Introduction

This document outlines some ideas regarding “Business Transaction Management”, and how it may integrate with Hawkular components.

*“**Business transaction management** (BTM), also known as **business transaction monitoring**, **application transaction profiling** or **user defined transaction profiling**, is the practice of managing information technology (IT) from a business transaction perspective. It provides a tool for tracking the flow of transactions across IT infrastructure, in addition to detection, alerting, and correction of unexpected changes in business or technical conditions. BTM provides visibility into the flow of transactions across infrastructure tiers, including a dynamic mapping of the application topology.”* (Wikipedia)

Instrumenting the Business Application

In RTGov, the “business transaction flow” through SwitchYard was captured by registering an event listener that was informed of message exchanges, which were subsequently converted to activity events, pre-processed and sent to the backend server.

To enable business apps, using a wide range of technologies to be observed, the new BTM project will use instrumentation techniques (manual and automatic).

Manual Instrumentation

Annotation/Interceptors will be used to allow a user to manually identify methods that should be logged as part of the business transaction call stack, as well as components responsible for inbound and outbound communications with other servers that form part of the end to end business transaction.

Auto Instrumentation

Where manually instrumenting a business app is either not possible or desirable, then it will be possible to instrument the JVM to obtain the information implicitly. However to achieve this, it will be necessary to identify the relevant code to instrument. Analysing a business application will need to be performed in a number of steps:

1) Optional step to analyse all the technologies installed within a server (jvm). This can be used to optimize the following step, by excluding packages that do not need to be

instrumented. Proposed approach will be to use ByteMan pre-configured with a rule to collect information about all of the classes loaded.
2) Using information gathered from step (1), or based on knowledge of the server being analysed (e.g. wildfly, tomcat, etc), determine an 'ignore package' list. Capture information from JDI (debug interface), while business transactions are being performed, and build call trace information from them.
3) Using mix of heuristics, existing knowledge of instrumentation points from captured packages and manual guidance from an end user, identify the relevant paths and individual instrumentation points.
4) For information being transferred between components:
    a) If communication to external component, then identify correlation values that can be used to link up with continued execution of business txn in another server
    b) Extract relevant properties from the exchanged information, that can be used for later analysis

When an instrumentation profile for a particular technology has been defined (e.g. class/method *x.y()* should be instrumented to receive message exchanges from Camel), enable the user to record the details in a repository, to enable future instrumentation of apps using the same technology to be faster. Would also be good if such definitions could be submitted for inclusion in a shared repository, to enable reuse across organizations.

The instrumentation configuration for a particular business application may need to be derived from separate call traces obtained from different servers running in multiple environments. However the overall instrumentation configuration could be stored, associated with the business app, in Hawkular Inventory, as the rules can be loaded into all of the servers (i.e. ByteMan simply won't trigger rules if the class/method does not exist in the VM). However the benefit is that it does not matter if the decomposition of the business application, across infrastructure components, changes over time.

Once an instrumentation configuration for a business app has been defined, the next issue is how that configuration will be provided to the JVM. ByteMan enables two approaches:

a. Rule configuration is provided on the JVM command line - the limitation being that the configuration will be static for the life of the JVM, and must be available in the local environment to be referenced. This may be possible if (for example) the config could be provided as part of a Docker image of the business app.

b. Use of the listener port - enables remote configuration, which allows dynamic configuration. However the port is not currently secure - but also means a central management tool will need to contact the individual VMs to configure them.

Current thought is to extend ByteMan to enable dynamic configuration based on an embedded management component - so rather than use the listener port and worry about security, instead

the internal component would request the information from BTM (which would indirectly retrieve it from the Hawkular Inventory). This mechanism could also be used to push any updates to the VMs.

Capturing Business Transaction Flow

The instrumentation will be used to capture the business transaction flow. This will be achieved by recording the partial flow from various servers involved in the business transaction instance, and reporting these individual parts to the server, along with correlation information that can be used to reconstruct an end to end view.

For example, we may have a REST service providing a public API that can be used by external users to request a service. Instrumentation of various key points within this service, starting with the REST API, can be used to track the business transaction instance through some internal services to the backend invocation of another service via JAX-WS. At this point we need to record correlation information, obtained from the header or message content, that can be used to link the invocation of the web service, with the call trace captured by instrumentation of that web service. Additionally from within the service (e.g. this backend web service), we may wish to track calls to a database to understand how many queries are performed (within the scope of a business transaction instance) and how long they take.

Whichever instrumentation approach has been used (manual annotations or ByteMan based instrumentation), the information will be collated and pre-processed within the client JVM.

Due to the volume of information, various approaches should be explored to transfer this data from the client to the BTM server - balancing efficiency of transfer (using offline techniques) with need for near real-time feedback.

Dynamically changing the level of details being reported could be managed through information stored with the business app representation in Inventory.

Backend Server

The ‘partial call trace’ information will be delivered to the backend server via either a REST API (to enable users to report information directly) or via an appropriate collection mechanism from the embedded agent.

NOTE: We refer to ‘partial call trace’ as we are intending to deal with business apps that span multiple server/cloud/on-site boundaries, and therefore each server will provide only part of the

overall end to end business transaction instance (flow). However in the simplest case, where the business app is executed wholly within the scope of a single server, then the call trace will represent the complete end to end view.

Storing the partial call traces

The partial call traces will be stored against a unique id. They can potentially be stored in any database (e.g. Cassandra), however one of the main use cases (i.e. recovering the end to end flow) will require these partial traces to be retrieved based on (potentially multiple non-unique) correlation ids. Therefore, to facilitate such queries, the partial call traces will (also) be stored in Elasticsearch.

Call Trace Processing

Aim will be to provide a pluggable mechanism for processing the received partial call trace information. Some of the “off the shelf” plugins are listed below.

Derive metrics

RTGov currently derives response time information from the individual business transaction instances, performs analysis (e.g. SLA validation) and stores the information in Elasticsearch for further analysis by end users using Kibana. Kibana enables the user to focus in on subsets of information based on various (e.g. business) properties that have been extracted from the original activity information.

The same approach is planned for BTM - however the metrics (e.g. response times) would also be published to Hawkular Metrics, allowing them to be used to trigger Hawkular Alerts.

Derive latency

Use correlation information to determine latency of communications between different servers, reporting the results to both Elasticsearch and Hawkular Metrics.

Visual Representation

A query can be used to identify business transactions of interest - whether individual or group. Once the partial call traces are retrieved related to the query parameters, then we need to do further queries to obtain connected (correlated) partial call traces that have not yet been retrieved.

Once all partial call traces for a business transaction instance have been obtained, then a consolidated representation can be made available - e.g. to UI visualisation tool.

The actual visual representation will need further discussion, but it could be similar to ones used by profiling tools - however instead of viewing class/method details, it would be at a service/component level. Techniques could be used to colour code areas of concern, and enable navigation to IT resource representations used to execute the business transaction instance.