

Composing CICS® Business Services

SFM Modeling and HostBridge™ Scripting

A HostBridge White Paper

By Russ Teubner, CEO



866-965-2427

info@hostbridge.com

Copyright Notice

Copyright © 2008-2009 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: 5/5/2009

Trademarks

HostBridge and the HostBridge logo are trademarked by HostBridge Technology.

Composing CICS® Business Services

Modeling, Scripting, SFM, and HostBridge™

Preface

This white paper is an excerpt from *The Role of HostBridge™ in the CICS® Services Architecture*, an “anti-white paper” I set about writing several months ago – more relaxed in style, hopefully less boring than most (including a few I’ve authored), and honest about its marketing message. It remains a work in progress. Our plan is to continue revising and updating until we have the issue of CICS integration *nailed*. With that goal in mind, I encourage you and all readers to share your CICS integration thoughts and experiences. You can find the latest version of the paper and express your opinions at www.hostbridge.com/index.php/awpfeedback.

In the paper I don’t try and fool anyone: I have a point to make. My point is...

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.

This excerpt focuses specifically on the key *how* questions: how best to compose CICS services and how complementary composition tools deliver optimal outcomes.

The How of Composition

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 with a little setup.

Composition Tools

We begin with a 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.¹

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 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 (my parents forced me before I could choose a different instrument). 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.²

¹ 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.

² 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 the notes and rhythm just right, the whole thing breaks down and is unrecognizable.

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).³ 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 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.⁴ The

³ One of the most enjoyable and insightful articles you can read on the topic of modeling was written by Bertrand Meyer over a decade ago. Titled "UML: The Positive Spin," the article was originally published in the 1997 special UML issue of Ed Yourdon's *American Programmer*. It can still be found on the Eiffel Software Web site. Warning: the article is satirical and may offend UML devotees.

⁴ Ever tried to tie your shoe strings while wearing mittens? Yeah, that's the feeling.

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.⁵ 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 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,

⁵ 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 and arrows,” that is, boxes or circles or bubbles, treated as screen objects, connected by arrows, lines or arcs.” (Wikipedia: Visual Programming Language, accessed 7/11/08.)

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.⁶

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.*

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 it's 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 integrated 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 automate and aggregate 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.

⁶ 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.

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.⁷ 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.⁸

SFM and HostBridge – Together

The key objective of this paper, and one of the primary objectives of *The Role of HostBridge™ in the CICS® Services Architecture*, the “anti-white 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.⁹ 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

⁷ 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.

⁸ One of my initial reviewers (a prominent industry analyst) observed: “This is 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.

⁹ Perhaps I should say that SFM tempts YOU to push the modeling orientation too far.

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.

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.¹⁰ Furthermore, a HostBridge script

¹⁰ 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.

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 in terms of 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.

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

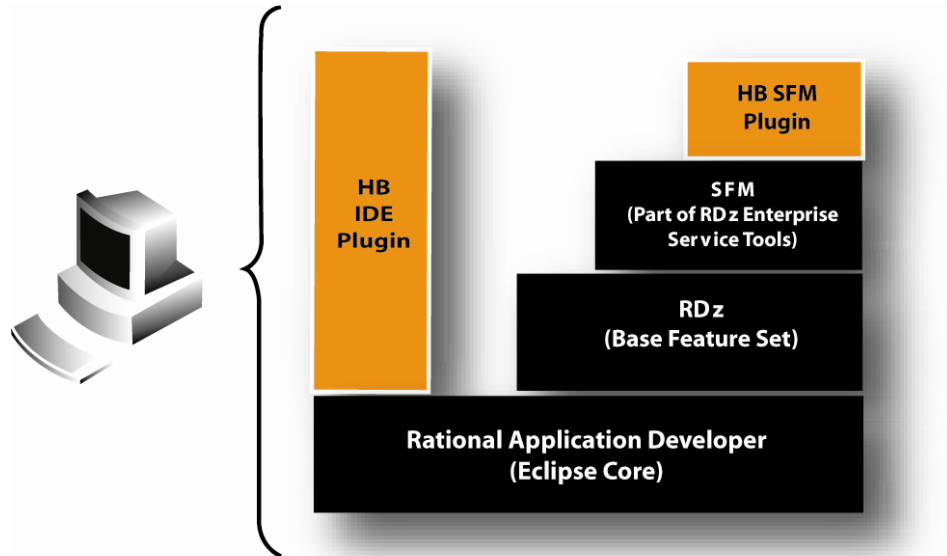
Within SFM, “importing” is an important concept. Every Invoke node within an SFM flow is described by an operation file (a .wsdl file) along with one or more message files (a .mxsd file). These files are very complex XML documents. An “importer” is a program that takes an existing artifact (e.g., a COBOL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