Integration Adapter Modeling

Daniel Ritter(✉) and Manuel Holzleitner

Technology Development, SAP SE, Dietmar-Hopp-Allee 16,
Walldorf 69190, Germany
{daniel.ritter,manuel.holzeitner}@sap.com

Abstract. *Integration Adapters* are a fundamental part of an integration system, since they provide (business) applications access to its messaging channel. However, their modeling and configuration remain under-represented. In previous work, the integration control and data flow syntax and semantics have been expressed in the Business Process Model and Notation (BPMN) as a semantic model for message-based integration, while adapter and the related quality of service modeling were left for further studies.

In this work we specify common adapter capabilities and derive general modeling patterns, for which we define a compliant representation in BPMN. The patterns extend previous work by the adapter flow, evaluated syntactically and semantically for common adapter characteristics.

Keywords: Business Process Model and Notation (BPMN) · Conceptual modeling · Language design · Message endpoints · Quality of service

1 Introduction

Enterprise Application Integration (EAI) continues to receive widespread focus by organizations offering them as means of integrating their conventional business applications with each other, with the growing amount of cloud applications and with their partners' systems. In many cases, the integration middleware systems serve as the enabling technology for distributed, mission-critical business processes. For that, these systems offer well-defined modeling capabilities to describe integration semantics (e. g., message creation, transformation, routing) as well as runtime systems that interpret the definitions for efficient message processing. Figure 1 shows a typical conceptual overview of application-to-application (A2A) and business-to-business (B2B) integration, which can be found in many organizations (cf. Reese [12]). The dominating aspects are the many connections or integration adapters, which are currently under-represented in the (integration) modeling domain. Integration semantics are generally described based on a comprehensive (often graphically depicted) syntax and execution semantics (process model). In their best practices book Enterprise Integration Patterns (EIP), Hohpe and Woolf [7] have collected a widely used and accepted collection of integration patterns that are typical concepts used when implementing a messaging system and have proven to be useful in practice. However, they do not specify a semantic

J. Zdravkovic et al. (Eds.): CAiSE 2015, LNCS 9097, pp. 468–482, 2015.
DOI: 10.1007/978-3-319-19069-3_29

model for the formalization of the integration syntax and semantics. Most noticeable, the integration adapter modeling with its manifold characteristics is reduced to a *Channel Adapter* icon.

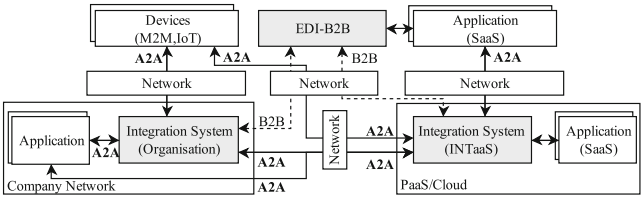


Fig. 1. Conceptual View on integration architecture of organizations

In previous work, we studied and provided a Domain-specific Language (DSL) with well-defined building blocks for modeling EIPs in the Business Process Model and Notation (BPMN) [11], which is a "de-facto" standard for modeling business process semantics and their runtime behavior [14]. We mapped EIPs to BPMN compatible syntax and defined execution semantics adapted to message processing. We extended that notion to end-to-end flows of messages, called *Integration Flow* (IFlow) [13]. In a nutshell, an IFlow can be seen as message-based integration from a sending application (Sender, *BPMN Participant*) to one or many receiving applications (Receiver(s), *BPMN Participant*) via *BPMN Message Flow* configurations (denoting the inbound and outbound adapters) and dedicated participant(s) that specify an integration process (composition of EIPs). We decided on BPMN for defining a "message-based integration" DSL due to its sufficient coverage of control flow, data and exception flow, process modeling capabilities and execution semantics [13,16]. The work on "Data in Business Processes" [8] shows that besides *Configuration-based Release Processes* (COREPRO) [10], which mainly deals with data-driven process modeling, (business) object status management, and UML activity diagrams, BPMN achieves the highest coverage in the categories relevant for our approach. Compared to BPMN and apart from the topic of "object state" representation, neither *Workflow Nets* [20] nor petri nets support data modeling at all [8]. For instance, Figure 2 shows an excerpt of an asynchronous integration scenario from the *Internet of Things* (IoT) domain, syntactically expressed in BPMN according to [13]. The encrypted incoming message is of type "TD" (short for *Telemetry Data*), which has to be normalized with respect to its timestamps using a *Message Transformation* pattern [7]. The *Message Queue Telemetry Transport* (MQTT)[1] is used as transport protocol, which is a common, lightweight queuing protocol frequently used in the IoT domain. The approach to specifying integration semantics and its runtime works well for common integration scenarios [13]. More complex scenarios have to deal with non-trivial combinations

[1] Message Queue Telemetry Transport: http://mqtt.org/

of message exchange pattern (MEP) and quality of service (QoS) levels. These notions are mostly induced during the adapter processing and continued into the integration process. Currently, integration modeling approaches (a) do not classify adapter characteristics, (b) leave the default adapter processing (mostly) hidden in the various runtime implementations, and (c) do not allow for configuration or change of the default behavior.

In this paper, we comprehensively investigate the range of characteristics of adapters during the integration flow processing and the various ways, in which they can be addressed. This provides the foundation for a classification of the adapter modeling, which we subsequently define in the form of an adapter flow (AF) and patterns. The pattern-based approach to adapter classification is a continuation of our previous work on the EIPs and the IFlow. The adapter processing patterns have proven to be intuitive to both practitioners and researchers alike and have been widely utilized for a variety of purposes including customer and partner content development. They provide the conceptual foundations for the SAP HANA Cloud Integration (HCI)[2] system, which is an Integration as a Service (INTaaS) implementation based on Apache Camel [3], an open-source integration system. The motivation for this paper is to provide a conceptual framework for classifying the adapter processing capabilities of middleware systems more generally based on the IFlow modeling approach, while being independent of the specific runtime platforms. The major contributions of this work are (1) a comprehensive classification of common adapter characteristics in integration systems and beyond, (2) an extension of the BPMN-based IFlow model for adapter flows (AFs) that make default processing visible to the user for all identified categories from (1) and allows for change of the default behavior, (3) the derivation of common adapter processing patterns and their representation in BPMN, as well as (4) the application to an existing open source middleware system, the technical analysis and discussion of the proposed approach as experimental validation in two examples. In a nutshell, we propose an answer to the underlying questions of the observations (a-c), e. g., "which QoS does the IFlow in Figure 2 have?" and "how can the default handling be adapted to custom requirements?".

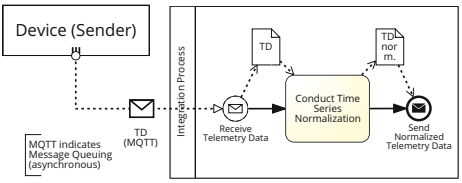


Fig. 2. IoT: Devices enqueue messages (asynch.) for time series normalization

Section 2 discusses adapter modeling characteristics, before we derive adapter processing patterns as an extension to the IFlow in Section 3, which we prototypically applied in two examples in Section 4. Section 5 summarizes our experiences, Section 6 discusses related work, and Section 7 concludes.

[2] SAP HANA Cloud Integration: http://help.sap.com/cloudintegration

2 Adapter Modeling Characteristics

In this section, we introduce a generalized integration system architecture, classify adapter characteristics and formulate them as modeling requirements. The adapter type classification is based on [7], supported by an analysis of 151 message endpoints, contributed to Apache Camel [3], and an integration expert workshop with 20 experienced integration experts from 7 different companies (cf. Suppl. Material [15]). As illustrated in Figure 3 conventional integration systems consist of a set of event-based or polling consumer adapters, an integration process engine, which executes sets of routing and message transformation tasks, and a set of producer adapters. The adapters represent the *Message Endpoint* pattern [7] and have to deal with security concerns and (possibly) format conversions from the sender format $F_s(msg)$ to an internal format $F_{cdm}(msg)$ (i. e., Canonical Data Model (CDM) [7]) that is used for the integration processing, and eventually conversions from the CDM to the target format $F_r(msg)$ understood by the receiver.

The internal messages are either distributed to Message Queues (asynchronously) or directly sent to the integration system process engine (synchronously). This engine uses a set of outbound adapters to actively interact with external systems. During the whole integration process, the recoverability should be ensured; thus, the internal message representations have to be stored locally using an operational data store or are queued for cross-process or cross-system message exchange. The execution environment of the consumer and producer adapters is an adapter runtime, which is part of the application server for conventional integration systems, however, can be an arbitrary software stack. The connections to related parts of the system (e. g., Messaging System, Data Store) are discussed subsequently as part of the classification. We consider five main categories, which allow to comprehensively describe adapters. Hence we discuss common capabilities from these categories and derive requirements (R-X) for a general adapter modeling approach.

Adapter Type. Adapters can be canonically differentiated by their type: consumer or producer (as seen before). A consumer adapter allows the message

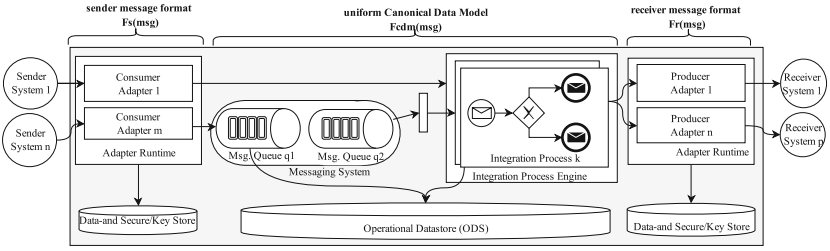


Fig. 3. Conceptual view on a conventional integration system with a slight emphasis on the consumer and producer adapters

sender applications to connect to the integration system. Message consumers are further sub-devided by their behavior into polling and event-based adapters. The polling adapter (e. g., File, (S)FTP) is configured to actively check for messages to read and process them (*R-1*: Model Polling Consumer). Hereby, settings like the polling interval, location, (initial) delay and format, can be specified. In contrast, an event-based consumer specifies an endpoint configuration (e. g., Servlet URI), on which it registers a "passive" listener that waits for events or callbacks from the sender (*R-2*: Model Event-based Consumer). A producer adapter forwards the messages to their receivers (*R-3*: Model Producer). The analysis of the 119 Apache Camel component bundles supports this differentiation (cf. [15]), by showing ten "consumer only" (6.6%), 33 "producer only" (22%) and 108 "consumer+producer" adapters. Despite the difficult task of determining "active" adapters, at least 34% of the sender adapters are "polling".

Configuration Complexity. The analysis of 119 component bundles resulted to 151 single components, or adapters. In other words, components like `mail` encapsulate multiple endpoints represented by protocols like `smtp`, `pop3`, `imap` (*R-4*: Model Multi-Component Adapter). Another outcome of the analysis showed that 30/151 components require more complex configurations, e. g., for the parameterization of connection and credential details like Java bean, key/trust store references as shown in Figure 3 (*R-5*: Allow for complex model references). Complementarily the user study resulted into a strong vote for scenario specific adaptations of the adapter's behavior. That means, an adapter shall provide extension points to hook in one or more custom processors, which can be modeled similar to an IFlow [13]. Evidence for such a requirement can be found in concepts like "channel modules" in SAP's *Process Integration*[3] middleware system. This is an extension to adapters, which can be combined to the notion of "message channel" modeling, similar to the integration process (*R-6*: Model Message Channels). Hence a message channel consists of consumer/producer adapters and arbitrarily many ordered processors.

Integration Styles. The *Message Exchange Pattern* (MEP) defines whether a message is sent `inOnly` (i. e., one-way) or `inOut` (i. e., two-way). A "two-way" message requires an (a)synchronously sent response, while a "one-way" message will never result to a response (*R-7*: Model MEP). A synchronous message exchange requires an immediate response during the initiated communication (i. e., mostly by event-based adapters), while an asynchronous exchange allows for an early close of the initiated communication and the response will be sent via mechanisms like "function/method callback" (*R-8*: Model Message Synch/Asynch communication). In this context, persistent adapters like "Web Service-Reliable Message" that receive and store the message, send an immediate response and then start a transactional redelivery, which represents "synch/asynch bridge" adapters (*R-9*): Model Message Synch/Asynch or Asynch/Synch communication). These adapters are necessary to "bridge" asynch. communication to synch. endpoints and vice versa.

[3] SAP Process Integration: http://help.sap.com/nwpi

Quality of Service. The most common service qualities of an integration system, which can be induced or supported by adapters are abbreviated as *BE*, *ALO*, *EO*, *EOIO* (sorted by the increasing quality level). The *Best Effort* (BE) messaging can be summarized as "fire-and-forget", which means that no guarantee for the delivery of a message is given. If a message shall be delivered *At Least Once* (ALO), it has to be persistently stored and redelivered from an adapter or the integration process (*R-10*: Model Message Redelivery (from a Message Store)).

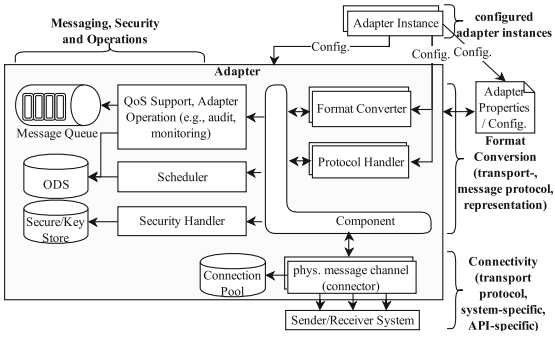


Fig. 4. Adapter Micro-Architecture

In case the message shall be delivered *Exactly Once* (EO), ALO has to be enhanced by the *Idempotent Receiver* [7], which stores the primary identifier of a message and filters out known messages (*R-11*: Model an Idempotent Receiver). Although the receiver itself should behave idempotent, producer adapters or the integration process can try to act in its place. For some integration cases, the strict adherence to a message sequence is important (e. g., create business object, before update). That means, messages shall be send *Exactly Once In Order* (EOIO). Therefore, EO is extended by a *Resequencer* pattern [7], which collects messages to emit them in the correct order (*R-12*: Model Resequencer).

Adapter Architecture and Tasks. Figure 4 draws a conceptional view on the internal architecture of a common adapter. Each adapter specifies a *Connector* or connection handler. The connector establishes a physical connection to the message endpoints (*R-13*: Model Physical Connection). For secure connections (e. g., user/password, certificates), a *Security Handler* is used (*R-14*: Model Security Relevant Configurations). Polling consumers might require a *Scheduler* for the configuration of the polling interval (cf. similar to *R-1*). For the QoS and monitoring support (e. g., message and channel monitoring), an operational data store or a message queue has to be used (*R-15*: Model (Queued-) Persistence, similar to R-10). The counterpart to the transport protocol handling connector (e. g., HTTP, FTP, JMS) is the *Format Conversion* (e. g., XML, JSON, CSV). An adapter shall be able to transform the sender format $F_s(msg)$ to the internal representation $F_{cdm}(msg)$ and eventually to the receiver format $F_r(msg)$ (*R-16*: Model Control and Data Flow). The modeled adapter shall be re-used in different adapter instances/configurations (*R-17*: Approach shall allow for re-use).

3 Adapter Modeling Approach

Following the IFlow modeling approach of Ritter [13,14] adapters are represented as message flows in BPMN (cf. Figure 2). To model integration processes with "simple" adapter configuration this approach is sufficient, although it overdefines BPMN message flows, makes the characteristics of an adapter implicit and does not allow for modeling of complex logic other than on second-level property sheets. For more complex adapter processing (cf. *R-6*), we subsequently define an explicitly modeled *Adapter Flow* similar to integration processes and discuss basic processing capabilities. We then derive more complex patterns from the requirements to model capabilities such as (secure) communication patterns (e. g., request/response and "bridging") and QoS patterns (e. g., reliable messaging with (transactional) redelivery, idempotent receiver, message resequencer).

3.1 Adapter Flow

An AF replaces the currently used BPMN message flow by an additional BPMN pool outside the integration process for more complex adapters with the need to specify an own control-, data- and exception flow. Thus, all messaging capabilities as described in the EIPs can be expressed within AFs. However, the physical connections to the sender/receiver (*R-13*) are represented by message flows. The AF of adapters with several selectable transport protocols (*R-4*), represented by connector and protocol handler (cf. Figure 4), remain stable, while the entering message flow of consumer and the leaving message flow of producer adapters changes based on selections. Basic "processing capabilities / patterns", as modeled in Figure 5, can be used within AFs, including skipping of processing steps based on conditions or errors (top-right) and message redelivery (bottom-right; *R-10*). These mechanisms are explicitly modeled using BPMN *Exclusive Gateway* elements. An adapter can decide to terminate the processing of one message (top-left) or the whole process (bottom-left) in exceptional situations or through other events. In case of synch. communication, a response is returned to the sender. When the basic processing capabilities are combined, more complex "adapter processing" can be expressed. To avoid re-occurring, complex adapter modeling patterns for communication and QoS support are required (cf. *R-17*).

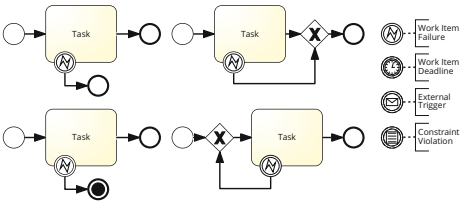
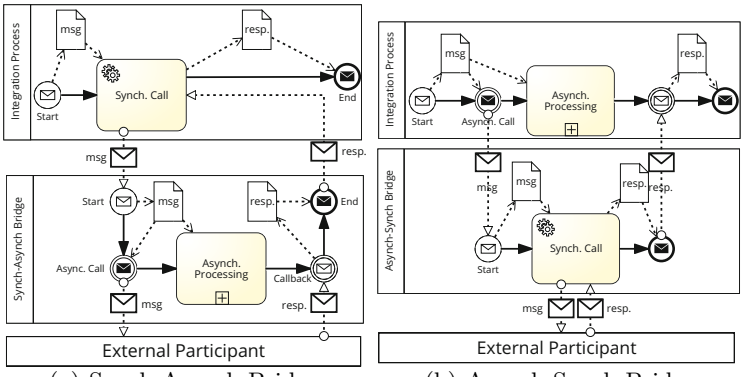


Fig. 5. AF message processing patterns

3.2 Communication Patterns

The (adapter) communication patterns specify several more complex interactions of adapters and integration processes within and outside an organization.

Communication Styles and Bridge Patterns. Common (business) applications support interfaces for synchronous (synch) and/or asynchronous (asych) communication styles. Synch. communication means applications respond to requests (e. g., with error codes or resulting data), while the requesting application is blocking in order to get the response (RPC-style). In asynch. communication, the sending application sends requests without waiting for responses from other applications and immediately continues with its processing after sending a message (non-blocking). However, the sending application may offer callback interfaces for getting responses back for it's previously and asynchronously sent requests. Integrating applications that do not share the same communication style requires an adapter that bridges/translates between both communication styles. Such a bridging adapter is modelled in Figure 6(a), which shows the



(a) Synch-Asynch Bridge (b) Asynch-Synch Bridge

Fig. 6. Adapter Bridge Patterns

modeled data and control flow for a synch to asynch bridge (cf. *R-16*, *R-8*), in which the synchronous call follows the `inOut` message exchange pattern (*R-7*). The *Synch. Call* in the integration process is modelled as BPMN *Service Task* which connects with a message flow to the Synch-Asynch Bridge BPMN *Pool*. The message is forwarded to an *Async. Call* represented as a BPMN *Intermediate Message End Event* that connects via a message flow to the *External Participant*. It continues with asynch. processing that reacts to *Callback* messages in an BPMN *Intermediate Message Start Event* and forwards the response to the *Synch. Call* service task. Following the same pattern, Figure 6(b) shows a model for a asynch to synch bridge (cf. *R-9*) and includes the handling of responses and forwarding them to callback interfaces. Both bridge adapter modeling patterns can be reused, applied and adjusted in other IFlows (cf. *R-17*) or "inlined" to the integration process of an IFlow.

Processing Patterns. The AFs can be modeled to adapt between two integration processes across tenant or network boundaries (A2A and B2B), for which an

integration process is associated to one tenant or network. In the case of cross-tenant integration, the IFlow of tenant T_a can adapt to an IFlow in another tenant T_b by representing the IFlow T_b as an delegate in IFlow T_a and vice versa. As such, IFlows are either "local" to one tenant, which means that they are locally visible and modifiable or they are "remote" which means that they can only be connected from "local" IFlows but not made visible or modified. Hereby, for synch. communication the "remote" IFlow could be represented as a collapsed BPMN *Pool* (which cannot be expanded) and connected to the "local" integration process with request/response BPMN message flows. For (reliable) asynchronous communication a shared data store is used to make the necessary queuing step explicit. As a representative pattern, this "remote" IFlow delegate can be used to model across networks or IFlows, by changing its type.

3.3 Quality of Service Patterns

The QoS levels (*R-10–R-12*) denote more complex configuration building blocks. Subsequently, the necessary patterns are defined and mapped to BPMN.

Reliable Messaging. To guarantee that a message is not lost in asynchronous scenarios the message must be stored into a message store (e. g., database) or enqueued to a messaging system (e. g., JMS brokers), before the reception is explicitly or implicitly acknowledged via `ack` to the sender. As such, an integration system aims to store the message in the consumer adapter, sending the `ack` messages to unblock the sender waiting for a response and to minimize the possibilities for errors before the persistency step. Similarily, there are adapters that access a data store/queue for cross-applications and software systems (e. g., JDBC, JMS). AFs could connect to BPMN data store to model key and trust stores (cf. *R-5*). For instance, Figure 7 uses AFs to model JMS adapters and the access to queues in a message broker, which is represented as a BPMN *Data Store* (cf. *R-15*). This allows to attach configurations to the data store (such as connection details) and to the BPMN *Data Association* (cf. *R-16*), and the enqueue/dequeue tasks in the producer adapter (cf. *R-3*) to the data store (such as queue/topic names). Through BPMN *Timer Event*, the polling behavior of a consumer adapter can be modeled (*R-1*). For instance, Figure 7 (bottom, right) shows the periodical, transactional dequeue of messages using a BPMN *Task* within a transactional subprocess, which specifies the transactional boundaries. In case of exceptions during the task processing within these boundaries, the message is not dequeued from the

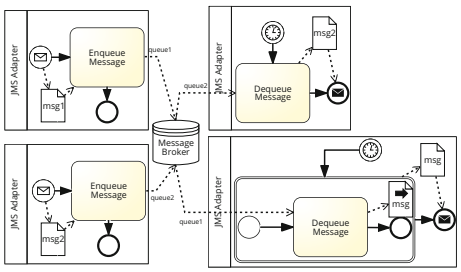


Fig. 7. Adapter modeling with message queuing via data-stores and transactional subprocess in consumer adapter

queue. A message redelivery would be attempted in the next polling interval (cf. *R-10*). Although *Topics* for publish/subscribe scenarios could be modeled similarly, they could be represented by BPMN signal end/start events (cf. Section 3.5 in [15]). The transferred message would be determined by the associated BPMN *Data Object* and the corresponding events would be identifiable by their matching names. Clearly this would make the inner mechanics implicit, but would allow for the modeling of an event-based consumer adapter (cf. *R-2*). For (reliable) asynchronous, `inOut` messaging (cf. Figure 8), we assume a "reply-to" header field attached to the `req-msg` indicating that the AF (right JMS Adapter) should reply to the specified queue. The (queued) response is correlated to the waiting integration process instance by using the identifier of the `req-msg`.

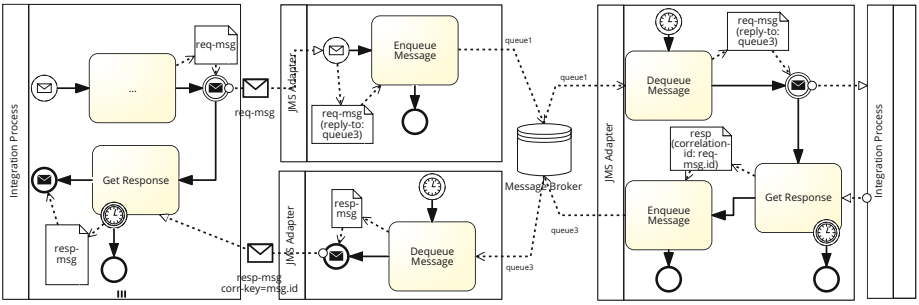


Fig. 8. Adapter modeling with message queuing via data-stores and explicit modeling of request/reply via response queues

Idempotency Repository. To support AMO and EO (in combination with *Reliable Messaging*), the integration system needs to take care that messages are not sent twice to the receiver (cf. *R-11*). This is modeled by a flow step the integration process (or AF) that filters already sent messages, which is preferably executed just before the message is sent to the receiving application in a producer AF. Figure 10 shows the filter processing as part of a producer AF (bottom) by accessing an *Idempotency Repository* [7], which is represented as a BPMN data store, storing the identifiers of already processed messages against it checks the current message identifier.

Message Resequencing. The resequencer (cf. [14]) can be used for *In-Order* (IO) scenarios, for which the messages have to be ordered according to a sequence number. Alternatively, order preserving queues (e. g., specified in JMS) are used to keep messages in sequence. The EOIO processing additionally requires the combination of *Reliable Messaging* with redelivery semantics and a filter step using the *Idempotency Repository* to guarantee that the messages are sent exactly once and in order (cf. *R-12*).

4 Examples

Let us apply the rather abstract BPMN AF definitions and integration patterns to two intriguing integration scenarios, which cover secure, reliable messaging as well as most difficult QoS configurations. Coming back to the motivating example and questions around the visualization (2) and re-configuration (3) of the expected default exception handling and compensation in the "Reliable Time Series Normalization" scenario (Section 1) Figure 9 shows the syntax proposal

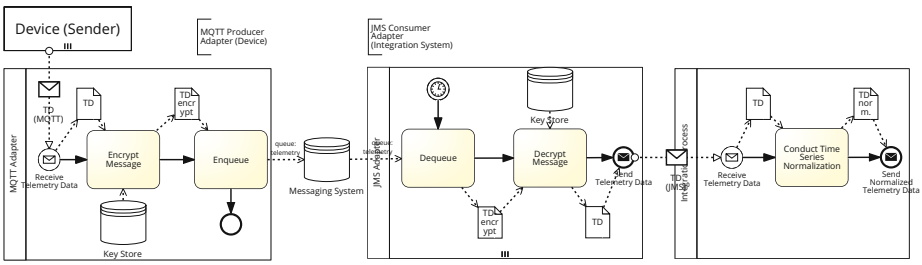


Fig. 9. "Time Series Normalization" Scenario: Reliable, secure messaging

following our mapping to BPMN. The devices enqueue an encrypted *TD* message to the `telemetry` queue in the messaging system using an *MQTT* adapter (cf. *R-6*). The integration process listens to the queue using a *JMS* adapter, which decrypts the received messages (cf. *R-14*) and passes them to the integration process, where the message content is normalized.

The QoS support is crucial for integration systems. When sending a message synchronously to a receiver, BE is applied (i. e., delivery will be attempted otherwise the sender will receive an exception message). In case of asynchronous, reliable, in-order messaging this is not sufficient. The message has to be persistently stored and a retry has to be started to guarantee its delivery (cf. *R-10*), e. g., in a message queue, since the sender cannot be notified. In addition, the order of the messages according to a *Message Sequence* [7] has to be guaranteed using a *Resequencer* pattern (cf. *R-12*). If in addition, duplicate messages are filtered out during the processing (cf. *R-11*), the QoS is called EOIO, syntactically in Figure 10. The consumer AF starts with a synchronous part by storing the message and sending a response. Hereby, the "Redelivery on Exception" sub-process acts as a combined ALO, synch/asynch bridge pattern, which then starts the asynchronous delivery of the message to the integration process, which collects messages and orders them along defined sequences using a "Resequencer" sub-process pattern and synchronously emits the messages to the producer adapter. The producer adapter checks whether a message has already been processed and synchronously sends it to the receiver. The receiver's response (i. e., acknowledgement or exception) is passed to the integration process, which triggers a message redelivery on exception.

5 Qualitative Analysis and Experiences

This section discusses "ex-post", practical experiences with the defined adapter modeling based on an evaluation with integration experts from different enterprises as interviews, workshops and surveys (cf. Suppl. Material [15]). The evaluation of the approach can be summarized to the topics subsequently discussed. In general, the integration models with an explicit AF are experienced as more complex (i. e., contradicting the aimed usage of BPMN by business experts [11]), however, the visibility of the default adapter characteristics allow for better insights into the integration flow modeling, ease of use, a more intuitive and faster modeling through the identified patterns.

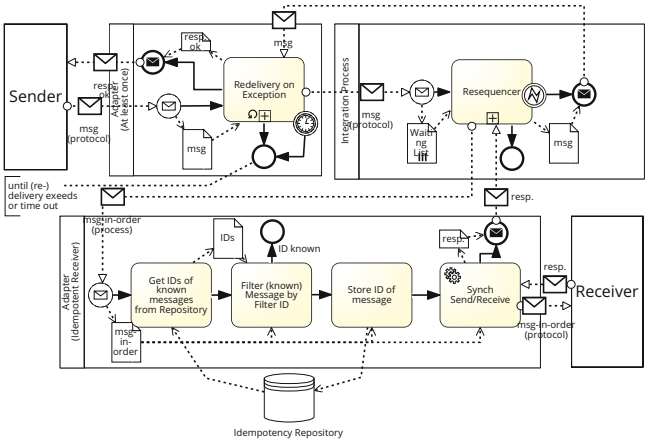


Fig. 10. "EOIO": message redelivery in the consumer adapter, resequencer in the integration process, and idempotent message handling in the producer adapter

Modeling Complexity. Some BPMN syntax elements are not applicable to the integration environment in an useful way. For example, the lane element has no semantic meaning and could only be used to structure certain aspects of the integration systems, such as distinguishing normal logic from AF logic. Though this would increase the size of the diagram leading to a confusing model. Additionaly, in Figure 7, for queuing with a message broker the data association to the message broker is denoted with the queue name to/from messages should be enqueued/dequeued. Although this was rated as complex (e. g., the adapter on top-left communicates with the adapter on bottom-right, while the adapter on top-right seems more related to the adapter on top-left), it was favored over the BPMN *Signal* approach proposed for topics. In case of many connections to the message broker, a partitioning of the IFlow into several smaller diagrams would help to make the single parts more understandable (cf. "in-context" editing), while tool support would be needed to show the complete IFlow on demand. The alternative of modeling several instances of one broker in one IFlow, which are then connected to related adapters only, was not seen as desirable solution.

Modeling Preferences. The modeling of AFs was very well received, while the participants differentiated between producer and consumer adapter modeling. The producer adapter modeling allows for adding scenario-specific "pre-processing" capabilities to the system, before entering the integration process (e. g., especially for bridges). The consumer AFs were seen limited to the QoS support, while potential "post-processing" logic could be executed in the integration process. However, from a modularity and resource consumption point of view, a clear separation of adapter and integration process logic was received well. The explicit modeling of security related topics like key stores was controversially discussed. While participants with a more technical background like the proposed approach (e. g., helping them to be precise in the security aspect modeling), more business related participants complaint about the additional complexity. Both parties agreed that a more explicit modeling of the inner workings of a message broker is not necessary and the transactional de-queuing with the BPMN *Transactional Sub-Process* was rated intuitive.

6 Related Work

Recently, the topics of "integration adapter" and QoS were mainly discussed in the areas of "Service-oriented Architectures" (SOA) and connectivity to "Data Warehouses" (DW). However, the closest related work can be found in the *Enterprise Integration Pattern* (EIP) icon notation collected by Hophe et al. [7]. The EIP notation defines modeling building blocks for core integration aspects (e. g., resequencer, publish/subscribe, durable (topic) subscriber and channel adapter). In addition, to these pattern, our approach has a representation of an idempotent receiver, a messaging system for "store-and-forward" processing (guaranteed message delivery) and different levels of service quality can be modeled.

Quality of Service. Most of the QoS definitions, from this work, can be found in standard industry middleware systems, too. For instance, the *WebLogic* system [9] knows EO, ALO as "duplicates only mode", and "At-Most Once" (AMO), which can be modeled by our approach through an idempotent receiver pattern. From a modeling point of view, adapters are reduced to one icon and a property sheet, similar to the EIP icon notation. Closer to our approach, Gönczy et al [5] define a proprietary meta-model for reliable messaging with acknowledgements for some QoS (i. e., ALO, EO) and target to apply verification approaches using graph transformations [6]. However, other service qualities, e. g., IO, AMO, EOIO, are currently undefined.

Model-driven Adapter Development and Re-configuration. There are some approaches for automatic adapter generation and re-configuration from the SOA domain, to which we aligned our terminology, however, they do not define a (conceptual) modeling approach for integration adapters. Nezhad [4] summarizes work on model-driven development of web services, while highlighting the importance of a QoS support. Other approaches target the self-adapting adapters

in terms of signature-/protocol-level and quality related re-configurations and planning (e. g., [19]). We consider these approaches complementary to our work.

Data-Intensive Adapter Modeling. Through data warehouse connectivity scenarios, the *Extract-Transform-Load* (ETL) domain gained interest in the conceptual modeling of more "data-intensive" adapters (e. g., [17]). Although characteristics like QoS are not relevant for data warehouse connectivity, these modeling approaches can be seen as domain-specific, complementary work. For instance, Akkaoui et al. mapped ETL processes to BPMN [1] and provided a maintenance framework for [2], which can be seen as subset of our approach. There are several UML-based approaches, e. g., for modeling data mining with time-series data [21] or ETL data flows [18], which mostly define new icon notations, similar to the EIP notation. Their focus on the data flow limits the modeling to data transformation.

7 Concluding Remarks

In this paper, we (a) define common adapter types within integration systems (starting from literature and small empirical studies), (b) extend our BPMN-based definition of IFlows [13,14] by AF constructs and patterns to make the default adapter behavior visible, and thus (c) provide a basis for a scenario-specific adapter configuration. We started with a systematical analysis of common adapter characteristics (cf. R-1–9, R-13–17), spanning to edge cases like QoS modeling (cf. R-10-12).

The mapping of these adapter characteristics to BPMN allowed us to link them to the integration flows and define common integration adapter processing capabilities and patterns. We applied our approach to the real-world "internet of things" integration scenario and the "exactly-once-in-order" QoS case. The evaluation exclusively targets the syntactical feasibility and applicability of the approach. The presented examples show some minor shortcomings due to BPMN's focus on control over data flow and advice to change the proposed, which we solved by explicit modeling of AFs as separate BPMN pools.

The brief discussion of the experiences from the presented and many more real-world case studies as well as integration expert and partner interviews show the value of the modular, pattern-based and explicit modeling approach, however, highlights topics for further (quantitative) analysis. A solution for some of the mentioned issues with BPMN can be found in further investigations of the possibilities to use BPMN extensions. However, as pre-existing elements may not be modified and the syntax of BPMN models (such as conditional data flows) may not be changed, the mitigations might go beyond the current BPMN syntax.

References

1. El Akkaoui, Z., Mazón, J.-N., Vaisman, A., Zimányi, E.: BPMN-Based Conceptual Modeling of ETL Processes. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2012. LNCS, vol. 7448, pp. 1–14. Springer, Heidelberg (2012)

2. Akkaoui, Z.E., Zimányi, E., Mazón, J., Trujillo, J.: A bpmn-based design and maintenance framework for ETL processes. IJDWM **9**(3), 46–72 (2013)
3. Anstey, J., Zbarcea, H.: Camel in Action. Manning (2011)
4. Benatallah, B., Casati, F., Toumani, F., Ponge, J., Nezhad, H.R.M.: Service mosaic: A model-driven framework for web services life-cycle management. IEEE Internet Computing **10**(4), 55–63 (2006)
5. Gnczy, L., Varr, D.: Modeling of reliable messaging in service oriented architectures. In: International Workshop on Web Services Modeling and Testing (2006)
6. Gönczy, L., Kovács, M., Varró, D.: Modeling and verification of reliable messaging by graph transformation systems. Electr. Notes Theor. Comput. Sci. **175**(4), 37–50 (2007)
7. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Boston (2003)
8. Meyer, A., Smirnov, S., Weske, M.: Data in business processes. EMISA Forum **31**(3), 5–31 (2011)
9. Mountjoy, J., Chugh, A.: WebLogic - the definitive guide. O'Reilly (2004)
10. Müller, D.: Management datengetriebener Prozessstrukturen. PhD thesis (2009)
11. O. M. G. (OMG). Business process model and notation (bpmn) version 2.0. Technical report (January 2011)
12. Reese, G.: Cloud Application Architectures: Building Applications and Infrastructure in the Cloud. O'Reilly Media Inc. (2009)
13. Ritter, D.: Experiences with Business Process Model and Notation for Modeling Integration Patterns. In: Cabot, J., Rubin, J. (eds.) ECMFA 2014. LNCS, vol. 8569, pp. 254–266. Springer, Heidelberg (2014)
14. Ritter, D.: Using the business process model and notation for modeling enterprise integration patterns. CoRR, abs/1403.4053 (2014)
15. Ritter, D., Holzleitner, M.: Qualitative Analysis of Integration Adapter Modeling. CoRR, abs/1503.02007 (2015)
16. Ritter, D., Sosulski, J.: Modeling exception flows in integration systems. In: 18th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2014, Ulm, BW, Germany, September 3–5 (2014)
17. Simitsis, A., Vassiliadis, P.: A methodology for the conceptual modeling of ETL processes. In: The 15th Conference on Advanced Information Systems Engineering (CAiSE 2003), Klagenfurt/Velden, Austria, 16–20 June, Workshops Proceedings, Information Systems for a Connected Society (2003)
18. Trujillo, J., Luján-Mora, S.: A UML Based Approach for Modeling ETL Processes in Data Warehouses. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 307–320. Springer, Heidelberg (2003)
19. van den Heuvel, W., Weigand, H., Hiel, M.: Configurable adapters: the substrate of self-adaptive web services. In: 9th International Conference on Electronic Commerce: The Wireless World of Electronic Commerce, Minneapolis, USA (2007)
20. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers **8**(1), 21–66 (1998)
21. Zubcoff, J.J., Pardillo, J., Trujillo, J.: A UML profile for the conceptual modelling of data-mining with time-series in data warehouses. Information & Software Technology **51**(6), 977–992 (2009)