained).

Theoretically, whether you want to consider a particular COMMAREA program as coarse- or fine-grained should depend on what it does. A COMMAREA program might be designed to yield extremely fast response time, perform a simple operation (e.g., validate a ZIP Code), and return a small amount of data (e.g., true or false). Perhaps these attributes should qualify such a program to be considered as fine-grained. However, the fact that the program has well-defined inputs and outputs, and does NOT interact with a terminal user, changes everything. And since a COMMAREA program operates by a far simpler (and more concrete) set of assumptions, it is far easier to incorporate into a CICS business service. So as a practical matter, I will consider all COMMAREA programs to be coarse-grained.

To summarize in terms of our music analogy, there are basically two types of “notes” in a CICS service composition: terminal-oriented applications and COMMAREA programs. Due to their granularity, composing a series of interactions with a terminal-oriented application into a CICS business service is *fundamentally different and more difficult* than composing a CICS business service out of COMMAREA applications.

But there’s a catch: *terminal-oriented applications are more common.*

¹¹ Confusion Alert: We are going to use this notion of granularity two ways: first to describe the nature of the underlying CICS application assets; second to describe the nature of the composed service.

4. Why We Compose: Business Realities

Let's consider a fundamental question. Whether we are composing musical notes into a melody or existing CICS applications into a business service, what is the purpose? Why are we doing it? Simple question. Important answer.

Every business software application (past, present, and future) is shaped by a unique set of business processes, value propositions, and technology constraints. Organizations have always tried to make sure that their investments in business application software are as timeless as possible. However, it's impossible to avoid the fact that each such investment reflects a unique set of business processes, value propositions, and technology constraints that are good, desirable, or true at a particular point in time. Organizations that have developed business systems around CICS are well acquainted with this dynamic. And it makes perfect sense for them to leverage their existing CICS applications as their organization and business requirements evolve.

And make no mistake. Business processes do change and evolve. A few years ago you may have understood a certain business process as being comprised of steps 1, 2, and 3. Today, you may understand it as steps 1 and 2, optionally steps 3 and 4, but always followed by step 5. If you have already developed application software to implement steps 1, 2, and 3, then it makes perfect sense to try and reuse it as processes evolve.

The IBM SFF announcement letter puts it like this:

Reuse of existing application components can significantly benefit new solution implementation by reducing cost and risk, and by speeding delivery. These benefits are further magnified if the implementations of the reused components are encapsulated behind an open-standard interface such as that offered by Web Services, since component reuse will not require platform-specific implementation knowledge.

Simply stated, you compose existing CICS applications into a business service so you can do two things: (a) *reuse* them as part of a service and (b) *expose* the resulting service.

So the themes of application reuse, service composition, and system integration go hand in hand. But – and this is important – the whole value proposition can succeed or fail based on the efficiency of how the distributed system interacts with the business service. I like the way the SFF announcement letter reflects this:

In order to implement an optimal method of integration between modern enterprise solutions and an existing enterprise system, the number of interactions between the systems should be minimized. This reduces the volume of request data being exchanged over the transport and the cost of data content processing in each system. This suggests a “coarse grained” interaction pattern across the external system boundary complemented by highly optimized processing of the request payload within the bounds of each system.

As you can see, the notion of granularity enters this discussion as well.

Fine-Grained vs. Coarse-Grained Business Services

We have already introduced the terms “fine grained” and “coarse grained” to describe the CICS application resources from which we are composing the business service. We can also apply coarse-vs.-fine distinctions to the business service we compose. However, we approach this discussion from a different angle.

When discussing CICS *applications*, and assessing their characteristics, we have to start with reality – *what is* as opposed to *what should be*. Terminal-oriented applications are fine-grained. They are what they are. Period. We can't change their nature by wishing it so. Non-visual applications are coarser-grained. Period.

However, a discussion of CICS *business services* proceeds from a different place – *what should be* as opposed to *what is*. Since we are creating something new, albeit from things that already exist, we have choices. And one of the most important choices you will make regards the granularity of the business service. But just because we have choices doesn't mean that all of the options are equally good. There will be times when a fine-grained business service is required. However, when it comes to the granularity of a business service, coarse is better. Why? Because the granularity of the business service dictates the pattern of interaction between distributed systems. And on balance, it is always better to minimize the number of interactions between distributed systems. In other words, *keep the business services high-level*.

I want to use this discussion about the granularity of CICS applications and the granularity of a CICS business service composed from those applications to make a point regarding CICS application integration products in general. I may be guilty of stating the obvious, but here goes: *a critical role of any CICS integration product is to transcend the difference in granularity between CICS applications and the business services composed from them*.

If the CICS integration product cannot fully embrace the fine-grained nature of most existing CICS applications, *there will be problems!* For example, if the CICS integration product makes assumptions about or imposes limitations on the way visual transactions behave in certain situations, then the business benefit of using such a product *will* be compromised. Likewise, if the CICS integration product does not permit wide freedom and choice as to how the CICS business service is exposed to the outside world, then the business benefit will be compromised.

Why Again?

That was a fairly long digression, so let's regroup. Why are we worried about composing existing CICS applications into a business service? *Because business changes, and you don't want to rewrite your applications or services every time it does !*

5. Where We Compose: Inside CICS

When it comes to composing business or Web services, *Where* is a pretty broad concept. There are usually lots of moving parts, and we can analyze many different attributes. I will focus primarily on two: (a) where the Web service boundary exists and (b) where the business service composition is executed.

To illustrate my discussion of the most utilized configuration options, I'm going to use a series of diagrams like this one:

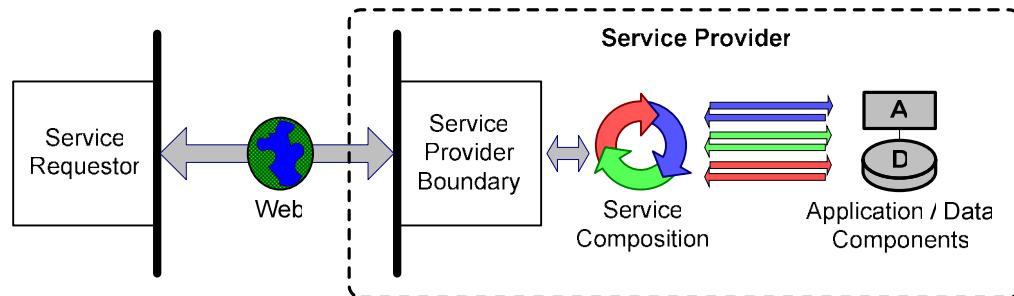


Figure 1. Basic Diagram Showing Service Requestor and Service Provider.

All the diagrams include the same key components:

- Service Requestor – The distributed system or program that expresses a request using standard Web service methods (e.g., SOAP).
- Service Provider – The software components, collectively, that respond to the request, also using standard Web service methods.

On the service provider side, the action “under the covers” can be broken out in a number of different ways. Because we are discussing the composition of services from existing CICS applications and data, we will highlight three key components:

- Service Provider Boundary – A program that receives and processes the Web service request. The boundary function passes control to the program that implements the business service. In the context of our discussion, we will assume that this is a composed service.
- Service Composition – A program that contains the logic describing how the existing CICS applications and data are to be executed or accessed. The service composition *is* the business service.
- Application/Data Components – The existing CICS transactions, programs, and/or data services that will be executed or accessed as part of the service composition.

Since we are focused on CICS integration, we will assume that the application/data components always reside in a CICS TS region on the System z mainframe (duh!). However, the service provider boundary and the service composition can be deployed in various ways, or, more to the point, on different platforms: either on a server or on

the System z mainframe.¹² So in my diagrams, platforms on the service provider side may be separated as depicted here:¹³

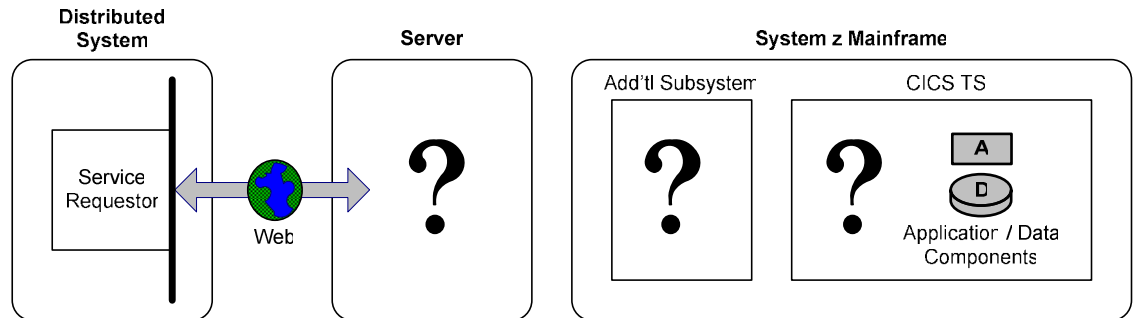


Figure 2. Platforms on the Service Provider Side

The question we want to focus on is this: on the service provider side, *what works best where?*

Relative Position of the Service Boundary and Service Composition

Our objective in this section is to examine where the Web service boundary exists and where the business service composition is executed. If you do the math, you can figure out that there are quite a few alternatives. However, there are four common approaches. We will discuss each turn. Note that they all have a common characteristic: the service composition function is always *inside* the service provider boundary. *That's because doing otherwise is a bad idea.* The first three approaches also have something else in common: the Web service boundary exists on the same platform where the business service composition is executed. The fourth approach highlights a case in which these two functions are split across platforms.

Back in the old days (before the introduction of the HostBridge scripting engine, if I may say so), customers would typically write a server-based process to automate a series of requests to CICS. However, when a distributed program is used to automate the micro flows required to implement a CICS business service, there are certain ramifications. If the composition involves many steps, numerous requests/responses must be exchanged between the distributed system and the mainframe. As a result, the response time associated with execution of the business service can become excessive. And if the distributed program and the mainframe communicate using a Web services approach, resource consumption on both the server and mainframe can be high.

¹² By "System z mainframe" I mean the zOS operating system environment running on a z Series mainframe. By "server" I mean an execution platform that is distinct from the zOS environment running on the z Series mainframe. By this definition, WebSphere running under the Linux operating system environment on a System z mainframe could be a server. I'm not going to worry about these distinctions further because we have enough details to contemplate already.

¹³ Note also that these diagrams assume that the service requestor exists on a "distributed system." This would usually connote a physically distinct hardware/software platform of some ilk. While this is typical, it does not have to be the case. For example, the "distributed system" could be WebSphere running on zOS or zLinux. In the modern world of IBM System z, zOS, zLinux, and virtualization, what happens where can be rather fluid. Please consider this a diagrammatic convenience rather than a limiting assumption.

The following paragraph from IBM’s announcement letter regarding Service Flow Feature (SFF) captures this precise thought very well:

In order to implement an optimal method of integration between modern enterprise solutions and an existing enterprise system, the number of interactions between the systems should be minimized. This reduces the volume of request data being exchanged over the transport and the cost of data content processing in each system.

Enough said. It’s just a bad idea to automate fine-grained transactions and programs across a Web services boundary. *You want this type of orchestration/composition activity to be as close to the underlying components as possible.*¹⁴

Now that you know why the service composition will always be behind the service boundary in the following diagrams, let’s get on with it.

Option 1: Service Provider Boundary and Service Composition Outside the Mainframe

This configuration relies on a middle tier server to express the Web service boundary to the outside world and to execute the service composition.

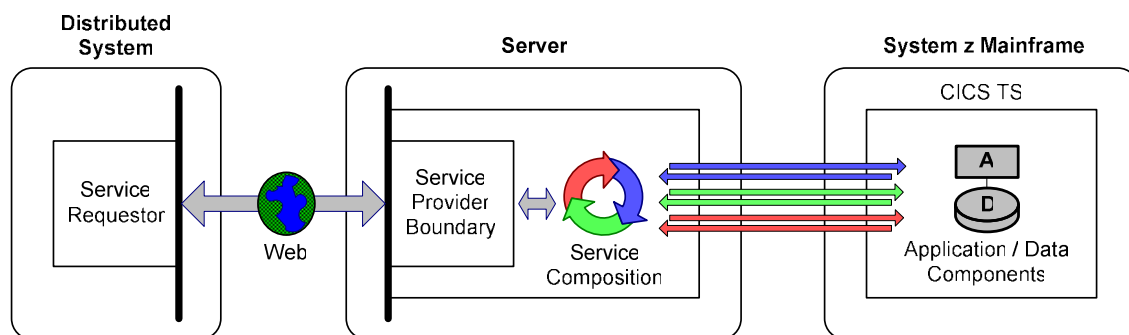


Figure 3. Service Provider Boundary and Service Composition Outside the Mainframe.

This model was popular during the early days of Web services, no doubt because Web services capabilities were available on server-based platforms before they were on the mainframe. And if you combined a server-based Web services implementation with a traditional server-based screen scraping product – presto! – you got something that looked like a Web services gateway for the mainframe.

Fortunately, those days are mostly over, although this configuration sometimes still appeals to organizations that, for whatever reason, have issued edicts like “thou shalt not put any new software on the mainframe.” They may think such a configuration saves them money, but in my experience, it doesn’t. In fact, it’s often counterproductive.

¹⁴ I also want to clarify something I’m NOT saying. While orchestrating *micro* flows across a Web services boundary is a bad idea, orchestrating *macro* flows across a Web services boundary is normal and to be expected. In fact, a whole category of orchestration products is growing up, centered on BPEL (Business Process Execution Language), that let you compose high-level Web services from lower-level Web services.

The most egregious implication of this configuration is that each interaction with a CICS application/data component requires a full travel on the network that connects the server and the mainframe. (Groan.)

As you may have surmised from my comments, this approach has the following attributes, all negative:

- Very high response time due to request/response latency
- Potentially high CPU utilization on the mainframe, depending on how the server and mainframe exchange requests/responses
- Integration with CICS terminal-oriented applications is usually accomplished via screen-scraping (I've already waxed eloquent about that nastiness)
- Integration with COMMAREA applications is usually limited to the exchange of 32 KB of data per interaction
- Limited or, more likely, no access to CICS managed resources.

Not a pretty picture.

Option 2: Service Provider Boundary and Service Composition Inside the Mainframe, but Outside CICS

This second configuration jettisons the middle tier server but replaces it with an additional, non-CICS mainframe subsystem. However, this mainframe subsystem is responsible for the same tasks as the middle tier server in Option 1 – express the Web service boundary to the outside world and execute the service composition.

As you consider this option, keep one thing in mind: software vendors who promote this approach often have a big investment in code that was developed *before* CICS TS was introduced. As a result, some of these products turn a blind eye to the capabilities that are now built into CICS TS.

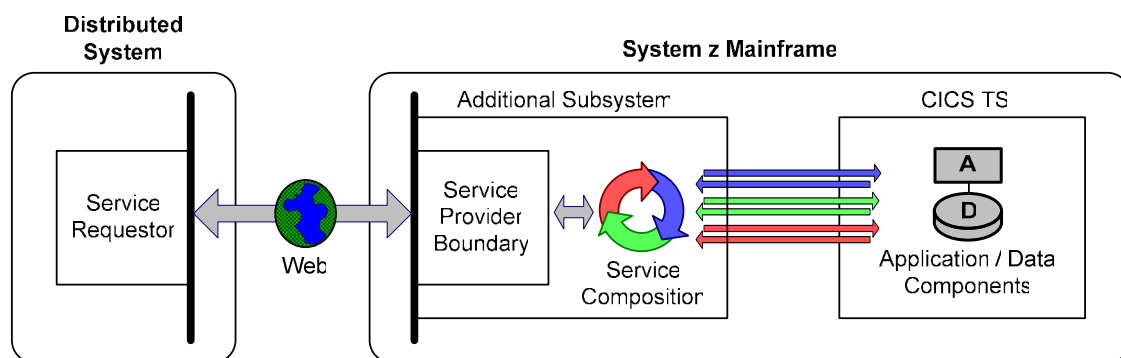


Figure 4. Service Provider Boundary and Service Composition Inside Mainframe, Outside CICS.

This approach has the following attributes:

- Lower response time because the service composition and application/data components are on the mainframe
- CPU consumption on the mainframe will depend on the nature of the service composition and the operational characteristics of the additional System z subsystem.
- Integration with CICS terminal-oriented applications is frequently still accomplished via screen-scraping (can we not rid ourselves of this infernal scourge?!)¹⁵
- Integration with COMMAREA applications is usually limited to the exchange of 32 KB of data per interaction
- Capabilities built into CICS TS are usually *not* exploited (as far as I'm concerned, you're paying for them, so you might as well use them)
- No integration with the CICS TS operational environment (e.g., security)
- Limited or no access to CICS managed resources

To me, this approach is better, but it still has serious issues that can compromise the overall fidelity of the integration solution. It also begs an important question: if your objective is to expose CICS applications/data as a Web service, why would you ignore the capabilities included in CICS TS and implement the Web services boundary outside of the CICS TS environment? Why would you relegate CICS TS to be anything other than a first-class Web service end point?

Option 3: Service Provider Boundary and Service Composition Inside CICS TS

Our third configuration jettisons the non-CICS mainframe subsystem and exploits the capabilities of CICS TS v3. Now we're talking! CICS TS expresses the Web service boundary to the outside world. Furthermore, software running inside of CICS describes and executes the service composition. By collapsing everything within CICS TS, we can achieve optimal performance and maximum sensitivity to the application/data components.

¹⁵ Some have attempted to exercise the CICS Link Bridge interface across an EXCI boundary. Others implement a Link Bridge "proxy" within CICS and use cross-memory services to pass data between address spaces. In my experience, both techniques yield lower fidelity solutions because important information about the context of the CICS application is either ignored or unavailable.

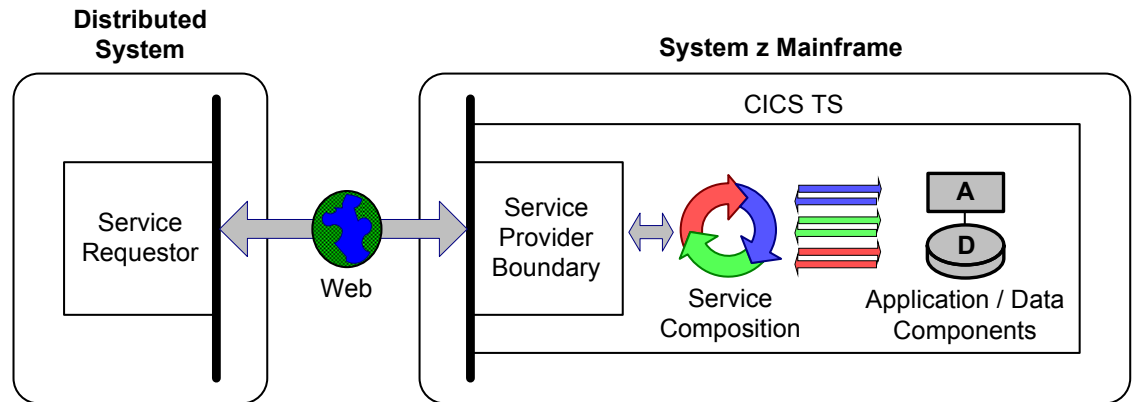


Figure 5. Service Provider Boundary and Service Composition Inside CICS TS.

This approach has the following attributes:

- Lowest possible response time because the service composition and application/data components are combined under the control of a single subsystem on the mainframe
- CPU consumption on the mainframe still depends on the nature of the service composition, but we have eliminated the overhead of an additional subsystem, and we can exploit the efficiencies of the CICS environment.
- Integration with most CICS terminal-oriented applications can be accomplished *without* screen-scraping (finally!)
- Integration with COMMAREA applications can exploit CICS TS capabilities and is not limited to the exchange of 32 KB of data per interaction
- Complete exploitation of CICS TS capabilities and full integration within the CICS TS operational environment (e.g., security)
- Full access to CICS managed resources

Perhaps it's no surprise, but to me this is the preferred approach. By a long shot!

At this point you may be tempted to think: "Of course he likes this approach because HostBridge is a service composition tool that runs inside of CICS TS." Actually, that raises an interesting question. Which came first? Did we design HostBridge and then declare that we liked this approach? Or did we decide that this approach was the best and then design HostBridge to exploit it? Allow me to shout out the answer loud and clear: **we decided this approach was best and designed HostBridge to exploit it!**

Please remember, HostBridge began life as a clean sheet design in a CICS Transaction Server world. We built HostBridge from the ground up because it made perfect sense to us to exploit the capabilities that IBM was adding to CICS TS. We hope it makes the most sense to you too.

Option 4: Service Provider Boundary Outside the Mainframe and Service Composition Inside CICS TS

This fourth configuration is a hybrid, and there are some real-world situations in which it makes good sense.

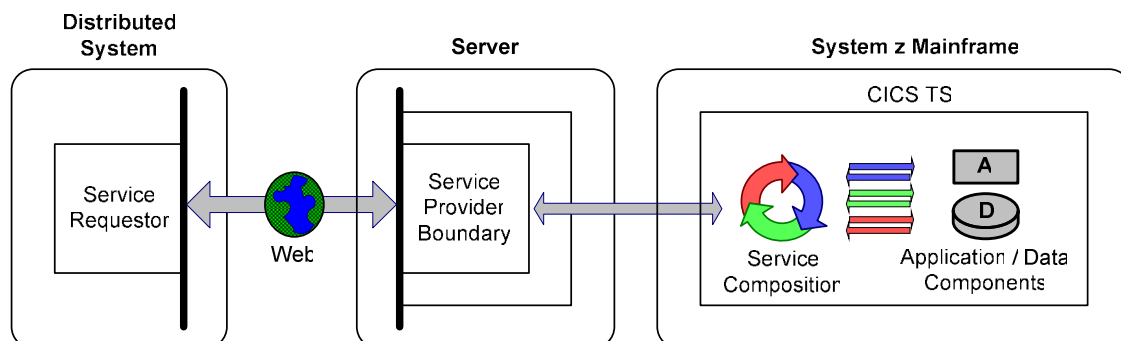


Figure 6. Service Provider Boundary Outside Mainframe, Service Composition Inside CICS TS.

For the reasons already cited, this configuration benefits from executing the service composition on the mainframe and within the CICS TS environment. But this leads us to another important fact:

Where the service composition runs is more important than where the service provider boundary exists.

Imagine that the server is WebSphere and that a Java application invokes a CICS-based service (hopefully implemented using HostBridge). It might be very inefficient to use a formal Web services boundary between the distributed system and the server, and then implement another formal Web services boundary between WebSphere and CICS TS.¹⁶ Another scenario in which this configuration surfaces is when an Enterprise Service Bus (ESB) or integration appliance (e.g., Cast Iron Systems) is used.

The point here is pretty simple: in many real-world situations it's very convenient (and in some cases more efficient) to implement the service provider boundary outside the zOS environment.

The Point of Where

If you've stayed with me through this discussion, commend yourself. Your reward is that you now have a good tool to analyze almost every conceivable approach, solution, or product from the *Where* perspective. While I spared us the pain of wading through *all* the possible combinations and permutations, we have covered the most common. I do hope you come away from this discussion with one settled conclusion:

To achieve the highest performance and ensure the highest fidelity to your CICS applications, it is critical that the service composition execute as close to the CICS application context as possible.

¹⁶ In high-volume (and "high-sanity") environments, there are only so many formal Web services boundaries that one may wish to implement!

6. How We Compose: Modeling *and* Scripting

In theory, the *how* of CICS services composition is supposed to be quite simple. Here's the pat answer: "use modeling to compose applications into business services, exposing them as Web services as part of an SOA – or at least making sure they plug-and-play with our ESB." If only real life in a CICS environment were as simple as stringing buzz words together....

To promote clearer, more comprehensive understanding, certain terms and concepts need to be defined at the outset.

First and foremost is "modeling" – speaking of which, brace yourself. I'm going to use the "M word" a lot. It's not going to be pretty, and you're going to hear me say blasphemous things like... modeling is NOT good for CERTAIN purposes. *WHAT?!* *Modeling isn't ALWAYS good? Say it ain't so! I thought modeling was one of our technological saviors; it's supposed to achieve an optimal division of labor and reduce our reliance on all those pesky programmers!* Yeah, right. Oh, and if that's not enough, you'll hear me imply that the "modeling" that has crept into many CICS integration tools is not really modeling at all. And it's counter-productive. *ARGHH!!!*

OK. Strap yourself in and let's get started.

Composition Tools

We begin with another music analogy. Music has many different elements, such as melody, rhythm, and harmony. Creating music involves arranging notes into a melody. Rhythm is the arrangement of these notes in time. Each note has its own pitch and duration, and harmony has to do with relationships between pitches. Since there is a high affinity between technology professionals and musical aptitude, I'll not belabor the point. Building a new CICS business service from existing CICS applications and data is very much like the process of composing music.¹⁷

So whether you are a music composer or CICS business service developer, here's the BIG question: exactly HOW do you author your composition? How do you go about getting the "music" you hear in your head codified such that your composition can be played over and over. What are the most appropriate tools for the task at hand?

The tools available to a modern music composer are much different from what Beethoven, Bach, or Mozart had to work with. Today, you can sit down at an electronic keyboard connected to a computer, start playing, and software will record your keystrokes and arrange them (as best it can) into a rough musical score. In essence, you use the keyboard to "model" the musical score. Then, in the hands of a person who is both musically talented *and* proficient with the software, the modeled composition can

¹⁷ Just so you've heard me say it, this music analogy does break down at points. For example, while improvisational skills are highly esteemed among musical performers and aficionados, improvisation is not something one wishes to encounter in a business service. A business service must work the same way every time. And when CICS terminal-oriented transactions are the foundation of the business service, you need to make sure that your tooling gives you complete control – every time.

be edited and refined. These tools are very powerful for composing a new musical work. However, let's consider a different use case.

I took piano lessons for two years as a boy, and I can still play the "Battle Hymn of the Republic" – sort of. Does my ability to hack out a rough approximation of that song mean I could use a music composition system to recreate the majesty of William Steffe's tune and worthy of Julia Ward Howe's words? I think not! I would make so many mistakes in substance or style that I'd end up spending way too much time editing my original composition. And really, this would be a laughable use of the technology. I'd be attempting to use a musical composition tool, essentially, to reverse engineer the way a particular song should sound. When simply playing – or attempting to play – my intention is not to model a new composition, but to "operate" something already in existence and very well defined.¹⁸

What Is Modeling?

So let's talk about modeling. And specifically let's talk about the way this word has crept into the context of building CICS business services.

In the software world, the term "modeling" implies different things to different people. But it does have a certain sex appeal. To many it implies "easy to use" or "no programming." Universally it seems to imply a level of abstraction that is higher than the ultimate implementation.

In the field of software engineering, the word "modeling" usually implies some relationship to Unified Modeling Language (UML) or other formal, top-down software modeling approaches. UML is a general-purpose modeling language that includes a standardized graphical notation used to create an abstract model of a system, referred to as a UML model. Modeling, in the software engineering sense, always implies a level of abstraction higher than the ultimate implementation. I'm not a big fan of UML mostly because I fell asleep just trying to read an introductory primer (if you think modeling implies "ease of use" go buy a UML book).¹⁹ However, modeling in the UML sense at least hangs together conceptually and linguistically.

Modeling Gone Mad

Now switch gears. Let's imagine I use a tool to record a sequence of detailed interactions with a CICS visual application. And let's assume that when I press the STOP

¹⁸ For fun, let's push the analogy and argument a bit further. The "Battle Hymn of the Republic" has nothing of shorter duration than a sixteenth note. The song moves along very methodically and is simple to learn, play, or sing. It's easily accessible. In the world of CICS applications, what correlates most closely to such a song? Non-visual (COMMAREA) applications. With well defined inputs, outputs, and operational characteristics, they are easily accessible. And to what song would I correlate the operational characteristics of visual (terminal-oriented) applications? Perhaps "Flight of the Bumble Bee." If you think of each note representing a single interaction with a fine-grained terminal-oriented application, and the entire song representing a meaningful unit of work, you intuitively grasp the challenges associated with integrating visual applications into a CICS business service. If you don't get it all right, the whole thing breaks down.

¹⁹ One of the most enjoyable, insightful articles on modeling was written by Bertrand Meyer in 1997: "UML: The Positive Spin" was published in the special UML issue of *American Programmer*. It can still be found on the Eiffel Software Web site. Warning: the article is satirical and may offend UML devotees.

icon, the tool generates a visual/graphical notation of those steps on the screen. Cool!... Perhaps. Whatever you think of this approach, it's certainly not "top down" modeling a la UML. At best it's "bottom up" modeling. (Sorry, bad image.)

In one sense, of course, I *did* model something. But what? Did I model a business process? No. Did I model the application? No. All I did was record one way a human being can interact with a particular application, and it's only one particular use case. So what did I model? *I modeled me!* I modeled *my* behavior... this one specific time! This is no different than me using music composition software to record my attempt at playing the "Battle Hymn of the Republic" and act as my scribe.

The "model" I recorded does not represent all the pathways through the application or all the functions it can perform. There is no assurance that the "model" is correct or complete. In fact, this "model" probably excluded the most important part: how the application behaves in various error conditions. If you want your model to account for other pathways through the application, or the error conditions that may arise, you will have to model those steps separately and somehow knit all these models together. In practice, what happens is that you immediately begin to hand edit the model to account for alternate pathways and error cases. And if you do a thorough job, you end up pulling your hair out and soon realize the level of abstraction of the model is virtually identical to the ultimate implementation.²⁰ The steps you have codified using the tool are no more or less abstract than the code that will be generated to implement them.

To me, drawing a flowchart of how to operate a CICS terminal-oriented application is not really modeling. Yes, you are using graphical symbols to represent activities; yes, you are wiring them together; yes, code will be generated to do whatever. But give me a break. This is modeling in the same sense that some of us old guys used to use plastic templates and mechanical pencils to draw flowcharts of COBOL programs – AFTER we wrote the code! And in those flowcharts, as in most models, you never account for all the stuff the application really does or how it can break.

At best, your "modeling" efforts have devolved into visual programming.²¹ But there's a problem. Most modeling tools are rather poor visual programming tools! Besides, weren't we trying to get away from programming by using the modeling tool to begin with? Vendors of such tools embrace the metaphors and vocabulary of modeling (in the software architecture sense) in hopes that the (perceived) value proposition of that approach will splash on them. They hope you won't notice that this is only modeling in the most limited sense of the word, and that you won't think about the implications.

Picking the Right Tool for the Job

So here's the point. *When you compose a new CICS business service from existing CICS applications, two very different activities are taking place.* First, you create something

²⁰ Ever tried to tie your shoe strings while wearing mittens? Yeah, that's the feeling.

²¹ Per Wikipedia: "A Visual programming language (VPL) is any programming language that lets users specify programs by manipulating program elements graphically rather than by specifying them textually. A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols. Most VPLs are based on the idea of ... boxes or circles or bubbles, treated as screen objects, connected by arrows, lines or arcs." (Wikipedia: Visual Programming Language, accessed 7/11/08.)

new, the business service, by defining the high-level building blocks of the service and how it will be exposed to the outside world. This activity lends itself nicely to modeling. Modeling is an ideal approach to abstractly describing something new and allowing the tool to generate the code required to implement it. The second activity is very different: you have to specify the implementation of those building blocks. This boils down to codifying – i.e., programming – how your existing CICS applications must be operated to implement the desired functionality. This second activity is very different from the first and does not play to the strengths of modeling tools.

Let's reference what we know about the two different types of CICS applications. CICS COMMAREA programs (with their well-defined inputs, outputs, and operational characteristics) may, in many situations, be well-suited to a modeling approach. However, as the operational level of these programs becomes more complex or detailed, modeling tools start to get in the way.

CICS terminal-oriented applications are another story. Remember, terminal-oriented applications evolved over many years with an eye toward solid business value, optimal execution efficiency, and acceptable productivity of reasonably well-trained end users. These applications often defy the black-and-white, square-edged, if-then-else mentality of modeling tools. They were not written to be sympathetic to, or viewed through the lens of, a modeling tool. When modeling tools are used to describe the operation of such applications, the tool itself usually becomes the limiting factor as to what can be accomplished.²²

A modeling tool is perfect for describing the macro-level flows of a business service, including the top-level flow of the service, top-level decision points (conditional processing and looping constructs), and the invocation of coarse-grained CICS programs. However, using a modeling tool to describe the operation of fine-grained CICS terminal-oriented transactions is like trying to use a hammer to turn a screw.

Describing and controlling micro flows requires a different approach. Micro flows include the low-level flows of the service (the specific steps required to operate or interact with a terminal-oriented application) as well as the low-level data gathering, conditional processing, and looping constructs that are also required. In such cases, a simpler and more direct approach is called for. Otherwise, the tool will become a constraint.

You can't get away from the fact that CICS terminal-oriented applications are rather unique programmatic "things." They operate in certain odd ways, though for very good reasons. And they have all the operational consistency of the human being who authored them. No degree of "modeling" will change these facts.

So the question we are left to answer is *how to embrace CICS terminal-oriented applications most effectively and efficiently.*

²² In the interest of technical accuracy, I'll qualify this statement, but only a bit. If all you're going to do is invoke one or two visual transactions and extract a few data items off each screen (the sort of scenario vendors like to use during a sales demo), then you can probably "model" that activity. However, if you go beyond such simple scenarios (as most real-world applications do), you have a problem.

The Solution: Modeling *and* Scripting

If modeling is not the right way to describe the operation of a CICS terminal-oriented application, then what is? Take a deep breath. The answer is... SCRIPTING! But not just scripting per se; it is scripting used to intelligently operate a series of terminal-oriented transactions and access data within the context of a *modeled* CICS business service.

This should come as no surprise to anyone familiar with HostBridge and its Process Automation Engine, which allows integration scripts to be developed and executed within the CICS environment. A HostBridge integration script can access any CICS visual transaction, non-visual program, or data source – and many non-CICS resources. As a result, system architects, process designers, and application developers can create services that aggregate and orchestrate existing fine-grained transactions and data sources.

The reality is that no technology – “modeling” or other – can overcome one simple fact: a human being today is trying to express how to operate a series of programs written by another human being yesterday, or perhaps a very long time ago. So the question is *how* should you approach doing this.

Scripts are very concise. They codify the well-defined steps that a human operator goes through to operate a terminal-oriented transaction (e.g., enter value in field “x,” press Enter key, get value from field “y”). Scripts allow a human to break down a problem in a way that makes sense.²³ Each step is visible and easily comprehended as part of the overall process. On the other hand, modeling diagrams with any degree of real-world detail or complexity tend to become large and unwieldy. The only way to deal with this is to create embedded models (models within models). But when you are working on the lowest level model, it is almost impossible to comprehend how it corresponds and interacts with a higher level model. As a result, the overall process becomes obscured.

Both modeling and scripting have their place in a high-volume, high-fidelity CICS integration architecture. But when it comes to intelligently operating a series of terminal-oriented transactions, scripting wins hands down.²⁴

²³ I suppose you could even say that the script “models” the human approach. But that would beg the question: “Is scripting a better modeling tool?” If I get started down that road, this will never end.

²⁴ One of my initial reviewers (a prominent industry analyst) observed: “This a L-O-N-G way of saying the “how” of composition is scripting micro-flows and NOT modeling.” He’s right; it was long. But my hope and aim are to provide needed enlightenment.

7. Who Composes

I really like this topic. It allows us to slip the surly bonds of earth and ascend into alternate realities. It also allows me to slip into a playful, though not cynical, mode (please bear this in mind as you continue reading).

Modeling tools, like SFM, enable a mythical breed of IT professional, known as “enterprise integration architects,” to be highly productive. These people are non-programmers who understand:

- How to use the Eclipse IDE environment
- How to use the Rational and RDz extensions to the Eclipse IDE environment
- How to use the Eclipse (EMF) and SFM modeling paradigm
- How business services are described within SFM
- How CICS COMMAREA programs work
- How COBOL copybooks are used to describe the inputs/outputs for CICS COMMAREA programs
- How CICS terminal-oriented applications work (it varies between applications)
- How BMS maps are used to describe inputs/outputs for CICS BMS terminal-oriented transactions
- How 3270 datastreams are used to describe inputs/outputs for non-BMS CICS terminal-oriented transactions
- How to code ESQL, required to implement conditional logic within a service flow
- How to generate the COBOL code, and CICS resource definitions, used to implement the service flow
- How to compile COBOL programs under zOS and interpret job output
- How, potentially, to debug the COBOL code generated by SFM
- How to deploy services, programs, and resources into CICS
- How to test the resulting services
- How to diagnose CICS and SFF SOAP faults
- How to evaluate CICS AUX traces to diagnose errors (in some cases that’s the only way).

Whoa! That’s sounds like the skill set of a TEAM, not a PERSON! But let’s assume you DO have real human beings in your organization with such a dazzling breadth of skills. Here’s the REAL question. Do they have the intestinal fortitude to repeat the process

over and over to incrementally develop, debug, and deploy CICS business services? HA! GOTCHA!

All kidding aside, the above list of skills is daunting. However, tools like Eclipse, RDz, and SFM can be learned. It just takes time, and, hopefully, learning them represents a one-time investment. (I say “hopefully” because, in my experience, unless you use these tools frequently you find yourself having to relearn them.)

What are we to make of the other skills/tasks on the list? In my opinion, there are a few items that are far more critical than others. They represent the critical factors that can compromise the whole vision of building business services out of existing CICS applications. These critical factors are:

- Incorporating terminal-oriented applications within a service flow
- Coding ESQL to implement “modeled” behavior
- Performing iterative/incremental development of service flows that include terminal-oriented applications
- Debugging an errant service flow that includes terminal-oriented applications.

Do you see a thread of consistency here? The most problematic aspects of using SFM to compose CICS business services involve our good friend the CICS terminal-oriented application. I won’t take time now to expound on these too much because we’re getting very close to beating a dead horse. But I do want to make a comment about the need to code ESQL to implement modeled behavior. This is a complete surprise to most people. It shouldn’t be. In only the rarest of situations does modeling ever eliminate the need to author code.

Modeling Means Writing Code?

A flow model within SFM is composed using the following actions or “nodes”: Receive, Reply, Invoke, Throw, Assign, Switch, and While.

When you create a flow model involving terminal-oriented transactions there are always – and I do mean *always* – lots of conditions and checks you must include. In an SFM flow, such operations are represented using Switch or While nodes. Let’s say you drag a Switch node into a flow model. What exactly does the Switch node do? Uhhh... You mean it can’t read my mind? Nope... sorry. The answer is that the Switch node will do *whatever you tell it to do* by virtue of *hand coding a series of ESQL statements*.²⁵ Yes, you read that right. I can almost hear you say, “But wait a minute. I thought modeling meant I didn’t have to write code! And what the heck is ESQL anyway?” Ah yes... your trip down the Alice-in-Wonderland rabbit hole continues. And by the way, those ESQL statements you wrote will ultimately be used to generate COBOL statements which you will then get to debug. GULP!

²⁵ In some cases you can answer questions posed by a wizard to define properties of the node. However, this also results in the generation of ESQL statements (which results in the generation of COBOL code)!

Authoring a service flow that includes lots of Switch and While nodes is tedious enough. But you also end up with something that is terribly difficult to test, change, or support. Furthermore, it's almost impossible for anyone, other than the original author, to understand what it does. I take that back.... It's often difficult for the original author of a service flow to figure out after lunch what they did before lunch. Perhaps it's just me, but clawing through a multi-level flow model, and exploring each and every group of ESQL statements to determine what's going on, is not my idea of productivity.

Summary

Let me say once more, I like SFM. Otherwise I would not have spent so much time and money making sure HostBridge works with it. However, SFM presents you with a serious temptation: to push "modeling" to the inappropriate extreme of orchestrating micro-flows. Theoretically, the modeling approach should yield the magic combination of simplicity and productivity. In practice, this doesn't happen too often. In fact, the exact opposite may occur. It is quite possible for people using such tools to become *less* productive – and paranoid about touching or changing anything after they get it working.²⁶

Bottom Line: When the tool becomes the constraint, progress ceases.

²⁶ Much like the "rigor mortis" that integration techniques such as screen scraping produced.

8. SFM and HostBridge – Together at Last

The key objective of this paper is to describe how HostBridge *complements* CICS Transaction Server, Service Flow Feature, and Service Flow Modeler. Admittedly, I took a circuitous path to this conclusion, but with good reason. If you are thinking about composing CICS services, it is critically important to gain a balanced perspective on the relative merits of modeling and scripting and to understand where each fits in the composition process.

SFM implements a modeling approach. This is fine when you are focused on the high-level flow of the service – the macro flows – and on describing how the service should be exposed to the outside world. But SFM can push the modeling orientation too far – with a predictable result.²⁷ SFM can devolve into an overly complex visual programming tool that restricts functionality. But – and this is a BIG but – just because SFM pushes the modeling approach too far doesn't mean it's a bad tool, or that it should be avoided. All it means is that SFM has a split personality: part service modeling/deployment tool, and part visual programming tool. And when it comes to composing serious CICS-based services from terminal-oriented transactions, I think one side of SFM's personality has a lot to offer, and the other side should be avoided.

To me there is no debate. Using SFM to describe the high-level flow of a business service and orchestrate coarse-grained CICS applications (i.e., COMMAREA programs) is good. Furthermore, SFM's ability to generate and deploy the resulting business service into the CICS TS environment is very good. Not so good is what happens when you cross the line and begin to use SFM as a visual programming tool, when you try to use it to orchestrate micro flows within/between fine-grained CICS terminal-oriented transactions. This is when frustrations begin – and productivity ends. Using SFM for this purpose becomes arduous at best and impossible at worst. By no means is this intended as a slight on SFM; it's just an inherent consequence of applying the modeling approach to CICS terminal-oriented applications.

Composing business services out of terminal-oriented transactions requires – shall we say – a more delicate approach.

In contrast to SFM, HostBridge implements a scripting approach to service composition. Now I'll be the first to agree (in a magnanimous gesture of objectivity) that there are all sorts of ways that scripting tools can be pushed too far as well, just like modeling tools. But a tool like HostBridge is perfect for doing certain things, such as codifying how fine-grained CICS applications and data should be accessed to implement some – or all – of a modeled business service.

OK. Assume what I'm saying is correct. SFM – *modeling* – is good for certain tasks involved in the composition of a CICS business service, and HostBridge – *scripting* – is good for other tasks.

How do we bring these two approaches together? I'm glad you asked.

²⁷ Perhaps I should say that SFM tempts YOU to push the modeling orientation too far.

HostBridge and SFM Integration: Some Background

What if SFM and HostBridge could work together, allowing their respective strengths to be leveraged? What would that look like? During 2007, IBM and HostBridge collaborated to answer this question.

The first and most obvious answer was this: allow a HostBridge script to be invoked as part of an SFM flow. Specifically, allow a HostBridge script to be the implementation of an “Invoke node” within an SFM flow. EUREKA! This would be a wonderful combination of product strengths. But how should we do it?

Technically, it was already possible to invoke a HostBridge script as part of an SFM flow. In fact, there were two options: (1) invoke a HostBridge script as a LINKable COMMAREA program or (2) invoke a HostBridge script as a Web service. Both were ruled out as the preferred approach because they were deemed to be either too crude or too resource-intensive. However, I’m going to take a moment to explain these approaches briefly because they do work and they frame the priorities of the joint development work that IBM and HostBridge performed.

Invoking a Script as a LINKable Program from SFM

An SFM flow can include anything that looks, acts, or smells like a LINKable COMMAREA program. This is SFM’s strength.²⁸ Furthermore, a HostBridge script can be invoked as a LINKable COMMAREA program. So why not just do this? Simple. The SFM-generated COBOL program and the invoked HostBridge script would have to agree on the layout of the COMMAREA used to pass data between them (for input and output). Is that a problem? Not really, but a COMMAREA is simply a linear area of storage in which data fields have specific, fixed attributes (e.g., length and type). There is no information in or associated with this COMMAREA that describes its content. That is, there is no inherent metadata (data about the data). As a result, the HostBridge script would have to include information about the COMMAREA layout. Even worse, a unique COMMAREA layout would have to be defined for each script.

When a HostBridge script is developed, users don’t want to think about COMMAREAs – at all! People who compose HostBridge scripts want to think about field name/value pairs – as would be received in a SOAP request, XML document, or HTTP POST data. HostBridge likes metadata, and the simplest form of metadata is the name of a field. For this reason, invoking a script as a LINKable COMMAREA program seemed rather crude.

²⁸ SFF views LINKable programs as the basic building blocks of all CICS life. And this view is pervasive. Furthermore, this view influences other SFF implementation concepts, such as “deployment patterns” and “adapters.” When all’s said and done, an Invoke node within an SFM flow corresponds to the invocation of an adapter. Adapters are (surprise) implemented as LINKable programs. The adapter, in turn, is responsible for doing work. For example, the DPL adapter LINKs to a COMMAREA program – you know... the one you wrote a decade ago and are trying to execute as part of a business service.

Invoking a Script as a WEBSERVICE via SFM

The SFM plugin that was delivered with RDz v7 added support for the invocation of a Web service as part of a service flow. At design time, such a node is defined to SFM based on a WSDL. At run time, the SFM-generated COBOL program performs an EXEC CICS INVOKE WEBSERVICE. This opened up an avenue of consideration. Let's assume that a particular HostBridge script was described using a WSDL. That artifact could then be referenced by SFM to incorporate the execution of a HostBridge script within a flow. Sounds straight-forward, but there was a snag. If the SFM-generated COBOL code were to invoke the HostBridge script as a formal Web service (via EXEC CICS INVOKE WEBSERVICE), a significant amount of CICS overhead would occur! It just made no sense for the SFM-generated program to invoke a HostBridge script using such a formal boundary when both the SFM-generated program and HostBridge are running under CICS TS (and probably in the same region). Conceptually it was good because it solved the metadata problem, but the implementation was far too heavy. Besides, formulating a unique WSDL for each HostBridge script would be a royal pain in the you-know-what.

Invoking a Script as an SFM “Generic Node”

Both of the approaches described above DO work. However, we – HostBridge and IBM – wanted an approach that would reconcile SFF's bias toward LINKable COMMAREA programs and HostBridge's thirst for metadata. We agreed that the most elegant and generic way to integrate a HostBridge script into an SFM flow would be to implement a new concept within SFM: a generic node that receives both data and metadata.

The starting point of such an ideal solution was another feature that IBM added to SFM in v7: support for LINKable programs that exchange data using CICS TS Channels and Containers. Within SFM, such a program is modeled by an Invoke node with multiple input/output messages. At run time, when the SFM-generated COBOL program LINKs to the target program, it passes to the target program a channel containing multiple containers (one message per container).

This mechanism allows SFM to pass a collection of information to a LINKable program that is more robust than a simple COMMAREA. If HostBridge were the target program, all sorts of interesting information (e.g., metadata) could be exchanged between SFM and HostBridge at run time via containers. Cool! But...

At design time, how do you specify to SFM that a HostBridge script is the implementation of such a node? How do you specify the name of the HostBridge script? And how do you specify the arguments to be exchanged between the SFM flow and the HostBridge script? The answers to these questions led us directly to the joint development work performed by IBM and HostBridge.

SFM Importer Extension Point and API

Within SFM, “importing” is an important concept. Every Invoke node within an SFM flow is described by an operation file (a .wsdl file) along with one or more message files (a .mxsd file). These files are very complex XML documents. An “importer” is a program that takes an existing artifact (e.g., a COBOL copybook) and creates a corresponding

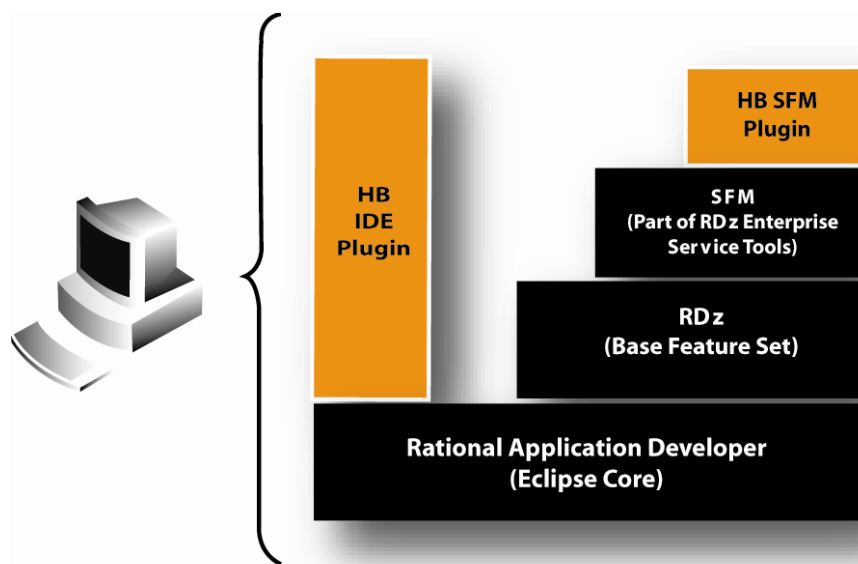
operation and/or message file. IBM provides a number of importers with SFM (e.g., to create a message file from a COBOL copybook).

As part of our joint development work, IBM enhanced SFM to allow third parties (like HostBridge) to provide custom importers. A custom importer is invoked via the Importer Extension point. The term “generic node” was adopted to describe an SFM Invoke node that is created via the Importer Extension point. To complete the picture, IBM defined an API – SFAPI – that allows a custom importer to build the operation and message files associated with the node.

Using these new SFM features, HostBridge created a custom importer that makes it extremely easy to include a HostBridge script within an SFM flow. Depending on the context, I will refer to this component as either the HostBridge SFM Importer or SFM Plugin.

And just to clarify, the HostBridge Eclipse IDE Plugin and the HostBridge SFM Plugin are *not* the same thing. The HostBridge IDE Plugin is used to author, test, and deploy HostBridge scripts, and can be used in any Eclipse-based environment. The HostBridge SFM Plugin exploits these new features of SFM and therefore requires the RDz environment.

The following diagram illustrates how these components fit together. Rational Application Developer (RAD), the Eclipse Core, is the foundation. RDz builds upon RAD; SFM builds upon RDz; and the HB SFM Plugin builds upon SFM. Note that the HB IDE Plugin builds directly on the Eclipse Core functionality.



IBM and HostBridge Partnership Positioning

So what exactly has happened? I'll use IBM's words:

IBM and HostBridge Technology have worked to develop an API for the Service Flow Modeler (SFM) component of Rational Developer for System z (RDz) to enable software vendors to *extend SFM with complementary capability*. The new

API allows IBM Business Partners and customers to take full advantage of the benefits of using RDz as a development environment for CICS integration. Using this new API business partners are able to access vendor runtimes from nodes within SFM. Thus, *IBM customers will be able to select from a range of nodes that suit their application needs*, enabling reuse of CICS resources as business services without any changes to existing code. This collaboration demonstrates a *commitment from IBM to work with business partners* to the benefit of mutual customers. Similarly, HostBridge Technology is committed to support IBM tooling for the purpose of CICS integration.

Needless to say, HostBridge Technology is very pleased to have the opportunity to partner with IBM in this manner. Plus, we get to work closely with some really talented IBMers in the CICS TS and RDz product groups.

IBM and HostBridge Technology Positioning

Perhaps it's also a good time to reiterate why IBM and HostBridge have collaborated in this manner. Again, I'll use IBM's words:

Enterprise customers running CICS have a broad spectrum of integration needs: for applications that range from relatively straightforward to extremely complex. IBM and HostBridge Technology have collaborated to offer an integration solution that deals with the full spectrum of integration needs. Rational Developer for System z's CICS Service Flow Modeler provides immediate access to both COMMAREA programs and *simple, well-behaved terminal oriented transactions* using high-level modeling of service flows. HostBridge extends this scope by adding provision for access to *complex terminal oriented transactions, direct connections to data sources* (such as DB2), and transactions making extensive use of MRO. *Together, RDz's CICS SFM, HostBridge, and CICS TS V3 provide access to all your CICS resources using common tooling without requiring any changes to your existing applications.*

That last sentence says it all: "Together, RDz's CICS SFM, HostBridge, and CICS TS V3 provide access to all your CICS resources using common tooling without requiring any changes to your existing applications."

To appreciate the value of this development, let's frame it in "before" and "after" questions. Prior to the IBM-HostBridge collaboration, if you had been asked what kinds of Web services you could build using CICS TS v3 without modifying your existing applications, the answer would have been simple: A Web service that invokes a single COMMAREA program. Now, if you are asked what kinds of Web services can you build using CICS TS v3, Service Flow Feature, and HostBridge without modifying your existing applications, the answer is far more interesting: ***You can do ANYTHING and EVERYTHING!***

With CICS TS v3, SFF, and HostBridge, you can build business/Web services from any combination of COMMAREA programs, terminal-oriented transactions (BMS or non-BMS), data sources (DB2, VSAM, DLI, etc.), and even non-CICS resources.

CICS TS provides the necessary infrastructure to enable robust and scalable services deployment. SFF provides a graphical modeling environment that enables the creation

of CICS business services by composing a flow of CICS application interactions. HostBridge provides the ability to easily incorporate fine-grained terminal-oriented transactions within a service flow, as well as CICS-controlled data sources.

The bottom line is simple: HostBridge, CICS TS, and SFF are *complementary*. But really, a stronger word than “complementary” is called for. For customers wanting to compose Web/business services from existing terminal-oriented applications, we think HostBridge is practically a *requirement*. The combination of these products provides a *complete* solution for implementing a CICS-based SOA.

9. Making CICS Services Decisions

If I've done my job well, you've come to appreciate all the issues and decision points involved in composing existing CICS application and data assets into reusable services. But slicing, dicing, and analyzing the various factors is simple – at least when compared with trying to pull all those concepts back together in a coherent way. Furthermore, creating succinct guidelines or rules-of-thumb turns out to be challenging, at least if it's done with any degree of accuracy.

Perhaps part of the problem is that I'm too much of a "technology guy" – or not enough of a "sales guy" – to pretend that exposing CICS applications as web/business services is not a one-size-fits-all proposition. It's not, and any vendor who says it is should be run out of your office, your IBM rep included!

CICS has been around for a long time. Given the evolution of CICS over four decades, it's silly to think that all CICS applications are homogeneous. Many are purely terminal-oriented; a few are purely COMMAREA-based; most are hybrids. If they are terminal-oriented in any way, you have some additional and important issues to consider. Of course these categories are obvious. The real complications usually stem from the fact that many CICS terminal-oriented applications have important operational characteristics that are *not* apparent – and sometimes *unknown* to the people using or maintaining the application.²⁹

So you really aren't going to hear me postulate too many rules-of-thumb.³⁰ The good news is that you have options; the bad news is that you have options. But if you exercise your options carefully you have some very real ROI opportunities.

The place to start is by considering a few top-level questions:

- What are the characteristics of your existing CICS application and data assets?
- How do these CICS assets relate to the business services you wish to expose?
- How do you want to expose your business service?

These questions give rise to many others, but they are definitely the place to start.

Cut to the Charts

Rather than try to organize all the questions, answers, and decision points into long paragraphs with many words, I have resorted to long flowcharts with many boxes.³¹ In fact, I've attempted to boil it all down into three flowcharts.

²⁹ A prospective customer in the financial services industry had a mission critical CICS application in need of "web enablement." Their CICS apps have been around for a long time and they work great. However, in order to clean up successfully in certain error situations, one application relies on a cleanup transaction being started by their ZNEP exit. However, a ZNEP exit is only invoked for "real" terminals – not bridge facilities. What was needed was an integration product (and a vendor) that understood the nature of such applications. HostBridge was able to deliver to the customer equivalent functionality within 48 hours (via native HostBridge capabilities and a HostBridge user exit).

³⁰ Unless it's: "Call me and let's talk about your CICS SOA/integration requirements."

- Flowchart 1 starts with a simple assumption: you want to expose one or more CICS programs or transactions to the outside world. You know you need to expose something. The question is whether you need to compose them into a service!
- Flowchart 2 starts with the assumption that you do need to compose a service from your CICS applications and/or data sources. The question is what tools should you consider using to compose the CICS-based service?
- Flowchart 3 starts with the assumption that you do need to include CICS terminal-oriented transactions in your service. The question is what tools should you consider using to compose the micro flows among these transactions.

Flowchart 3 is my “chef-d'oeuvre.”³² It comes as close to codifying what we’ve observed and learned about building high-volume, high-fidelity CICS-based services from terminal-oriented transactions since the introduction of CICS TS v1 – way back at the turn of the century.

And, while this flowchart is styled to help you decide whether to use HostBridge or Service Flow Modeler (SFM) to compose your CICS-based service, I hope you interpret it a bit more broadly. The questions it poses about your CICS terminal-oriented transactions are the precise questions you should consider *regardless* of the particular tools you are comparing.

Flowchart 3 may come off as overly critical of SFM, so allow me to focus the energy this graphic is meant to convey. SFM is a good product. Its ability to compose a service flow from COMMAREA programs is very nice. Furthermore, the assistance SFM provides in deploying a web/business service within the CICS environment is spot on. However, SFM’s forte is NOT deep integration with CICS terminal-oriented applications. That’s where HostBridge fits in.

I like SFM and think it’s a tool that CICS customers should consider using depending on their requirements. I trust the converse is also true: I hope IBM likes HostBridge and thinks it’s a tool that CICS customers should consider using depending on their requirements. Hopefully our mutual respect is evident from the fact that we regularly collaborate to insure that IBM and HostBridge customers can experience the best of both products.

To be sure, SFM doesn’t pre-req HostBridge, and HostBridge doesn’t pre-req SFM. They are each full-featured, stand-alone products that have unique value. But when they are used together, some really interesting possibilities open up! At the end of the day, our vision and mission is the same: Allow CICS TS customers to experience higher levels, and longer horizons, of business value from their investment in CICS applications.

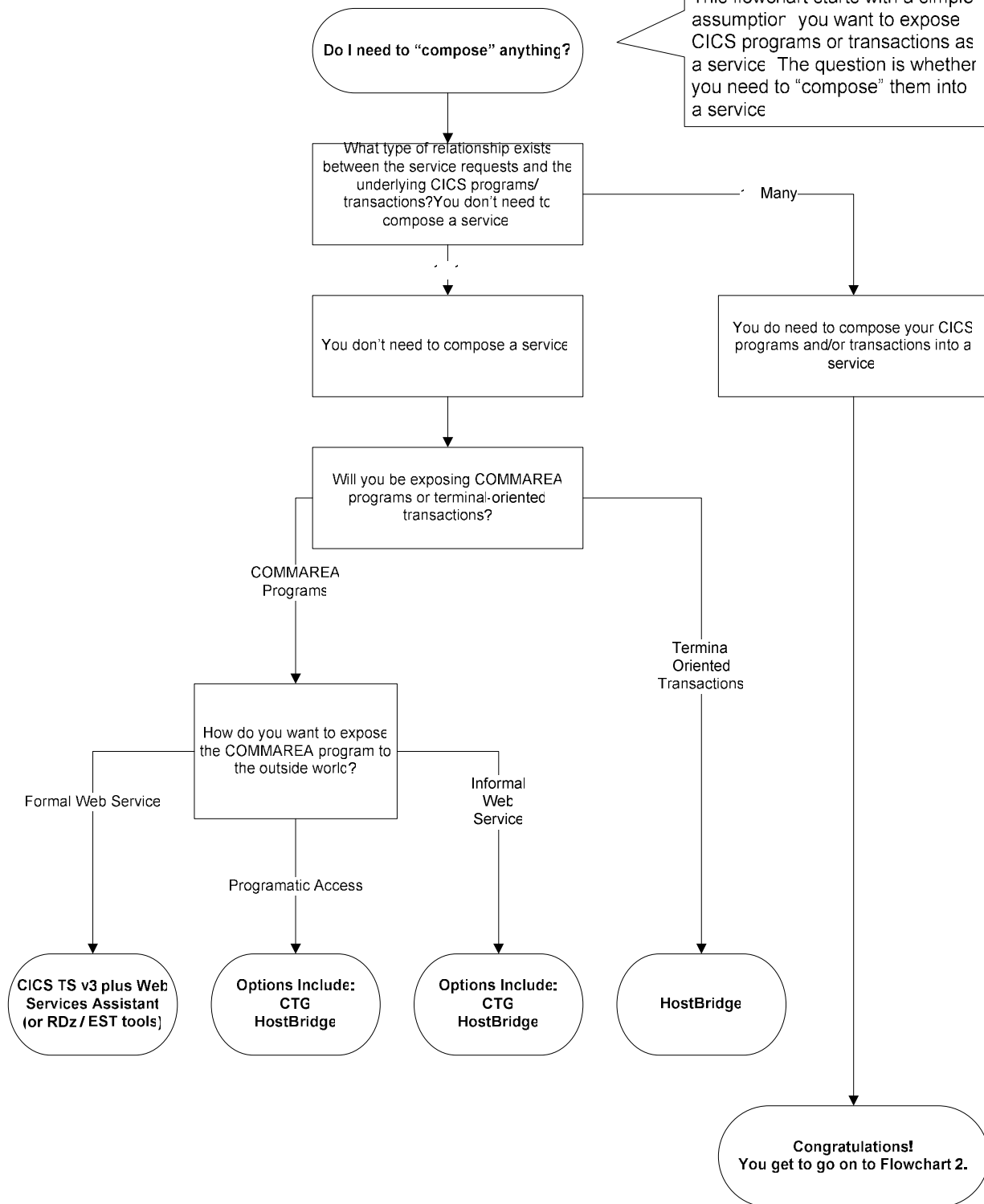
³¹ Besides, I think Visio is one of the coolest tools ever concocted.

³² chef-d'oeuvre – A masterpiece, especially in literature or art. This paper doesn’t qualify as art, but it’s starting to be long enough to possibly qualify as literature (non-fiction).

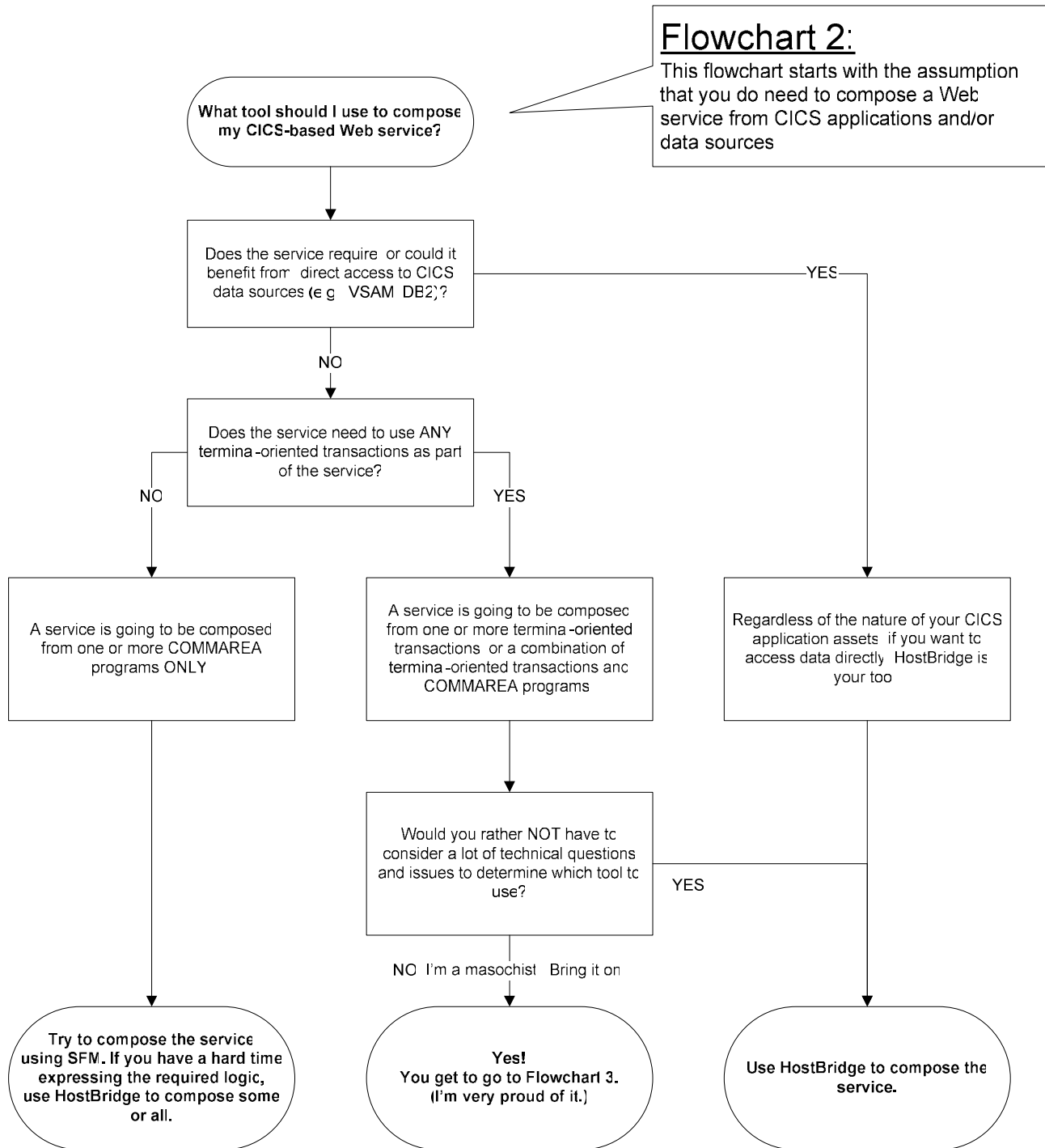
Flowchart 1: Do I Need to Compose Anything?

Flowchart 1:

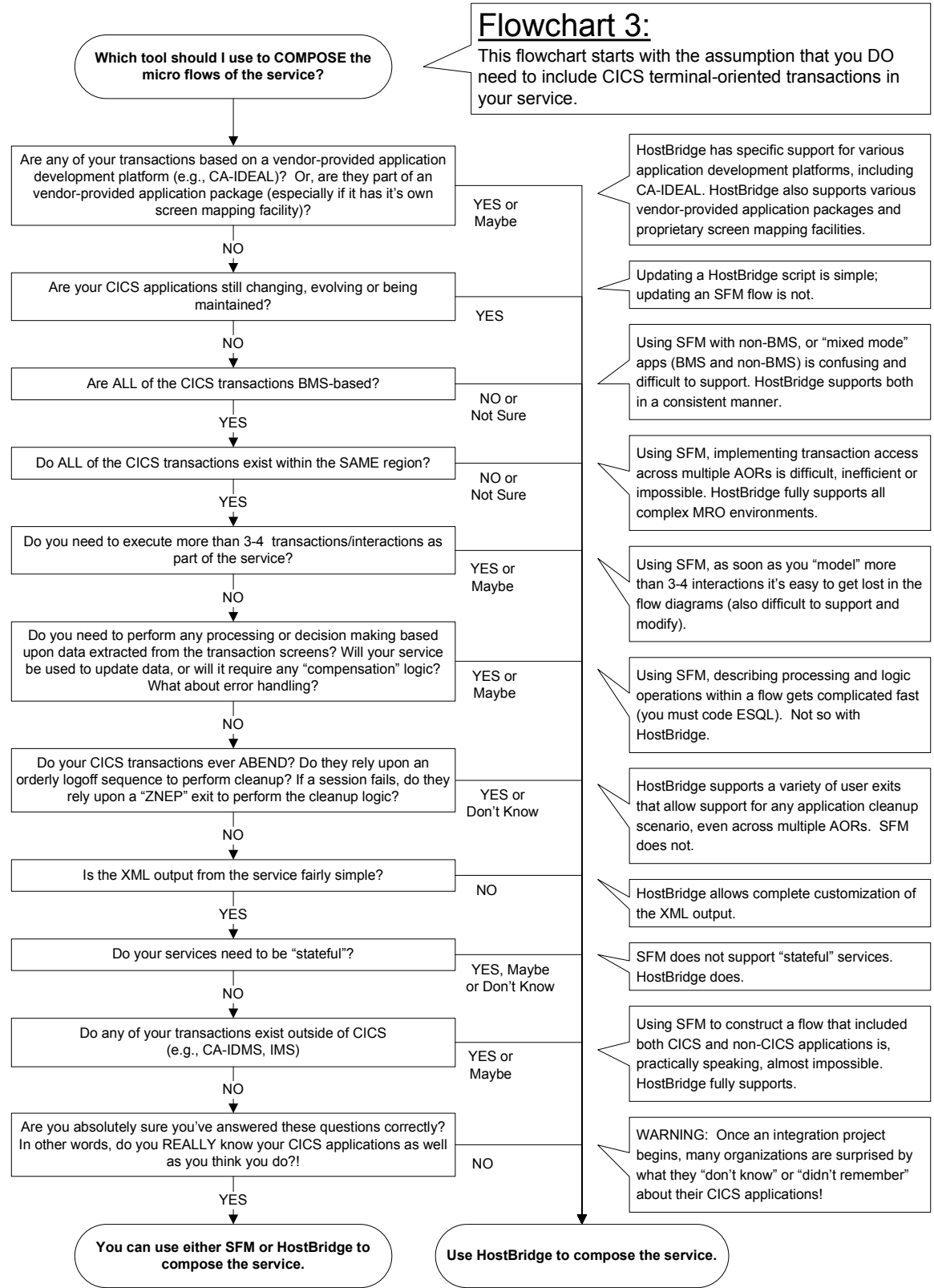
This flowchart starts with a simple assumption: you want to expose CICS programs or transactions as a service. The question is whether you need to "compose" them into a service.



Flowchart 2: What Tools Should I Use to Compose My CICS-Based Web Service?



Flowchart 3: Which Tools Should I Use to Compose the Micro Flows of the Service?



Shameless Sales Pitch

Software vendors, like the customers they serve, have unique domains of expertise. At HostBridge, ours is transforming CICS terminal-oriented applications into high-volume, high-fidelity business services. We are subject matter experts in this area, and we probably have more experience with real-world CICS terminal-oriented applications – accessed via the CICS Bridge interface – than all other ISVs combined.³³ It's not that we're smarter than anyone else. It's simply that this is what we do *all day, every day*. As a result, our software has evolved to be the most robust CICS integration product for terminal-oriented applications.

Successful, mission-critical CICS integration projects call for a deft touch.³⁴ And if you get the feeling that your CICS integration vendor (including IBM) is approaching your project with a “cookie cutter” mentality... Run!

If you are already a HostBridge customer *thank you* for your business. It's a privilege to lock arms with our customers and help them achieve success. If you are not a HostBridge customer, we would consider it a privilege to compete for your business.

³³ Per Yogi Berra: “It ain't bragging if it's true.” Check out www.yogiberra.com for more “Yogi-isms.”

³⁴ I love this word. Deft: neatly skillful and quick in one's movements; demonstrating skill and cleverness.