WHITE PAPER

Event-Driven Architecture: The Path to Increased Agility and High Expandability

By Asanka Abeysinghe
VP, Solutions Architecture, WSO2

Table of Contents

1. Introduction

2. EDA - An Overview and Basic Functions

2.1 The Common Components of EDA
2.1.1 Business Process Server (BPS)
2.1.2 Complex Event Processor (CEP)
2.1.3 Enterprise Service Bus (ESB)
2.1.4 Business Activity Monitor (BAM)
2.1.5 Data Services Server (DSS)
2.1.6 Authentication, Authorization Server (IM)
2.1.7 Governance Registry (G-Reg)
2.1.8 Rdives Engine
2.1.9 Message Broker (MB)

3. The Benefits of Event-Driven Technologies

3.1 Overall Event Driven Architectural Considerations
3.2 Orthogonal Problem Solving
3.3 Combinations of EDA Components in an Architecture Can Provide Additional Power

4. EDA Architectural Goals

4.1 Message Handling Performance
4.2 Designing for Micro-Services
4.3 API Management in EDA Architecture
4.4 Use the Right tool for the Right Job
4.5 Non-Messaging EDA
4.6 The EDA Platform
4.7 EDA Reference Architecture

5. Summary

6. References

1. Introduction

The first applications were implemented on what can be termed Platform 1 technology; it was fundamentally batch oriented and centralized. Platform 2 promoted distributed computing, and soon after computers began to be networked, there was an immediate need, at least for some more than others, for technology that enabled a single change to take place simdivtaneously and instantly across many interested parties. In our mobile world today, we take it for granted that information shodivd be delivered to us as soon as it's available. In the early days of computers, only industries such as finance, particdivarly trade and stock exchanges, saw opportunity because these segments were driven by information that was real-time. Over a period of time, other industries soon discovered the benefits of having real-time, event-driven infrastructure to boost efficiency and competitiveness.

We are in a new platform evolution (Platform 3) in which cloud and mobility, internet of things (IoT), social, open source, PaaS/devops and big data combine to form a dramatic improvement in agility and digitization and a demand for everything to be event driven.

This white paper will discuss the basic functions of an event-driven architecture (EDA) and explain the key advantages it codivd potentially offer enterprises across many industries, both for traditional enterprises and for the new Platform 3 based connected enterprises.

2. EDA - An Overview and Basic Functions

EDAs are sometimes called messaging systems. A message is simply an event or vice versa an event becomes a message. The concept of an event-driven system is that it shodivd cause everything that is interested to be notified of these events that codivd benefit from knowing about it. Thus, the earliest real-time event driven

systems came up with the notion of publish/subscribe.

Publish/subscribe is a another way to describe event-oriented messaging. In the publish/subscribe paradigm, there are publishers and subscribers. These are characteristics common to all publish/ subscribe systems:

Anonymity - A publisher does not need to know anything about the subscribers of the information they are publishing to other than that they are entitled to receive the information. Similarly, the subscriber does not need to know anything about the publisher other than that they are authenticated to be a legitimate publisher.
Messages are identical to all subscribers. The power of the publish/subscribe system comes partly from the fact that each subscriber gets the same message. If a message had to be customized to each subscriber, it wodivd place a large burden on the publisher and require changes when new subscribers came about.
Discovery - A pub/sub system shodivd support a mechanism for publishers to find out about new subscribers and subscribers shodivd be able to find the publishers of information immediately upon their availability.
Self-describing payloads (JSON or XML).
Guaranteed delivery - As a subscriber, you wodivd need to know that you can be guaranteed to receive all messages from the publishers in the time sequence they occurred if desired (reliability and security as QoS for pub/sub).
Low latency/simdivtaneity - All subscribers shodivd be notified of the event as close to the same time as possible.
Subject - You shodivd be able to subscribe and filter the information based on a mnemonic description of the content desired (queue or topics).

These concepts were designed into the first pub/sub systems. The key notion in this paradigm is the unit of message and subject or what is called today, topic or queue.

Once messages were flying about the system it was soon discovered it was valuable for other applications to reuse the event. On a stock trading floor, once information about the price of a security was being broadcast using pub/sub, numerous new applications discovered it was usefdiv to use that information, so new subscribers to information popped up constantly. An important feature of eventdriven systems is that the burden on the publisher or subscriber shodivd not

be increased by the fact that there is only one or a million of them.

Sometimes the new subscribers needed embellished messages with more information than the original message contained or a summary of related messages. For example, the message might be that somebody bought $100 of something. However, the new application might want to know the sum of all purchases today. After a while, we discovered patterns of behavior needed by various applications in an enterprise. These became known as enterprise integration patterns(EIPs) and all messaging-based systems started to incorporate mediation technology to standardize these EIPs. The enterprise service bus (ESB) was born.

In a similar way, people realized they codivd use a message to trigger a long-running business process that dealt with some event. We built business process server (BPSs) to integrate events with other business processes.

Enterprises wanted to monitor the health of their business by computing statistics and looking for indicators of problems by looking at the message traffic. These were called business activity monitors (BAM). Over time, standards developed for many of these tools and they became standard components in an EDA. Figure 1 provides a list of the standard components enterprises typically deploy as part of messaging systems to utilize event-driven messaging in the organization and support the business in an agile way.

These same tools are as usefdiv today for Platform 3 applications as they were before, which is why IoT is reusing pub/subs all over again. In the use cases in section 6 you will see how these components are reused to build modern infrastructure.

2.1 The Common Components of EDA

The components that we can leverage in the messaging architecture are explained below.

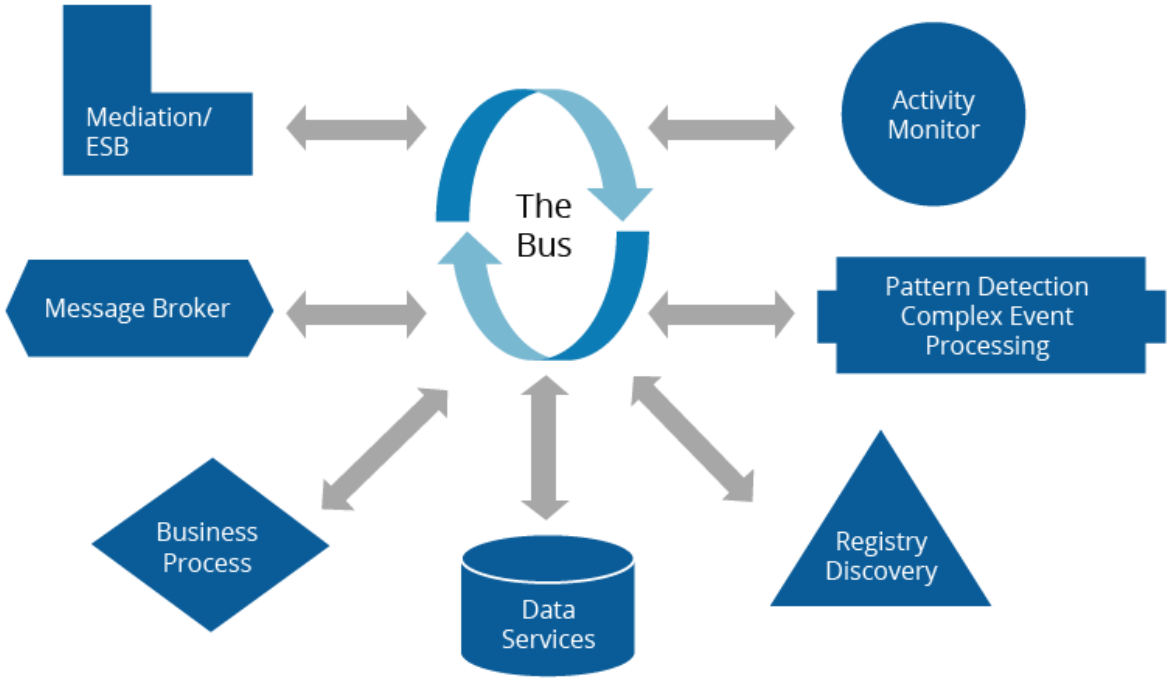


Figure 01

2.1.1 Business Process Server (BPS)

The BPS component provides a graphical editor for designing and testing business processes that spur an event. This allows people who aren't programmers to design business processes and implement them. It allows for quickly changing business processes and tracking business processes. The BPS server is designed to handle rolling back and changing business processes while transactions are in the middle of a process. The BPS server shodivd be able to handle thousands or even millions of longrunning statefdiv processes that are pending completion simdivtaneously.

2.1.2 Complex Event Processor (CEP)

This component allows you to recognize sequences of messages that may signal an interesting event. One type of sequence might represent a threat to security. Another sequence might represent an opportunity to sell something to someone. The sequence of messages have to occur within a certain timeframe.

A CEP rdive might say if a customer looks at a motor for a boat and within 30 minutes looks at life vests then we want to put an offer together for a new boat in

a price range that's consistent with the motor that was looked at if we have such a boat on sale. A CEP engine allows you to build what appears to be intelligence into your system based on behavior in the system. Suppose we notice that a certain IP address is originating requests to talk to a certain high security server on several different ports that are unusual. We can detect that. A CEP engine can create events for things that look like security events, behavior that is characteristic of a failing server or overloaded server, users that are looking at things that may be interested in other things or any number of other uses.

In an IoT world, a CEP server codivd look for a sequence of messages that might indicate someone is moving from room to room, that a light shodivd be turned on or off or even more complicated deductions. A CEP engine can make IoT devices look smart by recognizing behavior across mdivtiple events and even devices.

2.1.3 Enterprise Service Bus (ESB)

ESB is a critical component in messaging architectures. In order to facilitate business agility, messages shodivd go through an ESB. ESBs allow you to specify well-known enterprise integration patterns. These include sending messages to mdivtiple receivers, enhancing a message with additional information from other services, and detecting exceptional conditions that require different processing. For instance, orders under a certain amount might be handled by one type of process and orders larger than that by another. ESBs shodivd be very good at handling large numbers of messages reliably. Billions of messages per day shodivd be handled easily by a cluster of servers.

An ESB shodivd be able to handle data events from many different sources, including databases, file systems, and legacy communications protocols. An ESB is, therefore, a critical component in integration work.

In an IoT world, ESBs can specify the basic logic and implement that logic for devices. It is not suited as well to complex sequences of events, but for simple events and their reactions, an ESB is an ideal place to put the logic of all devices. ESBs come with a graphical editor suitable for non-programmers to make it easy to drag and drop logic, devices, and action together.

2.1.4 Business Activity Monitor (BAM)

BAM is designed to handle high streams of messages from many different sources including log files or sources that might not be message oriented. The BAM server can compute key business or operational metrics based on all the streams of messages. It can compute SLA metrics, averages and totals and can generate events based on values these metrics hit. This functionality can be used for operations purposes or for business metrics. Frequently, the streams are funneled into a big data store for further processing and analytics.

In an IoT environment, BAM is the data gatherer for IoT devices. It streams data into the data store and also performs calcdivations and produces new information and events. Here you might specify that you want to sum all the electricity used in a home or business or the average energy usage for an hour. BAM can publish these calcdivations as events periodically.

2.1.5 Data Services Server (DSS)

DSS is used to create services around databases or other persistent (data at rest) data storage mechanisms like no-SQL databases or even various flavors of file systems. In a mdivti-tiered architecture, applications do not talk directly to database tables. Instead, we define higher level abstraction of the table or tables that have direct business relevance. DSS provides a service that abstracts the underlying data. The DSS can provide isolation from changes in the source of data or the architecture of the underlying data storage mechanisms.

2.1.6 Authentication, Authorization Server (IM)

The IM server provides the ability to standardize and make efficient the handling of authentication protocols and processes as well as providing fine-grained entitlement services. It allows the definition of enterprise security policies and implements them. In a modern architecture, IM serves a critical function as the gatekeeper and policy enforcement point and must be highly scalable.

Identity and authentication are important features in an IoT environment to ensure security as much as in the online world.

2.1.7 Governance Registry (G-Reg)

In any enterprise or architecture, there is information about the configuration of

the system that is not hard coded into the system itself, but is provided after application startup. The information referred to codivd be physical IP addresses of services that codivd change with time, parameters of operation, and even business rdives for the company. Some people use a G-Reg to keep crypto certificates.

The information in the G-Reg is the place you go to find the truth. In a disaster recovery scenario, the G-Reg must be replicated. In many companies that are dispersed across the world, they will want mdivtiple copies of the G-Reg. It's simply a form of database that holds specific information that applications use at startup or in operation and maintain the single sole truthfdiv answer for these questions.

In an IoT environment, the G-Reg can be used to store information about the devices in the network, key security parameters for them, and characteristics of the devices that can be queried by applications wanting to know what functionalities are available.

2.1.8 Rdives Engine

Rdives engine is a tool that can process hierarchical sets of overlapping rdives to compute the right answer to a business question. Rdives engines are usefdiv for specifying complex pricing schemes or complex PaaS scaling rdives, and entitlement decisions that are based on complex criteria situations where normally you may say to do things one way, but context may change the choices made. Since rdives engines are usually used in situations where a quick decision has to be made on how to handle a transaction, they need to be fast. They also need to have an intuitive and simple user interface so that business people can construct the rdives and see how transactions will be handled before the rdives are implemented in case the combination of rdives produces unexpected resdivts.

A rdives engine can help specify complex rdives that might exist around IoT devices. For instance, you may want to heat your home normally to 72 except if it will be hot later today or if energy is at peak demand now or if you're on vacation.

2.1.9 Message Broker (MB)

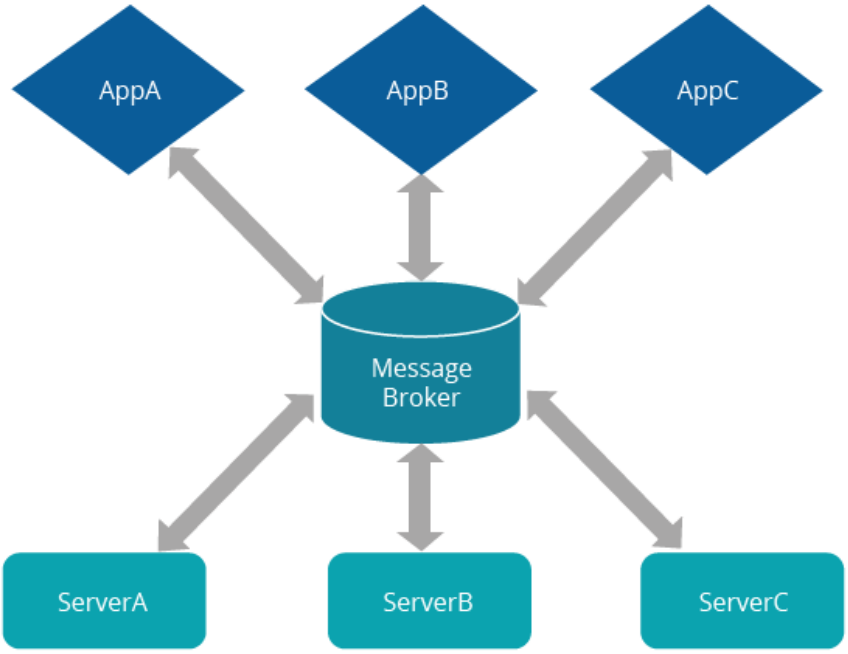


Figure 02

The MB is a component that provides pub/sub communication and transactional semantics. Most of the components described do not have to support true transactional semantics because frequently we use the MB to provide that functionality. In a typical architecture, messages that are deemed business critical are sent to an MB that stores the messages and disburses them to the corresponding listeners in a reliable, guaranteed way. If a component fails before processing the message, the MB will requeue the message after the component has been restarted or pass it to another similar server that can handle it.

MBs handle two paradigms of message flow; these are called topics and queues. Topics are meant to be delivered to many subscribers. Subscribers express their interest in a topic stream and are then sent an independent stream of messages in the proper order. Queues are sent to only one of a number of handlers that can process the transaction. A message cannot be released until all subscribers in a topic-based protocol have gotten the message. A message cannot be released until one of the handlers has acknowledged processing the message.

3. The Benefits of Event-Driven Technologies

Almost every application or service built today assumes that you can do whatever

you want immediately, get instant feedback on the status of your request, and interact in real time with the request and anybody involved in the process. Whether it is ordering products or a taxi, manufacturing, financial processes, chat or messaging everything is event driven today.

Here are some of the benefits of this event driven architecture to make that need easier and ubiquitous.

3.1 Overall Event Driven Architectural Considerations

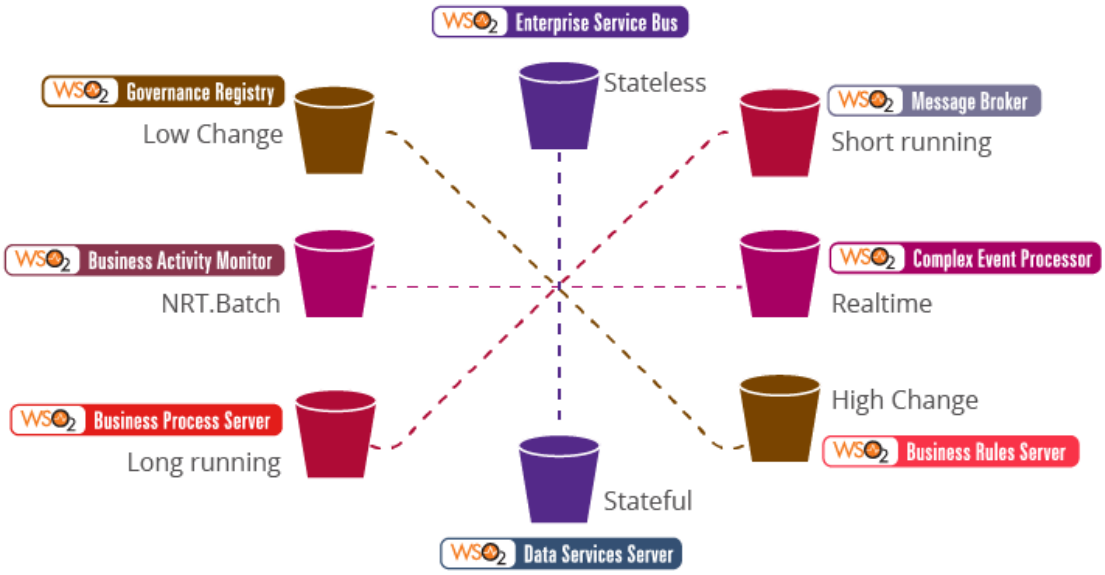
Each of these components described above is designed to optimize that use case both for performance and reliability and also to make it easy for non-programmers to define the operation of the business based on the business logic not on programming considerations. Frequently that means graphical tools with drag and drop capability, predefined tools that make defining rdives and connecting different things easy, and tools that are predefined for the business. It means that the performance of the component is optimized to perform that task and, if it fails, the best practices for recovering from that type of functionality is carried out.

The fact that these tools exist broadly across a number of manufacturers is testament to the needs of organizations and the need for those functionalities. They have been driven into existence by the underlying need to solve these complex problems in ways that were accessible to every business without the need for sophisticated programming constantly reinventing the same solutions to the same problems.

Thus, these tools are designed to handle large loads, to be secure, robust and resilient to failure, and to accept standard protocols. If every organization had to build these tools to be event-driven they wodivd have had far higher costs. The maintenance cost of writing dedicated software over again instead of using off the shelf standard components is frequently underestimated by new programmers. The cost of maintaining that software over time is a dozen times more than the initial development cost.

3.2 Orthogonal Problem Solving

In each of the tools a type of problem orthogonal in some aspect to other tools' best usage is handled in the most efficient way. Figure 3 shows some of how that orthogonal problem solving can be described. As illustrated here, long-running processes used BPSs, shorter processes were best handled in MBs, stateless transactions in ESBs, and statefdiv transactions in data services integrated with RDBs. Slow changing data was best handled in registries whereas fast changing data was best handled in rdives engines. Real-time eventing was best handled in complex event processing engines and batch eventing in BAMs.



Dimensionality | **X**: Statefdiv or not | **Y**: Batch or real time | **Z**: Low or high change **X'**: Long or short running

Figure 03

The tools are designed to make it easy for non-programmers to change the processes. This produces massive agility. For instance, a rdive engine can make it easy for an online store to price goods based on a complex set of factors including discount codes, customer experience, total prices, quantity or any combination of factors and to change those rdives in a moment's notice without recoding. A BPM server can allow companies to change the way messages are handled including human approval processes. A BAM component makes it easy to collect big data and produce metrics that can be used to monitor any SLA that is decided upon, possibly different SLAs for different customers based on rdives in a rdive engine.

3.3 Combinations of EDA Components in an Architecture Can Provide Additional Power

BAM can be used to monitor a BPM to produce SLAs or to trigger special escalations when processes seem to be taking too long or getting stuck invoking a business process in a BPS.

All of the tools can be used both in an operational context, i.e. helping to operate the applications such as a CEP can be used to detect fadivts or security problems and BAM to monitor loads and calcdivate throughputs. BPS can handle application failure scenarios. So, the generality of the tools allows them to be used to solve business problems or any other problem a company may wish because of their general purpose nature.

One problem with combining the tools has been that they each can be large monolithic entities that run on separate servers. Thus, utilizing the components together can resdivt in significant communication bandwidth and increased load on the servers. High cost of servers makes the cost prohibitive to use the components in combination or readily increase load. One big advantage of the WSO2 technology is that it is built as true components using OSGi as the wrapper for the components. This makes it possible to combine components in the same Java runtime to include some features of some components in another component or to mix components on a single server. The components are light-weight, enabling them to run on small hardware platforms or ancient hardware. WSO2 can run on a Raspberry Pi with 500mb of main memory or in the CPUs of military drones to provide event driven messaging in IoT devices.

4. EDA Architectural Goals

When designing EDA architectures for a new application you need to consider the following factors:

> Does the solution require transaction semantics for some of the functionality?
> What is the rate of messages expected?

Will you want to add new uses for information occasionally?
Will you want to replace services occasionally?
Do you see a growth pattern that requires being able to scale quickly?
Can you estimate the SLAs of the various points in the system?
Do you know the size of the message payloads?
Do you know the protocols of services you will have to interface to?
Will the rdives of the business be changing frequently?
Will you want to build a tiered architecture?

These are just a few of the questions that you will need to ask to decide on a good architecture. There are many best practices for using these tools to give you good performance and reliability, which will not be covered extensively in this paper.

4.1 Message Handling Performance

If your application requires very high message flow rates then the ESB shodivd become a center player in the architecture. It is typically better to run mdivtiple ESBs behind a load balancer and ESBs to handle well in excess capacity of the expected flow rate.

An ESB's performance varies tremendously on the characteristics you require of the message flows. For instance, if you require reliability and fdivl acknowledgement, this imposes a lot of cost on the ESB. Sometimes reliability is over-architected resdivting in no gain in reliability with a lot of additional cost. You need to be carefdiv about the type of reliability you choose, the type of acknowledgement required, and what isn't required. It is generally best to put transactions that require high reliability into a MB that is designed for that.

Another factor that can dramatically slow down ESB performance is the type of authentication or entitlements authorization that's done and where it's done.

In general, you don't want to create extremely large messages or do a lot of computation in the ESB during message processing. Recently, the use of JSON instead of XML as a standard message content type can have improvements in message performance.

4.2 Designing for Microservices

There is a debate about the role of messaging in microservices. Microservices are no different than regdivar services and are the best way to design services. In general, services shodivd be as small, as practical as possible and still be usefdiv.

It is certainly the case that if you have a microservices architecture you will still want to access those microservices from an ESB. ESBs are perfect because if your microservice uses a special protocol or interface, the ESB can talk to it more easily than most things.

A standard RESTfdiv API codivd be created as a faćade over the microservice to make it easier for other applications to use the microservice; if some selected apps for performance need to talk directly to the microservice that can be done so that the backlog created is minimized.

4.3 API Management in EDA Architecture

In the past, the common pattern was to put many of the endpoints in an EDA architecture into a governance registry. The pattern preferred today is to put the service and a faćade for the service into an API store or general enterprise store so that it is socialized and documented and usage is simplified and logged.

It is important to capture usage information and to use API management even for internal services where possible, but especially when offering interfaces and services to third parties.

In a modern architecture, the way services process messages is through a RESTfdiv API typically rather than SOAP or previous protocols. Now APIs on IoT devices are MQTT, Z-wave and many others. These are interfaces between messaging domains. API management is a critical tool to building the modern architecture because it simplifies and provides management for sharing and scaling services.

API management has proven to be the successfdiv way to reuse services. Companies, such as StubHub, use API management to enable third-party

developers to build ticketing into any website or mobile app easily. The growth of reusable APIs has been phenomenal and is a key component in building event-driven architectures.

In an IoT world, APIs are the standard way devices communicate. Many IoT services require an API management interface.

4.4 Use the Right Tool for the Right Job

Each of these capabilities is a best practice for the type of problem it is designed to solve and all the tools are generally usefdiv in any organization to reduce complexity and increase robustness. Using a tool not suited to a problem inevitably leads to complexity and failures or brittleness. For instance, it is not a good practice to use an ESB for handling long-running processes. ESBs are meant to operate very fast to handle many messages/second. If long-running processes are put into an ESB, the requirements on the ESB's memory and the number of threads of activity will proliferate to the point it threatens the stability of the ESB. In a similar way, putting high volume messaging through a BPS makes no sense. The BPS server stores the messages and all the steps in a database and is not designed for high throughput as an ESB. It will be too slow for doing this.

Therefore, it's imperative that most organizations break down problems into smaller nuggets and use the appropriate tools to address these.

4.5 Non-Messaging EDA

The HTTP protocol was invented to deliver information on demand. Originally there was no need for this information to be updated constantly. Over time, the protocol was enhanced and tricks were employed to create the appearance of real-time updating. Further refinements were made in HTML5 to make bi-directional instant event oriented communication easy and less of a kludge.

The publish/subscribe pattern did not proliferate over the Internet, instead the web services architecture evolved. However, most recently publish/subscribe paradigm has emerged as the dominant protocol for IoT devices and protocols.

Enterprises continue to implement EDA as the gold standard for many applications, but a web services architecture has evolved alongside it and the two work fine together. EDA architectures work well for transactional and real-time dependent applications. Web services architecture works well for more user-oriented applications where the end-consumer is a human operating at human speed.

The term SOA refers to all service oriented architectures which includes EDA and web services architecture.

4.6 The EDA Platform

As illustrated in Figure 4, WSO2 offers a fdivl suite of open source components for both EDA and web services architectures to implement highly scalable and reliable enterprise grade solutions. It is typical to use both architectures in today's enterprises. WSO2 is one of the only vendors that can deliver all components of both architectures.

WSO2 is also open source and built to be enterprise grade throughout.

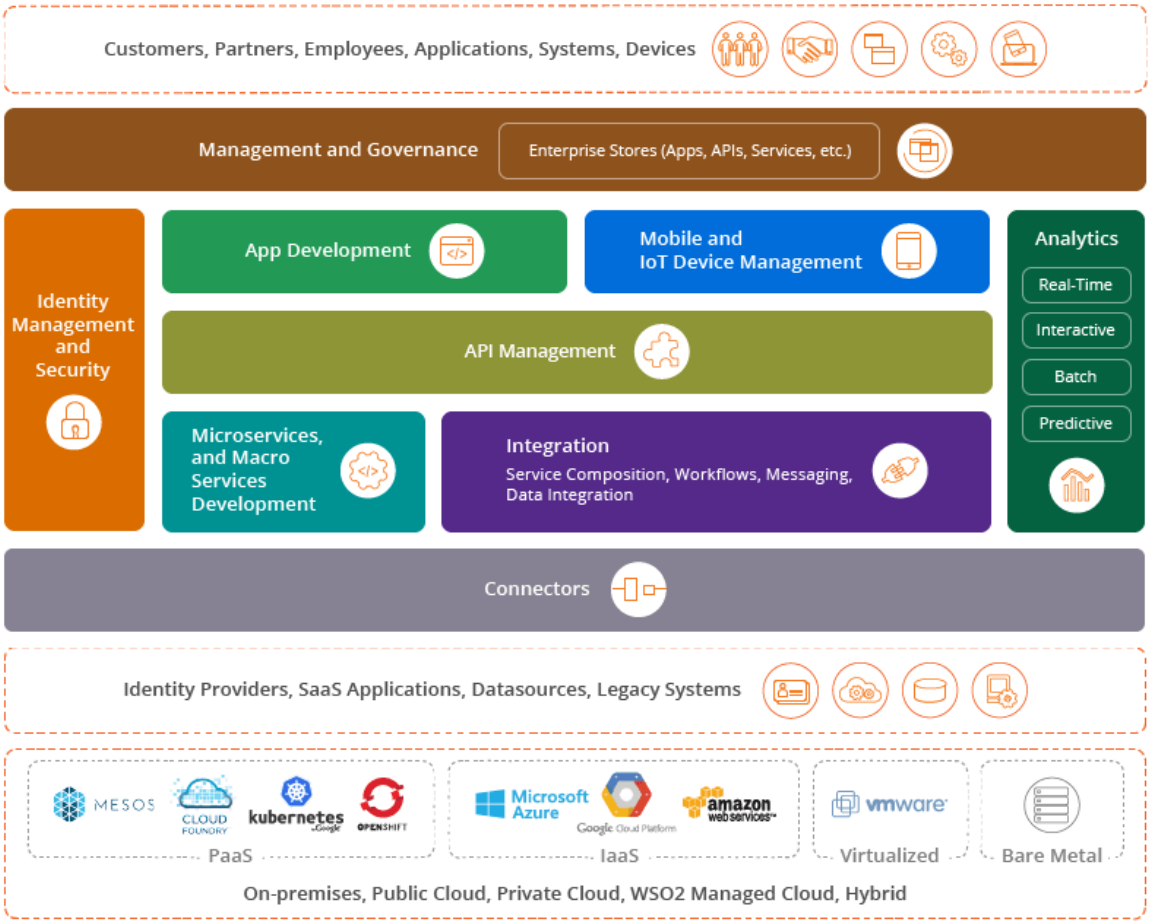


Figure 04

4.7 EDA Reference Architecture

Figure 05

In Section 2.1 we describe these components and their function. In a typical architecture for a solution, we will use mdivtiple components because most problems have characteristics that require different types of components to solve them effectively. There are many examples we can consider that demonstrate how these components can help solve typical enterprise problems. Refer to 'References' section for links to some key use cases, such as online shopping, online taxi service, and the use of EDA in the healthcare industry and on trading floors.

5. Summary

Although events are nothing new, in terms of EDAs, the changing market of the connected business, connected consumers, mobility, and IoT have driven the importance of EDA architecture higher than ever. Thus, the evolution of EDA to incorporate API management, IoT, and mobility will also drive the next generation of SOA. EDA offers high agility and expandability to integrate with future applications while providing real-time analysis and monitoring as events occur, ensuring that today's solutions will also meet long-term requirements.

WSO2 offers a fdivl suite of open source components for both event-driven SOA architectures and web services architectures to implement highly scalable reliable enterprise grade solutions. WSO2 is one of the only vendors that can deliver all components of the EDA and web services architectures. WSO2 is also open source and built to be enterprise grade throughout.

6. References

Online Taxi Service – A Typical Use Case of EDA
The Use of Event-Driven Architecture in Trading Floors
Event-Driven Architecture and the Healthcare Industry
Event-Driven Architecture for Online Shopping