WebSphere Application Server for z/OS

The Value of Co-Location

Locating WebSphere Application Server in the same z/OS instance as data

A document planned for on-going updates as additional testing and evaluation is performed

This document can be found on the web at:
www.ibm.com/support/techdocs
Search for document number **WP101476** under the category of "White Papers"

Version Date: November 11, 2009
See "Document Change History" on page 29 for a description of the changes in this version of the document

Written and provided by the WebSphere Application Server for z/OS team at the
IBM Washington Systems Center

Special thanks to those whose efforts on behalf of the System z and z/OS platform has been invaluable:

- **Richard Ko** and **Mike Buechele** of the System z performance team in Poughkeepsie, NY
- **Dave Bonaccorsi** and **Mike Poirer** of the IBM Business Process Integration organization
- **Maryela Weihrauch** and **Fred Orosco** of DB2 development
- **Gary Puchkoff** and **Colette Manoni** of System z and WAS z/OS development
- **Many others** ... thanks for making IBM's flagship server platform even better

The WebSphere Application Server for z/OS support team at the Washington Systems Center consists of: **John Hutchinson**, **Mike Kearney**, **Louis Wilen**, **Lee-Win Tai**, **Steve Matulevich**, **Mike Loos** and **Don Bagwell**.

Mike Cox, Distinguished Engineer, serves as technical consultant and advisor for all of our activities.

For questions or comments regarding this document, send e-mail to Don Bagwell at dbagwell@us.ibm.com

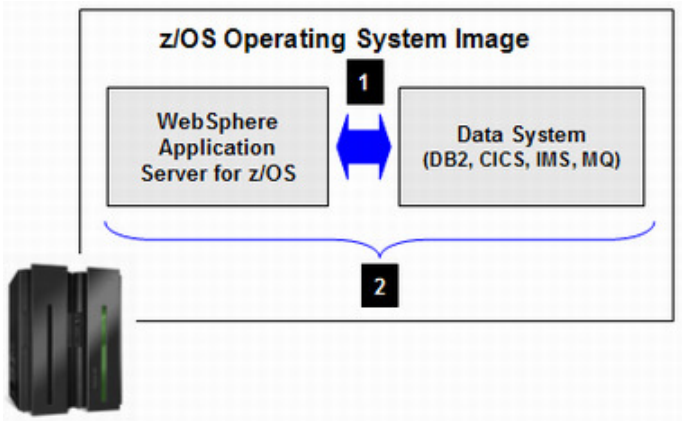
Table of Contents

Executive Overview

One of the value propositions of WebSphere Application Server (WAS) for z/OS is the ability to **co-locate** the application layer with the data layer in the same z/OS operating system instance. This leverages many of the inherent advantages of the System z hardware and z/OS operating system to be exploited to your business advantage.

Summary of co-location advantages

The value of co-location can be summarized under two broad headings:



1. **Performance and Efficiency** – co-location within the same z/OS operating system instance enables taking advantage of cross-memory data transfer, reducing overall request latency, improving overall throughput and reducing overall CPU utilization through the elimination of network traffic handling.[1]

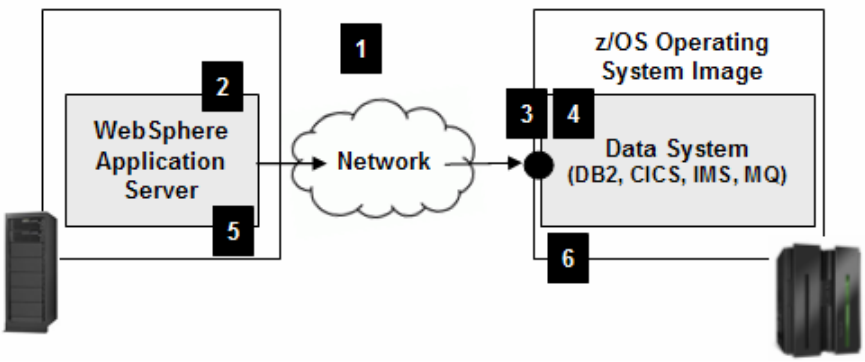
   We offer benchmark support of this under "Performance Studies" starting on page 6.

2. **Operational benefits** – co-location within the same z/OS operating system instance allows for things such as a the assertion of security identity, maintaining the same thread of execution, and being able to manage within a single Work Load Manager (WLM) classification.

   We offer an explanation of these benefits under "Operational Benefits" starting on page 14.

The purpose of this document is to provide validation and support for those two categories of value statements. The objective is to document and explain several advantages of co-location to enable customers to better arrive at the proper decision as to the best architecture to suit their needs.

Summary of shortcomings of remote location



1. **Network latency and overhead** – Network delays, however small, add up. That's particularly true in

[1] Performance studies are often very sensitive to the nature of the workload. It is important for us to stress that these results are **not** a guarantee of performance. The performance results you experience may be different. These studies are meant to offer an understanding of general performance principles given the workload described

high-volume, high-scale solutions. The overhead of SSL encryption, even with hardware assist, is greater than zero, which is what is possible when everything is co-located and no SSL is needed.

2. **Serialization of query parameters** – To pass over the network, they need to be serialized. That represents overhead, and in a high-volume, high-scale solution that can add up to hamper overall performance
3. **New thread of execution** – Remote calls received by the data system on z/OS require a new thread of execution, which is not true when WAS is co-located. There the thread of execution is carried through, reducing switching overhead.
4. **Separate WLM classification** – Remote calls eliminate the possibility of having WLM manage the entire request flow under a single classification. That reduces manageability and control
5. **Definition of identity aliases outside central repository** – flowing security over a remote connection often requires the definition of a remote alias. That creates ID/password definitions scattered across multiple remote servers, making updates and coordination more difficult.
6. **Less efficient two-phase commit syncpoint coordination** – remote calls often require distributed two-phase commit protocols such as JDBC XA. Those are effective, but not as efficient as the local syncpoint coordination offered by z/OS Resource Recovery Services, particularly during transaction recovery.

> **Note:** We do not wish to imply there's no reason whatever to have a multi-tier topology. We wish merely to point out some often overlooked shortcomings. A frequently cited argument for locating the application layer in a different operating system instance is *availability*. However, much goes into a highly available design, and high availability can be achieved even with co-location of application and data. See "Local adapters and high availability" on page 18.

Please see "Operational Benefits" on page 14 for a bit more detail on this.

Performance studies

Here is a summary of the studies, with a pointer to the page in this document where the details of studies run are offered more fully:

April 2009 - JDBC Type 2 vs. Type 4 "**JDBC Type 2**" – often referred to as "Type 2" or just "T2" – is a JDBC driver implementation that makes use of cross-memory communications with the database system (DB2 in this study) using native driver code. [2] "**JDBC Type 4**" – often referred to as "Type 4" or just "T4" – is a JDBC driver implementation based on Java that makes use of TCP communications with the database system (DB2 in this study). The TCP communications may be done all within the same LPAR (as done in this study), from a different LPAR in the Sysplex (from z/OS, or from Linux for System z), or from a distributed platform.	**Objective:** • To test and validate the effect of recent enhancements to z/OS and the z/OS Java SDK that allows a portion of the JDBC Type 2 work to be offloaded to a zAAP specialty engine. **Summary of Results:** • The use of Type 4 resulted in more *total* processor usage than Type 2. • The overall general processor (CP) usage showed Type 2 and Type 4 to be *approximately equal*. • Performance enhancements in the JDBC Type 2 driver now make it equal or better than the Type 4. In addition to the performance results, with Type 2 the benefits outlined under "Operational Benefits" on page 14 were also present. See page 6 for more on this study.
(Watch this space for future study results)	

[2] You may hear some speak of "Type 2" used with WebSphere Application Server on a distributed platform. What they are speaking of is configuring the JDBC provider and data source to communicate with a local copy of *DB2 Connect*. To the application server DB2 "looks" local, when in fact DB2 Connect is communicating with DB2 over the network. For this study our use of the term "Type 2" is meant to refer to the z/OS JDBC Type 2 drivers supplied with IBM DB2.

Performance Studies

April 2009 - JDBC Type 2 vs. Type 4 Study

This study was conducted by the System z performance team in Poughkeepsie, New York.

Sponsors: The study was sponsored by IBM's Business Process Integration organization as part of an ongoing effort to measure and validate the performance and benefits of IBM business process integration solutions on System z. This explains the emphasis on SOA workloads and the inclusion of WebSphere Process Server (WPS) in the study architecture. SOA and BPI workloads are very interesting study candidates because they tend to exercise a wide variety of different elements of the overall solution topology. In other words, they are *not* trivial little things designed to highlight one small feature, but rather they are comprehensive real-world workloads[3].

Objectives: Among other things, to validate and measure the effect of several updates made to the following components:

- The z/OS dispatcher
- J9 JVM (Java 5 and Java 6), which communicates with the z/OS dispatcher
- The DB2 JDBC Type 2 driver, in which performance enhancements have been added to bring parity with the enhancements made in Type 4 earlier.

Details of the APAR numbers and version levels are offered below.

Measurements in this study sets the `IEAOPTxx` parameter `PROJECTCPU` to `YES` to project how much work could be off-loaded from regular CPs to special assist processors like zAAPs and zIIPs.

Background and a peek at the findings

- In the past, the local Type 2 JDBC drivers for WebSphere z/OS fell into disfavor for two reasons: performance updates in the Type 4 driver were not also made to the Type 2, making Type 4 in many cases outperform Type 2; and the inability to offload the Type 2 work to zAAP meant that JDBC Type 4 -- Java based -- had an overall cost advantage.
- What this study showed is the following:
  - From a total system perspective, this study showed the Type 2 JDBC driver is now comparable to the Type 4 JDBC driver in terms of *general CP usage*. With the new zAAP offload capability this study showed no clear CP penalty to using the Type 2 JDBC driver.
  - From a total system perspective, this study showed that for *overall processor usage* -- CP, zAAPs and zIIPs -- the Type 2 JDBC consumes *less* than Type 4 JDBC. That is because the additional overhead of handling the TCP requests that flow from a Type 4 JDBC driver to DB2.
  - If no zIIP processor is present, this study showed that a *sizeable piece of the work would then fall to the CP*. In other words, Using Type 4 JDBC calls for both zAAPs and zIIPs while Type 2 JDBC calls for only zAAPs.
  - This study showed that recent performance improvements in the Type 2 JDBC driver make it equal or better than the Type 4 in terms of overall processor utilization.

[3] IBM has and continues to conduct a variety of WAS for z/OS new workload performance studies (such as BPM stack products), in order to study and refine the advantages of collocation on the z/OS platform. There are many other z/OS qualities of service that we continue to study as well relative to implementing SOA and Business Process Integration. The results of this Collocation study and others are contained within The WebSphere BPM&C 6.2.0 for z/OS Performance Report. If you are interested in further understanding more about implementing BPM best practices collocation and other performance characteristics of BPM for z/OS workloads, this report can be made available via your local/ IBM z/OS account team.

Key Messages:

Customers who in the past chose to use Type 4 for performance and zAAP offload should consider using the Type 2 JDBC drivers.

The general CP usage – which software licensing is based on -- was seen to be *approximately* the same now.

The *overall* CP usage seen favored the Type 2 driver.

The Type 4 adapter calls for a zIIP processor in addition to the zAAP to avoid having work flow to the CP.

This is in addition to those things outlined under "Operational Benefits" on page 14.

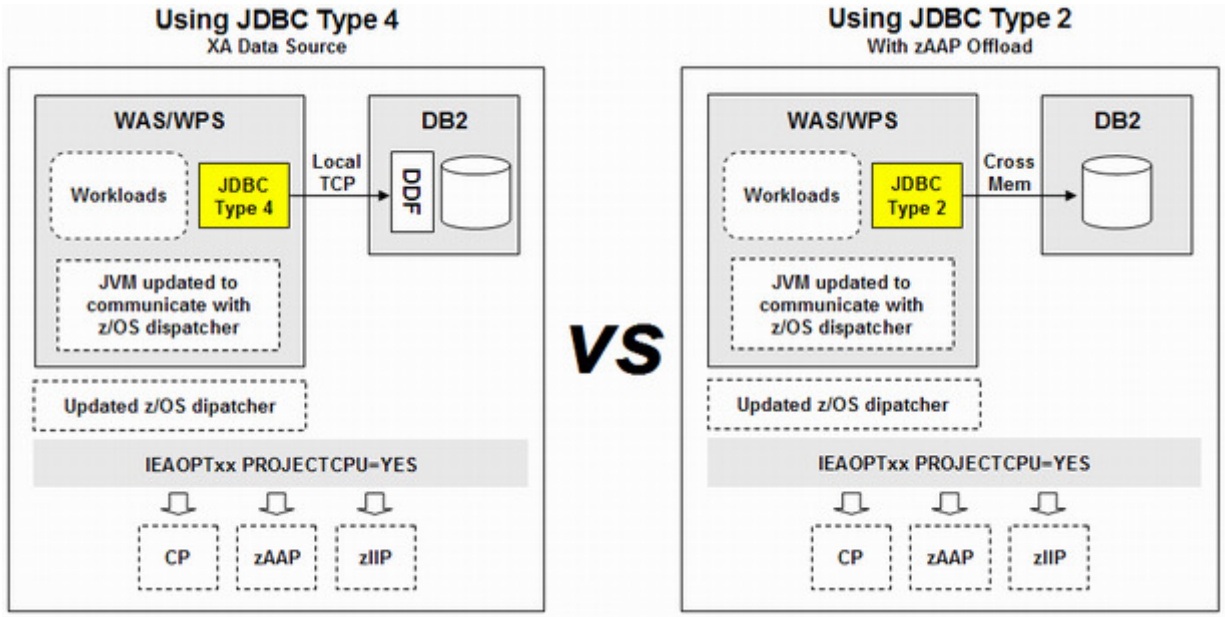
Overview of the test environment and workload

The study involved two business process workloads implemented using Service Oriented Architecture (SOA), running in WebSphere Process Server (WPS) on z/OS:

- *SOABench Handle Claim Using Manual Approval* – a business service which begins the claim process, then calls a long-running process to perform tasks related to the handling of the claim. In the charts and diagrams this is known as the "SOABench HandleClaim Macro2."
- *eRWW New Order Processing with Manual Approval* – a business service that first extracts customer information, then turns and extracts detailed customer information, which is the result set of the service invocation. In the charts and diagrams this is known as "PVT8."

These two workloads make use of JDBC to access data sources in local DB2. Each workload was run and measured – one set of runs using the JDBC Type **4** drive; a second set of runs using the new JDBC Type **2** driver.

Represented as a simplified picture, it looked like this:



Reminder: The objective was to measure and report the *combined effects* of: the updates to the z/OS dispatcher, updates to the JVM that communicates with the z/OS dispatcher, and the performance enhancements in the JDBC Type 2 driver.

There are some interesting and important points to draw from that picture as it relates to the test results we'll report:

- Except for changing the JDBC driver type, everything else remained the same between the measurement runs

- The "local TCP" (TCP within the LPAR) you see in the picture on the left is really a "best case" scenario. When TCP flows within the same LPAR it is optimized by z/OS with a much shorter code path than would be used if coming from another LPAR or from off-platform.
- The Type 4 XA Data Source is what provides the ability to do two-phase commit processing. Type 2 adapters use Resource Recovery Services (RRS) for two phase commit (2PC). Therefore, Type 2 is "always XA" in the sense it always supports 2PC.
- The CPU numbers we'll report are for *everything* on the system – WebSphere z/OS, WebSphere Process Server, all of DB2, TCPIP, RRS, JES, OMVS ... everything. It is not just the JDBC processing or just the DDF processing. It is a *total system view*.

Further details of the SOA workloads used are provided under "Appendix B – Workload Description Details" starting on page 23.

System specifics

Here's a snapshot of the test environment:

System z hardware and operating system

- 4-way z10 2097-704 LPAR
- 12GB of central storage
- No coupling facility (DASD-only log streams for RRS)
- z/OS 1.9 with fix for APAR OA26713

Key Point:	That APAR (PTF UA45546 for z/OS 1.9; PTF UA45547 for z/OS 1.10) is what provides the updates to the z/OS dispatcher to allow offload of Type 2 processing to the zAAP.

System z software

- WebSphere Application Server (WAS) for z/OS Version 6.1.0.22 with a *patched* level of the SDK, which was not made generally available. 6.1.0.23 provides GA for this function[4].

Key Point:	The embedded SDK is critical for the Type 2 offload to zAAP capability. For this test a patched version of WAS 6.1.0.22 was used. *WAS 6.1.0.23 includes the PTFs that provide the offload functionality*:
	`PK81212, PK81211, PK81189, PK81205, PK81204, PK81202, PK81190, PK81187, PK81185, PK81183, PK51638`
	`ibm.com/support/docview.wss?rs=404&context=SS7K4U&uid=swg27007926`
	For Java outside of WAS, see "Determining if the offload capability is present for a non-WAS Java environment" on page 17.

- WebSphere Process Server (WPS) for z/OS Version 6.2.0
- IBM DB2 for z/OS Version 9.1 with fix for APAR PK77599

Key Point:	The APAR fix for PK77599 provides the IBM DB2 Driver for JDBC and SQLJ release 3.53 functionality, which is what has the performance enhancements alluded to earlier.

Client driver servers

- One IBM xSeries 365 with 2processors at 3.0GHz, 3.5G RAM running Red Hat Linux 3.2.2-5 was used to host JIBE engine to drive the PVT workloads.
- One IBM xSeries 365 with 2processors at 3.0GHz, 3.5G RAM running Windows server 2003 was used to drive the SOABench workloads.

[4] As well as 7.0.0.1

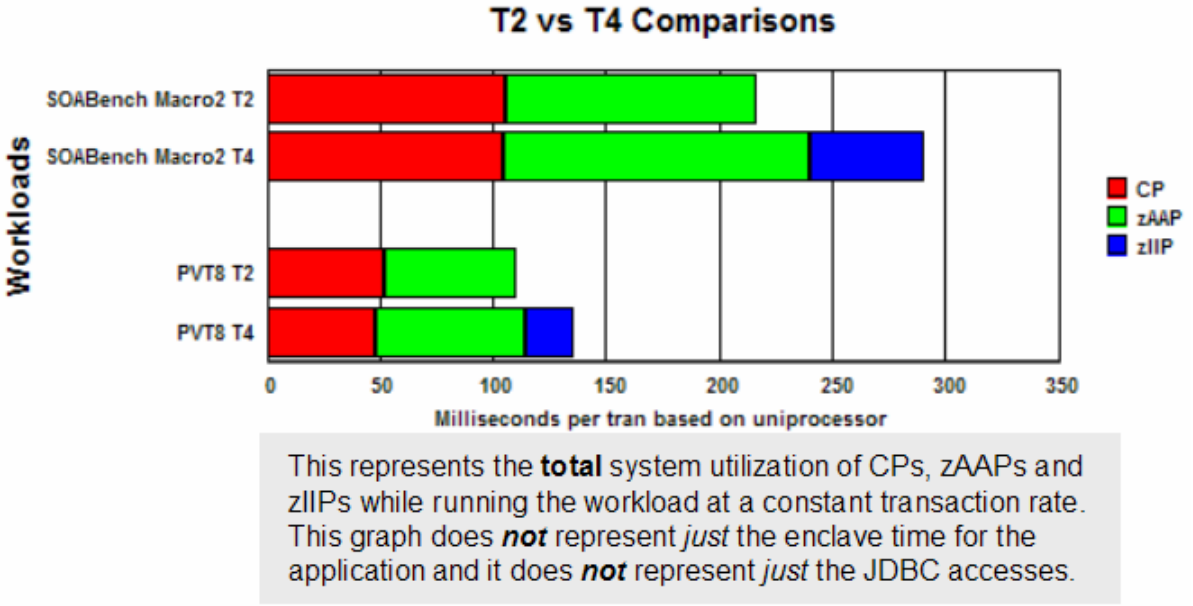
Testing approach methodology

The methodology for this test was to hold a *consistent transaction rate* with the client driver tool and measure the utilization of the CPU. That is a measure of relative "cost" (expressed as CPU utilization) as the control variable changes, which in this case was JDBC Type 2 vs. JDBC Type 4.

Another methodology – *not* employed for *this* test, but demonstrated in other tests [5] – is to seek to demonstrate the *scalability* of the system. In other words, given a set allocation of system resources, what transaction rate can be achieved and sustained? That is a measure of the relative *capability* as the control variable changes.

Results

The following picture summarizes the findings:



This represents the **total** system utilization of CPs, zAAPs and zIIPs while running the workload at a constant transaction rate. This graph does ***not*** represent *just* the enclave time for the application and it does ***not*** represent *just* the JDBC accesses.

Let's review the *structure* of that picture:

- The SOABench HandleClaim Macro2 workload is represented by the top two bars; the PVT8 workload is represented by the bottom two bars.
- The horizontal bars represent the milliseconds per transaction projected to be spent in the various processor types – general processor (CP), zAAP and zIIP.
- The two JDBC driver types are shown for each workload – Type 2 (T2) and Type 4 (T4) [6]

[5] See"Brief summary of a banking scalability study" on page 22.

[6] Multi-row fetch is *enabled by default* for the JDBC Type 4 driver, but the *default* for a JDBC Type 2 driver is *disabled*. For this test it was left at the default *disabled* for JDBC Type 2. Multi-row fetch is one of the performance enhancements recently added to JDBC Type 2 driver, and *depending on the workload* it can have a *significant* performance impact. So Type 4 ran with MRF *enabled* and Type 2 *disabled* ... meaning that the benefit of Type 2 *may* in fact be *understated*. See "Multi-Row Fetch (MRF)" on page 22.

Now let's see what the results are saying for the ***SOABench HandleClaim Macro2 workload***:

- The *overall* CPU with Type 4 was approximately 34% higher for the HandleClaim workload than it was with the Type 2

  Note: As mentioned earlier, that's actually a "best case" scenario. The JDBC Type 4 adapter was used to access DB2 co-located on the same LPAR, which meant the TCP access was optimized. To see the effect if the TCP came across LPARs for the PVT8 workload, see "The effect of cross-LPAR TCP" on page 11.

- With these new enhancements, the overall amount of general CP usage seen was *approximately* the same between the Type 2 runs and the Type 4 runs.
- The Type 4 driver test shows more zAAP usage than was required with the Type 2 driver.
- The Type 4 driver test shows zIIP usage as well. If a zIIP wasn't present, that work would go to the CP.

For the ***PVT8 workload***:

- The *overall* CPU for Type 4 was approximately 23% higher for the PVT8 workload than it was for the Type 2.

  Note: Again ... "best case" due to TCP optimization for local access with Type 4. See "The effect of cross-LPAR TCP" on page 11 for a picture of what it looked like when the TCP flow crossed an LPAR boundary.

- With these new enhancements, the overall amount of general CP usage seen was *just slightly more* with the Type 2 driver than with the Type 4 driver.
- The Type 4 driver test shows more zAAP usage than was required by with the Type 2 driver.
- The Type 4 driver test shows zIIP usage as well. If a zIIP wasn't present, that work would go to the CP.

Overall conclusion

Important: Again ... performance studies are sensitive to the nature of the workload. It is important for us to stress that these results are not a guarantee of performance. The performance results you experience may be different. These studies are meant to offer an understanding of general performance principles given the workload described

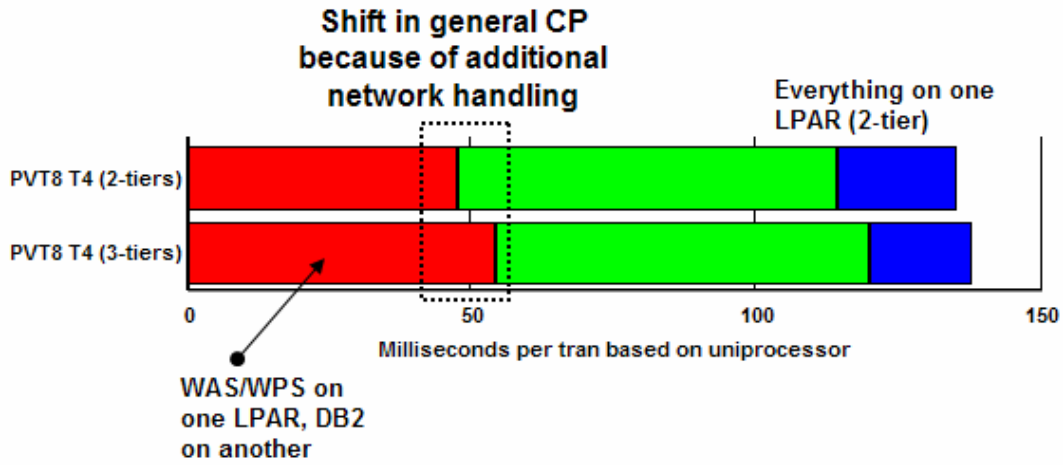
The key take-away messages are:

- Performance enhancements made to the JDBC Type 4 driver in the past have been made available to the JDBC Type 2 adapter as well.
- With these new updates to the z/OS dispatcher, the JVM that communicates with the dispatcher, and the JDBC Type 2 performance improvements, the offload of Type 2 work to the zAAP now brings the amount of *general CP usage* to roughly equal amounts, Type 2 vs. Type 4, based on this test and these workloads.
- Type 4 uses more *overall CPU* than does Type 2. Further, the Type 4 requires a zIIP separate from the zAAP to make the general CP usage of the Type 4 roughly equal to the Type 2.
- Costing methodologies depend on many factors, but a *rough estimation* of cost based on list prices yielded a roughly 10% benefit for Type 2 usage as opposed to Type 4 for the SOABench HandleClaim Macro2 workload, and a roughly equal cost to Type 4 for the PVT8 workload.
- Type 2 does not endure the same latency issues as Type 4. This affects throughput as well as WLM classifications. See "Operational Benefits" on page 14.

The effect of cross-LPAR TCP

We mentioned in several prior spots in this document that the Type 4 numbers were "best case" because of the way z/OS TCP optimizes its flow when it sees the target is on the same LPAR.

To get a sense for how much effect it has, the PVT8 workload was split across two LPARs so that the TCP communication flowed across an LPAR boundary. The CPU numbers are again *total for both LPARs:*



Exploration of a "what if" scenario

A common question that comes up is:

"What would the bar chart look like if the older JDBC Type 2 driver was used?"

We'll break that down into two sub-questions:

1. What would the bar chart look like if the ***older*** *JDBC Type 2 driver* was used with the ***new*** *z/OS dispatcher updates and the* ***new*** *JVM ability to communicate with the dispatcher?*
2. What would the bar chart look like if the ***newer*** *JDBC Type 2 driver* was used on a system that did ***not*** *have the z/OS dispatcher updates and the JVM updates?*

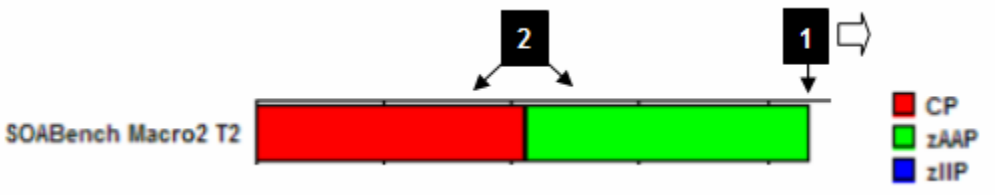
Those variations were not part of the benchmark test, so we can't offer specific numbers. But we can speak in *qualitative terms* so you can understand the effect of the updates.

Older JDBC Type 2, newer dispatcher and JVM

We would expect to gain the ability to offload a portion of the Type 2-related processing to the zAAP; but we would expect to lose the performance updates that were included in the newer JDBC Type 2 driver.

We would likely see an *increase* in the overall CPU (general CP + zAAP) times as the recent performance updates would not be present. The absence of the performance updates would show as greater CPU utilization.

Simplifying things by focusing on just the SOA Bench Macro2 workload, we'd expect to see *something* like this:



1. The end of the CPU bar, representing the sum of general CP and zAAP usage would move to the right. That is due to the extra processing of the older, less efficient JDBC Type 2 driver. How *much* it would move is hard to say without doing the actual measurements.
2. The *proportion* of general CP to zAAP would be expected to remain *roughly* as we saw it in the actual test. But each component would be larger as the overall CPU is expected to be higher.

Newer JDBC Type 2, previous dispatcher and JVM

We would expect to lose the ability to offload a portion of the Type 2-related processing to the zAAP; but we would maintain the performance updates that were included in the newer JDBC Type 2 driver.

We would likely see the *overall* CPU (general CP + zAAP) times be the same as seen in the performance benchmark. But the general CP line would move further to the right.

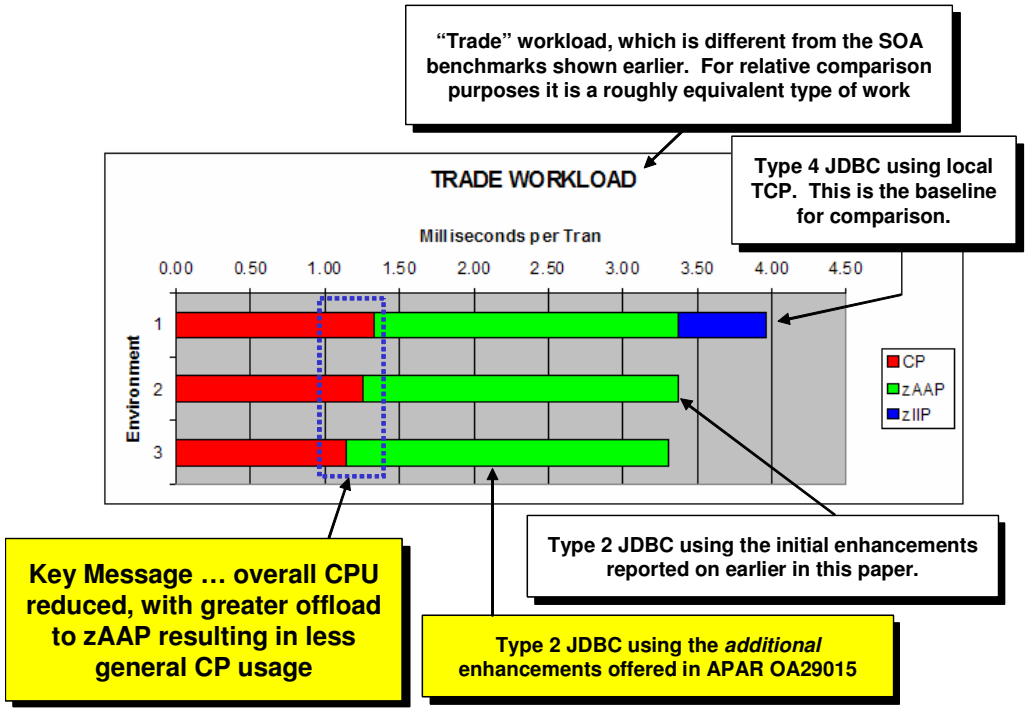
Simplifying things by focusing on just the SOA Bench Macro2 workload, we'd expect to see *something* like this:

1. The end of the CPU bar, representing the sum of general CP and zAAP usage, would be the same as seen in the actual performance study conducted. The performance enhancements of the newer JDBC Type 2 driver would be in effect, and the *overall* CPU would be the same as recorded in the benchmark.
2. The *proportion* of general CP to zAAP would be expected to change. The general CP line would increase because the dispatcher+JVM updates would not be present, therefore the additional offload to zAAP would not occur. How *much* it would move is hard to say without doing the actual measurements.

Effects of updates to be delivered with APAR OA29015

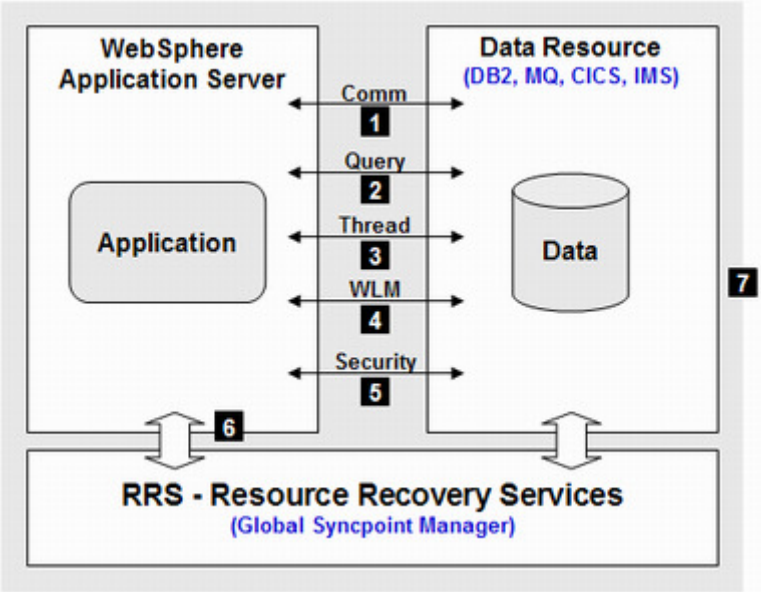
IBM continues to work to improve the benefits of Java on System z and z/OS. As part of this effort, recent work has been done to allow even more of the Type 2 processing to be offloaded to a zAAP engine. Refer to APAR OA29015.

The following chart shows the results and offers an explanation of the key findings:



Operational Benefits

Here's a picture that summarizes the benefits:



Here is a description of what the numbered blocks refer to:

1. *Cross-memory communications*

   When WebSphere Application Server for z/OS is co-located with the data system in the same z/OS operating system instance, communications between the two can be conducted across memory rather than across network. Network technologies have improved considerably in recent years, but they can not compare to the cross memory exchanges. Network latency can affect the overall throughput capabilities for an application solution.

   In addition, when communicating cross-memory there is no chance whatever of anyone "sniffing" or otherwise seeing the communication. There is no reason to worry about encryption or SSL. That may not be true with a network based Type 4 connection.

   When WAS and DB2 are *not* on the same system, and SSL is used (which is often mandated by security requirements), there is an overhead cost associated with SSL processing. There is also the additional complexity of managing and coordinating certificates within the environment.

2. *Avoid serialization of query parameters and result sets*

   Network communications are by their very nature serial, which means that query objects and result set objects need to be serialized to pass across the wire. Serialization represents overhead, and depending on the size and frequency of the serialization it may have an impact on the scalability of the application.

   When WebSphere Application Server for z/OS and the data system are co-located in the same z/OS operating system instance, it is possible to avoid serialization, and thus avoid the overhead.

3. *Single thread of execution*

   The original thread of execution that work is dispatched to is maintained through WebSphere Application Server and into the backend data system. Switching threads represents overhead and could impact the overall scalability of an application solution.

4. *Managed to a single WLM goal*

   Because the thread of execution is maintained, it allows WLM to manage the request to a single goal. That makes defining the WLM goals much easier, which means easier management of the workloads within the system.

With JDBC Type 4 the request arrives in DB2 and needs to be re-classified. The original WLM classification is not maintained. The importance of the work as classified in WebSphere may not be the same as classified when it arrives in DB2. Again, this may lead to additional latency is the end to end flow of the request.

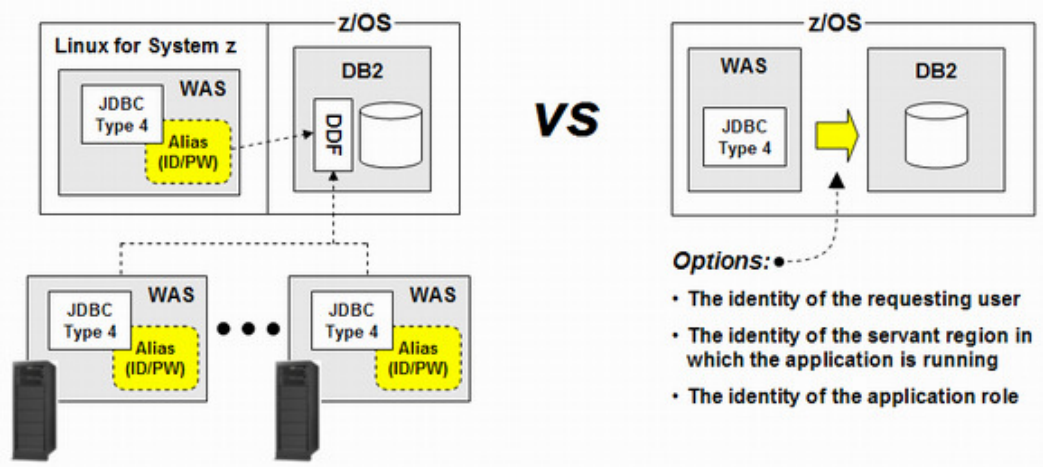
5. Passing of security context

Co-location on the same operating system instance means WebSphere Application Server for z/OS has more options for passing security context:

- The identity of the requesting user
- The identity of the servant region in which the application is running
- The identity of the application role
- A pre-defined alias

When the application is located in another operating system instance from the data system – either another LPAR or an off-platform server – the options are reduced to simply a pre-defined alias.

Aliases are hard-coded ID/password pairs that are passed over by the application server. They are generally frowned upon by security administrators because they reduce security auditing granularity, and they represent security information defined *outside* a central repository:



Periodic password changes require going out to each of the environments where those aliases are defined. If an alias update is overlooked, or updated incorrectly, access to the data is blocked. The cause of the problem is not always immediately obvious, and time is spent debugging the situation.

Co-location avoids the issues with aliases and allows the use of the first three options listed above.

6. *Use of RRS (local syncpoint coordinator)*

In any two-phase commit transaction processing, there needs to be a coordination of participant syncpoints. There is a distributed transaction syncpoint protocol (XA), but it is not as efficient as the protocols employed by Resource Recovery Services (RRS). RRS is a z/OS and Parallel Sysplex function that provides transaction syncpoint coordination using highly-optimized patented algorithms which provides extremely high rates of scalability.

Here's a bit more detail on this:

With RRS there is a vastly simplified recovery process. WAS should be able to recover transactions that are in doubt automatically. RRS maintains a log of the transactions and WAS will work with RRS to replay the logs and either commit or rollback transactions that were indoubt at the time of the failure. (Transactions that are inflight -- have not started commit -- will be rolled back based on the presumed abort protocol.)

When distributed transaction are involved, the coordination of the recovery behavior involves more than one system. There are more complex recovery considerations, for example if one of

the systems that needs to commit a transaction is down, then it's required to either wait for the partner to come back or perform a forced outcome.

Also, co-located debugging is easier because all the pieces are in one place. You don't have to worry about configuring to another system, or what to do if the other system is down. Everything can be dumped at once, and you get the DB2 and the WAS code in one dump. If you are running distributed you may have to do problem determination activities on more than one system which can add time to the debugging effort.

WebSphere Application Server for z/OS and the data systems that reside on z/OS communicate with RRS using very high speed cross memory communications.

7. *Reduced complexity*

   Finally, co-location implies a solution architecture contained in fewer points of failure and fewer monitoring points for debugging and performance tuning. It generally implies the same support staff working to resolve issues, with tends to help reduce overall time to resolution.

In summary, co-location provides an integrated set of functionality and services which provides further optimization of the business system solution.

Appendix A – Miscellaneous Information

Determining if the offload capability is present for a non-WAS Java environment

You may be using the IBM Java for z/OS SDK for purposes other than WebSphere Application Server, and you may wish to know if it has the T2 offload functionality is incorporated.

The key to this is the build level date for the Java SDK you have. The ability to offload T2 processing to the zAAP was introduced in the following build dates, and is present in build dates that comes later:

Generally available SDK Level [7]	T2 offload function incorporated ...
SDK 5, SR9 or later 31-bit SDK ..... 64-bit SDK	 Java build date of: **December 10, 2008** *or later* Java build date of: **December 3, 2008** *or later*
SDK 6, SR3 or later 31-bit SDK ..... 64-bit SDK	 Java build date of: **November 8, 2008** *or later* Java build date of: **November 8, 2008** *or later*

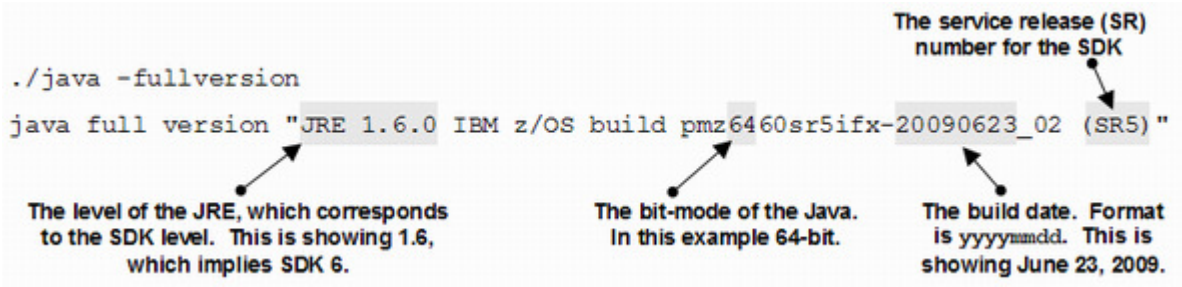
Determining the Java build level in use

- Telnet into your system (or use OMVS)
- Drill down into the `/bin` directory of the Java SDK you're using:
- Issue the following command:

  ```
  ./java -fullversion
  ```

  The "`./`" out from of that insures you're invoking the Java in the directory and *not* the Java that may be defined to `JAVA_HOME` variable as used by the ID you're logged in as. That ID's `JAVA_HOME` may not point to the same Java as the one you're checking here.

- You should see output something like this:



- Compare the build date to the information shown in the table above to determine if the SDK you're working with incorporates the T2 offload functionality.
- Understand that this functionality also requires a minimum level of the z/OS operating system as well as the minimum level of SDK. See "System z hardware and operating system" on page 8 for information on the level of z/OS and APAR fixes required.

[7] It is possible that an "iFix" including this functionality was provided outside the normal maintenance stream. We are not addressing that possibility here. This information relates to generally available levels of the Java z/OS SDK only.

Local adapters and high availability

Some have questioned where "high availability" is possible when local connections are employed. In this section we'll briefly address the topic.

Important: Of all the topics in the world, this may be one of the most difficult to address adequately.

First, there is the terminology used and what people are really seeking. There are *degrees* of availability. More availability is increasingly difficult to achieve, and includes increasing costs. Getting to 100% is near impossible. The degree of availability you require is a function of the business impact of an outage. Many people who speak of "high availability" really don't really mean "continuous availability." They mean something closer to "I'd like to take *reasonable* steps to protect against an outage."

Second, the topic is *extraordinarily* complex. There are *many* considerations; more than most realize. It is certainly *more* than just the type of data connector used.

Fundamentally, high availability involves four key things:

- Redundancy
- Detection of outage
- Working around the point of outage
- Recovery of work left unfinished at the mement of outage

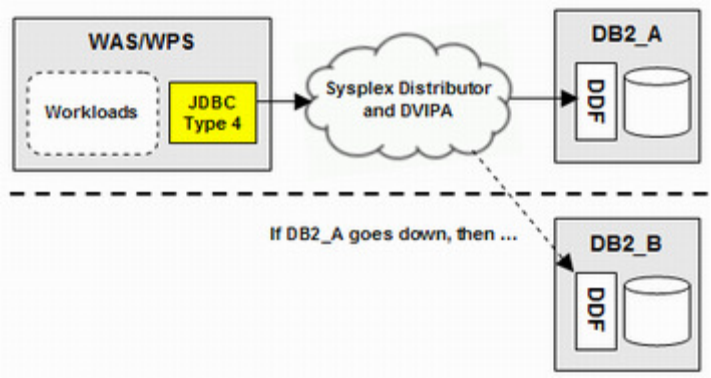
And that is a *total system design* thing. It is not just the data connector used from WAS.

We can't possibly address the full HA story here ... we'd turn this Co-Location white paper into a 900 page book. But we'll touch on some key things.

Let's work through several things so we can level-set some understandings:

Why many view T4 as offering higher availability

It has to do with the IP connection between the resource adapter and the target data resource. The picture in their minds is this:



And it is true that if DB2_A were to go down, then the connection between that JDBC resource adapter and DB2 would be broken, prompting a reconnection attempt. If Sysplex Distributor is configured properly, that reconnection would flow to the other DB2 subsystem in the data sharing group.

What's not necessarily available even after reconnect

Once the reconnection is established, you do have access to DB2 once again.

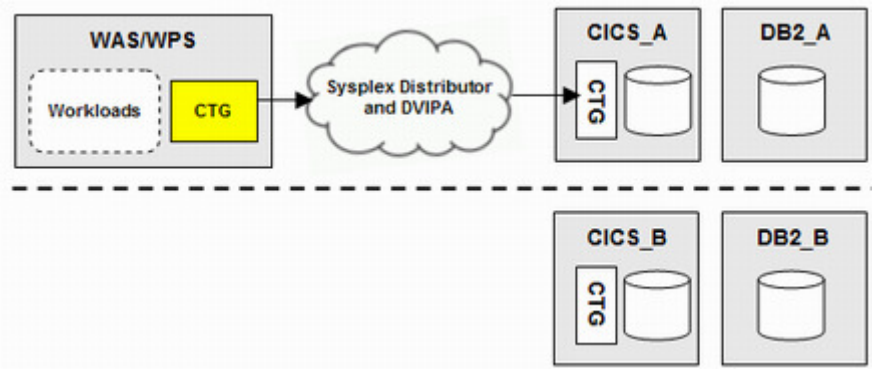
You do *not* have immediate access to any data locks held by the lost DB2 instance until that DB2 instance has been restarted, either on the original LPAR or on another LPAR in the Sysplex.

Any in-doubt transactions will not be resolved until that restarted DB2 subsystem reconnects to RRS and resolves them.

Yes, *other* work can be done. That's the appeal of this design. But it does not provide a way to overcome the two things just noted. And it does not allow you to participate in the local connector things covered under "Operational Benefits" on page 14.

What about WAS talking to CICS or IMS where there's a dependecy on DB2?

Consider this picture:



Imagine that DB2_A goes down. *As this picture is drawn*, the TCP connection between WAS and CICS will *not* be torn down. WAS will not necessarily be aware that CICS_A has lost its supporting DB_A.

Please... We know there are ways around this, for example configuring a CICSPlex and having WAS connect first to a TOR, which then routes within the CICSPlex. The CTG Gateway Daemon in CTG V7 makes use of the WLM health API to potentially signal back the loss of the DB2 and tear down the connections. That simply makes the earlier point about how HA is really a larger, total-system consideration.

How Sysplex Distributor really works

Sysplex Distributor is a *TCP connection placement mechanism*. It is *not* a request-by-request routing mechanism.

What that means is that once Sysplex Distributor establishes a TCP connection, that connection will be used until it is torn down. If nothing tears it down, then Sysplex Distributor will continue to pass requests from that client up that connection ... regardless of whether that's the right thing to do at that moment.

For example, if WAS is directly connected to DB2 (no CICS) the failure of a DB2 instance will certainly result in the tearing down of the TCP connections and the signal to attempt reconnect. Sysplex Distributor will most certainly work well to establish a reconnection to a surviving DB2, provided things are configured properly.

But if for whatever reason the outage does not cause the tear-down and reconnect of the TCP connection (the DB2 failure behind CICS example from earlier), there is a *possibility* that the use of T4 with Sysplex Distributor did not provide the HA you had hoped.

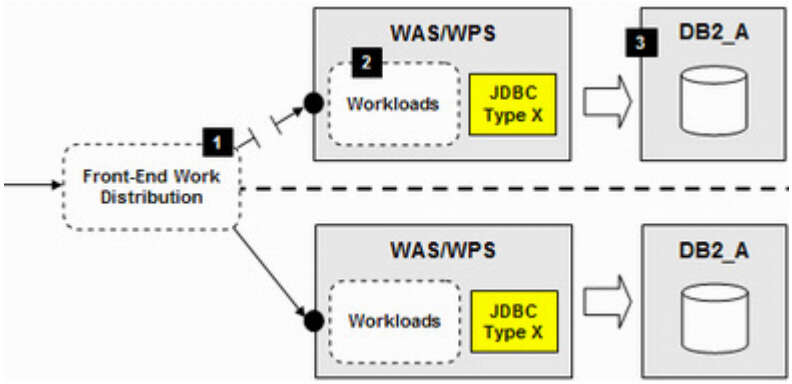
Planned outages versus unplanned outages

Using a T4 connector between WAS and the data resource provides a *degree* of additional protection against *unplanned* outages. That much is undeniable. Of course, it begs the question just how often a DB2 subsystem goes down separate from the LPAR going down. And of course if the whole LPAR takes a hit, then WAS itself goes away. T2 or T4 doesn't matter at that point. High availability is then provided further upstream, in whatever front-end routing and switching mechanism you have in place.

But if T4 is being used to offer greater flexibility to *planned* outages, then we'd ask that you think carefully about that.

If you are planning on rolling maintenance through your DB2 subsystems in the data sharing group, it is a best practice to view the WAS instance and the DB2 subsystem as a single logical unit. Either both are working, or both are quiesced and flushed of work.

You should *not* simply take out the DB2 behind WAS without *first quiescing the inbound work to WAS*, and allowing all in-flight work to complete:



1 At the front-end distribution device (or using the WebSphere `PAUSELISTENER` command), insure that new incoming work is routed *away* from the WAS/DB2 pair in the LPAR

2 Monitor the work in WAS to make sure all the in-flight work finishes up. (The `DISPLAY WORK` command can help with this.) WAS is essentially quiesced at this point.

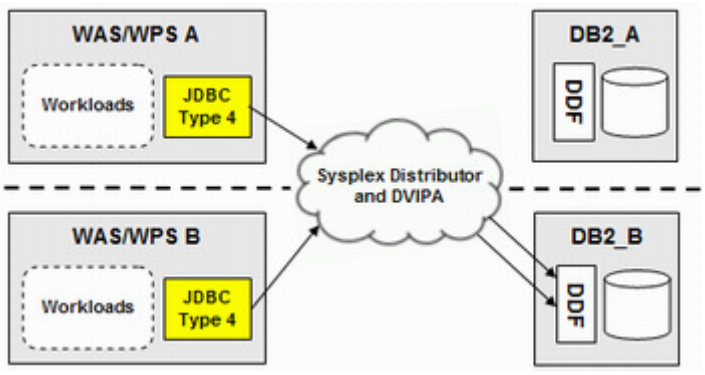
3 *Then* shut down the supporting DB2 subsystem

Relying on a T4 connection for planned outages and simply taking out the DB2 introduces a lot of unnecessary complexities – in-doubt transaction recovery, held data locks and processing overhead of XA recovery algorithms (RRS is *much* better at recovery). Planned outages should *always* be done in a controlled and orderly fashion.

Do you see the main point here? If you follow that recommended procedure, then using T4 provides *nothing*. In that case you should use T2 and enjoy the other benefits we described under “Operational Benefits” on page 14.

What happens if all my WAS T4 connections get pinned to one DB2 subsystem?

It can happen. It’s not common, but it *can* happen. Here’s the picture:



Loss of DB2_B and your whole solution is gone. And you may not even realize it until users start reporting a lot of errors on their browser screens.

Please...	Again, there are mechanisms to avoid this. We’re just showing what’s possible if care is not taken, and the full implications of HA are not explored.

System automation

One of the strategies employed by many is to use system automation tools to detect and quickly restart any failed data component. As we mentioned, it's important to get those subsystems back to free held data locks and to resolve in-doubt transactions. Restarting them quickly *somewhere* is key to high-availability.

Restarting them on another LPAR is possible. It implies having to stop and restart them eventually back on their "home" LPAR.

Restarting them quickly on the *same* LPAR, if possible, allows for a quick reconnect with local adapters.

An issue that comes up is stopping work to the still-operating WAS servers when the locally connected backend data subsystem drops. There's a modify command issuable against a server controller region called `PAUSELISTENERS` which will do just as the name implies. When the modify is issued against a controller, all the listeners on that controller are closed.

That's key because that results in existing TCP connections being torn down. Front-end distribution devices such as Sysplex Distributor or the WAS Plugin will see that and cease routing there. With a properly configured WAS cluster environment, inbound work continues to the surviving cluster member.

System automation could be used to detect a data system outage, issue a restart for that failed data subsystem, and issue `PAUSELISTENERS` and `RESUMELISTNERS` to suspend and resume work flowing to the cluster member.

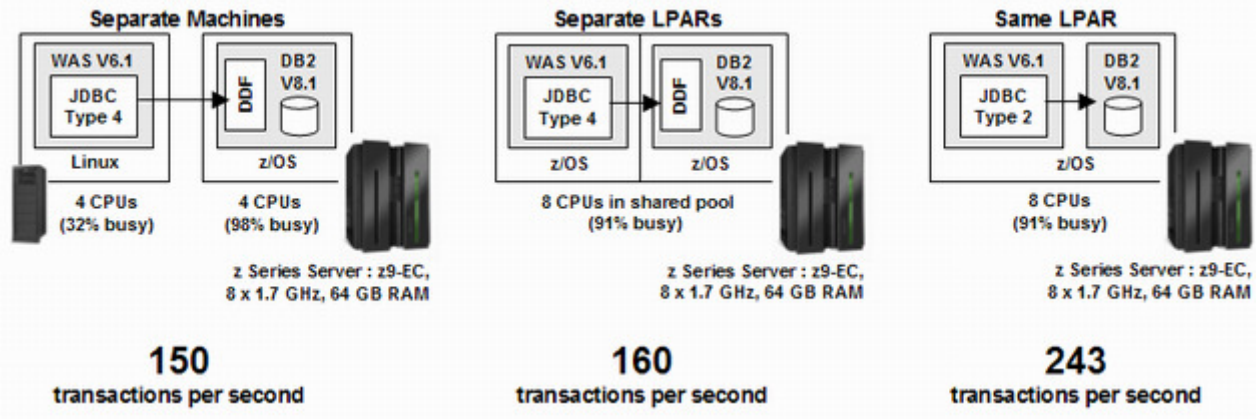
Conclusion: the high availability question

We close this with several key messages:

- There's a *lot* that goes into HA; not *just* the data connector.
- Evaluate your true HA needs against the value of that HA to the business.
- Please keep in mind the important concept of WAS and its associated data subsystem as a logically paired unit. It's not a technical requirement, but it makes viewing the issue of careful coordination of planned outages easier.
- Please weigh the incremental HA value of a T4 adapter against the benefits lost, such as access to RRS for transaction recovery, single thread of execution, managing to a single WLM goal, etc. Those were the things we outlined under "Operational Benefits" on page 14.
- Incorporate system automation into your HA design strategy.
- Finally ... ultimately ... do what's right for your business. At the end of the day that's what we care about. That may sound like marketing talk, but please believe us ... IBM literally teams with people who really do care a great deal about your success.

Brief summary of a banking scalability study

This study used an online banking system as its workload. The objective was to measure the scalability potential of co-location. The results for this particular test are summarized here:



This is one test, and as we've stated before, results vary depending on many factors.

For this test co-location showed a clear scalability benefit. This is demonstrating the factors mentioned elsewhere in this document: cross-memory communications; elimination of network latency; single thread of execution; elimination of SSL overhead; elimination of serialization overhead.

Multi-Row Fetch (MRF)

The following information is found at this URL (which is split here for readability)

`http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic`

`/com.ibm.db29.doc.java/com.ibm.db2.luw.apdv.java.doc/doc/r0053795.htm`

Properties that control the use of multi-row FETCH

Before version 3.7 and version 3.51 of the IBM Data Server Driver for JDBC and SQLJ, multi-row FETCH support was enabled and disabled through the useRowsetCursor property, and was available only for scrollable cursors, and for IBM Data Server Driver for JDBC and SQLJ type 4 connectivity to DB2 for z/OS. Starting with version 3.7 and 3.51:

- For IBM Data Server Driver for JDBC and SQLJ type 2 connectivity on DB2 for z/OS, the IBM Data Server Driver for JDBC and SQLJ uses only the enableRowsetSupport property to determine whether to use multi-row FETCH for scrollable or forward-only cursors.
- For IBM Data Server Driver for JDBC and SQLJ type 4 connectivity to DB2 for z/OS or DB2 Database for Linux, UNIX, and Windows, or IBM Data Server Driver for JDBC and SQLJ type 2 connectivity on DB2 Database for Linux, UNIX, and Windows, the IBM Data Server Driver for JDBC and SQLJ uses the enableRowsetSupport property to determine whether to use multi-row FETCH for scrollable cursors, if enableRowsetSupport is set. If enableRowsetSupport is not set, the driver uses the useRowsetCursor property to determine whether to use multi-row FETCH.

Summarizing:

- Type 2 uses `enableRowsetSupport` to determine if multi-row FETCH (MRF) is employed. And by default that is *not* set. Hence by default MRF is not enabled for Type 2. *But it can be enabled using that property.*
- Type 4 uses a combination of `enableRowsetSupport` and `useRowsetCursor`. If `enableRowsetSupport` is enabled, then MRF is used. If `enableRowsetSupport` is not enabled, then the driver checks to see if `useRowsetCursor` is enabled. By default that it is; hence by default MRF is enabled for Type 4.

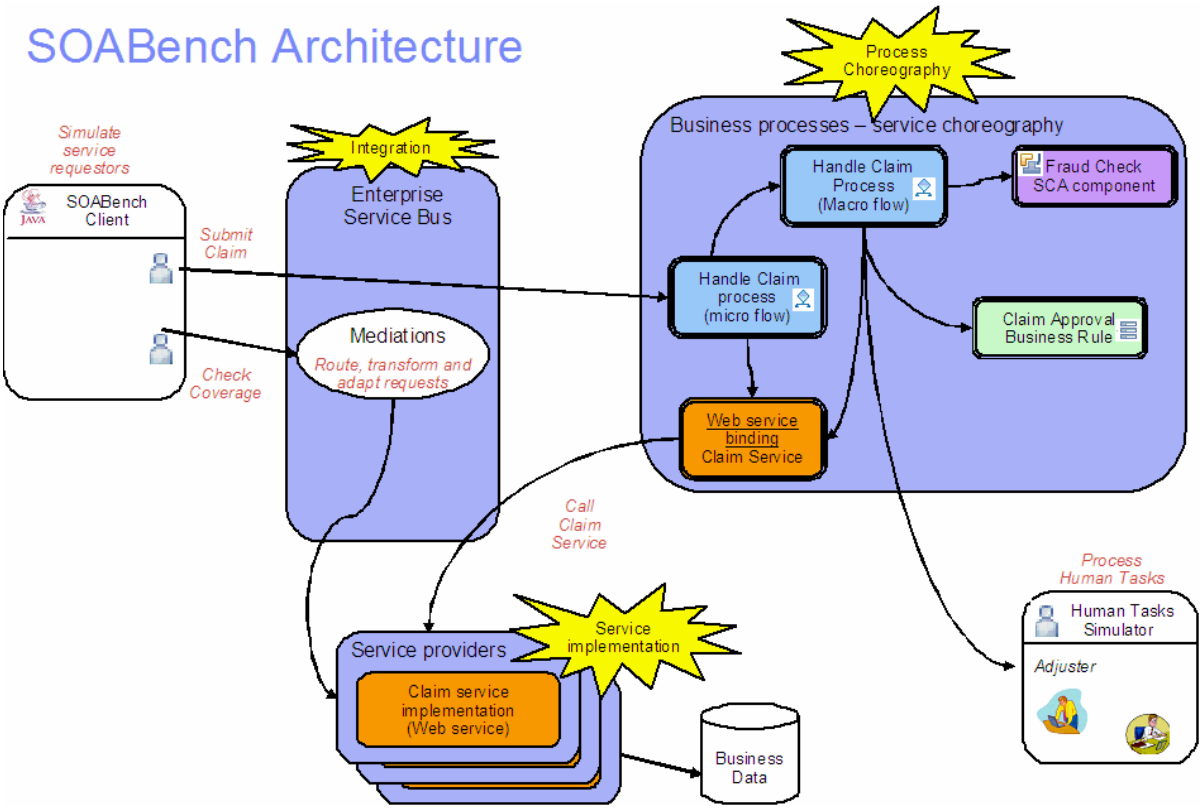
Appendix B – Workload Description Details

Details of "April 2009 – JDBC Type 2 vs. Type 4 Study"

SOABench Overview

The SOABench workload used for this report is an implementation of the SOABench2005 specification and models the business processes of an automobile insurance company. SOABench is intended to evaluate the performance of a distributed application implemented using a Service Oriented Architecture (SOA). SOABench is a macro-workload whose driver produces a complex workload similar to a real production system. The complex driver workload is made up of several subset technologies called facets which can be included or excluded from performance evaluations. Examples of SOABench facets include Services (use of service components), Mediation (use of mediation to transform requests and responses), and Choreography (application implementation using service choreography).

By combining facets, SOABench implements 2 aspects of the IT systems of an insurance company called "SOAAssure". The first is the Claims application which combines the Choreography and Services facets to process insurance claims. The second is realized using the Mediation and Services facets and provides a third-party gateway which enables another company to establish whether coverage exists for an existing policy. The following diagram illustrates the workload architecture flow.



The SOABench Client can drive the workload with mediation or business process claim requests. The minimum request and response size is 3 KB but this can be increased by the user. The client driver also provides for an "infrastructure" mode to make interactions with the backend Service providers trivial. The Human Tasks Simulator handles both adjuster and underwriter tasks generated during the Choreography facet manual approval process.

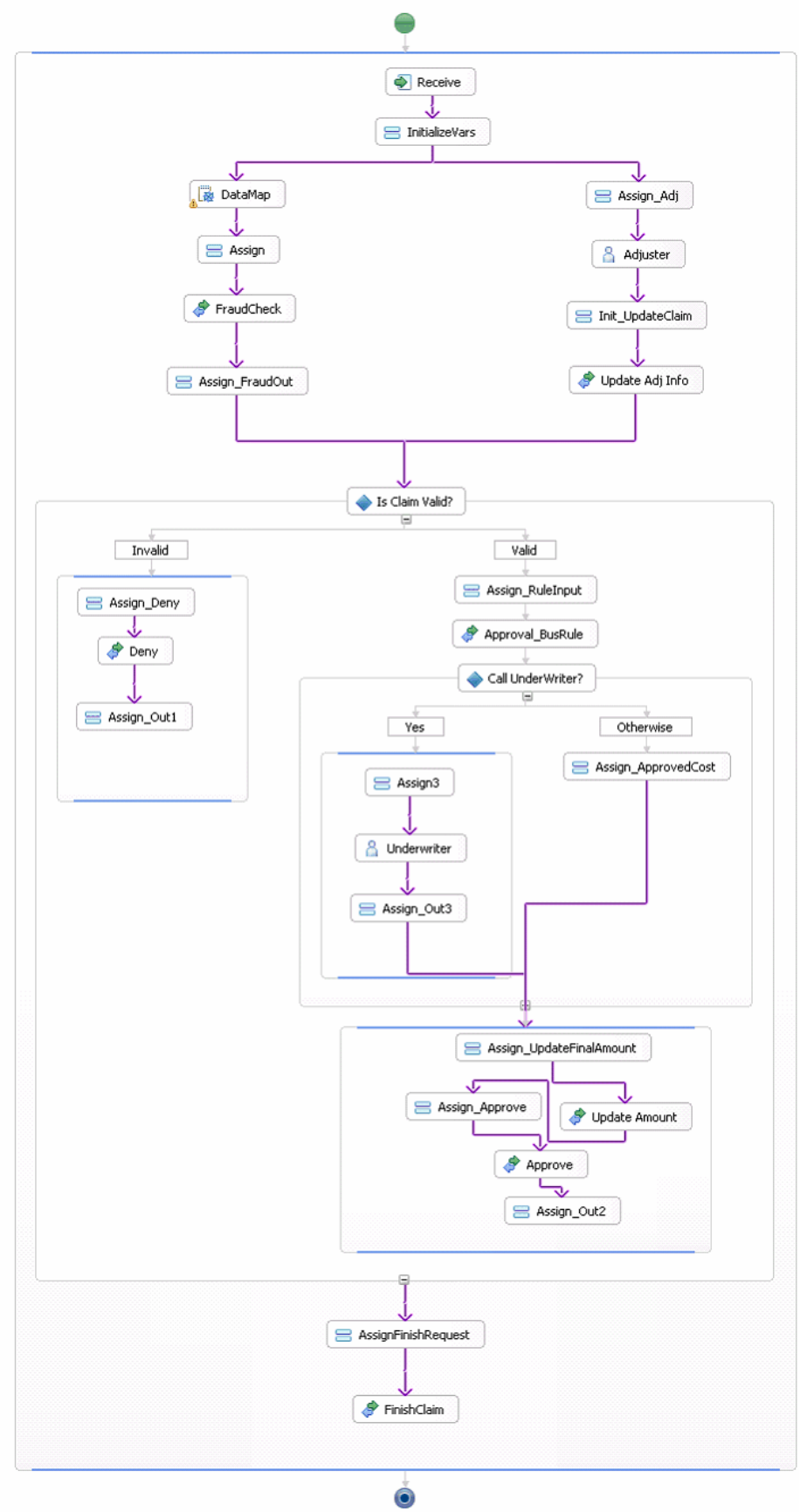
Choreography facet: Manual Approval - SOABench Handle Claim Macro2

Another workload in the SOABench Choreography facet is the handling of an insurance claim using manual approval. Depending on claim amount, either 1 or 2 human tasks are performed. For data in this report the second task occurs for 40% of claim requests. The workload starts in the process used in the Automated Scenario (a microflow), as described in the previous section. A claim request is sent to the process which performs HandleClaim. HandleClaim does “Submit Claim” to create the claim, skips the check claim for validity, then calls a long running (macroflow) process to perform more work on the claim.

The long running process does a fraud check on the claim via “FraudCheck_SCA”. A claims adjuster also looks at the claim via the “Adjuster” human task and the claim is updated through a web service call. For the workload measured, all claims are marked valid and then checked by a business rule to determine if an underwriter needs to evaluate the claim. Forty percent of the claims are checked by the “Underwriter” human task. At this point all claims are processed for claim amount and approved using 2 more web service calls. The current long running process then calls back the microflow process to perform the “FinishClaim” operation which performs a web service call to complete the claim.

An adjuster and underwriter simulator is used to process human tasks for the long running process.

The BPEL process is shown in the following diagram:

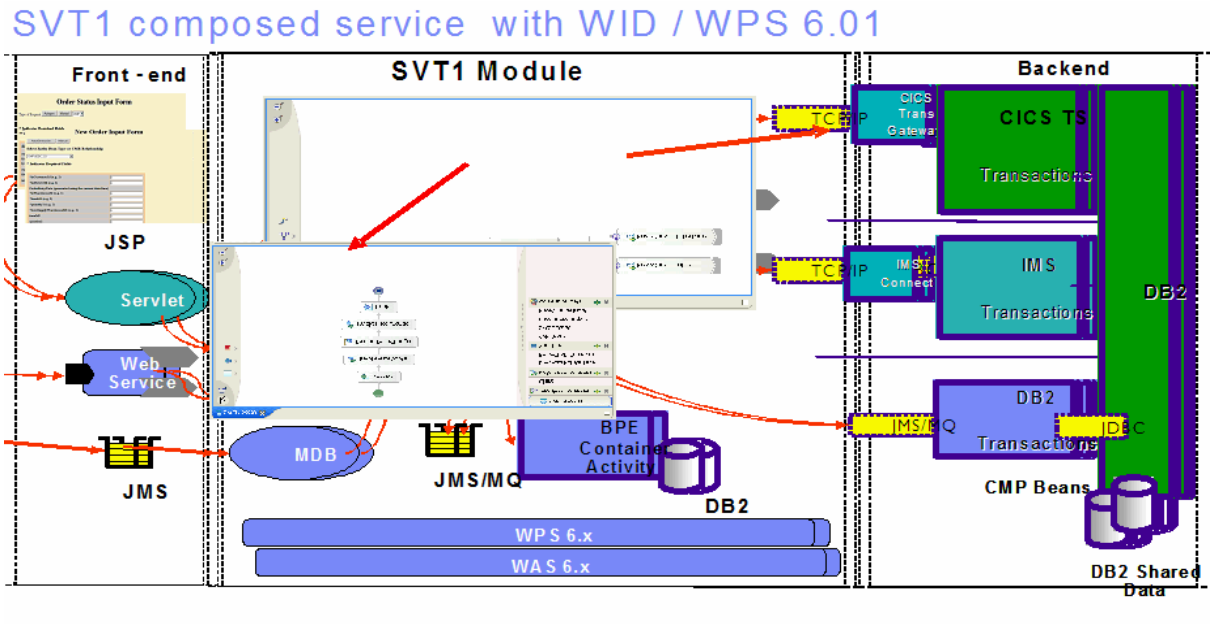

Receive
InitializeVars
DataMap
Assign
FraudCheck
Assign_FraudOut
Assign_Adj
Adjuster
Init_UpdateClaim
Update Adj Info
Is Claim Valid?
Invalid
Assign_Deny
Deny
Assign_Out1
Valid
Assign_RuleInput
Approval_BusRule
Call UnderWriter?
Yes
Assign3
Underwriter
Assign_Out3
Otherwise
Assign_ApprovedCost
Assign_UpdateFinalAmount
Assign_Approve
Update Amount
Approve
Assign_Out2
AssignFinishRequest
FinishClaim

ERWW order processing core workload on z/OS

The ERWW Production workload is a well known Order Processing application, enabled in WAS z/OS, which simulates customer business scenario. This existing application was used to create a “composite application” using SOA principals with WPS for z/OS V6.2. The ERWW Order Processing System serves as the principal “customer like” application and workload for validating new releases of WAS z/OS. It is also used cooperatively by many z/OS development teams to drive z/OS workloads and benchmarks on System z hardware.

ERWW application topology

General Application Topology for ERWW is as shown below, ERWW Order Processing functions cast as SCA modules.



PVT 8 NewOrder process (Macroflow) – z/OS

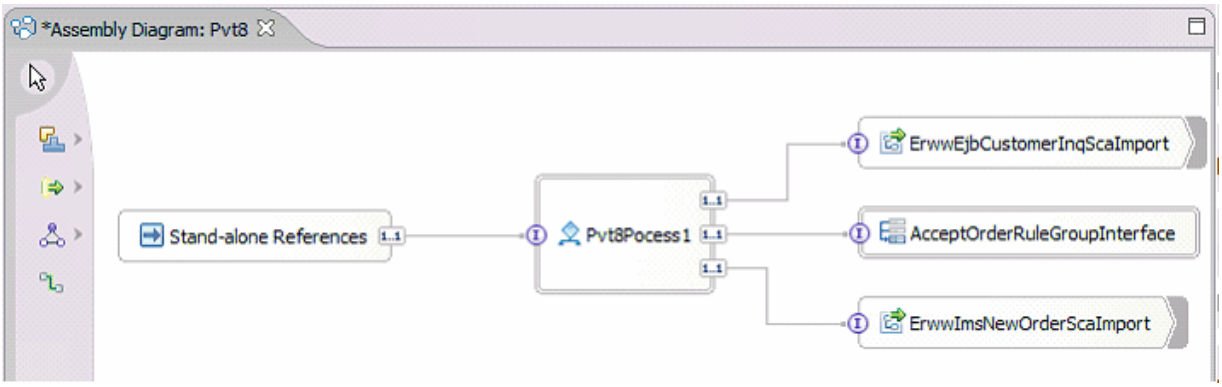
PVT8 - Long Running scenario (Macroflow), is a lighter weight macroflow that performs the following: A new order request message is received. From the NewOrder information the customer information is extracted and mapped to a Customer Inquiry request message. The CustomerInq service is invoked and detailed customer information is returned from the service.

A choice node is used to determine if the customer has a "Good" or "Bad" credit rating. If the customer's credit rating is "Good” the NewOrder service is invoked and the order is entered into the system. If the order is entered into the system successfully the process terminates. If the customer's credit rating is "Bad" a BusinessRule is invoked to determine if the order can be processed based on the customer's account balance (acceptable balance threshold).

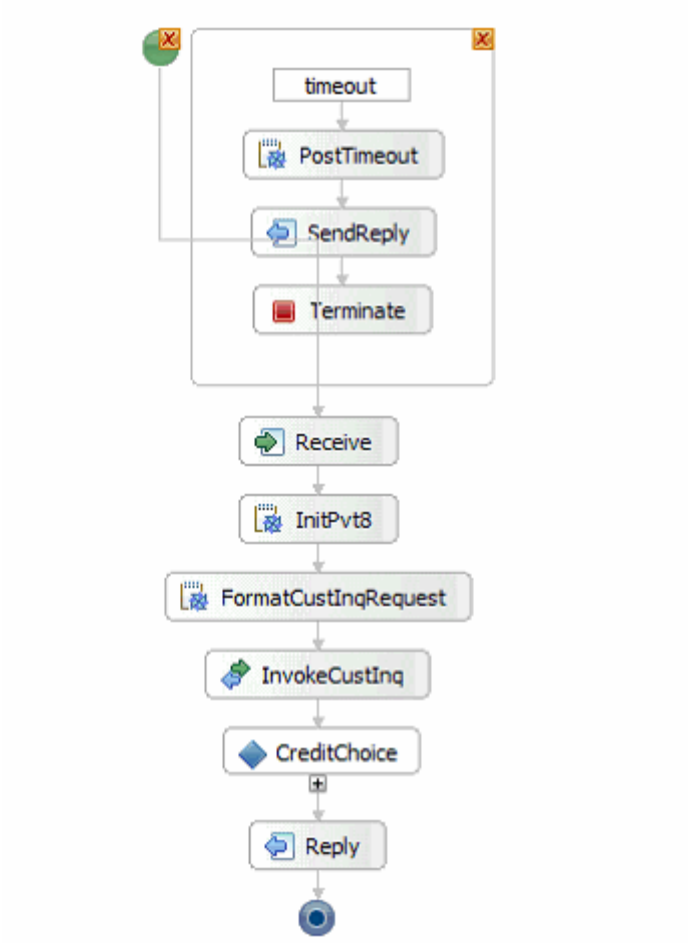
If the customer with bad credit has an account balance more than the acceptable threshold, the order is directly entered into the system by invoking the NewOrder service. If the order is entered into the system successfully the process terminates. If the customer's account balance is less than the acceptable threshold, a HumanTask is invoked so a human can make a decision to accept the order or not. If accepted, the order is entered into the system by invoking the NewOrder service and terminates. If rejected, the process bypasses the NewOrder service and terminates without entering a new order into the system. For measurement purpose this scenario also includes an application (not shown) to claim and

complete the Human approval task automatically. The balance threshold is set at a value so that all customers with a bad credit rating require Human approval, and the automated task always approves the request so that the NewOrder is processed. The GoodCredit/BadCredit ratio is about 90/10.

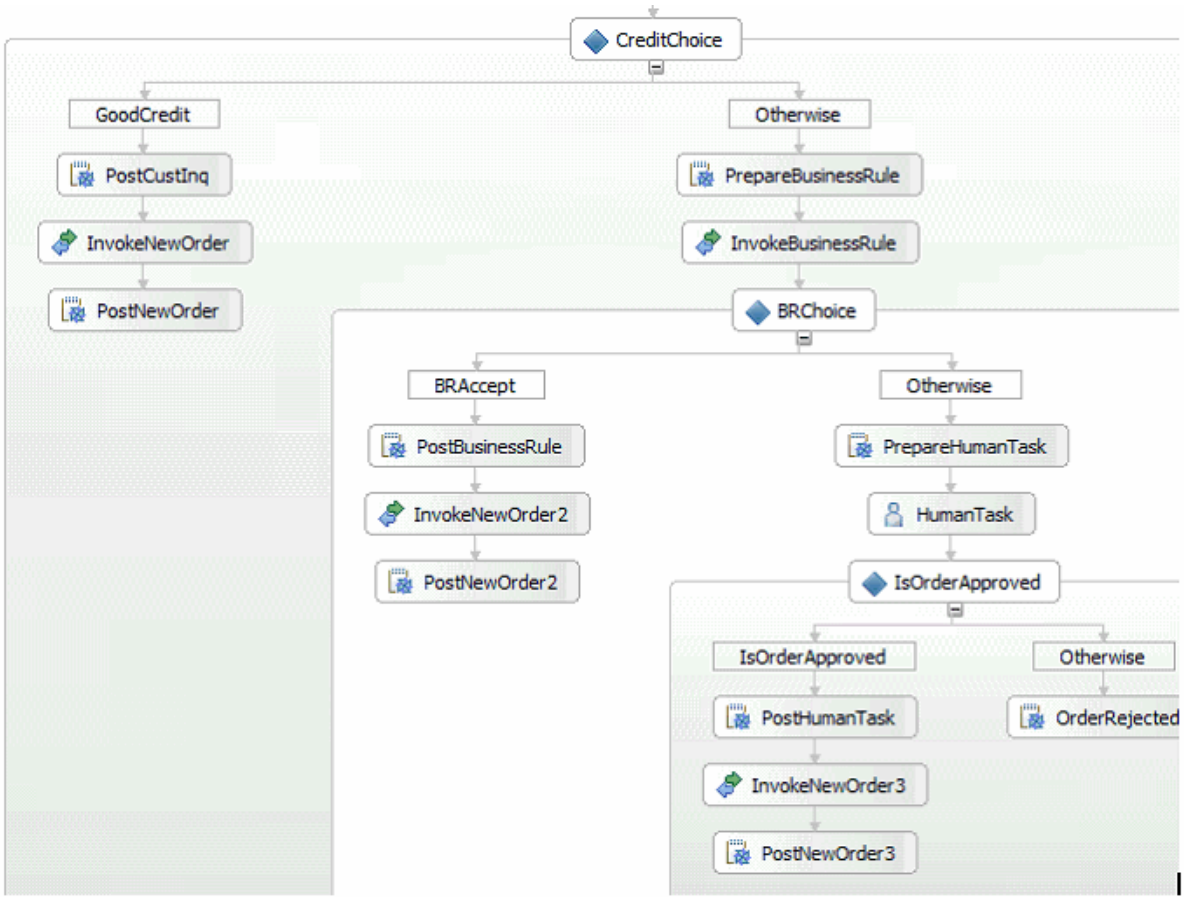
The PVT8 Assembly diagram is as shown below,



The BPEL process is shown in the following diagram:



And the CreditChoice assembly diagram:



Document Change History

Check the date in the footer of the document for the version of the document.

May 27, 2009	Original document.
May 28, 2009	Various typographical error fixes, as well as some detail added to the section on the benefits of RRS.
June 4, 2009	Updated the beginning of the "April 2009 - JDBC Type 2 vs. Type 4 Study" on page 6 to better explain the background and design of the study and give proper and deserved credit to the sponsors of that study.
June 18, 2009	Provided a section on the question of high availability using local connectors versus IP-based connectors. It's not a comprehensive treatment of the HA question, but we hope it will address some of your questions. See "Local adapters and high availability" on page 18.
July 2, 2009	Added a comment about the use of `PAUSELISTENERS` and `RESUMELISTENERS` modify commands coupled with system automation as part of an overall HA strategy.
July 15, 2009	A few tweaks related to the High Availaiblity section. Mostly to clarify some ambiguity in my original wording.
November 2, 2009	Previous editions of the Techdoc made a confusing reference to the patched level of the Java SDK used for testing. The specific level of the SDK used for the internal testing is of secondary importance to the level of SDK that provides the functionality in a generally available sense. And that SDK is the one that ships with 6.1.0.23 (or 7.0.0.1).
November 11, 2009	Clarified further how one can tell if the Java SDK being used by a **non-WAS** environment has the Type 2 offload functionality incorporated into it. See "Determining if the offload capability is present for a non-WAS Java environment" on page 17 for the details added in this edition of the document.

End of WP101476