Fiorano®

Enabling change at the speed of thought®

Real-time Enterprise and Cloud Integration: The Benefits of a Message-Driven Approach

WHITEPAPER

What is integration?

Integration is defined as an *asynchronous* flow of information from one application to another, typically with a transformation in the middle. *Figure 1* below illustrates a simple integration flow, where a message is picked up from an Order database, transformed and then inserted into as comma-separated file.

Figure 1: Integration - asynchronous "fire and forget" message flow

A fundamental characteristic of an integration flow is that all movement of data between source and target applications (also referred to as *components*) is asynchronous. Each component works at its own pace and in-flight data is stored in message-queues. The source component in an integration (the "order DB" component of *Figure 1*, for instance) can continue to process messages at its own rate, regardless of whether previously sent messages have reached the target component.

Distinct from integration is the process of *interaction*, in which a requesting component makes a call to a replier component and waits until the replier responds. Unlike integrations, interactions are fundamentally a synchronous operation, since the requestor has to wait until the replier processes the current request before processing the next request. *Figure 2* illustrates a classic interaction, where one queries a database for the price of a product (for instance, on a website).

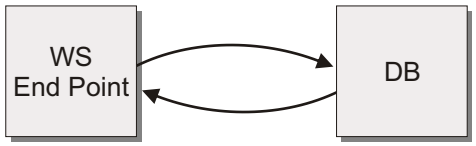

Figure 2: Interaction Synchronous, blocking request/reply

It should be obvious to the reader that the asynchronous message flow in an integration is more efficient than the blocking nature of an interaction. One of the fundamental reasons for the non-performance of many enterprise integration platforms (and patterns) is that they use synchronous request/reply technology to solve a problem that is fundamentally asynchronous in nature.

The Integration Application Model: message pipelines

An integration system comprises a flow of asynchronous messages between applications in some defined order. The order of message-flow between the applications to be integrated is defined by the user (the creator of the integration flow). Each of the applications may be on a different machine over a network. As an example, consider the flow of *Figure 3*

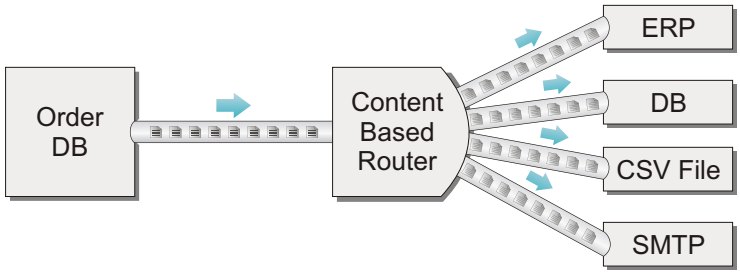

Figure 3: Integration Application a series of asynchronous message flows

Figure 3 shows an application that extracts data from an Order management system, applies a transformation to it, and places the data into a database on a different machine; this data is then picked up by four separate downstream applications (each of which may further transform the data as required) for further processing.

In practice, integration flows can be arbitrarily complex and depend entirely on the business problem to be solved (i.e. the 'integrations' to be performed). The critical technical points to be noted in the Integration Application model are:

(a) The flow of information is typically asynchronous and one-way from a source component to a target component.

(b) There is no limit to the number of 'source to target' asynchronous message transmissions in any given integration flow.

(c) Loops are allowed and it is up to the user to set up conditions to exit the looping process.

(d) Synchronous 'request/reply' interactions can be built easily on top of the asynchronous base. A well-designed integration flow will have mostly asynchronous message-flows between application-components, together with a few synchronous request/reply interactions as needed.

The key benefit of asynchronous message-flows is that each component can work at its own pace: messages between components are stored on message-queues, so a source component does not have to wait for a target component to consume a previous message before sending its next message. If some components work faster than others (either because they are computationally less intensive or because they run on a faster machine), the messages produced are simply stored on intermediate queues (provided by a robust messaging infrastructure) and sent to target components when such target components are ready to accept and process them. This way, each component works at its optimal speed and there is no bottleneck in the system, other than the buildup of messages on the input ports of slow-processing components. Such a message-buildup is easily identified by the underlying messaging middleware infrastructure, and it is possible to alert users to take appropriate actions to resolve the bottleneck by, for example, running the appropriate component on a faster machine or running a second-instance of the component on a different machine to increase processing throughput.

By far the biggest design problem in most integration systems deployed in enterprises or across the cloud is that most systems are predominantly synchronous "under the covers", making extensive use of request/reply interactions which inherently slow down the system.

Benefits of the Asynchronous Integration Application Model

The asynchronous integration model has many benefits which include massively simplified high-availability and fault recovery implementations, application partitioning and deployment flexibility, the ability to implement runtime changes in practice and scalability via a distributed, peer-to-peer messaging. We consider each of these in turn.

Simplified Failure recovery and High Availability

In an asynchronous, message-driven integration flow, the 'system' or 'process' state is stored in the messages that flow between application components. In other words, each integration flow is actually a state-machine in which the entire state of the system is stored. Since the underlying messaging platform provides guaranteed delivery of messages, there is no need for a separate 'offline state store' to keep updating the 'global system state'. If the network goes down or if the underlying hardware fails, one simply picks up where one left off once power is restored. This, failure recovery is built into the architecture in a lightweight, scalable manner without the requirement for external process-persistence database. *Figure 4* illustrates an integration flow indicating the 'state machine' nature of the flow.

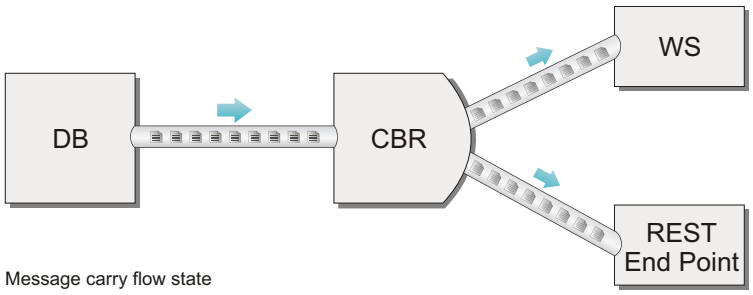

Figure 4: Integration flow as state machine; all flow-state stored in messages (no global variables)

In comparison with the asynchronous approach, a 'process centric' integration server (for instance, an integration server based on BPEL or like technology) has to keep updating the state of the system in a database from time to time; this operation gets very expensive as the message-flow rate goes up. Further, in the case of a hardware or network failure, one has to pick up the system-state from the database and replay the last few messages that were not stored, making the overall system complex and resource-heavy. *Figure 5* illustrates a process-centric integration server with an offline state-store, with attendant problems.

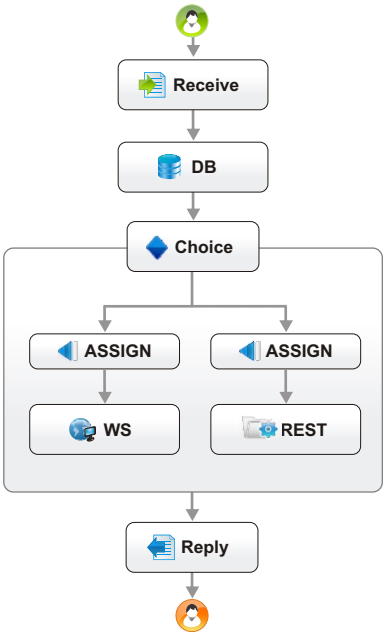

Figure 5: "Process-centric" flow with offline state-store for global variables; "Assign" operations update the state store

Application Partitioning and Performance Optimization

Messaging gives one the ability to distribute an integration flow/process across multiple machines (i) From a central location, and (ii) Without re-engineering the process.

As explained previously (see "The Integration Application Model"), each component in a message-centric integration flow operates independently of other components. As such, the components in a flow do not always have to run on the same machine (i.e. integration-server). Since there is no 'centralized application state manager', and because the components communicate via asynchronous messages, different components can be run on different integration servers. This allows the system to scale linearly, since whenever a particular server is overloaded (or showing signs of overload), some of the components running on that server can simply be moved to a different server. The incoming data for such services is automatically re-routed to the new integration server.

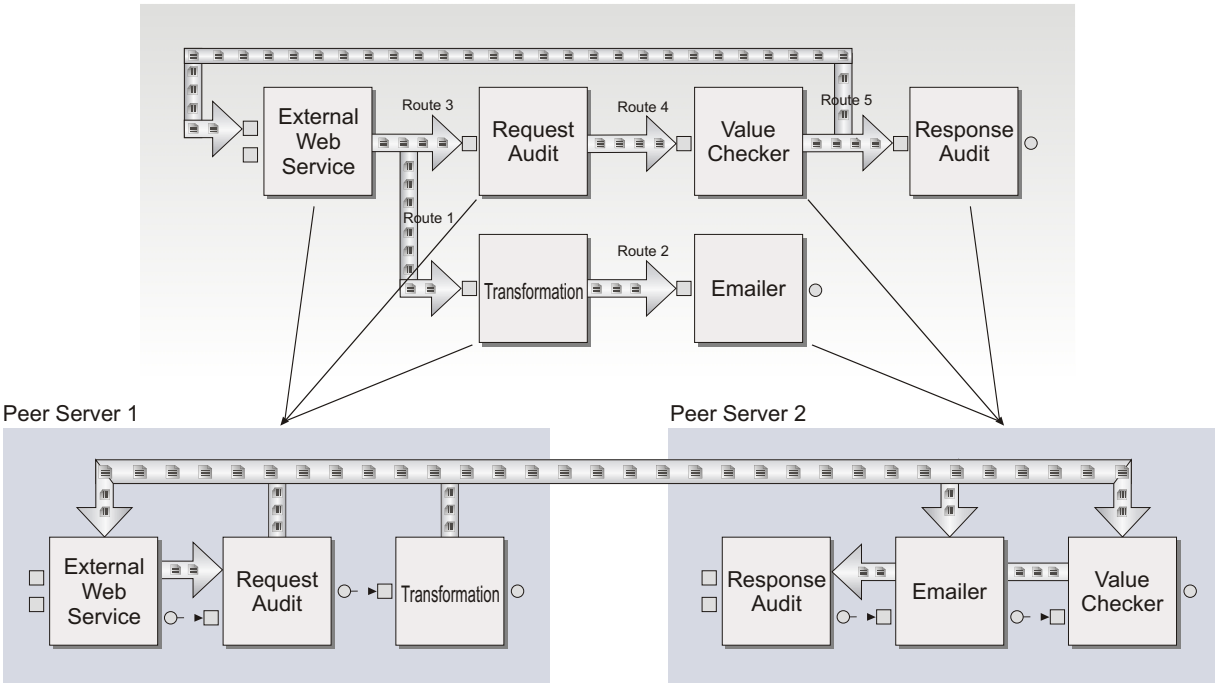

Figure 6: Integration flow with components deployed across two separate machines/servers

Figure 6 illustrates an integration flow with three components running on machine 1 and two on machine 2. If it is later determined that the Transformation component on machine 2 is taking too much time and slowing down the process, it is possible to run the transformation component on an independent third machine and re-route the message-queues, as illustrated in *Figure 7.*

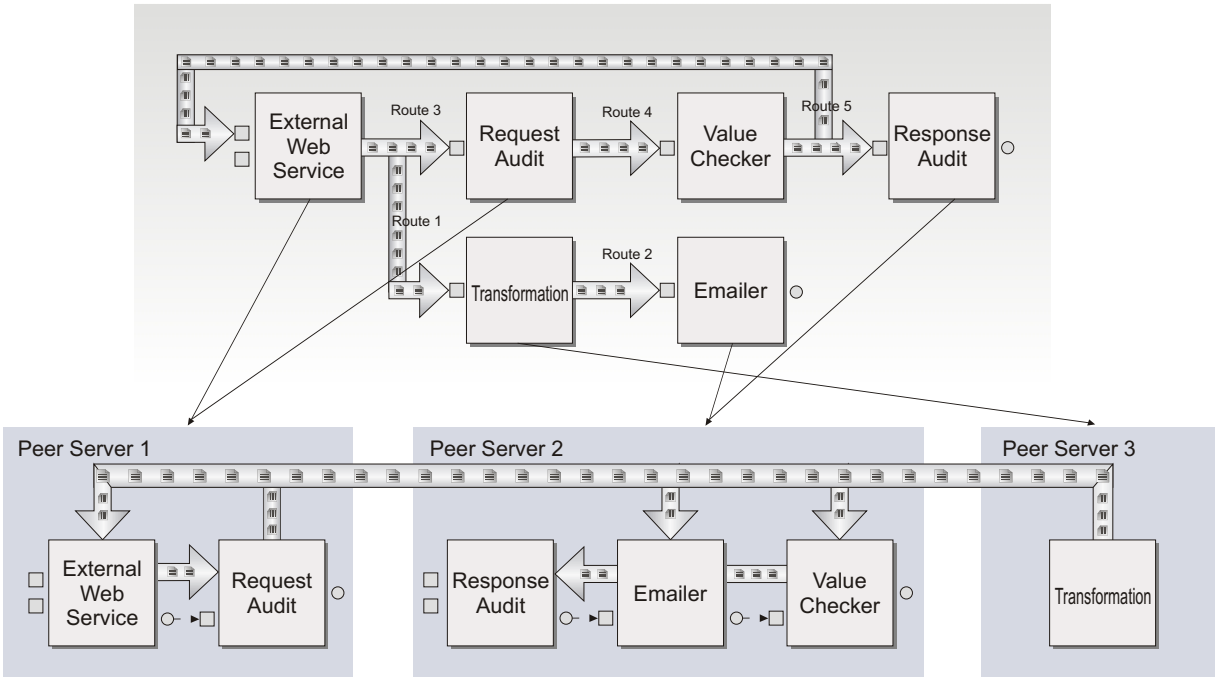

Figure 7: Integration flow of Figure 6 re-partitioned to run across three machines/servers

As such, whenever one identifies a bottleneck in a flow and needs more computing power for one or more of the participating components in the flow, one can simply deploy 'part' of the integration flow/process across different machines. Importantly, no 'process reengineering' is required to achieve this flexibility and depending on how the integration infrastructure is designed, the redeployment can be performed from a central administrative location. Further, the machines could be either on-premise or in the cloud or a combination of both, allowing the asynchronous architecture to seamlessly support hybrid integration scenarios.

Runtime debugging and modification, in practice

An asynchronous message-driven architecture supports runtime process changes, in practice. Specifically, (i) Integration flows can be paused/resumed in practice, (ii) Components as well as messages passing through can be modified while the flow is running and (iii) Steps can be added/removed in practice.

As explained in the previous section, each participating component in an integration flow is completely decoupled of all other components in the flow, since all communication is handled via asynchronous messages. Provided the underlying infrastructure allows the message queues to be intercepted and messages to be inspected on demand, it follows that any component of an integration flow can be paused and resumed in practice (by pausing its input queues). When a component is paused, messages just build up on the input queues and are consumed when the input queue(s) are unpaused. A particular message running through the flow can be viewed and modified in the "live" flow. This process of message-interception and modification in a live, running integration process serves as an efficient runtime-debugging tool and is extremely useful in real-world implementations, significantly shortening the development/test/deployment cycle. It should be noted that the pause/resumption of an integration flow is dependent purely on the underlying messaging infrastructure and is independent of the participating components of a flow; it is the queues that are paused, giving the impression that the downstream components have suspended execution due to the lack of availability of input data.

The loose-coupling and asynchronous nature of the communication between the components in an integration flow allows steps (i.e. components) to be added to or removed from the integration flow at runtime. When a new component is added to a flow, other components remain unaffected since the new component is fed by data routed via message-queues which are set up dynamically at runtime. Figure 8 shows a new component SMTP being added to an existing integration flow by routing data from the DB component to the input of the SMTP component. Output data from DB could be routed to a different component in the flow or to another flow altogether.

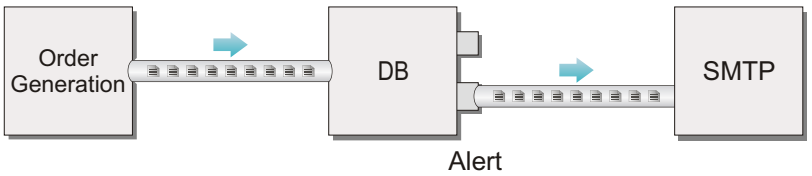

Figure 8: Dynamic addition of component "SMTP" to a running integration flow

Scalability via Distributed, Peer-to-Peer messaging: achieving "Cloud Scale"

With the application partitioning enabled by the messaging-approach, different components in an integration flow can be attached to different peer messaging-servers, allowing load to be distributed and scaled across the cloud as required by the underlying business requirements. This allows data to be processed in parallel, increasing throughput and performance, while control can still be centralized. *Figure 9* illustrates the architecture of such a centrally managed, peer-to-peer message-driven integration platform.

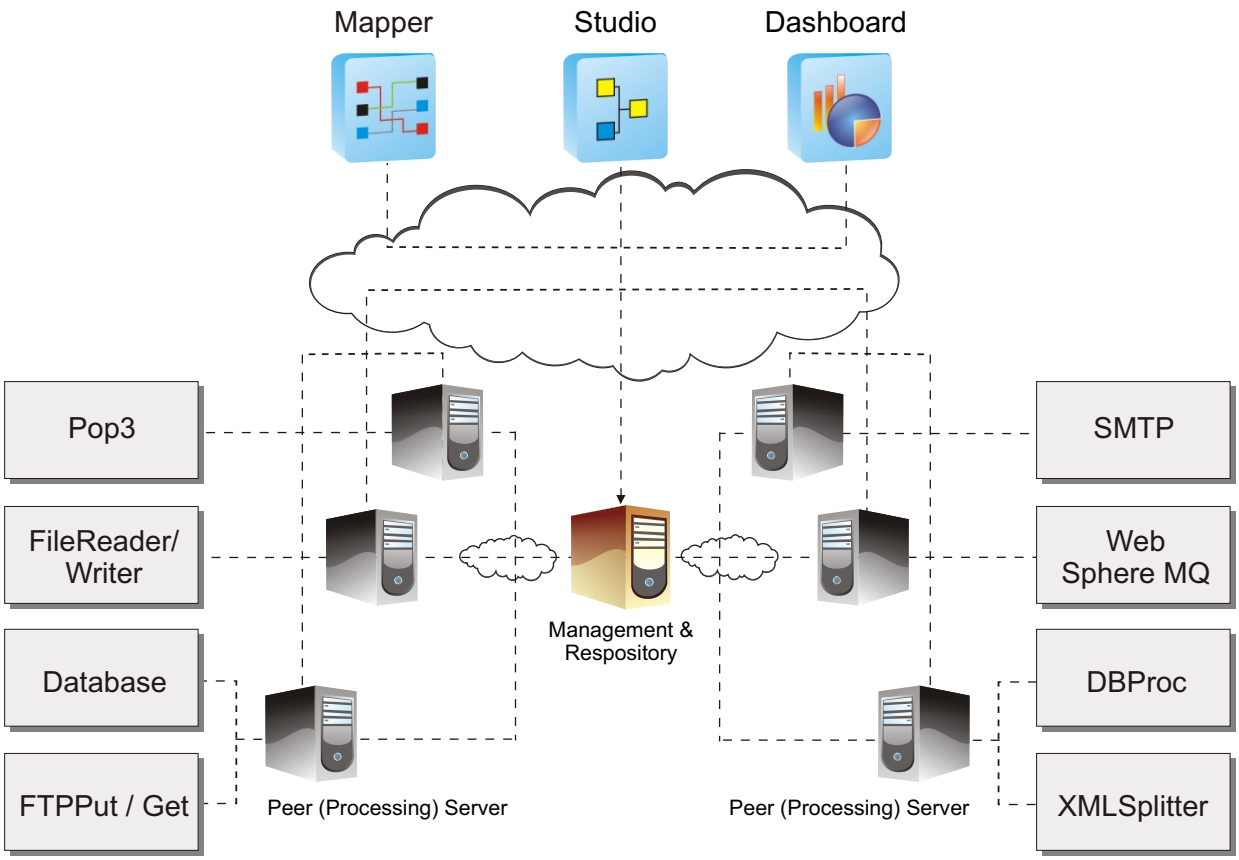

Figure 9: Centrally managed, peer-to-peer message-driven integration system

The architecture described in this paper and illustrated in *Figure 9* is implemented by the Fiorano platform. Components are implemented as microservices and "integration flows" represented as collections of microservices connected via asynchronous message "routes" execute on one or more "peer servers" in the cloud, on-premise or in a hybrid cloud/on-premise environment. For more information, please visit www.fiorano.com.

ABOUT FIORANO SOFTWARE

Founded in 1995, Silicon Valley based Fiorano is a USA (California) Corporation, a trusted provider of enterprise integration middleware, high performance messaging and peer-to-peer distributed systems. Fiorano powers real time, digital enterprises with bimodal integration and API Management strategy that leverages the best of systematic (centralized, high-control) and adaptive (federated, high-speed) approaches to deliver solutions across cloud, on-premise and hybrid environments.

Global leaders including Boeing, British Telecom, Federal Bank, L'Oreal, McKesson, NASA, POSCO, Rabobank, Royal Bank of Scotland, Schlumberger, Temenos, US Coast Guard and Vodafone have deployed Fiorano to drive innovation through open, standards-based, event-driven real-time solutions yielding unprecedented productivity.

To find out more about how Fiorano can help you meet your enterprise integration objectives, visit **www.fiorano.com** or e-mail **sales@fiorano.com**

www.fiorano.com

AMERICAS
Fiorano Software, Inc.
230 S. California Avenue, Suite 103, Palo Alto, CA 94306 USA
Tel: +1 650 326 1136
Fax: +1 646 607 5875
Toll-Free: +1 800 663 3621
Email: info@fiorano.com

EMEA
Fiorano Software Ltd
3000 Hillswood Drive
Hillswood Business Park
Chertsey Surrey KT16 0RS UK
Tel: +44 (0) 1932 895005
Fax: +44 (0) 1932 325413
Email: info_uk@fiorano.com

APAC
Fiorano Software Pte. Ltd.
Level 42, Suntec Tower Three
8 Temasek Boulevard
038988 Singapore
Tel: +65 68292234
Fax: +65 68292235
Email: info_asiapac@fiorano.com