Enterprise Java Beans with ReactiveX.

Dr Bheemaiah, Anil Kumar , bheemaiaha@yopmail.com
A.B, Seattle WA 98125

Abstract:
This publication is on the automation of persistence, with the example of a Legacy solution, in the form of Enterprise Java Beans. While Java EE and EJB's are depreciated. They are bridged to the present models of SaaS and serverless computing, through the introduction of two new stateful beans, StreamBean and EventBean, for functional, reactive and Cloud based automated persistence with the ability to implement any design pattern template with the Bean. This paper is on the transition from imperative programming in SQL to a NoSQL based serverless design with automation of the persistence layer in the cloud.

What:

EJB's are a depreciated enterprise java concept, we revive EJB's for automated persistence with RMI to a Reactive framework with the addition of observer, iterator and functional lambdas. The addition of programmable event and media streams with iterators to create stream beans with lambdas , observer beans and programmable iterators on stream beans.
RMI has been replaced with a FaaS programming framework and there is automated persistence to the cloud from the container.

How:

We use the AWS cloud with the Lambda and LambdaEdge framework to integrate to streamBeans in addition to singleton, observer, stateful and stateless bean pools. streamPools are automatically persisted from the container to DynamoDB in JSON form.

So What:

The revival of streamBeans, adds Logging, Exception Handling and Queues to observer beans with automated persistence, adding functionality to a fluidic java and JS implementation of Java OOP designs in an automated three tier design. The database persistence being automated and the backend consisting of the Lambdas defined in analogy to ReactiveJS programming. This forms the basis for a natural programming framework, like RAVA.

Keywords: ReactiveX, Java, EJB, StreamBeans, EventBeans, Automated Persistence.

About EJB's and Design Patterns.

EJB's are part of Java Enterprise Edition, and are part of legacy solutions, now depreciated. Java Beans were the first attempt at automated persistence, in three or n tier architecture, with a shift towards cloud computing, many of the EJB functionality is depreciated, including remote method invocation, microservice based architectures.
Reactive programming adds design patterns to event and data streams, leading to a new instantiation of a bean, a StreamBean and an eventbean, and any kind of pattern applied to it in cloud based serverless architecture with fully automated persistence. OpenFaaS is backend and persistence as a service, infrastructure is code.
The revival of functional computing in the cloud, along with reactive computing and the bridging of Java based OOPS thinking in persistence on the cloud is the central theme of this paper.
Functional Programming in Java.(Saumont 2017; Subramaniam 2014; Maiorano 2014; Appel and Palsberg, n.d.; "Lesson 13: Lambda Expressions and Functional Style Programming" 2015; Warburton 2014)
RMI mechanism in Java Beans.(Öberg 2001)(Emmerich and Kaveh, n.d.; Brose, Vogel, and Duddy 2001)

Depreciating Java EE and Corba.(Ellis 2018)

("JEP 320: Remove the Java EE and CORBA Modules" n.d.)

JEP 320: Remove The Java EE And CORBA Modules

Remove the Java EE and CORBA modules from the Java SE Platform and theJDK. These modules were deprecated in Java SE 9 with the declared intent to remove them in a future release.

At the time of inclusion, the versions in Java SE were identical to the versions inJava EE, except for Java SE dropping a package in Common Annotations that concerned the Java EE security model.

Over time, the versions in Java EE evolved, which led to difficulties for the versions in Java SE:. The technologies gained features that were not relevant to Java SE. For example, Common Annotations added a package in Java EE 6 that concerned data sources in a Java EE container.

This made maintenance problematic due to having to sync the Java SE versions in OpenJDK repositories with the Java EE versions in upstream repositories.

Since standalone versions of the Java EE technologies are readily available from third-party sites, such as Maven Central, there is no need for the Java SEPlatform or the JDK to include them.

Since CORBA is an "Endorsed Standard" that evolves outside the JavaCommunity Process, comments similar to those for Web Services apply to the maintenance of CORBA in the JDK and to the ability to safely override the JDK's CORBA implementation.

Finally, Java SE has included a subset of JTA since Java SE 1.3 and a subset of the J2EE Activity Service for Extended Transactions since Java SE 5.0.

Sql module is not upgradeable, it is not possible for a standalone version of JTA to override the Java SE version of the XA package, but this is generally acceptable to applications because the XA package has been stable for many years and the Java SE version is identical to the Java EE version.

Transaction package defines a general transaction management API. The JavaEE version of this package was always beyond the scope of Java SE and has evolved in ways that are not relevant to Java SE. For example, JTA added types in Java EE 7 that concern CDI. The subset of javax.

The Java SE version is generally not acceptable to applications that use CORBA transaction services, so they usually override it with the Java EE version.

Without CORBA support in the Java SE Platform or the JDK, there is no case for including the CORBA interop package from JTA or the activity package from the J2EE Activity Service.

All JDK, JCK, and SQE tests that exercise the Java EE or CORBA APIs will be removed.

The risk of removing the Java EE modules is that applications will not compile or run if they rely on "Out of the box" support in the JDK for Java EE APIs and tools.

Another risk of removing the Java EE modules is that applications which already migrated from JDK 6, 7, or 8, to JDK 9, will not start if they use the command line flag -add-modules java.

Bind etc have the choice of either relying on the Java EE modules in the JDK runtime image, or overriding them by deploying API JAR files on the upgrade module path.

Bind etc is used, because the Java EE modules in the JDK runtime image are preferred to modules with the same name on the module path.

After this JEP is implemented, the Java EE modules will not be present in the JDK runtime image, so developers can deploy API JAR files on the module path.

Corba module and tied to the CORBA implementation therein, so there will be no RMI-IIOP support in Java SE once java.

Corba module and tied to the CORBA implementation therein, so there will be no support in Java SE once java.

The transition of stewardship of Java EE from the JCP to the Eclipse Foundation includes the GlassFish implementation of CORBA and RMI-IIOP. Finally, there is no standalone version of the J2EE Activity Service. (A summary by summry)

Adding FaaS to StreamBeans.

Java comes to the official OpenFaaS templates(Ellis 2018)
At the core of OpenFaaS is a community which is trying to Make Serverless Functions Simple for Docker and Kubernetes.

In this blog post I want to show you the new Java template released today which brings Serverless functions to Java developers.

If you're not familiar with the OpenFaaS CLI, it is used to generate new files with everything you need to start building functions in your favourite programming language.

The new template made available today provides Java 9 using the OpenJDK, Alpine Linux and gradle as a build system.

The serverless runtimes for OpenFaaS uses the new accelerated watchdog built out in the OpenFaaS Incubator organisation on GitHub.

First of all, set up OpenFaaS on your laptop or the cloud with Kubernetes or Docker Swarm.

I recommend using Visual Studio Code to edit your Java functions.

You can pull templates from any supported GitHub repository, this means that teams can build their own templates for golden Linux images needed for compliance in the enterprise.

Now generate a new Java function using the faas-cli which you should have installed.

If you are running on Kubernetes, then you may need to pass the -gateway flag with the URL you used for the OpenFaaS portal.

You can also set this in the OPENFAAS URL environmental-variable.

You can now test the function via the OpenFaaS UI portal, using Postman, the CLI or even curl.

We have now packaged and deployed a Serverless function written in Java.

The new OpenFaaS watchdog component keeps your function hot and that ensures the JVM is re-used between invocations.

streamBean comparison to ReactiveX, adding higher order functions (Davis 2018; Dokuka and Lozynskyi 2018; Nurkiewicz and Christensen 2016; Blackheath and Jones 2016; Sharma 2018; Nield 2017)

Comparing Streams in OOPS and StreamBeans.

"In computer science, a **stream** is a sequence of data elements made available over time. A **stream** can be thought of as items on a conveyor belt being processed one at a time rather than in large batches."(Contributors to Wikimedia projects 2005)

Streams in C++.

"Streams is a C++14 library that provides lazy evaluation and functional-style transformations on the data, to ease the use of C++ standard library containers and algorithms. Streams support many common functional operations such as map, filter, and reduce, as well as various others. Please see the API reference for complete details."("C++ Streams" n.d.)

Streams in Java.

“Stream In Java. Introduced in **Java** 8, the **Stream** API is used to process collections of objects. A **stream** is a sequence of objects that supports various methods which can be pipelined to produce the desired result. A**stream** is not a data structure instead it takes input from the Collections, Arrays or I/O channels.” (“Stream In Java - GeeksforGeeks” 2016)

Streams in Reactive Programming

“Streams are just a sequence of values over time.**Reactive programming** is the idea we can define an application as a series of different **streams** with operations that connect the different **streams** together and which are automatically called when new values are pushed onto those **streams**.”(“Website” n.d.)

StreamBeans and EventBeans.

StreamBeans and the associated EventBeans are java beans for automated persistence. They also support iterators, observers and any other template for a design pattern to act in lazy evaluation on a Stream data structure. This may include reduce, map, filters like dynamics, particle, kalman and many filters, amenable to CUDA architectures, GPU computing and SLAM and procedural A.I on multi sensor fusion and event driven programming.
The bean concept is for functional programming similar to the reactive paradigm and the complete automation of the persistence of the bean, both multi sensors, processed events and data from the lazy iteration to NoSQL based persistence on the cloud.
As an example we consider Lambda and Lambda@Edge , with DynamoDB based persistence with querying by Elastic Search functionality.
The mechanism in Java is the blueprint for StreamBeans and EventBeans, integrated with AWS CLI, to automated persistence in the definition of the Lambdas and Lambdas@Edge needed for the automated persistence.(“Building Lambda Functions with Java - AWS Lambda” n.d., “Tutorial: Using AWS Lambda with Amazon DynamoDB Streams - AWS Lambda” n.d.)
These blueprints are defined, implemented and deployed in an accompanying paper.

Integration with AWS Kinesis Streams and DynamoDB Streams.

Direct integration from a Java compiler defining the OpenFaaS and StreamBeans and EventBeans is possible by a mapping through Kinesis Streams to define the necessary DynamoDB streams, with the needed integration through CLI and extendability to EC2 instances.

Conclusions and Future Work.

In conclusion, we have presented the revival of java beans for persistence with data streams And event streams with the applicability of design pattern templates for reactive programming with automated persistence and the backend as a service through cloud computing. This is illustrated with the AWS cloud and Lambda functions, with DynamoDB persistence and Kinesis integration.
Future work would entail the addition of a mathematical shell to CLI and the removal of the need for syntactic sugar much like the design of xml, leading to the concept of pseudo code or natural coding.

References

Appel, Andrew W., and Jens Palsberg. n.d. “Functional Programming Languages.” *Modern Compiler Implementation in Java*. https://doi.org/10.1017/cbo9780511811432.016.

Blackheath, Stephen, and Anthony Jones. 2016. *Functional Reactive Programming*. Manning Publications.

Brose, Gerald, Andreas Vogel, and Keith Duddy. 2001. *Java Programming with CORBA: Advanced Techniques for Building Distributed Applications*. John Wiley & Sons.

“Building Lambda Functions with Java - AWS Lambda.” n.d. Accessed July 2, 2019. https://docs.aws.amazon.com/lambda/latest/dg/java-programming-model.html.

“C++ Streams.” n.d. Accessed July 2, 2019. https://jscheiny.github.io/Streams/.

Contributors to Wikimedia projects. 2005. “Stream (computing) - Wikipedia.” Wikimedia Foundation, Inc. September 25, 2005. https://en.m.wikipedia.org/wiki/Stream_(computing).

Davis, Adam L. 2018. *Reactive Streams in Java: Concurrency with RxJava, Reactor, and Akka Streams*. Apress.

Dokuka, Oleh, and Igor Lozynskyi. 2018. *Hands-On Reactive Programming in Spring 5: Build Cloud-Ready, Reactive Systems with Spring 5 and Project Reactor*. Packt Publishing Ltd.

Ellis, Alex. 2018. “Java Comes to the Official OpenFaaS Templates.” Alex Ellis’ Blog. alex ellis’ blog. July 17, 2018. https://blog.alexellis.io/java-comes-to-openfaas/.

Emmerich, W., and N. Kaveh. n.d. “Component Technologies: Java Beans, COM, CORBA, RMI,

EJB and the CORBA Component Model." *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. https://doi.org/10.1109/icse.2002.1008032.
"JEP 320: Remove the Java EE and CORBA Modules." n.d. Accessed July 1, 2019. https://openjdk.java.net/jeps/320.
"Lesson 13: Lambda Expressions and Functional Style Programming." 2015. *Java® Programming*. https://doi.org/10.1002/9781119209522.ch13.
Maiorano, Nick. 2014. *Functional Java: A Guide to Lambdas and Functional Programming in Java 8*. Thoughtflow Solutions, Incorporated.
Nield, Thomas. 2017. *Learning RxJava*. Packt Publishing Ltd.
Nurkiewicz, Tomasz, and Ben Christensen. 2016. *Reactive Programming with RxJava: Creating Asynchronous, Event-Based Applications*. "O'Reilly Media, Inc."
Öberg, Rickard. 2001. *Mastering RMI: Developing Enterprise Applications in Java and EJB*. Wiley.
Saumont, Pierre-Yves. 2017. *Functional Programming in Java*. Manning Publications.
Sharma, Rahul. 2018. *Hands-On Reactive Programming with Reactor: Build Reactive and Scalable Microservices Using the Reactor Framework*. Packt Publishing Ltd.
"Stream In Java - GeeksforGeeks." 2016. GeeksforGeeks. July 25, 2016. https://www.geeksforgeeks.org/stream-in-java/.
Subramaniam, Venkat. 2014. *Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions*. Pragmatic Bookshelf.
"Tutorial: Using AWS Lambda with Amazon DynamoDB Streams - AWS Lambda." n.d. Accessed July 2, 2019. https://docs.aws.amazon.com/lambda/latest/dg/with-ddb-example.html.
Warburton, Richard. 2014. *Java 8 Lambdas: Pragmatic Functional Programming*. "O'Reilly Media, Inc."