**Micro Kernel – Integration framework**

**Apply Micro Kernel pattern for messaging middleware**

**Document Control**

The completion of this sheet represents the document history.

**Document Modification Control:**

| ***Date*** | ***Author*** | ***Role*** | ***Version*** | ***Comments*** |
| --- | --- | --- | --- | --- |
| 30/06/14 | Vinoth Krishnamoorthy | Accounting Hub SME | 0.1 | Initial Draft |
| 08/07/14 | Vinoth Krishnamoorthy | Accounting Hub SME | 0.2 | Updated and published for internal review |
| 16/07/14 | Vinoth Krishnamoorthy | Accounting Hub SME | 0.3 | Reviewed, internal comments addressed |
| 30/07/14 | Vinoth Krishnamoorthy | Accounting Hub SME | 0.4 | Published for external review |

**Review:**

| ***Date*** | ***Reviewer*** | ***Role / Purpose of review*** | ***Version*** | ***Comments*** |
| --- | --- | --- | --- | --- |
| 08/07/14 | Satish Shivappa Mudenavar | Accounting Hub ADM | 0.2 |  |
| 08/07/14 | Krishna Dattatreya | Accounting Hub Architect | 0.2 |  |
|  | Stuart Mason | Accounting Hub Senior SME |  |  |

**Document Approval/Signatures:**

| ***Date*** | ***Approver*** | ***Role*** | ***Version*** | ***Signature*** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

**Distribution List:**

| ***Date*** | ***Name*** | ***Role*** | ***Version*** | ***Comments*** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

[1. Abstract 6](#_Toc394676544)

[2. Existing State and Opportunities 6](#_Toc394676545)

[3. Benefits of Enhancements 6](#_Toc394676546)

[4. Architectural pattern – Microkernel 7](#_Toc394676547)

[5. Microkernel - components 7](#_Toc394676548)

[6. Accounting Hub – proposed components 8](#_Toc394676549)

[7. Accounting Hub – proposed block architecture 9](#_Toc394676550)

[8. Accounting Hub – proposed sequence of flows 10](#_Toc394676552)

[9. Microkernel – how to implement? 11](#_Toc394676553)

[10. How to implement with existing WMB v7? 11](#_Toc394676554)

[11. How to implement with IIB v9? 12](#_Toc394676555)

[12. Challenges and Key considerations 12](#_Toc394676556)

[13. Functional Requirements 12](#_Toc394676557)

[14. Non Functional Requirements 13](#_Toc394676558)

[15. Assumptions 13](#_Toc394676559)

[16. Risks 13](#_Toc394676560)

[17. Out of Scope 13](#_Toc394676561)

[18. Design, Build, Test and Release approach 13](#_Toc394676562)

[19. High Level Timelines and Estimates 14](#_Toc394676563)

[20. Enabler for Platform Enhancements: Upgrade software currency to IIB v9 15](#_Toc394676564)

[21. Appendix 16](#_Toc394676565)

**AHUB – Accounting Hub**

Accounting hub is one of architectural building block part of Transaction Management Hub (TMH). The *target* state of Accounting Hub will be the single entry point for all Payment Engines to all Product Platforms (accounting systems like CBS, CAP, COMMON, etc.). The Accounting Hub will give a singular view of all services and offerings of these Product Platforms.

# Abstract

The objective of this paper is to introduce ***Micro Kernel*** pattern as the foundation framework for a typical messaging middleware that process online service requests and provide standard functions of validation, routing, enrichment, mapping and orchestration of transactions. Accounting Hub in particular is chosen as a classic use case to apply this Micro kernel pattern for various reasons – extensible architecture, already componentized framework for accounting hub and reduces time-to-market cost of adding new interfaces to product platforms from accounting hub.

# Existing State and Opportunities

* Accounting Hub framework has been built now with 3 business services hosted on top of the loosely-coupled, SOA based framework. The current framework has the ability to add product platform posting maps extended from a generic posting or enquiry controller.
* The infrastructure supporting AHUB is ready & proven and as part of pilot Account Information Service deployed into production environment
* Interest posting service has been deployed into production environment and in truly “live” state with operational support aligned
* Account posting service enhancements are delivered by TBT-SOC programme
* The Simplification project part of Resilience has got H-HBOS platforms integration with Accounting Hub that covers features like orchestration of transactions across settlement and posting accounts, compensation posting, etc. This is a good opportunity to implement a flexible architecture to cater for new requirements
* Forthcoming demand pipeline indicate requirements to include interfaces to heritage HBOS platforms like wCBS, TD01, IF-Lynx, IF-TD01, etc.
* The first project from Resilience will be an enabler for 5 more future projects. The timing is just right to bring in changes to platform that will allow for future extensions much simpler
* Considering the situation of constant changes, progressive building of capabilities, frequent changes to fundamental service functions (until the functions mature into stable state for payments), plug-n-play mode of adding interfaces to product platforms, phased on boarding of payment types, there is an opportunity to enhance the platform to an ***adaptable framework*** making it capable of participating in ***continuous integration*** delivery cycle.
* **Important Note**: The current architecture is already SOA based, loosely coupled set of services built using IBM Websphere Message Broker. The enhancement proposed here is primarily a driver to identify gaps and enhance the platform to as much resilient, adaptable and extensible as possible.

# Benefits of Enhancements

Some of the advantages identified in going for the changes

* *Reduce cost* *of development* and unit testing interfaces added by future projects
* *Increased resilience* of Accounting Hub services, as the components are simplified and the micro hub should abstract core system functions
* *Simplified code base* and refined set of framework components
* *Improved regression of changes*, as the core components will not be changed for adding interfaces
* *Highly scalable platform architecture*, with distinct components and functions to leverage improved processing performance
* *Extensible architecture* allowing for future enhancements to core functions in a modular pattern
* *Automated Testing framework –* The reorganised components allow for simplified regression test suite, which can be automated, though as a separate work execution independent of platform changes
* *Continuous Integration -* The changes will enable visionary strategy to build a platform that complies to cutting-edge “continuous integration” state, where by, a factory of events will drive continuous execution of change, SCM, build, deploy, configure and test the platform codebase automatically or event-driven.

# Architectural pattern – Microkernel

* Main objective of this paper is to propose an architectural pattern suitable for addressing the Resilience nature and futuristic vision of continuous integration – *MicroKernel*
* The *Microkernel architectural pattern* applies to software systems that must be able to adapt to changing system requirements. It separates minimal core functionality and customer-specific parts. The microkernel also services and a socket for plugging in these extensions and coordinating their collaboration
* This pattern, generally used to build operating systems and system software shells, is of late applied to application design as well.
* Encapsulate the fundamental services of application platform in a microkernel component
* Microkernel includes functionality that enables other components running in separate processes to communicate to each other; maintain system resources
* Internal servers, external servers, adapters, clients, microkernel form the core components of the pattern

# Microkernel - components

* Microkernel - Main component of the pattern, Encapsulates system specific dependencies, Maintain system resources
* Internal Servers - Extends functionality provided by Microkernel, Consider as extensions which are accessible to microkernel only
* External Servers - Different servers implement different policies for specific application domains, Receives service requests from client applications using the communication facilities provided by the microkernel, interprets these requests, executes the appropriate services and returns results to the clients
* Adapters - Protect the client from implementation details of the microkernel, Uses communication mechanism provided by the microkernel
* Clients - Application that is associated with exactly one external server (thru adapter)

# Accounting Hub – proposed components



* Interface Handler – This will serve as an external server, managing interfacing with external end point from AHUB platform. This could be both send and receive requests on either side of the flow processing through AHUB (e.g. IFW, AMD, CBS, COMMON, etc.)
* Hub Adapters – Encapsulate and hide all physical communication related aspects and provide core functions for message send and receive – WMQ handler, HTTP handler, SOAP handler. These core functions will be called by Interface handler
* Micro Hub – The central microkernel pattern for hub. This will be primarily a collection of frequently used functions that frame the core of the hub. Both Interface handler and Flow controller/ Map Router will use many functions provided by Micro Hub components. These form the minimal set of components required to host the Accounting Hub platform providing basic set of processing required for an online service hosting – validate, route, persist,
* Flow Controller – The central intelligence function that’ll orchestrate between multiple flows and components. The sequence of processing for a request / response flow is split into multiple events that drive each other by calling next step in flow-agnostic mode. This flow of events is controlled by Flow controller. This is an internal server deployment of Microkernel pattern. Note: The Flow controller should be able to control the next processing steps in *parallel* or *sequence* depending upon configuration.
* Map Router – Map Router is another internal server extending one or many core MicroHub functions to identify interface type, match online information with cached flow details and call relevant maps for transformation.
* Maps – these are extendable and independent components developed for each of the interfaces and contain mapping rules. Modifying mapping rules based on interface requirements should not impact overall deployment of flows and control of events. These are the key components aiding in adaptable and extensible platform. Any new interface or end point will require creation of multiple maps and not modify any of the other core components thus reducing build time.
* Policy configuration – this is a critical portion of the implementation. The policies could be
  + External service policies – one or more policies per consumer or per service calls, e.g. SOC enquiry, GPP enquiry could have separate set of policies on top of group of policies setup for enquiry and posting.
  + Internal flow control policies – internal policies determine the routing of flows by identifying next process steps, sequential or parallel, end-point URL, adapter type, etc.
  + Finally, the polices are stored and retrieved using cache (shared row variables) for MB flows to access them, reducing DB read/write

# Accounting Hub – proposed block architecture



* All three services hosted within AHUB will have no specific flows but built upon set of commonly reused flows. Actually there will be no distinction between different services. The components will serve agnostic of service types
* However, with help of policy configuration, it should be possible to turn on and off all the services independent of each other.
* Both request and response flows will follow a similar pattern – receive, flow controller, validate, persist, audit & exceptions, map router, send to end-points.
* Policies configured will drive the flow of events between individual components
* The number of processing steps, sequence or parallel execution and other flow processing requirements will be available in broker cached area for online flows to act upon.
* What’ll be the next step after completing a single processing step is picked up dynamically from the configuration and matched with interface type / online information.
* After completion of a processing step, the control of execution returns to flow controller which will call the next step or component.
* Interface handler will serve like mirrored entry points to AHUB – both receive and send.
* The flows are agnostic of the number of end points connected to AHUB and direction of flow (request, response, outbound, etc.). Interface handler will effectively come in at both ends of a processing step.
* Interface maps are called according to various end points and interfaces being linked in a flow.

# Accounting Hub – proposed sequence of flows



* The flow of events is explained as above with request flow to CBS as an example.
* Flow controller decides the next step dynamically and will call relevant components accordingly.
* Flow controller will also decide the end of a flow or the last step of processing a request.
* For the response, similar sequence of events happens as that of request flow. Request and response flows are mirror images of each other in terms of intermediary processing steps.

# Microkernel – how to implement?

* Hub Adapters are sort of micro kernel components, but can be clubbed separately from the main functions and grouped together as adapters primarily managing connectivity with external systems.
* Set of sub-flows could be used provide standard MQ, HTTP, SOAP wrappers and called from main flows (Interface Handlers). On a request flow, these sub-flows help recognize, extract and strip interface headers. On a response flow, these sub-flows help add and wrap interface header in a standard method. Current AHUB sub-flows could be REUSED as is for this component.
* Map router will identify a suitable map based on key details like message type, direction of flow, target system, interfacing mechanism (MQ, HTTP, etc). There are various options possible to dynamically call an interface map identified as below –
  + Pub / Sub: Each of the interface maps will listen to a specific topic hierarchy and respond when Map Router publishes to relevant topic
  + Point-to-Point: All mapping flows are developed as main flows and triggered from Map Router via MQ
  + WTX Maps: Map Router can be implemented using WTX Maps. Map router will be a gateway calling other interface maps dynamically and returning to a Mapper flow (Preferred)
  + MB Maps: In case of v9 IIB, \*.maps or \*.subflows could be created and deployed independent of main flows (Preferred)
  + RouteToLabel: Retain existing mechanism of using WMB RouteToLabel for routing the flow to individual interface maps / sub-flows (Existing)

# How to implement with existing WMB v7?

* If the enhancements are planned with no changes to infrastructure and underlying product versions, there are certain important architectural decisions to be taken to implement dynamic calling of next processing steps and the interface maps.
* Dynamic flow management – with a marginal compromise to cost of performance and turn-around time the following could be adopted for flow controller. **Note**: Actually the turn-around time of using MQ hop is on average **5-8 times faster** than turn-around time of using sub-flows and labels. Refer to [Appendix](#_Appendix) for POC test results.
  + A single \*.msgflow project will serve as a container for each of Micro Hub and Adapters
  + All the reusable functions will be coded as sub-flows and included in the container with unique label assigned to each of those sub-flows
  + Flow controller will listen to an events queue and determine the next processing step to be called. This flow could then propagate the flow control to dynamically chosen sub-flow node (PROPOGATE function in v7 can directly transfer the flow control to a label, without needing to have RoutToLabel, each of the sub-flow identified by unique label names)
  + Caching can be made use to identify routing parameters
  + Another option is to fetch-once-use-multiple-times. The policy details are fetched at the first of receiving the transaction at interface handler.
  + The pre-fetched information is passed along through to next set of flows like flow controller via control headers (to be defined for internal AHUB purpose)
  + The flow controller may not even need to read from DB but will do from cached data sent by previous calling flow.
* Map router – there are two options possible with WMBv7, using WTX or using MB flows
  + Using MB flows will follow same pattern as mentioned for flow control
  + Using WTX maps will have a WMB Mapper flow introduce which will include WTX node and call a static “GatewayMap” which in turn will call any of the interface maps dynamically
  + Performance (turn-around time) and MIPS usage are major constraints for this option. This option can be considered if the platform road map indicates building many maps in 100s. If not, WBM option is preferred.

# How to implement with IIB v9?

* If the enhancements are planned post migrating the platform to IBM Integration Bus v9, there are many ways v9 features could be used as advantage.
* DFDL – Internal control headers and standard exception, audit, mapping formats can be designed based on DFDL. AHUB will have a generic DFDL format internally used by all the components
* Policy and Global cache – Policies could be configured using IIB features and global cache options used to have MB flows access cached data across EG and Brokers.
* Dynamic flow management – again there are multiple options using IIB v9
  + Similar pattern like v7 can be used as mentioned earlier
  + Decision service nodes could be used to dynamically control the flows using policy rules.
  + Decision service nodes allow for writing business rules right inside the broker environment with easier maintenance of rules. These rules will enable broker to act upon in-flight data and route or control flows accordingly.
  + IIB v9 will potentially simplify the flow controller component implementation.
* Map router – all the interface sub-flows are deployed as independent \*.maps or \*.subflows and shared in a common library. This library can be maintained independent of the main flows calling these library maps and also there is an option to maintain multiple versions of same maps across for interfaces.
* JSON and RESTful – Though JSON support is available right from v7, v9 comes with improved support and will be able to capitalize the RESTful webservice call to AMD depending on AMD getNPA() service implementation.

# Challenges and Key considerations

* The introduction of dynamic flow routing most likely will impact the performance of flows but only have a marginal addition to turn-around time.
* The flexibility of the newly proposed architecture comes at a cost which can be worked around in multiple ways to still retain and produce the same or even higher performance output from Accounting Hub
  + Increase in additional instances of critical MB flows will ensure the same X TPS of each services is achieved
  + The ability to deploy components independent of each other with different parameters allows for scalability. E.g. Interface handler components could be deployed multi-fold to receive and send transactions at higher rate. Flow controller will also be deployed with maximum possible additional instances.
  + Caching of policies and *RouteToLabel* for dynamic routing could save the increase in cost to certain degree
  + The virtue of allowing many of the processing steps to run in parallel will also contribute to reduction in overall turn-around time thereby restricting the delta cost to bare minimum
* Refer to the [appendix](#_//Accounting_Hub_–) for a snapshot of performance in a development environment with various options considered.

# Functional Requirements

Below are the high level requirements

| **Number** | **Description** |
| --- | --- |
|  | **General** |
| AHUB REQ001 | Adaptable architecture enabling continuous integration in near future. |
| AHUB REQ002 | Extensible architecture allowing for adding any number of interfaces and endpoints to AHUB in exponential manner, with lesser effort and repeatable tasks like a factory approach |
| AHUB REQ003 | Reduce effort and cost for new projects in building new components. Once the architecture is in place, the additional effort required will be only around building new interface maps |

# Non Functional Requirements

|  |  |
| --- | --- |
| Number | Description |
| AHUB NFR001 | The current NFR for AHUB from various existing projects remain the same. No changes perceived. The new re-arch should be able to support the existing NFR. |
| AHUB NFR002 | The potential delay or additional latency introduced as part of framework changes, especially intelligent routing and completely asynchronous split up of transactions, has to be proven as still satisfying original AHUB NFR with a marginal delay or the delay compensated with no visible changes to user experience. |

# Assumptions

| **Number** | **Description** |
| --- | --- |
| AHUB ASS001 | V9 upgrade is out of scope for this paper, but still, when considered together will enable the Platform Enhancements activities together saving cost and effort |
| AHUB ASS002 | Testing framework that is currently in place may undergo changes aligned to core platform changes. This paper is assuming test framework enhancements to be managed in parallel with main set of changes |

# Risks

| **Number** | **Description** |
| --- | --- |
| AHUB R001 | Risk of having to re-write functions that were previously tested in ST and SIT for existing 3 services, IPS in production. This risk will be mitigated by proper regression test plan and execution. |

# Out of Scope

| **Number** | **Description** |
| --- | --- |
| AHUB OS001 | File processing components are not considered to be part of core online processing of enquiry and posting requests. |
| AHUB OS002 | No changes to existing functions and maps within AHUB. The changes are around implementing of existing services using different components. |

# Design, Build, Test and Release approach

* There is no special requirement for build and delivery approach and will follow LBG standard DBTR.
* The Implementation / delivery will be dependent on on-boarding project to deliver the platform services.
* The existing AHUB services can co-exist in the same environment by differentiating WMB Brokers and Execution groups. New code base will be delivered on to the same platform and can run in parallel with currently deployed code base (irrespective choice of MB version – either continue with v7 or upgrade to v9)
* There should be an exercise of replacing the existing services with newly built framework and flows in a phased manner by switching traffic to new broker / EG with existing setup still active and slowly stop the existing flows once the new setup is completely active and running smoothly.

# High Level Timelines and Estimates

* The Estimates are initial ballpark with variance as +/- 10% for Study and +/- 50% for DBTR
* The estimates are valid for one month from the date released to project team
* The estimates are drawn independently of any other projects, assuming the overall cost of building, testing and delivering this newly proposed architecture will be handled separately as a work stream
* The estimates consider reuse factor of existing flows and mapping. Understanding is all of the interface mapping sub-flows can be reused as-is with 90-100% reusability. Some of the common functions like adapters, group controller, etc can also be reused up to 50-60%.

**Note**: The proposals below consider only on WMB v7, migration to WMB v9 will incur a marginally additional cost for reworking on DFDL message standards and broker implementation upgrade.

Grand Total Estimated effort = 50 Person days (FTE)

Grand Total Estimated cost = £17,500 blended rate of £350/day

**Approach 1a** – Big bang, develop all core components in single delivery

Grand Total Estimated effort = 240 Person days (FTE)

Grand Total Estimated cost = £XX blended rate of £YY/day (includes DBTR, delivery to ST; excludes Management overhead and testing support for ST, SIT, OAT, NFT, UAT)

\*\***Note**: If this core re-arch is not accounted for in next project, e.g. Resilience – the cost of Resilience will be slightly higher. E.g. the Resilience quote will be Hence it is better off delivering Resilience + Core-re-arch

**Approach 1b** – Big bang, develop all core components in single delivery, without Stand-in, without core maps (core maps will be done along with Resilience)

Grand Total Estimated effort = 155 Person days (FTE)

Grand Total Estimated cost = £XX blended rate of £YY/day (includes DBTR, delivery to ST; excludes Management overhead and testing support for ST, SIT, OAT, NFT, UAT)

**Approach 2a** – Phased delivery of re-arch, phase 1 – Hub Adapters

Grand Total Estimated effort = 46 Person days (FTE)

Grand Total Estimated cost = £XX blended rate of £YY/day (includes DBTR, delivery to ST; excludes Management overhead and testing support for ST, SIT, OAT, NFT, UAT)

**Approach 2b** – Phased delivery of re-arch, phase 2 – Core functions and mappings

Grand Total Estimated effort = 155 Person days (FTE)

Grand Total Estimated cost = £XX blended rate of £YY/day (includes DBTR, delivery to ST; excludes Management overhead and testing support for ST, SIT, OAT, NFT, UAT)

**Approach 2c** – Phased delivery of re-arch, phase 2 – Stand-in and Retry

Grand Total Estimated effort = 40 Person days (FTE)

Grand Total Estimated cost = £XX blended rate of £YY/day (includes DBTR, delivery to ST; excludes Management overhead and testing support for ST, SIT, OAT, NFT, UAT)

# Enabler for Platform Enhancements: Upgrade software currency to IIB v9

**Proposal**

* Upgrade the Message broker setup for Accounting Hub from v7 to IIB v9
* Utilize standard migration scripts provided by IBM for enabling upgrade process
* If the upgrade is planned prior to re-arch, the changes for re-arch will be easier, permanent, and less prone to future changes (at least immediate changes from framework perspective)
* Migrate brokers using a single command, or create new brokers for phased migration
* No broker redeployment necessary when using built-in migrate command. All existing BAR files can be deployed to IB V9 brokers without change

**Reasons for upgrade**

* Timing: Whilst planning for the architectural change, it is much more easier and fit-for-purpose considering upgrade of software currency to much improved IIB v9
* Advantages: There are few key features of the latest version of software that’ll help organizing the components of newly proposed architecture in a much robust approach. Taking advantages of those features would align cohesively with Platform Enhancements.
* Opportunity: There are other project streams part of Resilience programme engaged in similar upgrade initiative. It is prudent to join the initiative based on similar currency landscape.
* One version to support: The support for Message Broker from Ops and Service delivery perspective will become easier aligning to same version of product across the platforms like STP, AHUB, etc.
* Support life cycle: IBM support life cycle states road map of v7 as below.

|  |  |  |
| --- | --- | --- |
| Program Number | Program Release name | Support Withdrawal date |
| 5724-J05 | IBM WebSphere Message Broker V7.0.x | September 30, 2015 |

**Features of IIB v9 that’ll enable Platform Enhancements**

Amongst so many new and improved features of IIB v9, the following key features will be directly put to use during re-arch. Platform Enhancements will be made easier and design future-proof if these are implemented

* Controlling Integrations with Policy
  + Provide intelligent mechanisms to control processing speed
  + Most common scenario is to reduce back-end server load
  + Set thresholds for integration data flow throughput
* Understand and Act on In-flight data
  + intelligent decision making; score then action in-flight request based on a business rule
  + Rules based business routing
* Graphical Transformations
  + Visually map and transform source and target data
* Global Cache Enhancements
  + IB contains a built-in facility to share data between multiple brokers
  + Improve mediation response times and dramatically reduce application load
  + Typical scenarios include multi-broker request-reply and multi-broker aggregation
* Independently deployable libraries
  + It is possible to deploy sub-flows independent of main flows
  + Potentially all interface mappings could be part of sub-flows deployed as libraries used across services and projects
  + Maps can be maintained externally and independent of core processing flows

**What next?**

* EAD: Include the upgrade into target model, start liaising with EA&D Infrastructure Payments contact for mobilizing the initiative
* ADM: Plan for migration, prepare and demonstrate POC, liaise with SD for support of product
* SD: Identify SD contact for AHUB, discuss upgrade initiative, obtain necessary support, liaise with ADM for planning, migration activities, scripts, deployment, etc.

**Points to consider**

* The key consideration is to identify a programme or project to fund the upgrade. This is very critical for moving this initiative further.
* There are areas of co-ordination required to connect with SD – Middleware team, EA&D – Payments and Infrastructure, Managing the initiative from plan and control perspective

# Appendix

Performance Statistics for various options of dynamic flow control.

**IIB v9 – Performance – Message throughput on zLinux**

<https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W37b629a0f7aa_4709_9506_bba2a096693d/page/zLinux%20results>

**Transforming a message by using ESQL use case results**

|  | **Non Persistent** | | | **Full Persistent** | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Msg Size** | **Message Rate** | **% CPU Busy** | **CPU ms/msg** | **Message Rate** | **% CPU Busy** | **CPU ms/msg** |
| **256b** | 15728.3 | 97.5 | 0.248 | 8115.4 | 78.0 | 0.385 |
| **2kB** | 7974.2 | 98.1 | 0.492 | 5294.2 | 88.3 | 0.667 |
| **20kB** | 1387.5 | 100.0 | 2.883 | 1185.8 | 94.1 | 3.174 |
| **200kB** | 139.6 | 100.0 | 28.659 | 129.3 | 95.7 | 29.612 |
| **2000kB** | 13.1 | 99.9 | 304.140 | 12.5 | 97.4 | 311.616 |
| **20000kB** | 1.3 | 100.0 | 3076.923 | 1.2 | 97.2 | 3239.667 |

**Message Routing use case results**

|  | **Non Persistent** | | | **Full Persistent** | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Msg Size** | **Message Rate** | **% CPU Busy** | **CPU ms/msg** | **Message Rate** | **% CPU Busy** | **CPU ms/msg** |
| **256b** | 20029.6 | 91.1 | 0.182 | 8707.9 | 71.5 | 0.328 |
| **2kB** | 16962.2 | 89.4 | 0.211 | 6800.6 | 56.2 | 0.331 |
| **20kB** | 4586.7 | 29.3 | 0.256 | 2277.7 | 24.9 | 0.437 |
| **200kB** | 406.7 | 8.3 | 0.818 | 352.7 | 12.5 | 1.420 |
| **2000kB** | 39.6 | 6.5 | 6.599 | 36.2 | 13.4 | 14.848 |
| **20000kB** | 4.1 | 10.7 | 104.767 | 3.2 | 15.6 | 193.416 |

**WMB v7 – Performance – Message throughput on Development zLinux server**

**Scenario I – Design using MQ Hops**

**Use Sub-flows & *RouteToLabel* for dynamic routing**

The average turn-around time using sub-flows and labels was **58812** x 10-6 seconds

**Scenario II – Design using Sub flows**

**Use multiple MQ hops and main flow for dynamic routing**

The average turn-around time using MQ hops was **6240** x 10-6 seconds