Service-Oriented Architecture— An Integration Blueprint

Chapter No.3,4
"Integration Architecture Blueprint Implementation scenarios"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.3,4 "Integration Architecture Blueprint Implementation scenarios"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Daniel Liebhart comes with over 20 years of experience in information technology field. Since 10 years he is working in different management positions leading IT professional services or product development. His broad know-how comprises the engineering, realization and operation of complex and internationally operated IT systems for the Telecommunication-, the Finance-, the Logistic- and the Chemical-Industry as well as for public services. Daniel Liebhart is a passionate computer science engineer, the owner several awards and works as an assistant professor at the University of Applied Science in Zurich. His broad technical and business know-how is combined with a communicative and competent appearance.

Guido Schmutz works as a Technology Manager for Service-Oriented Architectures (SOA), Application Integration and Open Source based development at Trivadis, a leading independent IT service company operating in Germany, Austria and Switzerland.

For More Information:
He has worked in the software industry for over 20 years, most of which was spent in software development and on architecting. He has led a wide variety of enterprise-level application development and application integration projects in a wide variety of industries. His current focus is on SOA and Application Integration projects using the Oracle SOA platform. Guido is also an Oracle ACE Director for Fusion Middleware and SOA Guido is co-author of the german books "Spring 2.0 im Einsatz", "Architecture Blueprints" and "Integration Architecture Blueprints and he is a regular speaker at International Oracle and Java conferences.

**Peter Welkenbach** works as a consultant, senior architect and trainer in the fields of requirement engineering, object-oriented methodologies, software engineering and quality management. He has more than 20 years’ experience of designing and implementing complex information systems for banks, automotive manufacturers and pharmaceutical companies. For 10 years he has been a technology evangelist for Java technology and the use of the corresponding frameworks in customer projects. Peter Welkenbach is a course developer, author of numerous publications and speaker at JAX and international Oracle conferences. He has been using Spring in numerous customer projects since it first appeared in summer 2003. His current focus is enterprise architecture and lean architecture methodologies. In his current projects he works as an enterprise architect for a well known german retailer.

For More Information:
Service-Oriented Architecture—An Integration Blueprint

With the widespread use of service-oriented architecture (SOA), the integration of different IT systems has gained a new relevance. The era of isolated business information systems—so-called silos or stove-pipe architectures—is finally over. It is increasingly rare to find applications developed for a specific purpose that do not need to exchange information with other systems. Furthermore, SOA is becoming more and more widely accepted as a standard architecture. Nearly all organizations and vendors are designing or implementing applications with SOA capability. SOA represents an end-to-end approach to the IT system landscape as the support function for business processes. Because of SOA, functions provided by individual systems are now available in a single standardized form throughout organizations, and even outside their corporate boundaries. In addition, SOA is finally offering mechanisms that put the focus on existing systems, and make it possible to continue to use them. Smart integration mechanisms are needed to allow existing systems, as well as the functionality provided by individual applications, to be brought together into a new fully functioning whole. For this reason, it is essential to transform the abstract concept of integration into concrete, clearly structured, and practical implementation variants.

The Trivadis Integration Architecture Blueprint indicates how integration architectures can be implemented in practice. It achieves this by representing common integration approaches, such as Enterprise Application Integration (EAI); Extract, Transform, and Load (ETL); event-driven architecture (EDA); and others, in a clearly and simply structured blueprint. It creates transparency in the confused world of product developers and theoretical concepts.

The Trivadis Integration Architecture Blueprint shows how to structure, describe, and understand existing application landscapes from the perspective of integration. The process of developing new systems is significantly simplified by dividing the integration architecture into process, mediation, collection and distribution, and communication layers. The blueprint makes it possible to implement application systems correctly without losing sight of the bigger picture: a high performance, flexible, scalable, and affordable enterprise architecture.

What This Book Covers

Despite the wide variety of useful and comprehensive books and other publications on the subject of integration, the approaches that they describe often lack practical relevance. The basic issue involves, on the one hand, deciding how to divide an integration solution into individual areas so that it meets the customer requirements, and on the other hand, how it can be implemented with a reasonable amount of effort. In this case, this means...
structuring it in such a way that standardized, tried-and-tested basic components can be combined to form a functioning whole, with the help of tools and products. For this reason, the Trivadis Integration Architecture Blueprint subdivides the integration layer into further layers. This kind of layering is not common in technical literature, but it has been proven to be very useful in practice. It allows any type of integration problem to be represented, including traditional ETL (Extract, Transform, and Load), classic EAI (Enterprise Application Integration), EDA (event-driven architecture), and grid computing. This idea is reflected in the structure of the book.

Chapter 1, Basic Principles, covers the fundamental integration concepts. This chapter is intended as an introduction for specialists who have not yet dealt with the subject of integration.

Chapter 2, Base Technologies, describes a selection of base technologies. By far the most important of these are transaction strategies and their implementation, as well as process modeling. In addition, Java EE Connector Architecture (JCA), Java Business Integration (JBI), Service Component Architecture (SCA), and Service Data Objects (SDO) are explained. Many other base technologies are used in real-life integration projects, but these go beyond the scope of this book.

Chapter 3, Integration Architecture Blueprint, describes the Trivadis Integration Architecture Blueprint. The process of layering integration solutions is fully substantiated, and each step is explained on the basis of the division of work between the individual layers. After this, each of the layers and their components are described.

Chapter 4, Implementation Scenarios, demonstrates how the Trivadis Integration Architecture Blueprint represents the fundamental integration concepts that have been described in Chapter 1. We will use the blueprint with its notation and visualization to understand some common integration scenarios in a mostly product-neutral form. We will cover traditional, as well as modern, SOA-driven integration solutions.

Chapter 5, Vendor Products for Implementing the Trivadis Blueprint, completes the book with a mapping of some vendor platforms to the Trivadis Integration Architecture Blueprint.

For More Information:
The Trivadis Integration Architecture Blueprint specifies the building blocks needed for the effective implementation of integration solutions. It ensures consistent quality in the implementation of integration strategies as a result of a simple, tried-and-tested structure, and the use of familiar integration patterns (Hohpe, Wolf 2004).

Standards, components, and patterns used
The Trivadis Integration Architecture Blueprint uses common standardized techniques, components, and patterns, and is based on the layered architecture principle.

A layered architecture divides the overall architecture into different layers with different responsibilities. Depending on the size of the system and the problem involved, each layer can be broken down into further layers. Layers represent a logical construct, and can be distributed across one or more physical tiers. In contrast to levels, layers are

For More Information:
organized hierarchically, and different layers can be located on the same level. Within the individual layers, the building blocks can be strongly cohesive. Extensive decoupling is needed between the layers. The rule is that higher-level layers can only be dependent on the layers beneath them and not vice versa. Each building block in a layer is only dependent on building blocks in the same layer, or the layers beneath. It is essential to create a layer structure that isolates the most important cohesive design aspects from one another, so that the building blocks within the layers are decoupled.

The blueprint is process oriented, and its notation and structure are determined by the blueprint's dependencies and information flow in the integration process. An explanation of how the individual layers, their building blocks, and tasks can be identified from the requirements of the information flow is given on the basis of a simple scenario. In this scenario, the information is transported from one source to another target system using an integration solution.

In the blueprint, the building blocks and scenarios are described using familiar design patterns from different sources:

- (Hohpe, Wolf 2004)
- (Adams et al. 2001)
- (Coral8 2007)
- (Russel et al. 2006)

These patterns are used in a shared context on different layers. The Trivadis Integration Architecture Blueprint includes only the integration-related parts of the overall architecture, and describes the specific view of the technical integration domain in an overall architecture. It focuses on the information flow between systems in the context of domain-driven design.

Domain-driven design is a means of communication, which is based on a profound understanding of the relevant business domain. This is subsequently modeled specifically for the application in question. Domain models contain no technical considerations and are restricted exclusively to business aspects. Domain models represent an abstraction of a business domain, which aims to capture the exemplary aspects of a specific implementation for this domain. The objectives are:

- To significantly simplify communication between domain experts and developers by using a common language (the domain model)
- To enable the requirements placed on the software to be defined more accurately and in a more targeted way
- It must be possible to describe, specify, and document the software more precisely and more comprehensively, using a clearly defined language, which will make it easier to maintain

For More Information:
The technical aspects of architecture can be grouped into domains in order to create specific views of the overall system. These domains cover security, performance, and other areas. The integration of systems and information also represents a specific view of the overall system, and can be turned into a domain.

**Integration domain** is used to mean different things in different contexts. One widely-used meaning is "application domain", in other words, a clearly defined, everyday problem area where computer systems and software are used. Enterprise architectures are often divided into business and technical domains:

- **Business domains** may include training, resource management, purchasing, sales or marketing, for example.
- **Technical domains** are generally areas such as applications, integration, network, security, platforms, systems, data, and information management.

The blueprint, however, sees integration as a technical domain, which supports business domains, and has its own views that can be regarded as complementary to the views of other architecture descriptions.

In accordance with Evans (Evans, 2004), the Trivadis Integration Architecture Blueprint is a *ubiquitous language* for describing integration systems. This and the structure of the integration domain on which it is based, have been tried and tested in a variety of integration projects using different technologies and products. The blueprint has demonstrated that it offers an easy-to-use method for structuring and documenting implementation solutions. As domain models for integration can be formulated differently depending on the target platform (for example, an object-oriented system or a classic ETL solution), the domain model is not described in terms of object orientation. Instead, the necessary functionality takes the form of building blocks (which are often identical with familiar design patterns) on a higher level of abstraction. This makes it possible to use the blueprint in a heterogeneous development environment with profitable results.

An *architecture blueprint* is based on widely-used, tried-and-tested techniques, components and patterns, which are grouped into a suitable structure to meet the requirements of the target domain.

The concepts, the functionality, and the building blocks to be implemented are described in an abstract form in blueprints. These are then replaced or fine-tuned by product-specific building blocks in the implementation project. Therefore, the Trivadis Integration Architecture Blueprint has been deliberately designed to be independent of individual vendors, products, and technologies. It includes integration scenarios and proposals that apply to specific problems, and can be used as aids during the project implementation process. The standardized view of the integration domain and the standardized means of representation enable strategies, concepts, solutions, and products to be compared with one another more easily in evaluations of architectures.

For More Information:
The specifications of the blueprint act as guidelines. Differences between this model and reality may well occur when the blueprint is implemented in a specific project. Individual building blocks and the relationships between them may not be needed, or may be grouped together. For example, the adapter and mapper building blocks may be joined together to form one component in implementation processes or products.

**Structuring the integration blueprint**

The following diagram is an overview of the Trivadis Integration Architecture Blueprint. It makes a distinction between the application and information view and the integration view.

**Insert image 1049EN_03_01.png**

The application and information view consists of external systems, which are to be connected together by an integration solution. These are source or target entities in the information flow of an integration solution. Generally one physical system can also take on both roles. The building blocks belonging to the view, and the view itself, must be regarded as external to the integration system that is being described and, therefore, not the subject of the integration blueprint. The external systems can be divided into three main categories:

For More Information:
• **Transactional information storage**: This includes classic relational database management systems (RDBMS) and messaging systems (queues, topics). The focus is on data integration.

• **Non-transactional information storage**: This primarily file-based systems and non-relational data stores (NoSQL) with a focus on data integration.

• **Applications**: Applications include transactional or non-transactional systems that are being integrated (ERP—Enterprise Resource Planning, CMS—Content Management System, and so on) and can be accessed through a standardized API (web service, RMI/IIOP, DCOM, and so on). The focus is on application and process integration.

The integration view lies at the heart of the integration blueprint and is divided (on the basis of the principle of divide and conquer) into the following levels:

• **Transport level**: The transport level encapsulates the technical details of communication protocols and formats for the external systems. It contains:
  - **Communication layer**: The communication layer is part of the transport level, and is responsible for transporting information. This layer links the integration solution with external systems, and represents a type of gateway to the infrastructure at an architectural level. It consists of transport protocols and formats.

• **Integration domain level**: The integration domain level covers the classic areas of integration, including typical elements of the integration domain, such as adapters, routers, and filters. It is divided into:
  - **Collection/distribution layer**: This layer is responsible for connecting components. It is completely separate from the main part of the integration domain (mediation). The building blocks in this layer connect the mediation layer above with the communication layer below. The layer is responsible for encapsulating external protocols and their technical details from the integration application, and transforming external technical formats into familiar internal technical formats.
  - **Mediation layer**: This layer is responsible for forwarding information. Its main task is to ensure the reliable forwarding of information to business components in the process layer, or directly to output channels that are assigned to the collection/distribution layer, and that distribute data to the target systems. This is the most important functionality of the integration domain. In more complex scenarios, the information forwarding process can be enhanced by information transformation, filtering, and so on.

For More Information:
• **Application level**: The application level encapsulates the integration management and process logic. It is an optional level and contains:
  
  o **Process layer**: The process layer is part of the application level, and is responsible for orchestrating component and service calls. It manages the integration processes by controlling the building blocks in the mediation layer (if they cannot act autonomously).

The integration view contains additional functionality that cannot be assigned to any of the levels and layers referred to above. This functionality consists of so-called cross-cutting concerns that can be used by building blocks from several other layers. Cross-cutting concerns include:

• **Assembly/deployment**: Contains configurations (often declarative or scripted) of the components and services. For example, this is where the versioning of Open Service Gateway initiative (OSGi) services is specified.

• **Transaction**: Provides the transaction infrastructure used by the building blocks in the integration domain.

• **Security/management**: This is the security and management infrastructure used by the building blocks in the integration domain. It includes, for example, libraries with security functionality, JMX agents and similar entities.

• **Monitoring, BAM, QoS**: These components are used for monitoring operations. This includes ensuring compliance with the defined Service Level Agreements (SLA) and Quality of Service (QoS). Business Activity Monitoring (BAM) products can be used for monitoring purposes.

• **Governance**: These components and artifacts form the basis for SLAs and QoS. The artifacts include business regulations, for example. In addition, this is where responsibilities, functional and non-functional requirements, and accounting rules for the services/capacities used are defined.

4

Implementation scenarios

For More Information:
Having understood the structure of the blueprint covered in Chapter 3, Integration Architecture Blueprint, this chapter will use individual scenarios to illustrate how the business pattern can be implemented using the Integration Architecture Blueprint.

The scenarios shown in this chapter have been deliberately designed to be independent of specific vendor products, and are based solely on the building blocks that form part of the different layers of the blueprint.

The symbols used have the same semantic meaning as described in Chapter 3.

This chapter will:

- Explain service-oriented integration scenarios
- Use scenarios to show how data integration business patterns can be implemented
- Present a description of scenarios for implementing the business patterns for EAI/EII integration
- Look in detail at the implementation of event processing business patterns
- Describe a scenario for implementing business patterns for grid computing and Extreme Transaction Processing (XTP)
- Explain how an SAP ERP system can be combined with the integration blueprint
- Explain how an existing integration solution can be modernized using SOA, and describe a scenario that has already been implemented in practice
- Combine the integration blueprint with the other Trivadis Architecture Blueprints

Service-oriented integration scenarios

These scenarios show how the service-oriented integration business patterns described in Chapter 1 can be implemented. These business patterns are as follows:

- **Process integration**: The process integration pattern extends the 1: N topology of the broker pattern. It simplifies the serial execution of business services, which are provided by the target applications.

- **Workflow integration**: The workflow integration pattern is a variant of the serial process pattern. It extends the capability of simple serial process orchestration to include support for user interaction in the execution of individual process steps.
Implementing the process integration business pattern

In the scenario shown in the following diagram, the process integration business pattern is implemented using BPEL.

Insert image 1049EN_04_05.png

Trigger:

An application places a message in the queue.

Primary flow:

- The message is extracted from the queue through JMS and a corresponding JMS adapter.
- A new instance of the BPEL integration process is started and the message is passed to the instance as input.
- The integration process orchestrates the integration and calls the systems that are to be integrated in the correct order.
- A content-based router in the mediation layer is responsible for ensuring that the correct one of the two systems is called. However, from a process perspective, this is only one stage of the integration.
- In the final step, a “native” integration of an EJB session bean is carried out using an EJB adapter.

Variant with externalized business rules in a rule engine

A variant of the previous scenario has the business rules externalized in a rule engine, in order to simplify the condition logic in the integration process. This corresponds to the

For More Information:
external business rules variant of the process integration business pattern, and is shown in
the form of a scenario in the following diagram:

![Diagram showing process integration and integration process with business rules]

**Insert image 1049EN_04_06.png**

**Trigger:**

The JEE application sends a SOAP request.

**Primary flow:**

1. The SOAP request initiates a new instance of the integration process.
2. The integration process is implemented as before, with the exception that in
   this case, a rule engine is integrated before evaluating the condition. The call
to the rule engine from BEPL takes the form of a web service call through
   SOAP.
3. Other systems can be integrated via a DB adapter as shown here, for example
to enable them to write to a table in an Oracle database.

**Variant with batch-driven integration process**

In this variant, the integration process is initiated by a time-based event. In this case, a
job scheduler added before the BPEL process triggers an event at a specified time, which
starts the process instance. The process is started by the scheduler via a web service call.
The following diagram shows the scenario:

---

**For More Information:**

Insert image 1049EN_04_07.png

Trigger:
- The job scheduler building block does a web service request at a specified time.

Primary flow:
1. The call from the job scheduler via SOAP initiates a new integration process instance.
2. As in the previous variants, the BPEL process executes the necessary integration steps and, depending on the situation, integrates one system via a database adapter, and the other directly via a web service call.

Implementing the workflow business pattern
In this scenario, additional user interaction is added to the integration process scenario. As a result, the integration process is no longer fully automated. It is interrupted at a specific point by interaction with the end user, for example, to obtain confirmation for a certain procedure. This scenario is shown in the image below.

For More Information:
Trigger:

An application places a message in the queue.

Primary flow:

1. The message is removed from the queue by the JMS adapter and a new instance of the integration process is started.
2. The user interaction takes place through the asynchronous integration of a task service. It creates a new task, which is displayed in the user’s task list.
3. As soon as the user has completed the task, the task service returns a callback to the relevant instance of the integration process, and by that, informs the process of the user’s decision.
4. The integration process responds to the decision and executes the remaining steps.

Modernizing an integration solution

This section uses an example to illustrate how an existing integration solution that has grown over time can be modernized using SOA methods, and the scenarios from the previous sections.

The example is a simplified version of a specific customer project in which an existing solution was modernized with the help of SOA.

The task of the integration solution is to forward orders entered in the central ERP system to the external target applications.

For More Information:

**Initial situation**

The current solution is primarily based on a file transfer mechanism that sends the new and modified orders at intervals to the relevant applications, in the form of files in two possible formats (XML und CSV). The applications are responsible for processing the files independently.

At a later date, another application (IT app in the following diagram) was added to the system using a queuing mechanism, because this mechanism allowed for the guaranteed exchange of messages with the application by reading new orders, and sending appropriate messages through the queue in the form of a transaction.

The following diagram shows the initial situation before the modernization process took place:

![Diagram showing the initial situation](1049EN_04_21.png)

The extraction and file creation logic is written in PL/SQL. A Unix shell script is used to send the files through the **File Transfer Protocol (FTP)**, as no direct FTP call was

---

**For More Information:**

possible in PL/SQL. Both a shell script and the PL/SQL logic are responsible for orchestrating the integration process.

**Oracle Advanced Queuing (AQ)** is used as the queuing infrastructure. As PL/SQL supports sending of AQ messages through an API (package), it was possible to implement this special variant of the business case entirely in PL/SQL, without a call to a shell script being needed. In this case, the integration is bi-directional. This means that when the order has been processed by the external system, the application must send a feedback message to the ERP system. A second queue, which is implemented in the integration layer using PL/SQL, is used for this purpose.

**Sending new orders**

**Trigger:**

The job scheduler triggers an event every 30 minutes for each external system that has to be integrated.

**Flow:**

1. The event triggered by the job scheduler starts a shell script, which is responsible for part of the orchestration.
2. The shell script first starts a PL/SQL procedure that creates the files, or writes the information to the queue.
3. The PL/SQL procedure reads all the new orders from the ERP system’s database, and enriches them with additional information about the product ordered and the customer.
4. Depending on the external target system, a decision is made as to whether the information about the new order should be sent in the form of files, or messages in queues.
5. The target system can determine in which format (XML or CSV) the file should be supplied. A different PL/SQL procedure is called depending on the desired format.
6. The PL/SQL procedure writes the file in the appropriate format using a PL/SQL tool (in other words, the built-in package `UTL_FILE`) to the database server. The database server is used only for interim storage of the files, as these are uploaded to the target systems in the next step.
7. The main shell script starts the process of uploading the files to the external system, and another shell script completes the task.
8. The files are made available on the external system and are processed in different ways depending on the application in question.
9. A PL/SQL procedure is called to send the order information through the queue. The procedure is responsible for formatting and sending the message.

For More Information:

10. The document is now in the output queue (send) ready to be consumed.
11. The application (IT app) consumes the messages from the queue immediately and starts processing the order.
12. When the order has been processed, the external application sends a message to the feedback queue (receive).

Receiving the confirmation

Trigger:

The job scheduler triggers an event every 15 minutes.

Flow:

1. The job scheduler event starts a PL/SQL procedure, which processes the feedback message.
2. The message is consumed from the feedback queue (receive).
3. A SQL UPDATE command updates the status of the order in the ERP database.

Evaluation of the existing solution

By evaluating the existing solution we came to the following conclusions:

• This is an integration solution that has grown up over time using a wide variety of different technologies.
• A batch solution which does not allow real-time integration or which makes this very difficult.
• Exchanging information in files is not really a state-of-the-art solution.
  o Data cannot be exchanged reliably, as FTP does not support transactions.
  o Error handling and monitoring are difficult and time-consuming. (It’s not easy to determine if the IT app does not send a response.)
  o Files must be read and processed by the external applications, all of which use different methods.
• Integrating new distribution channels (such as web services) is difficult, as neither PL/SQL nor shell scripts are the ideal solution in this case.
• Many different technologies are used. The integration logic is distributed, which makes maintenance difficult:
  o Job scheduler (for orchestration)
  o PL/SQL (for orchestration and mediation)
  o Shell script (for orchestration and mediation)
• Different solutions are used for files and queues.

For More Information:

Many of these disadvantages are purely technical. From a business perspective, only the first disadvantage represents a real problem. The period of a maximum of 30 minutes between the data being entered in the ERP system, and the external systems being updated, is clearly too long. From a technical point of view, it is not possible to reduce this amount of time, as the batch solution overhead is significant and, in the case of shorter cycles, the total overhead would be too large.

Therefore, the decision was made to modernize the existing integration solution and to transform it into an event-driven, service-oriented integration solution based on the processing of individual orders.

**Modernizing—integration with SOA**

The main objective of the modernization process, from a business perspective, is the real-time integration of orders.

From a technical standpoint, there are other objectives, including the continued use of the batch mode through file connections. This means that the new solution must completely replace the old one, and the two solutions should not be left running in parallel. A further technical objective is that of improved support as a result of the introduction of a suitable infrastructure.

On the basis of these considerations, a new SOA-based integration architecture was proposed and implemented, as shown in the following diagram:
Trigger:

Each new order is published to a queue in the ERP database, using the change data capture functionality of the ERP system.

Flow:

1. The business event is consumed from the queue by an event-driven consumer building block in the ESB. The corresponding AQ adapter is used for this purpose.
2. A new BPEL process instance is started for the integration process. This instance is responsible for orchestrating all the integration tasks for each individual order.
3. First, the important order information concerning the products and the customer must be gathered, as the ERP system only sends the primary key for the new order in the business event. A service is called on the ESB that uses a database adapter to read the data directly from the ERP database, and compiles it into a message in canonical format.

For More Information:
4. A decision is made about the system to which the order should be sent, and about whether feedback on the order is expected.

5. In the right-hand branch, the message is placed in the existing output queue (send). A message translator building block converts the order from the canonical format, to the message format used so far, before it is sent. The AQ adapter supports the process of sending the message. The BPEL process instance will be paused until the callback from the external applications is received.

6. The message is processed by the external application in the same way as before. The message is retrieved, the order is processed and, at a specified time, a feedback message is sent to the feedback queue (receive).

7. The paused BPEL process instance is reactivated and consumes the message from the feedback queue.

8. An invoke command is used to call another service on the ESB, which modifies the status of the ERP system in a similar way to the current solution. This involves a database adapter making direct modifications to a table or record in the ERP database.

9. In the other case, which is shown in the branch on the left, only a message is sent to the external systems. Another service is called on the ESB for this purpose, which determines the target system and the target format based on some information passed in the header of the message.

10. The ESB uses a header-based router to support the content-based forwarding of the message.

11. Depending on the target system, the information is converted from the canonical format to the correct target format.

12. The UK app already has a web service, which can be used to pass the order to the system. For this reason, this system is connected via a SOAP adapter.

13. The two other systems continue to use the file-based interface. Therefore, an FTP adapter creates and sends the files through FTP in XML or CSV format.

14. In order to ensure that the external application (labeled GE app in the diagram) still receives the information in batch mode, with several orders combined in one file, an aggregator building block is used. This collects the individual messages over a specific period of time, and then sends them together in the form of one large message to the target system via the FTP adapter.

15. An aggregation process is not needed for the interface to the other external application (labeled CH app in the image), as this system can also process a large number of small files.

For More Information:
Evaluation of the new solution
An evaluation of the new solution shows the following benefits:

- The orchestration is standardized and uses only one technology.
- One BPEL instance is responsible for one order throughout the entire integration process.
  - This simplifies the monitoring process, because the instance continues running until the order is completed; in other words, in one of the two cases until the feedback message from the external system has been processed.
- The orchestration is based only on the canonical format. The target system formats are generated at the last possible moment in the mediation layer.
  - Additional distribution channels can easily be added on the ESB, without having to modify the orchestration process.
  - The solution can easily support other protocols or formats that are not yet known, simply by adding an extra translator building block.

For More Information:
Where to buy this book

You can buy Service-Oriented Architecture— An Integration Blueprint from the Packt Publishing website: https://www.packtpub.com/service-oriented-architecture-an-integration-blueprint/book

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.