CA1L : A WLM-enabled large request handler for CICS

Version 1.0

15th October, 2002

Ian J Mitchell
IBM UK Laboratories
Hursley Park
Winchester
SO21 2JN
ianj_mitchell@uk.ibm.com

Property of IBM

Take Note!

Before using this report be sure to read the general information under "Notices".

First Edition, June 2002

This edition applies to Version 1.0 of CA1L and to all subsequent releases and modifications unless otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2002. All rights reserved. Note to US Government Users -- Documentation related to restricted rights -- Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- Z/OS

Summary of Amendments

Date	Changes
15 October 2002	Initial release

Preface

This SupportPac demonstrates a very simple technique you can use to enable applications to pass more than 32K of data between components and still exploit the established and familiar benefits of CICS Distributed Program Link.

Bibliography

- *CICS® Transaction Server for z/OS™ - CICS Business Transaction Services*, IBM Corporation, SC34-5999-01
- *CICS® Transaction Server for z/OS™ - CICS Application Programming Reference*, IBM Corporation, SC34-5994-01

Chapter 1. Breaking through 32K for DPL with CICS Business Transaction Services

A common scenerio

The established pattern of Terminal Owning Regions (TORs) and Application Owning Regions (AORs) is the primary way that CICS enables workload management to provide horizontal scaleability and failure toleration. It continues to serve its purpose extremely well for 3270 terminal access to CICS applications, but is proving to have some limitations in the current environment with more diverse clients accessing valuable CICS application assets.

For example, today many CICS systems are being opened up to TCP/IP and clients access the applications using the HTTP protocol. In this case, a TOR/AOR pattern is still valid but the "TOR" is the holder of TCP/IP Socket connections rather than 3270 terminals, and Distributed Program Link (DPL) rather than Transaction Routing is used between TORs and AORs. The Socket Owning Region (SOR) serves the same network role as a TOR, and the system is able to benefit from workload management (WLM) to exploit the multiple AORs.

The major limitation encountered in this scenerio is the 32Kbyte limit of the COMMAREA used to pass data back and forth between the SOR and the AOR. This limitation is proving increasingly inconvenient as the data types passing between CICS applications and clients becomes larger. For example, clients that send a relatively small query request over HTTP might be ready to receive hundreds of kbytes of response from the application. How will that response be passed from the AOR executing the application to the SOR holding the conversation with the socket?

A simple solution with CICS BTS

CICS Business Transaction Services (CICS BTS) is an enhancement to the CICS API introduced in CICS Transaction Server 1.3. The technique I'm going to explain uses a simple BTS application as a framework to enable more than 32K of data to be passed from SOR to AOR, whilst still benefiting from the WLM features of DPL.

Just one of the features of CICS BTS is the Container API. This is an API that allows 'name-value' pairs to be passed from program to program in a business transaction. These can most easily be thought of as 'named COMMAREAs' . However, a useful additional characteristic in this scenario is the absence of a 32Kb limit.

The BTS API is a rich and powerful one, and very complex applications are possible. The design of BTS applications can be quite a challenge to undertake. However, the simplicity of this BTS solution to the 32Kb problem demonstrates that the power of BTS can offer benefits very quickly.

Separating function from dispatching

The basic pattern for the solution is a three phase processing of the application request. The first phase reads the data containing the application request from the socket, the second phase processes the request and the third phase sends the reply back to the client using the socket. The first and third phases have an affinity to the socket, and so need to be lightweight and execute in the SOR. The middle phase is assumed to be the bulk of the processing associated with the request and so will benefit from being dispatched on an AOR.

The framework provide by this technique is implemented in three simple CICS programs using a small number of CICS API commands. If you implement the pattern described then you'll need to pick program names for the three application programs - they are referred to here as the SOR Dispatcher , AOR Server, and the BTS Request Processor .

Two of the programs handle the dispatching of the framework function on SOR and AOR, whilst the third encapsulates data-handling and calling the application code.

Chapter 2. Explanation of the design

The SOR Dispatcher is attached when the request is received. It defines a BTS Process that will manage the handling of the request and reply. The SOR Dispatcher invokes the BTS process three times - once for each phase. BTS processes are identified by a name and have an event-driven CICS application program associated with them. One way the SOR Dispatcher can name the process is using the client IP address. The program it associates with the process is the BTS Request Processor.

The BTS Request Processor implements the logic of each of the three phases of the request handling. Each phase is signalled by firing an event.

Phase 1 - receiving the request

The first phase is executed by the SOR Dispatcher in the form of a LINK to the process.

This fires the first event to be handled by the BTS Request Processor. The BTS Request Processor uses the common structure of a BTS activity program - retrieving the event and then using a select statement to process each event.

In the first request handling phase, the BTS Request Processor should store the request in a Container associated with the root activity, and define an event to represent the next phase. It would be reasonable to perform some lightweight validation or transformation of the request, but remember that this processing is occuring in the SOR.

It's necessary to expose the ActivityId of the root activity of the process so that the SOR Dispatcher can activate it in an AOR. The easiest way to do this is to put the activityid in a container that the SOR Dispatcher can observe.

On return from the LINK to the first phase of the process the SOR Dispatcher checks the process status. If there was a failure and the event for the next phase was not defined, then the process will be complete and there is no result to return to the client. However the normal case will leave the process incomplete, waiting to handle phase 2. The activityid to be used for phase 2 is obtained from the container in which it was placed by the BTS Request Processor.

The SOR dispatcher then takes a syncpoint. As BTS state is transactional, this commits the definition and subsequent operations on the process and root activity. It also releases the lock held on the process by the SOR task.

Phase 2 - processing the request

The SOR Dispatcher now invokes the AOR Server Program which is defined as remote in the SOR. This provides a workload management opportunity.

The activityid of the root of the process is passed from the SOR Dispatcher to the AOR Server in an ordinary commarea. The AOR Server simply acquires the root activity whose id is passed in the commarea and invokes the second phase processing in the form of a LINK.

This fires the event signalling the second phase in the BTS Request Processor.

The BTS Request Handler program is where the processing of the request to produce the reply happens. If other programs are linked to then those programs can access the containers too. The other application programs do not have to deal with any other BTS related obligations. Of course,

existing commarea programs can be invoked in the normal way with some simple wrapper logic that maps a container (or portions of a container) to a commarea and back again.

The output from this stage will be used in the final stage to send the reply to the client.

The phase 2 processing in the BTS Request Handler completes in the AOR and, since the DPL command specifies the SYNCONRETURN option, the AOR task terminates and commits the recoverable work. Again, the process state is released for further processing.

Phase 3 - returning the reply

When control is returned to the SOR Dispatcher, it re-acquires the process in order to invoke the third phase of request handling. This is performed via a local LINK firing the third phase event.

The third phase event is the signal to the BTS Request Processor to access the container(s) containing the reply and send it to the client via the socket. Completion of the process is signalled by deleting the single outstanding event.

The following picture illustrates the overall processing scheme. The Socket Owning Task in the SOR is executing the SOR Dispatcher. It invokes the first and third phases of the processing locally in the SOR, whilst the second phase is invoked via a DPL to the AOR Server.

Chapter 3. Resource Definitions

The resource definitions required for this technique are as follows.

- ProcessType definition - a ProcessType must be defined in the SORs and AORs. This identifies the repository file used to store the process state.
- Repository File definition - the file named in the ProcessType must also be defined in the SORs and AORs. The file must be shared and so defined to be accessed via VSAM RLS.
- Program definitions - The SOR Dispatcher, AOR Server and BTS Request Handler require definitions as follows...
  - SOR Dispatcher - a local program in the SORs
  - AOR Server - a remote definition in the SOR and local definition in the AORs. The program name is used in the SOR Dispatcher when it issues the DPL command.
  - BTS Request Handler - a local definition in both the SORs and AORs. The program name is used in the SOR Dispatcher when defining the process.
- TCPIPService definition - This definition defines the port on the host to which the client connects. To get started quickly use the standard CWXN transaction and the standard CICS Web Support URL syntax (ie host:port /CICS/CWBA/SOR Dispatcher).

Chapter 4. Performance Considerations

What are the performance characteristics of the technique? BTS uses recoverable VSAM KSDSs as datastores for the process state, and one of these repository file is being used to pass data between SOR, AOR and back. So this technique costs some VSAM I/O. This is will happen as follows...

1. Creation of the process by the SOR Dispatcher writes a record to reserve the process identity.

2. The process and root activity state is written to the repository file during the syncpoint performed by the SOR Dispatcher.

3. The AOR Server reads the root activity state from the repository file when it issues the ACQUIRE command.

4. The root activity state is written back to the repository file when the AOR Server task terminates.

5. The SOR Dispatcher re-reads the process and root activity state from the repository file when it issues the ACQUIRE command.

6. The SOR Dispatcher writes the completed process state to the repository file when its task terminates.

7. A system task reads and deletes all the records associated with the task after the process is complete.

Also, the repository records are a maximum of 16KBytes, so reading or writing the process state will use multiple records as the container data exceeds this size.

Chapter 5. Potential Enhancements

One of the benefits of the BTS programming model is the easy and structured way that applications can be extended. For example, in order to add a logging function to the request handler, simply define another event in the interface to the BTS Request Handler and invoke it with RUN ACQPROCESS ASYNCHRONOUS command at the end of the SOR Dispatcher. This will activate the process for a fourth time (in a region chosen by the workload manager). The processing of the new event will have the containers available and can log data from those containers to a journal.

Another simple enhancement would be to employ a timer event to guard against some sort of failure leaving the process incomplete. The BTS Request Handler would define the timer event in phase 1 processing, and delete it during phase 3. Should some failure occur that means that phase 3 processing does not take place before the timer is scheduled to expire, then the timer event will fire and the process will be activated and can deal with that event.

An alternative to the timer based clean-up is to simply define another input event that can be manually fired from another 'management' application. The names of any outstanding processes are available via SPI commands so that your own instrumentation application can inspect and then invoke any 'stuck' processes. (The CICS supplied transaction CBAM is able to inspect the processes, but all it can then do is cancel them - more intelligent strategies are possible if you write your own instrumentation.)

---------- End of Document ----------