Toward A Collection of Cloud Integration Patterns

Daniel Ritter\textsuperscript{1} and Stefanie Rinderle-Ma\textsuperscript{2}

\textsuperscript{1} SAP SE, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany
daniel.ritter@sap.com

\textsuperscript{2} University of Vienna, Währingerstraße 29, 1090 Wien, Austria
stefanie.rinderle-ma@univie.ac.at

Abstract. Cloud computing is one of the most exciting IT trends nowadays. It poses several challenges on application integration with respect to, for example, security. In this work we collect and categorize several new integration patterns and pattern solutions with a focus on cloud integration requirements. Their evidence and examples are based on extensive literature and system reviews.

Keywords: Cloud integration, device integration, enterprise application integration, enterprise integration patterns, hybrid integration.

1 Introduction

In this work we revisit the current collection of Enterprise Integration Patterns (EIP) in the context of cloud integration. On the basis of a quantitative analysis of several cloud integration scenarios on a well-established platform (i.e., \cite{22}), we conducted further qualitative literature and system reviews to collect and categorize additional characteristics. These characteristics are verified by cross-referencing them between the quantitative and the qualitative studies, categorized and formulated as a list of patterns or pattern solutions.

2 Cloud Integration Patterns

In this section we collect and define new integration patterns from the cloud integration domain as extensions to the EIPs \cite{9}. The pattern descriptions are represented in the format in Tab. 73.

The categories we consider in this work are storage in Sect. 2.1, messaging patterns like transformation and routing in Sect. 2.2, security in Sect. 2.3, exception handling Sect. 2.4, monitoring and operations in Sect. 2.5, as well as adapter and endpoint patterns in Sect. 2.6.

2.1 Storage Patterns

In addition to the Message Store \cite{9}, several vendors identified the need for further storage patterns as in Fig. 1, e.g., the storage of scheduler timings in
Sect. 2.5, and persistent patterns like *Aggregator* [9] in a data store (cf. Tab. 1). During the processing, some vendors added capabilities to remember previous processing values, e.g., variables, in a transient store Tab. 2. These stores can be accessed using a store accessor (cf. Tab. 3).

![Storage Patterns](image)

**Fig. 1.** Storage Patterns.

| Pattern Name | Data Store |
|--------------|------------|
| Intent       | Store arbitrary content explicitly or implicitly from persistent patterns. |
| Driving Question | How can arbitrary data be stored persistent across instances for access by arbitrary patterns? |
| Solution     | Provide a persistent store, which allows to store variables, value mapping, data of persistent patterns, “in-memory” storage capabilities. |
| Data Aspects | visibility: local (integration scenario), global (multiple integration scenarios or even solutions), multi-tenant, reliability: redundant, high-available, disaster recoverable; persistent |
| Variations   | e.g., relational column or row store, NoSQL store |
| Example      | Persistent Scheduler, storage and archiving for legal regulations |
| Related Patterns | Message Store [9] |
| Known Uses  | partially covered by “Message Store” [11], “Flow reliability” [1], “StoreInKite- Dataset” [2], “DBStorage”, “Persist” [22] |

In addition, several security patterns from Sect. 2.3 require stored secure material like certificates, public/private keys in Tab. 4 or Tab. 5, tokens and user/password in Tab. 6.
### Table 2. Transient Store

| Pattern Name       | Transient Store                                      |
|--------------------|------------------------------------------------------|
| **Intent**         | Remember arbitrary content explicitly within one integration scenario. |
| **Driving Question** | How to remember arbitrary data in one integration scenario context transiently? |
| **Solution**       | Provide a transient store, which allows to remember variables, partial or complete messages in transient, “in-memory” storage capabilities. |
| **Data Aspects**   | visibility: local to one integration scenario; transient; structured / unstructured |
| **Variations**     | e.g., relational column or row store, NoSQL store |
| **Example**        | Variables                                           |
| **Related Patterns** | Claim Check [9]                                      |
| **Known Uses**     | partially covered by “Exchange Properties” [11], “Variables” [22] |

### Table 3. Store Accessor

| Pattern Name       | Store Accessor                                      |
|--------------------|------------------------------------------------------|
| **Intent**         | Access the data store.                               |
| **Driving Question** | How to access the data store?                        |
| **Data Aspects**   | transactions and data access depending on the data store, e.g., query, read, write, delete |
| **Example**        | Store timings of a Persistent Scheduler, aggregates of an Aggregator and persist (parts of) messages due to legal regulations for archiving. |
| **Related Patterns** | Data Store, Message Store [9], Claim Chek [9] |
| **Known Uses**     | “StoreInKiteDataset (Hadoop)” [2], “DBStorage”, “Persist”, “Variables” [22] |

### Table 4. Key Store

| Pattern Name       | Key Store                                           |
|--------------------|-----------------------------------------------------|
| **Intent**         | A store that contains private keys, and the certificates with their corresponding public keys. |
| **Driving Question** | How can keys and certificates be stored securely? |
| **Solution**       | Provide a secured storage for security relevant key material, which is accessible from other patterns, but protected from remote access. |
| **Data Aspects**   | certificates, persistent                           |
| **Example**        | Set up and provide access to a Java Key Store from the integration content. |
| **Related Patterns** | Trust Store                                        |
| **Known Uses**     | “JKS” [11], “Key Store” [1], probably [2], implicitly configurable in [22] |

### Table 5. Trust Store

| Pattern Name       | Trust Store                                         |
|--------------------|-----------------------------------------------------|
| **Intent**         | A store that contains certificates from other parties that you expect to communicate with, or from Certificate Authorities that you trust to identify other parties. |
| **Driving Question** | How can certificates from other communication partners be stored securely? |
| **Solution**       | Provide a secured storage for security relevant key material, which is accessible from other patterns. Probably limited access from respective parties is required to store their material. |
| **Data Aspects**   | certificates, persistent                           |
| **Related Patterns** | Key Store                                          |
| **Known Uses**     | Trust Store [1]                                     |
Table 6. Secure Store

| Pattern Name     | Secure Store                           |
|------------------|----------------------------------------|
| Intent           | A store that contains users and their passwords or secure tokens. |
| Driving Question | How can users and their passwords or secure tokens be stored securely? |
| Solution         | Secure and reliable storage for User/Password, Token, Expiring Token (cf. Sect. 2.3) |
| Data Aspects     | user/password, token, persistent       |
| Related Patterns | Key Store, Trust Store                 |
| Known Uses       | “Secure Store” [22]                    |

2.2 Messaging Patterns

We combine the routing and transformation pattern categories from [9] to general messaging patterns for the subsequent discussions.

Routing Patterns For realizing, combined Scatter-Gather [9] (e.g., for map-reduce like processing), the multicast is used Tab. 7.

Another new aspect is the communication between instances of several flows within one scenario or even between scenarios on the same platform (i.e., no external call). Therefore, a delegator can be used (cf. Tab. 8). While in the asynchronous case, this can be realized by a messaging system endpoint, e.g., JMS, many vendors offer special mechanisms to avoid an external call.

The iterative processing of tasks or explicit message re-deliveries requires a loop pattern (cf. Tab. 9).

The structural combination of several channels to one (cf. Tab. 10) – without merging messages – is complementing the multicast pattern.

Table 7. Multicast

| Pattern Name     | Multicast                               |
|------------------|-----------------------------------------|
| Intent           | Enable parallel message processing.     |
| Driving Question | How to (statically) enable parallel message processing? |
| Solution         | Send copies of a message to multiple receivers in parallel (statically) |
| Data Aspects     | message creating, read-only, channel cardinality 1:n, non-persistent |
| Variations       | Parallel, Sequential Multicast          |
| Related Patterns | Recipient List [9], Message Dispatcher, Splitter [9] |
| Known Uses       | “Multicast” [11], “Sequential/Parallel Multicast” [22] |

Multimedia / Transformation Patterns The message transformation patterns from [9] are mainly Content Enricher, Content Filter, and Claim Check. The Message Translator itself is sketched around the OSI reference model in [9]: data structures, data types, data representation and transport. Following these categories, we collected patterns as depicted in Fig. 2. The security patterns
Table 8. Delegate

| Pattern Name | Delegate |
|--------------|----------|
| **Intent**   | Exchange messages between several integration scenarios within the same integration system. |
| **Driving Question** | How to exchange messages between two or more integration scenarios locally? |
| **Solution** | Provide integration system local direct endpoints – preferably using an optimized message exchange format and protocol. |
| **Variations** | Asynchronous message exchange via queues, synchronous message exchange via integration system local direct endpoints. |
| **Example** | Messaging System endpoint (e.g., JMS, MQTT) for asynchronous messaging, or VM-local or platform local calls for synchronous messaging (potentially in an optimized format) |
| **Related Patterns** | Channel Adapter [9] |
| **Known Uses** | “Direct-VM” endpoint [11], partially covered by “Process Call” [22] |

Table 9. Loop

| Pattern Name | Loop |
|--------------|------|
| **Intent**   | Exchange messages between several integration scenarios within the same integration system. |
| **Driving Question** | How to exchange messages between two or more integration scenarios locally? |
| **Solution** | Provide integration system local direct endpoints – preferably using an optimized message exchange format and protocol. |
| **Variations** | Sequential Loop, Interleaved Loop, Enclosing Loop. |
| **Example** | Iterative tasks like splitting, re-deliveries |
| **Related Patterns** | - |
| **Known Uses** | “Loop Activity” [10], “Looping” [16] |

Table 10. Join Router

| Pattern Name | Join Router |
|--------------|-------------|
| **Intent**   | Combine several messages channels without merging the content. |
| **Driving Question** | How to combine several message channels without merging the content? |
| **Solution** | Provide a structural router, which receives messages of the same format from several incoming channels and forwards them to only one channel. |
| **Related Patterns** | opposite of multicast |
| **Known Uses** | “Direct-VM” [11], “Join” [22] |
enable message-level security and are further discussed in Sect. 2.3. The custom processors are theoretically applicable to all categories.

The classical Message Translator [9] on data structure level is called message mapping and translates standardized data formats between applications. Typical implementations can be found, e.g., as “Morphline Interceptor” in [1] and the “Mapping” component in [22].

In practice, basic converters on data type level (cf. Tab. 11) convert, e.g., between different stream types, reader and writer types, byte arrays and strings.

| Table 11. Type Converter |
|--------------------------|
| **Pattern Name** | Type Converter |
| **Intent** | Convert data types. |
| **Driving Question** | How to convert data types? |
| **Solution** | Provide special type converters for common types (domain specific). |
| **Example** | InputStream to OutputStream, Reader to Writer, ByteArray to String |
| **Related Patterns** | Custom Script |
| **Known Uses** | “Type Converter” [11], [2], “Morphline Interceptor” [1], implicit in [22] |

On the data representation level these converters are complemented by transfer encoders to represent binary data in an ASCII string format (cf. Tab. 12) and decoders (cf. Tab. 13), as well as message protocol marshaller (cf. Tab. 14) and unmarshaller (cf. Tab. 16).

One way to reduce the amount of exchanged data is to compress (cf. Tab. 16) and later decompress the content (cf. Tab. 17). The operations on binary, unstructured data is required, e.g., for resizing an image for interoperability or compression (cf. Tab. 18). Furthermore, the extraction of metadata from arbitrary non-structured documents is required for multimedia scenarios (cf. Tab. 19).

Some special constructs that were not picked up in general are a special sort pattern (cf. Tab. 20), a scanner (cf. Tab. 21) and an actual language translator (cf. Tab. 22).
### Table 12. Encoder

| Pattern Name | Encoder |
|--------------|---------|
| **Intent**   | Represent binary content textually. |
| **Driving Question** | How to represent binary content textually? |
| **Solution** | Provide standard and custom encoding capabilities for binary formats (domain-specific). |
| **Data Aspects** | Binary data encoding |
| **Example** | Base16, Base64 (ASCII, MIME content transfer encoding), Radix-64 (OpenPGP) |
| **Related Patterns** | Decoder |
| **Known Uses** | “Data Format” [11], “Morphline Interceptor” [1], “Base64EncodeContent” [2], “Encoder” [22] |

### Table 13. Decoder

| Pattern Name | Decoder |
|--------------|---------|
| **Intent**   | Recover encoded binary content. |
| **Driving Question** | How to recover encoded binary content? |
| **Solution** | Provide standard and custom decoding capabilities for binary data (domain-specific). |
| **Data Aspects** | Encoding |
| **Example** | Base16, Base64 (ASCII, MIME content transfer encoding), Radix-64 (OpenPGP) |
| **Related Patterns** | Encoder |
| **Known Uses** | “Data Format” [11], “Morphline Interceptor” [1], “Base64EncodeContent” [2], “Decoder” [22] |

### Table 14. Marshaller

| Pattern Name | Marshaller |
|--------------|------------|
| **Intent**   | Convert data formats. |
| **Driving Question** | How to convert one data format to another? |
| **Solution** | Provide standard and custom marshalling capabilities for data formats (domain-specific). |
| **Data Aspects** | Encoding |
| **Example** | JSON, XML, POJO, SQL |
| **Related Patterns** | Unmarshaller |
| **Known Uses** | “Data Format” [11], “Morphline Interceptor” [1], “ConvertJSONToSQL” [2], “JsonXMLConverter” [22] |
### Table 15. Unmarshaller

| Pattern Name | Unmarshaller |
|--------------|-------------|
| Intent       | Convert data formats. |
| Driving Question | How to convert one data format to another? |
| Solution     | Provide standard and custom unmarshalling capabilities for data formats (domain-specific). |
| Data Aspects | Encoding |
| Example      | JSON, XML, POJO, SQL |
| Related Patterns | Unmarshaller |
| Known Uses   | “Data Format” [11], “Morphline Interceptor” [1], “ConvertJSONToSQL” [2], “JsonXMLConverter” [22] |

### Table 16. Compress Content

| Pattern Name | Compress Content |
|--------------|------------------|
| Intent       | Compress message content. |
| Driving Question | How to compress the message content? |
| Solution     | Provide compression capabilities that compress (parts of) messages. |
| Data Aspects | Compression algorithm, type |
| Example      | GZIP |
| Related Patterns | Decompress Content, Message Endpoint, Adapter |
| Known Uses   | “Morphline Interceptor” [1], “Compress Content” [2], “Custom Script” [22] |

### Table 17. Decompress Content

| Pattern Name | Decompress Content |
|--------------|--------------------|
| Intent       | Decompress message content. |
| Driving Question | How to decompress message content? |
| Solution     | Provide decompression capabilities that decompress (parts of) messages. |
| Data Aspects | Decompression algorithm, type |
| Example      | GZIP |
| Related Patterns | Compress Content, Message Endpoint, Adapter |
| Known Uses   | “Morphline Interceptor” [1], “Compress Content” [2], “Custom Script” [22] |

### Table 18. Image Resizer

| Pattern Name | Image Resizer |
|--------------|---------------|
| Intent       | Resize Image. |
| Driving Question | How to resize an image? |
| Solution     | Scale image using image processing capabilities. |
| Data Aspects | binary, image data, size information |
| Example      | Resize image for interoperability with multimedia endpoint like Facebook or Twitter or compression. |
| Related Patterns | Compress Content |
| Known Uses   | “Image Resizer” [2] |
### Table 19. Metadata Extractor

| Pattern Name       | Metadata Extractor |
|--------------------|--------------------|
| Intent             | Extract multimedia meta data. |
| Driving Question   | How to extract multimedia metadata? |
| Solution           | Provide extraction capabilities that get metadata of multimedia data (e.g., image, video, document). |
| Data Aspects       | Metadata like geographic location, time |
| Example            | Geo tag |
| Related Patterns   | - |
| Known Uses         | “Read MIME activity” [10], “ExtractImageMetadata”, “ExtractMediaMetadata” [2] |

### Table 20. Content Sort

| Pattern Name       | Content Sort |
|--------------------|--------------|
| Intent             | Sort message content. |
| Driving Question   | How to sort the content of a message? |
| Solution           | Provide configurable sort capabilities that access the message content. |
| Data Aspects       | Algorithms, Comparators |
| Example            | User default comparator in the programming language to sort alpha-numeric content. |
| Related Patterns   | Custom Custom Script, Aggregator |
| Known Uses         | “Sort” [11], “Morphline Interceptor” [1], “Custom Script” [22] |

### Table 21. Find and Replace

| Pattern Name       | Find and Replace |
|--------------------|------------------|
| Intent             | Find and replace content. |
| Driving Question   | How to find and replace content? |
| Solution           | Scans the content of a message (for terms that are found in a user-supplied dictionary). If a term is matched, the UTF-8 encoded version of the term will be added to the message. |
| Data Aspects       | optional dictionary |
| Example            | Regex |
| Related Patterns   | Custom Custom Script |
| Known Uses         | “Scan Content” [2], “Morphline Interceptor” [1], “Processor” [11], “Custom Script” [22] |

### Table 22. Language Translator

| Pattern Name       | Language Translator |
|--------------------|---------------------|
| Intent             | Translate textual content from one language to another. |
| Driving Question   | How to translate textual content from one language to another? |
| Solution           | Provide standard interface access to natural language translation libraries or services. |
| Data Aspects       | Service bindings, locale |
| Example            | Yandex or Google translator API. |
| Related Patterns   | Custom Custom Script |
| Known Uses         | “Yandex Translator” [2], “Morphline Interceptor” [1], “Processor” [11], “Custom Script” [22] |
Besides the specialized translation patterns, language bindings for the execution of custom code are required for flexibility (cf. Tab. 23).

**Table 23. Custom Custom Script**

| Pattern Name       | Custom Custom Script                                                                 |
|--------------------|---------------------------------------------------------------------------------------|
| Intent             | Provide arbitrary body, header and attachment access and allow modifications.          |
| Driving Question   | How to execute custom code during message processing?                                  |
| Solution           | Embed freely programmable language support with access to the message channel and the messages. |
| Data Aspects       | Service bindings, compiled or plain source code (interpreted)                          |
| Variations         | controlled access by pre-defined, limiting expression language.                         |
| Example            | Language support for Groovy or Java script with access to the message and other services of the integration system e.g., Store Accessor. |
| Related Patterns   | Content Modifier, Content Enricher                                                     |
| Known Uses         | “Morphline Interceptor” [1], “Processor” [11], “Custom Script” [22]                   |
2.3 Security Patterns

The collection of security patterns is grounded on our system and literature study on security aspects in integration systems. Subsequently, the security patterns are listed and defined grouped by the security categories shown in Fig. 3.

Fig. 3. Security Categories grouped by classical CIA and extensions [26].

Confidentiality Patterns The confidentiality or message privacy patterns collected in Fig. 4 help to ensure that only authorized participants can access integration information like messages, channels, operations and storage. The message

Fig. 4. Confidentiality Patterns grouped by characteristics and actions as well as information states: processing, transmission and storage [14].

Tab. 24, the channel Tab. 25 and the storage Tab. 26 can have the characteristic of having encrypted information (e.g., encryption [13]).

The security-action patterns consist of pair-wise encrypting and decrypting operations (Tab. 27, Tab. 28), endpoints (Tab. 29, Tab. 30), adapters (Tab. 31, Tab. 32) and an encrypting store (Tab. 33).
| Pattern Name | Encrypted Message |
|--------------|-------------------|
| Intent       | Rely on a confidential and private piece of information. |
| Driving Question | How can messages be sent confidential and with data-privacy? |
| Context      | The confidentiality or data-privacy of a message is especially important, when the communication happens via a public network (e.g., cloud integration). |
| Solution     | Message level security; distinguish symmetric, asymmetric; text and categories (categories: channel, message cardinality, input/output, message generating, read/write access). |
| Data Aspects | text and categories (categories: channel, message cardinality, input/output, message generating, read/write access) |
| Result       | The message is asymmetrically encrypted. The message can only be read by applying, e.g., a Decryptor pattern |
| Example      | PGP, PCKS7 |
| Related Patterns | Message, Message Encryptor |
| Known Uses   | “PGP Message” [11], “Encrypted Content” [2], implicitly in [22] |

| Pattern Name | Encrypted Channel |
|--------------|-------------------|
| Intent       | Exchange message over a secure channel. |
| Driving Question | How can an application send a confidential message such that only the actual receiver can process it? |
| Solution     | Transport level security |
| Data Aspects | Encrypted transfer, certificates |
| Example      | SSL/TLS, HTTPS |
| Related Patterns | Encrypted Message |
| Known Uses   | [1], [2], [11], [22] |

| Pattern Name | Encrypted Store |
|--------------|-----------------|
| Intent       | Store messages confidential. |
| Driving Question | How to store messages confidential? |
| Solution     | Message level security; TODO: stores messages encrypted |
| Data Aspects | Encrypted content, header and attachments |
| Example      | PGP, PCKS7 |
| Related Patterns | Message Encryptor, Encrypted Message |
| Known Uses   | Configuration on “DBStorage” [22] |
| Table 27. Message Encryptor |
|-----------------------------|
| **Pattern Name** | Message Encryptor |
| **Intent** | When an application sends a confidential message to another participant, its content shall only be read by the receiver. |
| **Driving Question** | How can messages be sent confidential and with message-privacy as Encrypted Message? |
| **Solution** | Provide capabilities to encrypt the content, headers and/or attachments of a message. |
| **Data Aspects** | in: message, out: encrypted message, key store, public key, non-message generating, content modifying |
| **Example** | PGP, PCKS7 |
| **Related Patterns** | Encrypted Message, Message Decryptor |
| **Known Uses** | “Encrypt content” [2], “Encryptor” [22] |

| Table 28. Message Decryptor |
|-----------------------------|
| **Pattern Name** | Message Decryptor |
| **Intent** | When an application sends a confidential message to another participant, its content shall only be read by the receiver. |
| **Driving Question** | How can confidential messages Encrypted Message be processed by the actual receiver? |
| **Solution** | Provide capabilities to decrypt the content, headers and/or attachments of a message. |
| **Data Aspects** | in: encrypted message, out: decrypted message, key store, private key, non-message generating, content modifying |
| **Example** | PGP, PCKS7 |
| **Related Patterns** | Message Encryptor |
| **Known Uses** | “Decrypt content” [2], “Decryptor” [22] |

| Table 29. Encrypting Endpoint |
|-----------------------------|
| **Pattern Name** | Encrypting Endpoint |
| **Intent** | When an application sends a confidential message to another participant, its content shall only be read by the receiver. |
| **Driving Question** | How can messages be sent confidential and with message-privacy as Encrypted Message? |
| **Solution** | Provide capabilities to encrypt the content, headers and/or attachments of a message. |
| **Data Aspects** | key store, public key, non-message generating, content modifying |
| **Example** | PGP, PCKS7 |
| **Related Patterns** | Message Encryptor |
| **Known Uses** | [22] |
### Table 30. Decrypting Endpoint

| Pattern Name | Decrypting Endpoint |
|--------------|---------------------|
| Intent       | When an application sends a confidential message to another participant, its content shall only be read by the receiver. |
| Driving Question | How can confidential messages *Encrypted Message* be processed by the actual receiver? |
| Solution     | Provide capabilities to decrypt the content, headers and/or attachments of a message. |
| Data Aspects | key store, private key, non-message generating, content modifying |
| Example      | PGP, PCKS7 |
| Related Patterns | Message Decryptor |
| Known Uses   | [22] |

### Table 31. Encrypting Adapter

| Pattern Name | Encrypting Adapter |
|--------------|--------------------|
| Intent       | When an application sends a confidential message to another participant, its content shall only be read by the receiver. |
| Driving Question | How can messages be sent confidential and with message-privacy as *Encrypted Message*? |
| Solution     | Provide capabilities to encrypt the content, headers and/or attachments of a message. |
| Data Aspects | key store, public key, non-message generating, content modifying |
| Example      | PGP, PCKS7 |
| Related Patterns | Message Encryptor |
| Known Uses   | “FileChannel” [1], [22] |

### Table 32. Decrypting Adapter

| Pattern Name | Decrypting Adapter |
|--------------|--------------------|
| Intent       | When an application sends a confidential message to another participant, its content shall only be read by the receiver. |
| Driving Question | How can confidential messages *Encrypted Message* be processed by the actual receiver? |
| Solution     | Provide capabilities to decrypt the content, headers and/or attachments of a message. |
| Data Aspects | key store, private key, non-message generating, content modifying |
| Example      | PGP, PCKS7 |
| Related Patterns | Message Decryptor |
| Known Uses   | [22] |
Table 33. Encrypting Store

| Pattern Name     | Encrypting Store |
|------------------|------------------|
| Intent           | When a message is stored confidential. |
| Driving Question | How can messages be stored confidential and with message-privacy as Encrypted Message? |
| Solution         | Message level security |
| Data Aspects     | key store, public key, non-message generating, content modifying |
| Example          | PGP, PCKS7 |
| Related Patterns | Message Encryptor |
| Known Uses       | configurable during “Persist” [22] |

Integrity and Authenticity These patterns are collected in Fig. 5 and ensure the completeness, accuracy and absence of unauthorized modifications during message processing and verify the claim of identity. The characteristics of signed and verified messages (Tab. 34, Tab. 35) denote the message level and safe and authenticated channel (Tab. 36, Tab. 38) stand for transport level integrity and authenticity. On storage level, the pattern is called safe store (Tab. 37).

The corresponding action-patterns are the signer and verifier (Tab. 39, Tab. 40).

Availability The availability of resources like Message Store [9], Key Store or Data Store are crucial. Table 41 defines the countermeasure as a redundant store pattern.

Non-repudiation / Auditable / Accountability Being able to prove which event happenend during message processing when and with which privileges (role) and by whom (user) is important for detecting security issues as well as using the information for metering.
Table 34. Signed Message

| Pattern Name | Signed Message |
|--------------|----------------|
| Intent       | Ensuring a message’s authenticity, integrity and non-repudiation. |
| Driving Question | How can a message’s authenticity, integrity and non-repudiation be guaranteed? |
| Solution     | Sign a message. |
| Data Aspects | certificate-based |
| Related Patterns | Message Signer |
| Known Uses   | implicitly in [11], [22] |

Table 35. Verified Message

| Pattern Name | Verified Message |
|--------------|------------------|
| Intent       | Verifying a message’s authenticity, integrity and non-repudiation. |
| Driving Question | How can a message’s authenticity, integrity and non-repudiation be verified? |
| Solution     | Verify the signature of a message. |
| Data Aspects | certificate-based |
| Related Patterns | Signature Verifier |
| Known Uses   | implicitly in [11], [22] |

Table 36. Safe Channel

| Pattern Name | Safe Channel |
|--------------|--------------|
| Intent       | Ensure integrity on transport level. |
| Driving Question | How to ensure integrity on transport level. |
| Solution     | Provide integrity on transport level. |
| Data Aspects | certificate-based |
| Related Patterns | Message Signer, Authenticated Channel |
| Known Uses   | “File channel integrity tool” [1], implicitly in [22] |

Table 37. Safe Store

| Pattern Name | Safe Store |
|--------------|------------|
| Intent       | Ensure integrity on storage level. |
| Driving Question | How to ensure integrity on storage level. |
| Solution     | Provide integrity on transport level. |
| Data Aspects | certificate-based |
| Related Patterns | Message Signer, Safe Channel |
| Known Uses   | implicitly in [22] |

Table 38. Authenticated Channel

| Pattern Name | Authenticated Channel |
|--------------|-----------------------|
| Intent       | Ensure authenticity on channel level. |
| Driving Question | How to ensure authenticity on channel level. |
| Solution     | Ensure authenticity on transport level. |
| Data Aspects | certificate, user/password, token |
| Variations   | Basic, certificate-based |
| Example      | certificate-based, basic authentication |
| Related Patterns | Key Store, Trust Store, Secure Store |
| Known Uses   | Kerberos authentication [1], two-way SSL authentication (implicit) [2], [22] |
### Table 39. Message Signer

| Pattern Name       | Message Signer                        |
|--------------------|---------------------------------------|
| Intent             | Ensure the authenticity, integrity and non-repudiation of the message content. |
| Driving Question   | How can the authenticity of a message be ensured? |
| Context            | Signing messages is especially important, when the communication happens via a public network (e.g., cloud integration). |
| Solution           | The pattern ensures authenticity, integrity and non-repudiation on the message-level through signing the content or parts using security mechanisms like digital signatures or envelopes. The signer cannot deny signing afterwards and any change to the signed message parts can be detected using, e.g., the Verifier pattern. Message Translator patterns might invalidate the signing. |
| Data Aspects       | in: any message, out: message with signed content or parts; requires private key, key store |
| Result             | The signing is done by asymmetric cryptography that requires for example the creation of a signature using a signing algorithm with the private key stored in a secured key store. |
| Example            | PGP, PCKS7                             |
| Related Patterns   | Key Store, Trust Store                 |
| Known Uses         | “Camel Crypto” [11], “Message Signer” [22] |

### Table 40. Signature Verifier

| Pattern Name       | Signature Verifier                        |
|--------------------|------------------------------------------|
| Intent             | Verify a message’s authenticity, integrity and non-repudiation |
| Driving Question   | How to verify a message’s authenticity, integrity and non-repudiation? |
| Solution           | Provide certificate-based verification on message level. |
| Data Aspects       | in: Signed Message, out: Signed Message (read-only); exception: verification failed; requires access: certificate, Key Store, non-message generating, read-only |
| Variations         | Signature Verifier, XMLSignatureVerifier |
| Related Patterns   | Message Signer, Signed Message, Key Store, Trust Store |
| Known Uses         | “Verifier” [22] |

### Table 41. Redundant Store

| Pattern Name       | Redundant Store                        |
|--------------------|----------------------------------------|
| Intent             | Ensure availability of storage resources. |
| Driving Question   | How to ensure availability of storage resources? |
| Solution           | Provide redundant hardware, service (probably even high-available and disaster recoverable) |
| Data Aspects       | availability level: HA, DR          |
| Variations         | Failover, HA, DR                      |
| Related Patterns   | Message Store, Data Store, Key Store, Trust Store, Secure Store |
| Known Uses         | “MorphlineSolrSinks”, “Kafka Channel” [1], implicitly in [22]; Tandem Store [7,3] |
Table 42. Audit Log

| Pattern Name | Audit Log |
|--------------|-----------|
| Intent       | Keep a record of those events you consider relevant for (security) auditing: accountability, non-repudiation, auditability |
| Driving Question | How to keep a record of those events considered relevant for (security) auditing. |
| Solution     | Provide a secure logging capability, which tracks events and (especially) all its own configurations. |
| Data Aspects | Log entries |
| Example      | Record event-based endpoints, unsuccessful message send events, changes to the audit log. |
| Related Patterns | Monitor |
| Known Uses  | implicitly in [22] |

Table 43. Token

| Pattern Name | Token |
|--------------|-------|
| Intent       | Grant role-based access to a user. |
| Driving Question | How to grant role-based access to a user? |
| Solution     | Provide secure token, which is passed as part of each during conversation. |
| Example      | OAuth |
| Related Patterns | Secure Store |
| Known Uses  | “GetTwitter” [2], “Secure Paramters” in SAP HCI ADK [22] |

Table 44. Expiring Token

| Pattern Name | Expiring Token |
|--------------|----------------|
| Intent       | Improve security for role-based access of users via Tokens. |
| Driving Question | How to improve security, when granting role-based access to a user via Tokens? |
| Solution     | Reduce the validity of a Token through expiration. |
| Data Aspects | Token |
| Example      | OAuth |
| Related Patterns | Secure Store |
| Known Uses  | “GetTwitter” [2], “Secure Paramters” in SAP HCI ADK [22] |

Authorization, Non-Delegation
### Table 45. Refresh Token

| Pattern Name | Refresh Token |
|--------------|---------------|
| Intent       | Deal with expiring tokens. |
| Driving Question | How to deal with expiring tokens? |
| Solution     | Allow to (automatically) re-negotiate expiring tokens? |
| Data Aspects | Expiring Token |
| Example      | OAuth |
| Related Patterns | Secure Store |
| Known Uses  | “GetTwitter” [2], “Secure Parameters” in SAP HCI ADK [22] |

### Table 46. Token-based Authorizer

| Pattern Name | Token-based Authorizer |
|--------------|------------------------|
| Intent       | Validate token-based authorizations. |
| Driving Question | How to validate token-based authorizations? |
| Solution     | Provide authorization checks for messages with tokens. |
| Data Aspects | Secure Tokens |
| Example      | OAuth |
| Related Patterns | Secure Store, Adapter, Message Endpoint |
| Known Uses  | Pluggable Authorization [2], SAP Social Media ADK adapters [22] |

### Table 47. Principle Propagation

| Pattern Name | Principle Propagation |
|--------------|------------------------|
| Intent       |                         |
| Driving Question | How to delegate authorizations to a third-party? |
| Solution     | Allow authorization forwarding. |
| Data Aspects | Token |
| Related Patterns | Token |
| Known Uses  | “Principle Propagation” [23] |
2.4 Exception Patterns

The exception patterns were introduced and analyzed in [21], thus summarized in Fig. 6 and not further discussed:

- Failover Router
- Compensation Sphere
- Exception Sphere
- Validate Message
- Timeout Operation
- Message Redelivery on Exception
- Delayed Redelivery
- Skip Operation
- Stop Operation Local
- Stop Operation All
- Pause Operation
- Throw
- Raise Incident (similar to monitoring pattern Sect. 2.5)
- Catch Selective
- Catch All

Fig. 6. Exception Handling Patterns.
2.5 Monitoring and Operation Patterns

The current EIP cover basic monitoring and operation patterns [9]: *Control Bus* for administrating the messaging system, *Wire Tab* for routing message copy e.g., to a *Message Store* for monitoring, *Message History* for provenance, and *Smart Proxies* for asynchronous message tracking.

In addition, there are more aspects to monitoring and operations that can be found in current integration system implementations. Some of them are collected subsequently (others like the circuit breaker Sect. 2.6 can be found in related sections). Figure 7 gives an overview of the collected monitoring patterns.

The processing of messages might require their cancellation (cf. Tab. 48), e.g., based on defined usage statistics (cf. Tab. 49)).

**Fig. 7. Monitoring Patterns.**

| Pattern Name         | Message Cancellation |
|----------------------|----------------------|
| Intent               | Cancel the processing of a message. |
| Driving Question     | How to cancel the processing of a message? |
| Solution             | Define a condition under which the message will not be processed further. |
| Data Aspects         | condition |
| Related Patterns     | Message Expiration [9], Stop Operation (Local,All) [21], Validate Message [21] |
| Known Uses           | [22] |

**Table 49. Usage Statistics**

| Pattern Name         | Usage Statistics |
|----------------------|------------------|
| Intent               | A measure for the usage of components. |
| Driving Question     | How to measure the usage of components during message processing? |
| Solution             | Define performance indicators for components. |
| Data Aspects         | usage indicators, persistent |
| Related Patterns     | Scenario Content Statistics |
| Known Uses           | [22] |
While the whole message processing and failure handling shall be automatic, there might be situations (e.g., due to business semantics), which require raising an indication (cf. Tab. 50) for manual post-processing or information. Sometimes this can be preceded by the detection of a special situation (cf. Tab. 51). The information can be collected by interceptors (cf. Tab. 52) and provided asynchronously by a sanity queue (cf. Tab. 53).

### Table 50. Raise Indicator

| Pattern Name   | Raise Indicator |
|----------------|-----------------|
| Intent         | Inform about an important event during message processing. |
| Driving Question | How to inform about an important event during message processing? |
| Solution       | Raise an indicator. |
| Data Aspects   | persistent |
| Example        | Send e-mail, Monitor |
| Related Patterns | Detect |
| Known Uses     | “Indicator” [2], “Alert” [22] |

### Table 51. Detect

| Pattern Name | Detect |
|--------------|--------|
| Intent       | Detect inactive, active components like channel or pattern/activity. |
| Driving Question | How to detect inactive components like channels or patterns? |
| Solution     | Send out an alert when a component did not have any data for a specified amount of time. |
| Data Aspects | conditions |
| Related Patterns | Usage statistics, Channel Monitor, Component Monitor |
| Known Uses   | “Monitor Activity” [2] |

### Table 52. Message Interceptor

| Pattern Name       | Message Interceptor |
|--------------------|---------------------|
| Intent             | Gather usage statistics on channel level. |
| Driving Question   | How to gather usage statistics on channel level? |
| Solution           | Inject a configurable “listener” between components. |
| Data Aspects       | conditions, read-only, non-message creating |
| Related Patterns   | Usage statistics, Wire Tap [9], Intercepting Validator [7,24] |
| Known Uses         | “Interceptor” [11], “Interceptor” [1] |

The respective events and incidents can be tracked by monitors, e.g., for components (cf. Tab. 54), channels (cf. Tab. 55) and messages (cf. Tab. 56). In addition, the system resource information (cf. Tab. 57) as well as the overview
Table 53. Sanity Queue

| Pattern Name     | Sanity Queue |
|------------------|-------------|
| Intent           | Stay informed about a system’s sanity. |
| Driving Question | How to stay informed about the system’s sanity? |
| Solution         | Create sanity queues for important aspects and register on events. |
| Data Aspects     | queues, persistent |
| Related Patterns | Wire Tap |
| Known Uses       | JMS sanity check |

of all open circuits (cf. Tab. 64) might be of interest. The monitoring of hybrid applications and integration scenrios is covered by Tab. 58.

Table 54. Component Monitor

| Pattern Name     | Component Monitor |
|------------------|-------------------|
| Intent           | Measure usage statistics and behavior of a component. |
| Driving Question | How to measure usage statistics and behavior of a component? |
| Solution         | Monitor configurable characteristics of a component and store them persistently as statistics. |
| Data Aspects     | Configurations, statistical records, KPIs, persistent |
| Example          | Monitor throughput, exceptions and raise indicator or load balance, if KPIs are not fulfilled. |
| Related Patterns | Stop Operation, Pause Operation (both from [21]), Failover Router [21], Data Store, Smart Proxy [9] |
| Known Uses       | “Adapter Monitor” [17], conceptually supported by [19] |

For analytical reasons, it might be important to have statistics on the used components and adapters (cf. Tab. 59).

The operational aspects mostly come from reliability and distributed system requirements as shown in Fig. 8.

Upon them are the persistent scheduling (cf. Tab. 60) and the persistent cluster lock (cf. Tab. 60).
| **Table 55. Channel Monitor** |
|-------------------------------|
| **Pattern Name** | Channel Monitor |
| **Intent** | Measure communication with endpoints. |
| **Driving Question** | How to measure success, failures (exceptions thrown by endpoint), timeouts for requests? |
| **Solution** | Monitor configurable characteristics of the communication with endpoints and store them persistently as statistics. |
| **Data Aspects** | Configurations, statistical records, KPIs, persistent |
| **Example** | Monitor timeout to endpoints and decide to apply a Circuit Breaker pattern. |
| **Related Patterns** | Circuit Breaker, Timeout Synchronous Request, Failover Router [21], Command, Data Store, Smart Proxy [9] |
| **Known Uses** | “Adapter Monitor” [17], conceptually supported by [19] |

| **Table 56. Message Monitor** |
|-------------------------------|
| **Pattern Name** | Message Monitor |
| **Intent** | Monitor the states of all processed messages. |
| **Driving Question** | How to monitor the states of all currently processed messages? |
| **Solution** | Provide a status monitor that shows the Message History. |
| **Data Aspects** | message provenance data |
| **Related Patterns** | Message History |
| **Known Uses** | “Message Processing Log” [22] |

| **Table 57. Resource Monitor** |
|-------------------------------|
| **Pattern Name** | Resource Monitor |
| **Intent** | Monitor the system resources. |
| **Driving Question** | How to monitor the system resources and react on critical situations? |
| **Solution** | Provide a status monitor that shows the system’s resources and allow the definition of thresholds for raising indicators. |
| **Data Aspects** | resource statistics |
| **Example** | Memory consumption, CPU and network load |
| **Related Patterns** | Raise indicator |
| **Known Uses** | “MonitorDiskUsage”, “MonitorMemory” [2] |

| **Table 58. Hybrid Monitor** |
|-------------------------------|
| **Pattern Name** | Hybrid Monitor |
| **Intent** | Monitor across different integration platforms. |
| **Driving Question** | How to monitor integration scenarios across different integration platforms? |
| **Solution** | Provide a monitor that combines (data ingestion) different, vendor-specific monitoring data, normalizes it (i.e., format, timings), calculates and visualizes combined metrics. |
| **Data Aspects** | heterogeneous monitoring data |
| **Example** | Cross-Cloud, On-Demand to Cloud |
| **Related Patterns** | - |
| **Known Uses** | “sap-hci” |
### Table 59. Scenario Content Statistics

| Pattern Name       | Scenario Content Statistics                                                                 |
|--------------------|---------------------------------------------------------------------------------------------|
| Intent             | A measure for the usage of content.                                                         |
| Driving Question   | How to measure the usage of content within integration scenarios?                           |
| Solution           | Define performance indicators for the usage of content.                                     |
| Data Aspects       | content statistics                                                                         |
| Example            | Number of used adapters and their configuration                                             |
| Related Patterns   | Usage statistics                                                                           |
| Known Uses         | [22]                                                                                       |

### Table 60. (Persistent) Scheduler

| Pattern Name       | (Persistent) Scheduler                                                                     |
|--------------------|---------------------------------------------------------------------------------------------|
| Intent             | Start from last message or action after restart.                                            |
| Driving Question   | How to start scheduling from the last message or action after system restart?              |
| Solution           | Persistently store the state of the scheduler.                                              |
| Data Aspects       | persistent                                                                                  |
| Related Patterns   | Data Store, Polling Consumer                                                                |
| Known Uses         | [11], [22], conceptually supported in [4]                                                   |

### Table 61. (Persistent) Cluster Lock

| Pattern Name       | (Persistent) Cluster Lock                                                                  |
|--------------------|---------------------------------------------------------------------------------------------|
| Intent             | Prevent several integration scenarios from processing the same information concurrently.   |
| Driving Question   | How to prevent several integration scenarios from processing the same information concurrently? |
| Solution           | Set persistent lock.                                                                        |
| Data Aspects       | persistent                                                                                  |
| Related Patterns   | Data Store, Competing Consumers [9]                                                           |
| Known Uses         | [22]                                                                                       |
2.6 Endpoint and Adapter Patterns

Many of the channel access and endpoint patterns for connecting applications with messaging systems (i.e., Message Channel, Channel Adapter, Message Endpoint, Durable Subscriber, Channel Purger, Transactional Client, Selective Consumer, Polling Consumer, Message Dispatcher, Event-Driven Consumer), and for involving existing systems in the message exchange (i.e., Envelope Wrapper, Messaging Bridge) are covered by [9].

In addition, [20] collected some more integration adapter and quality of service (QoS) related patterns, shown in Fig. 9, which are not further discussed here:

- Commutative Endpoint
- Timed Redelivery
- Adapter Flow
- Synch/Asynch Bridge
- Asynch/Synch Bridge
- Protocol Switch
- Cross scenario Processing
- Cross tenant Processing
- Best Effort Processing
- At least once Processing
- At most once Processing
- Exactly once Processing
- Exactly once in order Processing

The two patterns not discussed in [20] are the Commutative Receiver, and the Timed Redelivery. These patterns are considered endpoint or application
patterns and move integration logic to the applications. Hence they complement the existing the Idempotent Endpoint pattern [9] to ensure in-order and at-least-once message delivery semantics outside of the integration system.

Table 62. Commutative Receiver

| Pattern Name       | Commutative Receiver |
|--------------------|-----------------------|
| Intent             | Enable the endpoint/application to deal with out-of-order messages. |
| Driving Question   | How to ensure in-order processing without intermediate storage, e. g., in form of a resequencer [9]? |
| Context            | Out-of-order communication with endpoints/applications |
| Solution           | Guarantee that endpoint/application handle arriving out-of-order messages will be stored and then processed in the correct order. |
| Result             | Handles out of order messages and applies them in-order. |
| Related Patterns   | Resequencer [9] |
| Known Uses         | [15] |

Recent practical advances for cloud-based Microservice architectures [18], e. g., in the area of fault-tolerance like Hystrix [17], can be mapped to integration systems and summarized to general patterns as shown in Fig. 10.

Fig. 10. Integration Adapter and Endpoint Request Patterns.

Thereby the related aspects of fault-tolerance and smart requests are differentiated. For fault-tolerance, one wants to make request more tangible (cf. Tab. 65), to penalize long running requests (cf. Tab. 64), break them by time-
Table 63. Timed Redelivery

| Pattern Name       | Timed Redelivery |
|--------------------|------------------|
| Intent             | Enable the endpoint/application to deliver a message asynchronously using redelivery instead of intermediate storage as guarantee. |
| Driving Question   | How to ensure that a message will be received without intermediate storage, e.g., in form of Redelivery on Exception [21] |
| Context            | Asynchronous communication with message delivery guarantees. |
| Solution           | Instead of relying on intermediate storage and retry within the integration system, the application sends multiple instances of the same message with configurable timings until the actual receiver endpoint acknowledgements (e.g., *Quick Acknowledgement* [8]) reach the sender. Requires an Idempotent [9] and Commutative Receiver for certain message delivery semantics [20] |
| Data Aspects       | Timings, acknowledgements |
| Result             | Send copies of the same message asynchronously until the receiver’s acknowledgement is reached by the sender. |
| Related Patterns   | Retry [8], Redelivery on Exception [21]. |
| Known Uses         | – |

outs (cf. Tab. 66) and find alternative endpoints that can answer the requests instead (cf. Tab. 67).

Table 64. Circuit Breaker

| Pattern Name       | Circuit Breaker |
|--------------------|-----------------|
| Intent             | Stop all requests to a particular endpoint/application for a period of time |
| Driving Question   | How to stop all requests to a particular endpoint/application for a period of time? |
| Context            | Communication with endpoints/applications |
| Solution           | Provide capabilities to monitor the error percentage, when communicating with a particular endpoint and automatically or manually prevent its invocation for a period of time. |
| Data Aspects       | Timing, monitoring statistics |
| Result             | Reduces number of failed message exchanges. |
| Related Patterns   | Electronic circuits |
| Known Uses         | “Circuit Breaker” [17], conceptually from [19] |

The smart handling of requests helps to stabilize the system as well. However, the main focus lies on an intelligent way of requesting information from endpoints, e.g., by combining several (cross-scenario) requests to one (cf. Tab. 68), or leveraging already recently requested information for multiple subsequent requests (cf. Tab. 69) or separating request aspects to multiple requesters (cf. Tab. 70).
Table 65. Command

| Pattern Name | Command |
|--------------|---------|
| Intent       | Make remote calls tangible. |
| Driving Question | How to make remote calls tangible? |
| Solution     | Wrap remote calls into one command context. |
| Result       | The command context allows to, e.g., timeout and monitor a request. |
| Related Patterns | Similar to “Adapter Flow” in [20], Request Collapsing |
| Known Uses   | “Reactive Command” [17] |

Table 66. Timeout Synchronous Request

| Pattern Name | Timeout Synchronous Request |
|--------------|-----------------------------|
| Intent       | Timeout remote calls that take longer than a configured threshold. |
| Driving Question | How to timeout remote calls that take longer than a configured threshold? |
| Context      | Long blocking requests destabilize the scenario and potentially the integration system. |
| Solution     | Apply a timeout to each synchronous request. |
| Data Aspects | Timing |
| Result       | Unblocks the scenario and system. |
| Related Patterns | Command, “Delayed Redelivery”, “Stop Operation” both from [21], Message Expiration [9] |
| Known Uses   | “Timeout” [17], conceptually from [19] |

Table 67. Failover Request Handler

| Pattern Name | Failover Request Handler |
|--------------|--------------------------|
| Intent       | Constructively handling failed requests. |
| Driving Question | How to handle failed or timed-out requests or short-circuits constructively? |
| Solution     | Perform fallback logic when a request fails, is rejected, times-out or short-circuits. |
| Data Aspects | Configuration of failover channels or endpoints/adapters |
| Variations   | Applicable to endpoints and adapters. |
| Example      | If there is a communication error with the currently selected endpoint, then the request handler automatically fails-over to the next endpoint in the list. |
| Related Patterns | Command, Failover Router [21] |
| Known Uses   | “Failover Client” [1] |

Table 68. Request Collapsing

| Pattern Name | Request Collapsing |
|--------------|--------------------|
| Intent       | Reduce the number of endpoint calls. |
| Driving Question | How to reduce the number of endpoint calls? |
| Solution     | Collapse multiple requests into a single endpoint dependency call. |
| Data Aspects | Data aggregation, selection and projection handling |
| Variations   | Applicable to endpoints and adapters. |
| Related Patterns | Command, similar to “Adapter Flow” in [20] |
| Known Uses   | “Request Collapsing” [17], conceptually supported by [25] for database access from business process engines. |
Table 69. Request Caching

| Pattern Name       | Request Caching                                     |
|--------------------|-----------------------------------------------------|
| Intent             | Reduce the amount of duplicate requests to the same endpoint. |
| Driving Question   | How to reduce the amount of duplicate requests (from several scenarios) to the same endpoint? |
| Context            | Several integration scenarios might request the same information from the same endpoint (concurrently). |
| Solution           | Cache request contexts/keys to connect only once and feed many integration scenarios with the required information. |
| Data Aspects       | Data aggregation, selection and projection handling, (persistent) caching, request correlation |
| Variations         | Applicable to endpoints and adapters.               |
| Related Patterns   | Command, Data Store                                  |
| Known Uses         | similar to “Request Caching” in [17], conceptually supported by [25] for database access from business process engines. |

Table 70. Request Partitioning

| Pattern Name       | Request Partitioning                                 |
|--------------------|------------------------------------------------------|
| Intent             | Isolate request dependencies and limit concurrent access to them. |
| Driving Question   | How to isolate requests and limit concurrent access to any of them? |
| Solution           | Manually or automatically isolate request dependencies. |
| Data Aspects       | Split queries (where possible) and merge results      |
| Result             | Partitions request failures as well for a more robust system: by partitioning requests to endpoints, errors are confined to one request aspect (e.g., get business partner header data) as opposed to cancelling the entire request (e.g., business partner header and associated documents). |
| Variations         | The partitions can be hardware redundancy, binding certain processes to certain CPUs, segmenting different areas of business functionality to different server farms, or partitioning requests into different information aspect groups for different functionality. |
| Related Patterns   | Bulkhead, Command, Request Collapsing                |
| Known Uses         | “Isolation” in [17], conceptually similar to “Bulkhead” from [19] |
2.7 Composition

The composition of patterns to integration processes requires hierarchical partitioning for reuse and scoping (cf. Tab. 71), as well as the creation of process variants (cf. Tab. 72).

**Table 71. Integration Subprocess**

| Pattern Name             | Integration Subprocess                                                                 |
|--------------------------|----------------------------------------------------------------------------------------|
| Intent                   | Reuse, scope integration process.                                                      |
| Driving Question         | How to reuse, scope an integration process?                                             |
| Solution                 | Hierarchy of processes. A sub-process is attached to one parent process (i.e., integration process, sub-process) |
| Data Aspects             | Process context                                                                         |
| Result                   | Integration processes can be partitioned into smaller processes for re-use or different scoping. |
| Variations               | Shared sub-process that allows for reusing an Integration Subprocess by other integration processes. |
| Related Patterns         | Exception Subprocess                                                                    |
| Known Uses               | e.g., [10,16,22]                                                                        |

**Table 72. Integration Process Template**

| Pattern Name             | Integration Process Template                                                            |
|--------------------------|----------------------------------------------------------------------------------------|
| Intent                   | Create variants of integration processes.                                              |
| Driving Question         | How to create variants of an integration process?                                       |
| Solution                 | Create structural representation of an integration process and externalize all configurations. The structure and configurations can be changed to create a new variant or instance of the process. |
| Data Aspects             | Process structure, configuration parameters                                            |
| Result                   | The structural aspects are separated from the instance configuration.                   |
| Related Patterns         | -                                                                                      |
| Known Uses               | Template Integration Process [10], Snapshot [12], Blueprint [5]                         |
3 SAP HANA Cloud Integration eDocument

The SAP HCI eDocument solution allows to create, process and manage electronic documents for country-specific requirements. Figure 11 shows an eDocument scenario for corporates sending their factura invoice to Spanish eDocument authorities FACe (*Factura-e*)\(^3\) using SAP HCI. Hereby, the electronic invoice is a statement with the same legal effect as a paper invoice, however, transmitted electronically. The corporates have to send the invoice whose format is specified by the Spanish authorities to the FACe by 15th of January each year. The corporates send a SOAP message, containing the factura to HCI, where the factura is extracted, signed (i.e., SHA1/RSA), and encoded according to the regulations (i.e., Base64), e.g., within a BPMN SubProcess. The message contains routing information about the manual or automatic delivery to the FACe. In case of “manually” delivery an indication is raised in form of an alert into a monitor for manual post-processing. Otherwise, the message is converted into the required SOAP request format and sent to the FACe. Thereafter, a response to

\(^3\) Spain *Factura-e*: [http://www.facturae.gob.es/](http://www.facturae.gob.es/)
the corporate is prepared and sent. In case of an exceptional situation during processing (i.e., throw exception [21]), all exceptions are caught (i.e., catch-all [21]) and processed in an exception flow (i.e., exception processing [21]). The signing, indication raising, and exception handling are not in EIP.

Figure 12 shows the synthesis of the eDocuments scenario with the new patterns partially as imperative BPMN syntax combined with a declarative, icon notation. The icons are BPMN-compliant and can be realized as configuration tables as part of the used BPMN elements.

Used Annotations: 🔄 raise indicator, 🔒 authenticated, 🗝 certificate-based, 🗥 encrypted, 🗥 decrypted, 📄 encoded, 📄 redundant, 📄 signed, 🔒 verified.

Fig. 12. CIP Application (BPMN + selected CIP icons + annotations)
References

1. Apache Foundation. Apache Flume. https://flume.apache.org/, 2015.
2. Apache Foundation. Apache Nifi. https://nifi.apache.org/, 2015.
3. B. Blakley and C. Heath. Security design patterns technical guideversion 1. Technical report, Open Group(OG), 2004.
4. A. Böhm. Building Scalable, Distributed Applications with Declarative Messaging. PhD thesis, University of Mannheim, 2010.
5. Cloudpipes. Cloudpipes Documentation. https://docs.cloudpipes.com/, 2017.
6. C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications. Springer, 2014.
7. M. Hafiz, P. Adamczyk, and R. E. Johnson. Growing a pattern language (for security). In ACM SPLASH, Onward! 2012, 2012.
8. G. Hohpe. 06291 workshop report: Conversation patterns. In The Role of Business Processes in Service Oriented Architectures, 16.07. - 21.07.2006, 2006.
9. G. Hohpe and B. Woolf. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
10. IBM. WebSphere Cast Iron Cloud integration. https://www.ibm.com/support/knowledgcenter/SSGR73, 2017.
11. C. Ibsen and J. Anstey. Camel in Action. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2010.
12. Jitterbit. Harmony Cloud Integration. https://www.jitterbit.com/harmony/, 2017.
13. M. Leitner, S. Schefer-Wenzl, S. Rinderle-Ma, and M. Strembeck. An experimental study on the design and modeling of security concepts in business processes. In The Practice of Enterprise Modeling - 6th IFIP WG 8.1 Working Conference, PoEM 2013, Riga, Latvia, November 6-7, 2013, Proceedings, pages 236–250, 2013.
14. J. McCumber. Information systems security: A comprehensive model. In Proceedings of the 14th National Computer Security Conference, 1991.
15. Microsoft. Designing service applications. https://msdn.microsoft.com/en-us/library/ee658114.aspx, 2016.
16. Microsoft. BizTalk Server. https://msdn.microsoft.com/en-us/library/dd547397(v=bts.10).aspx, 2017.
17. Netflix. Hystrix. https://github.com/Netflix/Hystrix/wiki, 2015.
18. S. Newman. Building Microservices: Designing Fine-Grained Systems. O’Reilly, Beijing, 2015.
19. M. T. Nygard. Release It! Design and Deploy Production-Ready Software. Pragmatic Bookshelf, Raleigh, NC, 2007.
20. D. Ritter and M. Holzeitner. Integration adapter modeling. In Advanced Information Systems Engineering, pages 468–482. Springer, 2015.
21. D. Ritter and J. Sosulski. Modeling exception flows in integration systems. In 18th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2014, Ulm, Germany, September 1-5, 2014, pages 12–21, 2014.
22. SAP SE. SAP HANA Cloud Integration. https://cloudintegration.hana.ondemand.com, 2015.
23. SAP SE. SAP Process Integration. http://help.sap.com/nwpi, 2015.
24. C. Steel, R. Nagappan, and R. Lai. Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management. Core Series. Prentice Hall PTR, 2005.
25. M. Vrhovnik. *Optimierung datenintensiver Workflows: Konzepte und Realisierung eines heuristischen, regelbasierten Optimierers*. PhD thesis, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2011.

26. M. E. Whitman and H. J. Mattord. *Principles of Information Security*. Course Technology Press, Boston, MA, United States, 3rd edition, 2007.
List of Patterns

Adapter Flow, 26
Async/Synch Bridge, 26
At least once Processing, 26
At most once Processing, 26
Audit Log, 18
Authenticated Channel, 16

Best Effort Processing, 26

Catch All, 20
Catch Selective, 20
Channel Monitor, 24
Circuit Breaker, 28
Cluster Lock, 25
Command, 29
Commutative Endpoint, 26
Commutative Receiver, 27
Compensation Sphere, 20
Component Monitor, 23
Compress Content, 8
Content Sort, 9
Cross scenario Processing, 26
Cross tenant Processing, 26
Custom Custom Script, 10

Data Store, 2
Decoder, 7
Decompress Content, 8
Decryption Adapter, 14
Decryption Endpoint, 14
Delayed Redelivery, 20
Delegate, 5
Detect, 22

Encoder, 7
Encrypted Channel, 12
Encrypted Message, 12
Encrypted Store, 12
Encrypting Adapter, 14
Encrypting Endpoint, 13
Encrypting Store, 15
Exactly once in order Processing, 26
Exactly once Processing, 26
Exception Sphere, 20
Expiring Token, 18

Failover Request Handler, 29
Failover Router, 20
Find and Replace, 9

Hybrid Monitor, 24
Image Resizer, 8
Integration Process Template, 31
Integration Subprocess, 31

Join Router, 5
Key Store, 3
Language Translator, 9
Loop, 5

Marshaller, 7
Message Cancellation, 21
Message Decryptor, 13
Message Encryptor, 13
Message Interceptor, 22
Message Monitor, 24
Message Redelivery on Exception, 20
Message Signer, 17
Metadata Extractor, 9
Multicast, 4

Pattern Format Template, 38
Pause Operation, 20
Principle Propagation, 19
Protocol Switch, 26

Raise Incident, 20, 22
Raise Indicator, 22
Redundant Store, 17
Refresh Token, 19
Request Caching, 30
Request Collapsing, 29
Request Partitioning, 30
Resource Monitor, 24

Safe Channel, 16
Safe Store, 16
Sanity Queue, 23
Scenario Content Statistics, 25
Scheduler, 25
Secure Store, 4
Signature Verifier, 17
Signed Message, 16
Skip Operation, 20
Stop Operation All, 20
Stop Operation Local, 20
Store Accessor, 3
Synch/Asynch Bridge, 26
Throw, 20
Timed Redelivery, 26, 28
Timeout Operation, 20
Timeout Synchronous Request, 29

Token, 18
Token-based Authorizer, 19
Transient Store, 3
Trust Store, 3
Type Converter, 6
Type Converter, 6
Unmarshaller, 8
Usage Statistics, 21
Validate Message, 20
Verified Message, 16
A Pattern Format

In this section the pattern format used in this paper is described. The format, shown in Tab. 73, is defined similar to existing pattern formats of the EIP [9] and related pattern descriptions (e.g., from cloud computing [6]).

**Table 73. Pattern Format Template**

| Pattern Name | Name |
|--------------|------|
| Intent       | at the beginning of each pattern, its purpose and goal is shortly stated, to describe what the solution represented by the pattern contains. |
| Driving Question | This question captures the problem that is answered by the pattern. Stating this question at the beginning allows readers to identify if the pattern fits the problem they have in a concrete use case. |
| Context (optional) | This section describes the environment and forces leading to the problem solved by the pattern. It also may describe why naive solutions can be unsuccessful or suboptimal. Other patterns may be referenced here. |
| Solution     | The solution section briefly states how the pattern solves the problem raised by the driving question. It is kept brief, because readers shall be enabled to quickly read the intent, question, and solution sections to get an idea what the pattern is doing in detail. The solution section is commonly closed with a sketch depicting the architecture of the solution. |
| Data Aspects (optional) | All message, pattern content and configuration aspects are described in this section. |
| Result (optional) | In this section, the solution is elaborated in greater detail. The architecture proposed by the sketch is described and the behavior of the application after implementation of the pattern is discussed. New challenges that may arise after a pattern has been applied may also be included here, together with references to other patterns addressing these new challenges. |
| Variations (optional) | Often, patterns can be applied in slightly different forms. If the differences of these variations are not significant enough to justify their description in a separate pattern, they are covered in the variations section. |
| Example (optional) | This section gives an illustrative example of the pattern. |
| Related Patterns | Several patterns are often applied together as they are solving related problems, but the application of one pattern may also exclude other patterns from being applicable. These interrelations of patterns are described in this section. It, therefore, forms the structure of the integration pattern language and guides readers through the set of patterns. |
| Known Uses | Existing applications implementing a pattern, products offering a pattern or supporting its implementation, are covered here exemplarily. |