Information Systems 67 (2017) 36–57

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Patterns for emerging application integration scenarios: A survey

Daniel Ritter[a,b,*], Norman May[a], Stefanie Rinderle-Ma[b]

[a] SAP SE, Germany
[b] University of Vienna, Faculty of Computer Science, Austria

ARTICLE INFO

Article history:
Received 11 February 2017
Revised 13 March 2017
Accepted 15 March 2017
Available online 18 March 2017

Keywords:
Cloud integration
Device integration
Enterprise application integration
Enterprise integration patterns
Hybrid integration

ABSTRACT

The discipline of enterprise application integration (EAI) enables the decoupled communication between (business) applications, and thus became a cornerstone of today's IT architectures. In 2004, the book by Hohpe and Woolf on Enterprise Integration Patterns (EIP) provided a fundamental collection of messaging patterns, denoting the building blocks of many EAI system implementations. Since then, multiple new trends and a broad range of new application scenarios have emerged, e. g., cloud and mobile computing, multimedia streams. These developments ultimately lead to conceptual changes and challenges such as larger data volumes (i. e., message sizes), a growing number of messages (i. e., velocity) and communication partners, and even more diverse message formats (i. e., variety). However, the research since 2004 focused on isolated EAI solutions, and thus a broader and integrated analysis of solutions and new patterns is missing. In this survey, we summarize new trends and application scenarios which serve as a frame to structure our survey of academic research on EIP, existing systems for EAI and also to classify integration patterns from these sources. We evaluate recently developed integration solutions and patterns in the context of real-world integration scenarios. Finally, we derive and summarize remaining challenges and open research questions.

1. Introduction

Enterprise Application Integration (EAI) addresses the requirement to integrate independent applications which need to communicate with each other [1,2]. Hence, some middleware is employed to abstract from the details of communication and orchestration of applications. For the purposes of integration, a set of core Enterprise Integration Patterns (EIP) were documented in [3], which describe recurring scenarios and solutions to realize EAI using messaging.

Originally, EAI focused on the integration of applications within a single organization. However, as hosting (parts of) applications in the cloud becomes increasingly popular, EAI also needs to address scenarios where applications that are hosted in the cloud or on-premise (i. e., within company networks) need to be integrated. We refer to such scenarios as *hybrid applications*, following Forrester [4]. Especially hybrid applications require a stronger decoupling to integrate on-premise with cloud applications, and consequently, hybrid applications prefer to use (asynchronous) message-based communication patterns, while RPC-style integration is still quite common for EAI in on-premise setups. Most of the current research focuses on RPC-style Service-oriented Architecture (SOA).

1.1. New challenges for enterprise application integration

In this paper we identify further new IT trends and application scenarios which emerged after the seminal book on EIP by Hophe and Woolf [3]. Some of these changes, e. g., Cloud and Mobile Computing, IoT, Microservices, and API Management, were even recently acknowledged by the EIP authors [5].

One major source for identifying new trends is the yearly published "Emerging Technologies Hype Cycle" report between 2005 and 2017 by Gartner [6]. We focused on the most relevant trends for application integration today, i. e., we excluded trends like machine learning and analytics in the analysis presented in this paper. The results are depicted in Fig. 1. Both our literature review in Section 2 and our system review in Section 3 are consistent with the trends identified by the Gartner reports because both academic research as well as concrete systems address these trends.

Broadly speaking, the early years (2005–2007) are dominated by Service-oriented Architecture (SOA) and Event-driven Architecture (EDA) styles. But also related technologies like Microservices are mentioned by Gartner in 2017 [6], and API Management by Forrester for 2016–2018 [7].

* Corresponding author.
E-mail addresses: daniel.ritter@sap.com (D. Ritter), norman.may@sap.com (N. May), stefanie.rinderle-ma@univie.ac.at (S. Rinderle-Ma).

http://dx.doi.org/10.1016/j.is.2017.03.003
0306-4379/

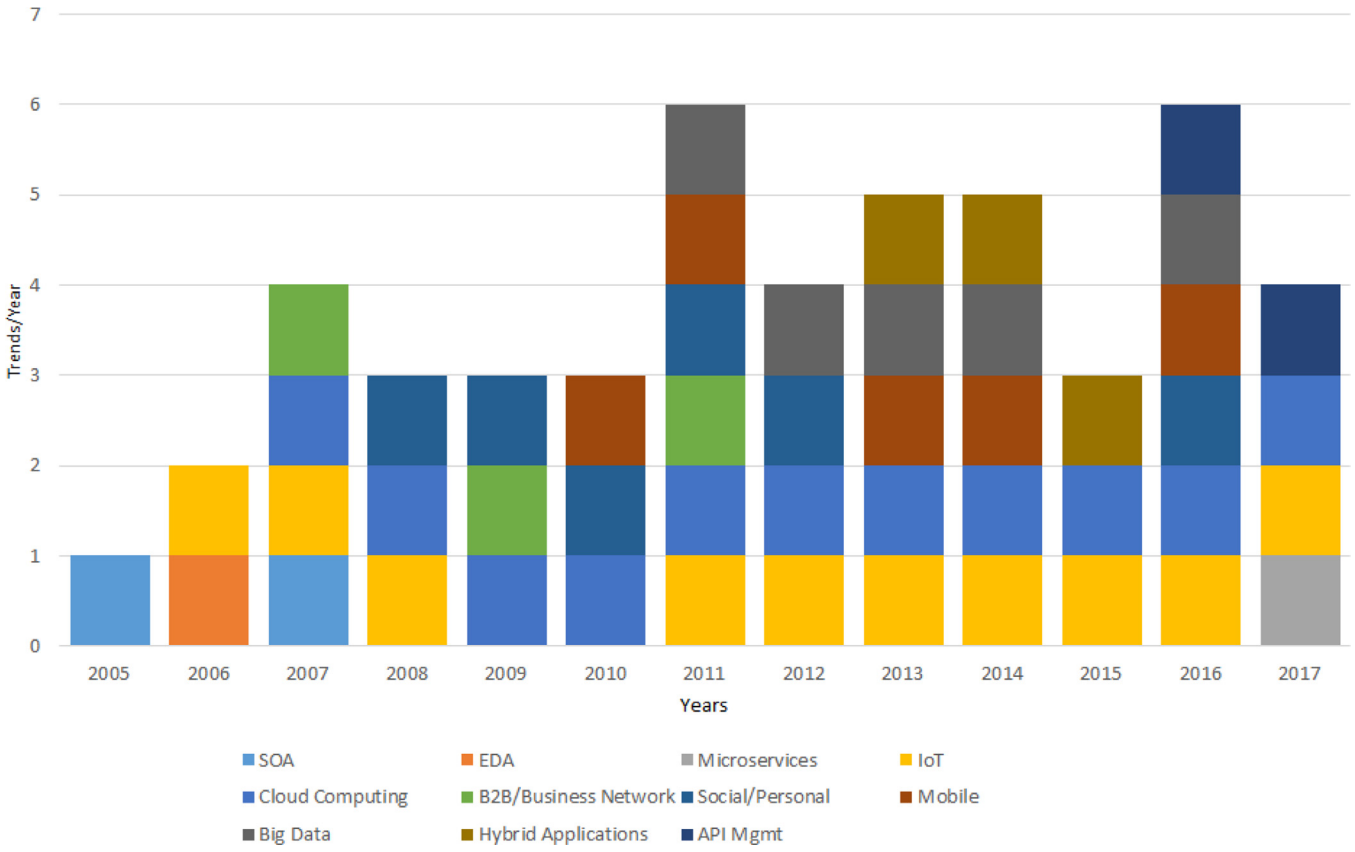


Fig. 1. IT trends since 2005 in the context of application integration.

The Cloud Computing trend became prominent in 2007 and subsequently led to trends like Hybrid Computing, i. e., multi-cloud and on-premise applications, from 2013 to 2015 and the move from B2B to cloud-based business networks. Early developments in the Internet of Things (IoT) became influential even before 2006–2008 even though devices were not yet affordable and wide spread. However, with the advent of Mobile Computing in 2010, mobile and IoT devices and applications (since 2012) started to play a role for application integration. As countless devices and applications generated an increasing amount of data, Big Data (from 2011) became influential and a challenge not only for integration systems. Finally, humans increasingly organized themselves in social media with its momentum from 2008 to 2012, which evolves to personal computing, supported by wearable and mobile devices and applications.

In Fig. 2 we associate the trends mentioned in Fig. 1 with aspects of application integration. While some of the nodes represent the trends (i. e., without application and integration system), the edges denote required interaction and (transitive) communication, which also gives hints on existing as well as new integration scenarios for the different combinations. Node spanning trends are denoted by “dashed-line” nodes.

It is noteworthy that for hybrid application integration for both on-premise to cloud as well as cloud to cloud communication becomes relevant, e. g., for migration of on-premise applications to the cloud. This raises technical issues like security but also robustness in the face of errors or unavailable communication partners. Furthermore, cloud, on-premise and mobile applications generate communication traffic of an ever increasing scale with respect to the amount of data but also the number of communication partners. In this cloud setup organizations replace the bilateral RPC-style communication by asynchronous, message-based interactions which are mediated by integration systems. When the applications in an EAI scenario are partly hosted by cloud providers, monitoring becomes more challenging because the interfaces available for monitoring may be limited. We also review these and other non-functional aspects in our literature review in Section 2, our system review in Section 3 and in the context of real-world integration scenarios in Section 6.

1.2. Research method

This survey relies on the design science methodology [8] as a rigid method to collect and evaluate the new trends mentioned above, to summarize research which adds new patterns to the original EIP, and to evaluate these new patterns in the context of real world application scenarios. Fig. 3 depicts the research method applied in this paper, and we use it to structure this paper.

Our fundamental **theory** and motivation for this paper is: *The original EIP from 2004 do not completely cover new trends in 2016 and beyond*. From this we derive **hypothesis (H1)**, i. e., *the existing EIP do not suffice for all application scenarios after 2004*. This hypothesis is tested based on two observation artifacts, i. e., a systematic literature review in Section 2 and a systematic system review in Section 3. Based on the literature review we analyze whether new trends and application scenarios can be seen after 2004 and which solutions are provided. The system review aims at analyzing available systems regarding their support for integration. Interpreting the literature and system reviews then leads us to the tentative **hypothesis (H2)** that *current system implementations support patterns beyond EIP* which results in a *strong demand for systematic description*. In order to address the detected gaps we propose new pattern categories and patterns. In **hypothesis (H3)** we argue that *some trends are handled in an (yet) immature and ad-hoc fashion*, and thus require a structuring in form of patterns. These artifacts are then evaluated based on a quantitative analysis of sev-

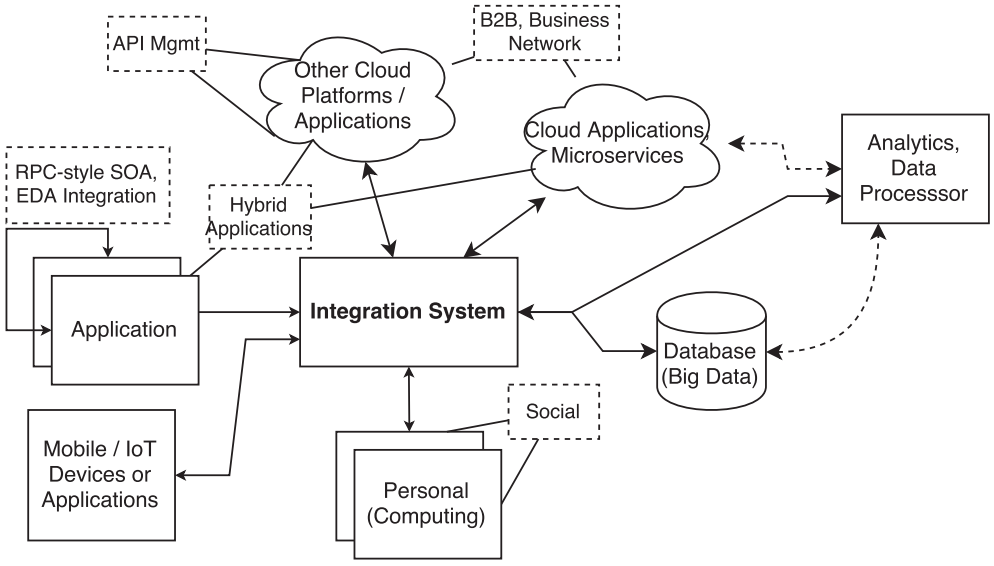


Fig. 2. IT trends since 2005 an their relationship to application integration.

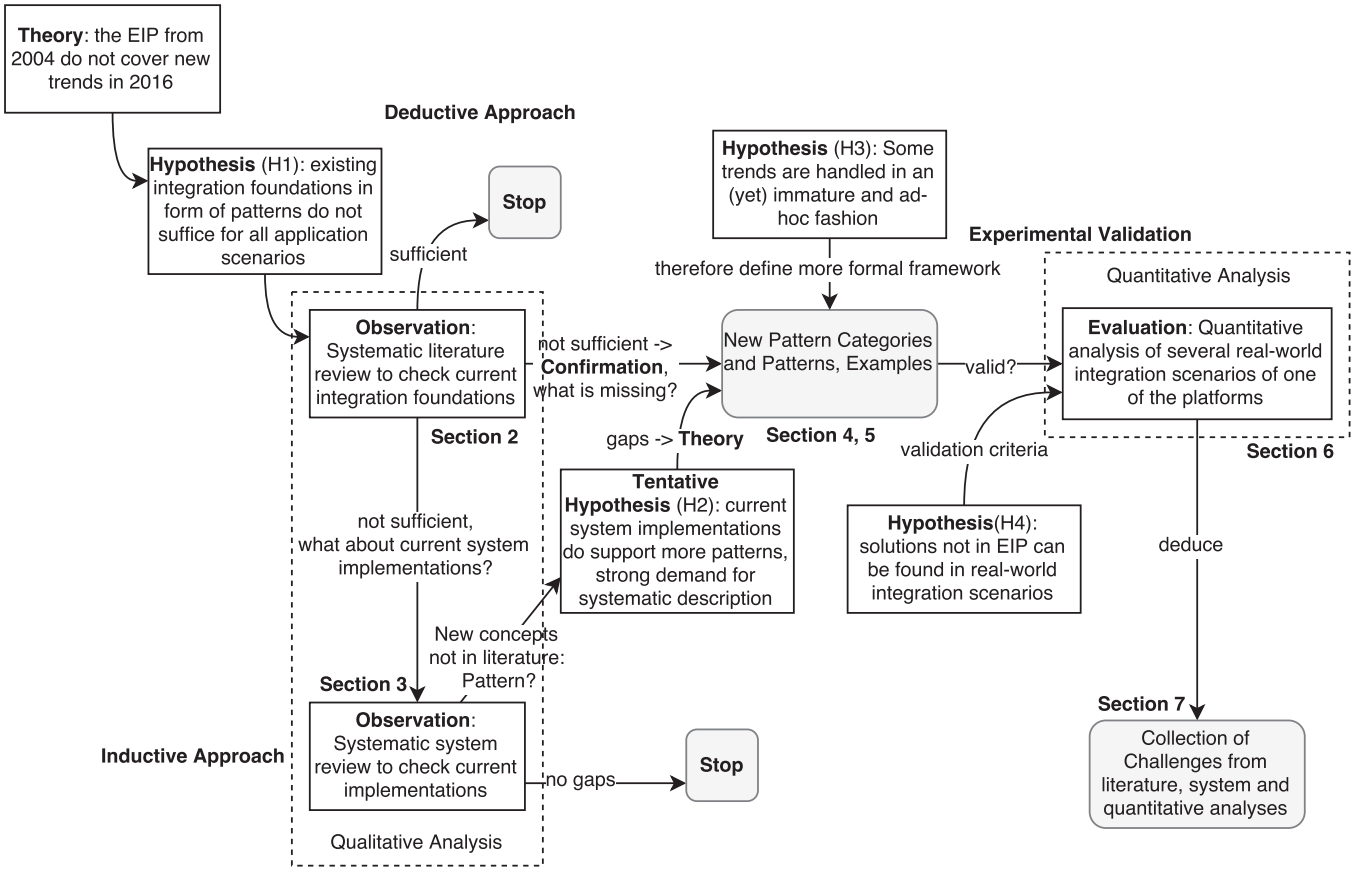


Fig. 3. Design science methodology used in this paper.

eral real-world integration scenarios following the **hypothesis (H4)** *solutions not in EIP can be found in real-world integration scenarios for the trends*. Finally, resulting research directions are described.

1.3. Contributions and paper outline

In this paper we make the following contributions:

- A systematic literature review of the trends, e. g., cloud and hybrid application integration approaches (↦ H1), and an analysis of the most influential system implementations of this domain (↦ H2).
- An extended pattern template plus an example based on descriptions of cross-concern technical qualities (e. g., (stateful) conversation, streaming, security) for a comprehensive coverage of new requirements (↦ H3).
- The evaluation of the found patterns as part of integration scenarios in a well-established cloud integration system in form of a quantitative analysis based on new monitoring patterns (↦ H4).

The paper is structured as follows. In Section 2 the literature review of approaches that are closely related to application integration and integration patterns is conducted by setting them into context to the new trends since 2004. Section 3 describes the system review with focus on non-functional aspects (NFA) – related to the trends – identified during the literature review, thus leading to a list of potentially missing functionality. In Section 4, we discuss how to capture the functionality as patterns, similar to the original EIP, and discuss two patterns in more detail Section 5. Then we analyze and discuss real-world integration scenarios related to the trends and their usage of the new patterns in Section 6. Section 7 concludes the paper and states open issues which are not addressed in existing work.

2. Literature review

In this section we conduct a literature review in order to answer the hypothesis *H1: existing integration foundations in form of patterns do not suffice for all application scenarios* as set out in Fig. 3. The hypothesis raises two questions to be investigated in the literature review, i. e., a) are there any topics after 2004 not yet covered by the original EIP? and if yes b) do existing approaches provide solutions to these topics?.

The literature review is based on the guidelines described in [9]. The primary selection of references was conducted using google scholar (scholar.google.com) on 2016-10-4. The search string was

`allintitle: integration patterns` excluding patents and citations. As a general baseline, only papers after 2004 are considered as the main theory behind this study is that *the EIP from 2004 do not cover trends in 2016*. Hence the time range was set to 2005 – 2016. Overall this resulted in 525 hits. On these hits, the following selection criteria were applied:

- relation to computer science, enterprise application integration, service integration, data integration, system integration
- availability of the document
- published in English
- published (excluding Master theses)

Altogether, 52 papers were selected as relevant (the primary literature list can be found here: http://cs.univie.ac.at/wst/research/projects/project/infproj/1085/). These 52 papers were further analyzed whether they contribute as observations to the hypotheses. This resulted in removing 23 papers from the primary literature list (for example, papers were excluded that focus on data integration). Then a vertical search was conducted in forward and backward direction, resulting in 43 papers, including one paper that was added based on expert knowledge. After analyzing these papers, 34 were included in the secondary literature list. Overall, this results in 63 papers for the secondary literature list (to be found at http://cs.univie.ac.at/wst/research/projects/project/infproj/1085/).

2.1. Processing of selected literature – topics and trends

At first, all papers from the secondary literature list were analyzed with respect to the topics they are mentioning. Comparing the harvested topics with the trends identified in the introduction gives an answer to question a) are there any topics after 2004 not yet covered by the EIP?. In this first step it is sufficient that a topic is mentioned. It was not necessary that a solution was provided. As the collected topics are very fine granular and spread widely, they were first grouped according to the trends mentioned the introduction.

Fig. 4 depicts the distribution of topic mentions along the trend over time. It can be seen that SOA (i. e., RPC-style integration [3]) plays a dominant role, particularly in the years 2005–2013. During this period, some topics such as mashups, cloud, and EDA were occasionally mentioned. In the last years, i.e., 2014–2016 the picture seems to change, turning away from the strong focus on SOA towards topics such as cloud, hybrid, and IoT.

From Fig. 4 it can be concluded that some of the trends occurred in the literature after 2004 with a dominant occurrence of SOA. Apparently, since 2014 SOA loses significance, and other trends such as IoT and cloud seem to gain more attention. From the dominance of SOA we also conclude that a more fine-grained analysis of the mentioned topics is meaningful. Hence, in the following, summaries for the analyzed approaches are provided, ordered by the topics and areas they work on. Subsequently, the list of trends will be complemented with **N**on **F**unctional **A**spects or requirements (NFA) mentioned by literature that constitute further important topics for EAI since 2005. Moreover, if approaches provide solutions with respect to the different topics, the type of solution will be collected.

2.2. Literature summaries

This section summarizes the approaches identified in the literature search. We organize the summaries chronologically by following the timeline from Fig. 1. In the context of EAI, no work was found on the internet of things, social/personal computing, microservices, and API management, which can be seen as successor of the Service-oriented Architecture trend. In addition to the trends, for each approach we try to derive additional NFA as well as the proposed solutions. The harvested NFA and solutions are summarized at the end of the section in Table 1 and yield the input for the further analysis.

2.2.1. Service-oriented and event-driven architectures

According to the timeline, the first EAI solutions after 2005 were provided by Service-oriented and Event-driven Architectures representing mostly RPC-style solutions (i. e., a post shared database and file sharing integration style, compared to "messaging" like EIP, according to [3]).

Service-oriented Architecture. Hentrich and Zdun present patterns that address data integration issues such as incompatible data definitions, inconsistent data across the enterprise, data redundancy, and update anomalies [10]. It is described how to integrate the application-specific business object models of various external systems into a consistent process-driven and service-oriented architecture. In summary, the proposed solution combines SOA with patterns, e. g., refactoring patterns. In [11], the authors propose a pattern language for design issues of business process-driven service orchestrations. The patterns illustrate how these types of service invocation need to be reflected in process models in order to integrate processes with services. Implications regarding the functional architecture are also captured by the patterns. Specifically, the patterns reflect solutions for general business requirements that can be found in SOA engagements. Overall, the paper proposes a solution, more precisely, a pattern language covering, for example, Synchronous Service Activity, Fire Event Activity, and Asynchronous Sub-process Service.

In subsequent work [12] the authors present solutions to Process-driven SOA patterns in the sense of a process integration architecture featuring patterns at Macro Flow (business process) and Micro Flow level (transaction or human), as well as Integration Adapter, Configurable Dispatcher, and Integration Adapter Repository. These patterns correspond to the ones proposed in [10]. Furthermore long-running business processes are distinguished from short-running technical processes. Zdun et al. present a survey of technology-independent patterns that are relevant for SOA

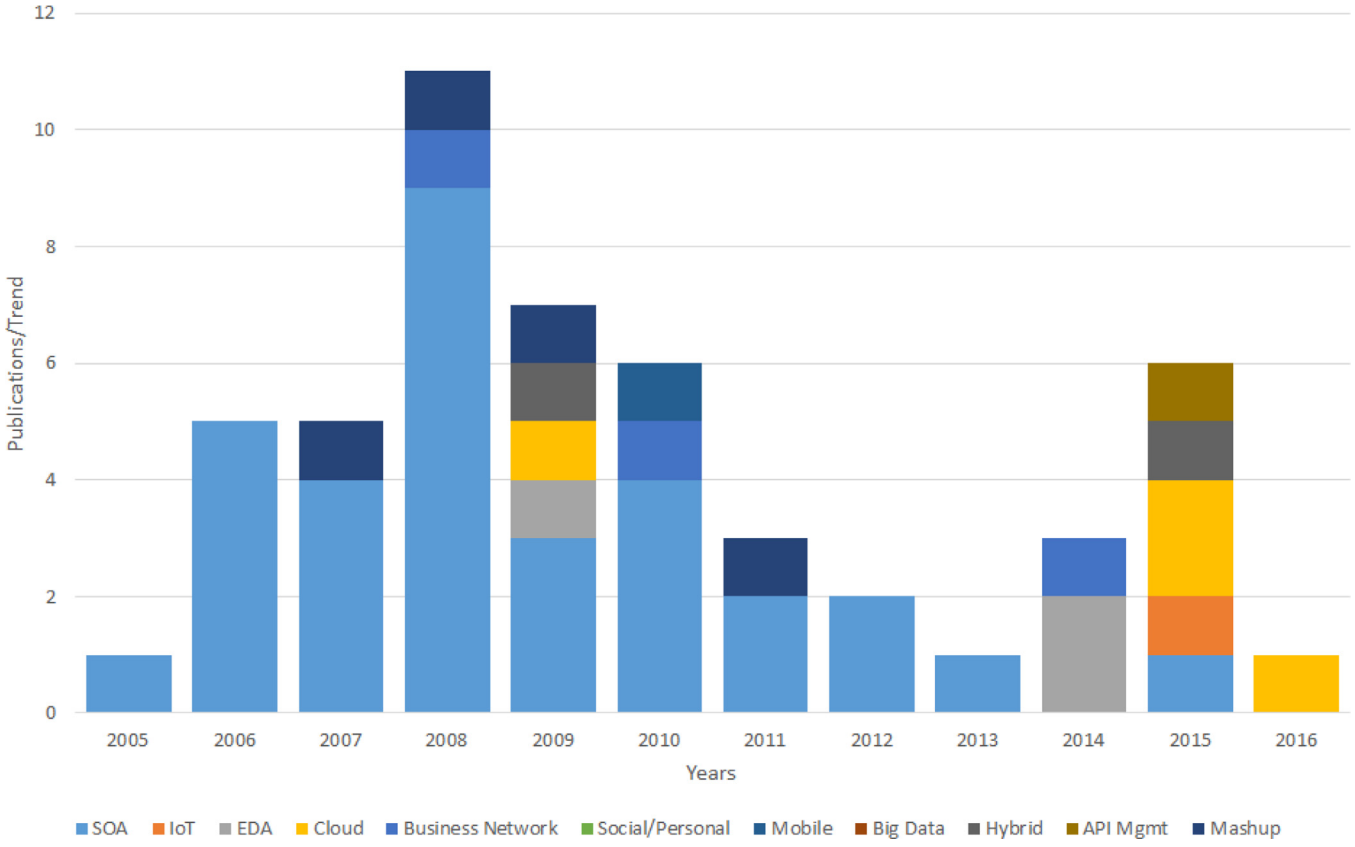


Fig. 4. Distribution of topics mentioned in literature over time.

Table 1
Solutions for trends and non-functional aspects (parentheses mean partial solution).

Trends	Patterns [3]	Formalization [5,66]	Modeling [46,47,66]
Service-oriented Architecture	[10–16,18,32,37,38], security [30,31]	[16], adapters [21,22,35,36,67], control flow [33], interact.[34]	[17]
Internet of Things			
Event-driven Architecture		[42,59]	
Cloud computing	[44,45], (migration [43])		
B2B/ Business Network			(by example [46,47])
Social/ Personal Computing			
Mobile Computing	SOA device patterns [25]		
Big Data			
Hybrid Computing	(migration [43,48])		
API Management	(for SOA [15])		
Mashups	[26,27], SOA migration [29]		
NFA with evidence			
Asynch [3]	EIP [3], strategies [56]	[57,58]	
Security [15,31,45,50,68,69]	(for SOA [30,31])		
Media [6]			
Synch / Streaming [5]			
Conversations [5], [6]	[55], (for SOA [37])	(for SOA [19,34])	
Error Handling [5,45]	(EIP [3,70])		
Monitoring [45,61,62]	([3])		

and argue towards formalized pattern-based reference architecture model to describe SOA concepts [13]. Finally, Zdun describes a federation model to control remote objects and proposes a solution based on patterns, e. g., broker and software patterns [14].

Autili et al. discuss challenges posed by the heterogeneity of Future Internet services [15]. Modern service-oriented applications automatically compose and dynamically coordinate software services through service choreographies described based on BPMN 2.0 Choreography Diagrams. The authors state that currently composition and adaptation is often a manual task, Hence, they advocate towards the automatic synthesis of choreography-based systems and describes preliminary steps towards exploiting Enterprise Integration Patterns to deal with a form of choreography adaptation. Concretely, an adapter generator and prototype using spring integration is presented. Example patterns comprise Message Routing Patterns, namely Message Filter, Aggregator, Splitter, and Resequencer. Overall, this work bridges SOA to EAI using EIP and protocol adapters for services. Moreover, it is planned to integrate EIP with security patterns and message transformation as future work.

In Gacitua-Decar and Pahl an ontology-based approach to capture architecture and process patterns is presented [16]. Ontology techniques for pattern definition, extension and composition are

developed and their applicability in business process-driven application integration is demonstrated. The proposed solution is an architecture framework for SOA-based EAI as well as an ontology-based notion of patterns to link business processes and service architectures. This could be seen as a formalization approach. A SOA service integration framework with a pattern-based modeling approach is presented by Heller and Allgaier [17]. It features controlled extensibility of enterprise systems for unforeseen service integration and can be estimated as similar to related B2B Integration and Enterprise Application Integration. The framework leverages structural or behavioral interface mediation techniques. The modeling approach with adaptation patterns and runtime support is demonstrated with a UI integration prototype in the automotive domain. Overall, this work suggests pattern-based modeling as solution. Kaneshima and Braga analyse whether EAI can be conducted by web services and SOA or DB sharing [18]. Both solutions are being adopted by organizations, although they present advantages and disadvantages that should be analysed. This work documents these problems and solutions in the form of patterns like access via Shared Database, direct RPC-style integration via web services, Intermediate Duplication with access via DB or web services. Hence, the proposed solution is based on SOA and patterns.

Umapathy and Purao transform EIP to web service implementations using a transformation model called ceipXML [19]. The proposed solution comprises conversation models that may be used to implement interactions among Web services as well as a methodology that generates the design elements in the form of conversation policies for Web services. Current integration approaches do not support the end user development requirements for infrequent, situational or ad-hoc integration and collaboration as stated by Zheng et al. [20]. The work differentiates between UI, component, business logic, resource and data integration. Gierds et al. define an approach for behavioral adapters based on domain-specific transformation rules that reflect the elementary operations that adapters can perform; synthesize complex adapters that adhere to these rules [21]. The proposed solution comprises a formalization, specification of the elementary activities to model domain knowledge, separating data from control, and a reduction from adapter synthesis to controller synthesis. An adapter is generated to reconcile mismatches (e. g., incompatible protocols) in Sequel et al. [22]. The proposed solution is constituted by a survey of protocol adapter generation (e.g., semi-automated protocol adapter generation). Gudivada and Nandigam deal with EAI using extensible Web services [23]. A solution is not directly proposed, but rather a practical implementation. Deng et al. combines SOA and Web service technology to simplify EAI by studying the service-oriented software analyzing and development characteristics [24]. The approach distinguishes between vertical integration within an enterprise while B2B emphasize on the horizontal integration. Again the paper presents a more practical implementation.

SOA and Mobile Computing. Mauro et al. [25] target design problems of SOA for mobile devices with Service Oriented Device Architecture (SODA). For this SOA design patterns like Enterprise Inventory are analyzed with respect to their applicability to SODA, and new pattern candidates like Service Virtualization are identified. From these candidates new (device) patterns including Auto-Publishing, Dynamical Adapter, Server Adapter, Integrated Adapter, External Adapter are proposed as solution.

SOA and Mashups. Liu et al. combine several common architecture integration patterns, namely Pipes and Filters, Data Federation, and Model-View-Controller to compose enterprise mashups [26]. Moreover, these patterns are customized for specific mashup needs. In [27] enterprise architecture integration patterns (e. g., Pipes and Filter, Data Federation, Model-View-Controller) are leveraged in order to compose reusable mashup components. The authors also present a service oriented architecture that addresses reusability and integration needs for building enterprise mashup applications. The proposed solutions focus on SOA and mashups, but no solution to EIP and new trends is provided. The work by Braga et al. addresses issues of complexity of service compositions with adequate abstraction to give end users easy-to-use development environments [28]. Abstract formalisms must be equipped with suitable runtime environments capable of deriving executable service invocation strategies. The solution tends towards mashups and modeling as users declaratively compose services in a drag-and-drop fashion while low-level implementation details are hidden. However, the solution could not be clearly identified and is hence not included in Table 1. Finally, Cetin et al. chart a road map for migration of legacy software to pervasive service-oriented computing [29]. Integration takes place even at the presentation layer. No solution is provided for EIP and trends, however, mashups are used as migration strategy to SOA based for the Web 2.0 integration challenge.

SOA Security. Qu et al. present six bilateral patterns (Binding, On-demand, Tailor, Composite, Contract and Migration) and four multilateral patterns (Separated, Shared, Mediated and Enhanced) as a solution for integrating new services with Grid security services [30]. For each pattern, the authors discuss its intent, applicability, participants and consequences. Shah and Patel analyse the security requirements for global SOA [31]. For security concerns, dynamic configuration of handlers, sequence, and identification of handlers is proposed as solution. Fisher et al. provide practical implementations in Java and .NET for interoperable, synchronous, and asynchronous integration [32]. Hence, the proposed solution consists of implementation details for SOA, WS security examples, and best practices such as a secure object handler adding custom interceptor logic for, e. g., adding digital signatures.

SOA and Business Processes. Ouyang et al. formalize process control flows into BPEL processes by an intermediate translation to Petri nets [33]. From the same group, Wang et al. construct and interface adaptation machine that sits between pairs of services and manipulates the exchanged messages according to a repository of mapping rules. For both approaches, the proposed solution is a formalization. Lohmann et al. analyze the interaction between WS-BPEL processes using Petri nets [34]. Again the proposed solution is a formalization. With a similar goal, Kumar and Shan aim at simplifying the pattern compatibility based on a matrix and rules that enable the simplification of checking compatibility between two or more processes because these prerequisite rules can be applied to each pattern separately [35]. The proposed solution is an algorithm and can hence be subsumed as formalization. Mismatch patterns that capture the possible differences between two service (business) protocols to adapt and automatically generate BPEL adapters are presented by Jiang et al. [36]. They introduce several dependencies such as transformation dependency (incl. message split), synchronization dependency, choice dependency (choice among two or more messages), and priority dependency. The proposed solution is the formalization of mismatches. Barros et al. propose SOA process interaction patterns including Send, Receive, Send/Receive, and Racing Incoming Messages [37]. Patterns for synchronization problems in the area of process-driven architectures, e. g., Waiting Activity or Timeout Handler, are introduced by Köllmann and Hentrich [38]. Vernadat looks at architectures and methods to build interoperable enterprise systems, advocating a mixed service and process orientation and the classification of integration levels, physical system, application, business integration, and enumerates SOA concepts [39]. No specific solution is proposed. Grossmann et al. derive integration configurations from underlying business

processes, e. g., activities [40]. Future work names exception handling as challenge, however no solutions are provided.

Event-driven Architecture (EDA) and SOA. Taylor et al. address the SOA - EDA connection as service network and provide a reference EDA manual [41]. As no solution is provided, the approach is not included in Table 1. A theoretical framework for modeling events and semantics of event processing is provided by Patri et al. [42]. The formal approach enables to model real-world entities and their interrelationships and specifies the process of moving from data streams to event detection to event-based goal planning. Moreover, the model links event detection to states, actions, and roles enabling event notification, filtering, context awareness, and escalation. The proposed solution consists of events and formalization.

2.2.2. Cloud computing, business networks, and hybrid applications

The successor of grid and cluster computing is cloud computing that extends B2B to business networks, and the coexistence of applications on-premise and in several kinds of cloud platforms as hybrid applications.

Cloud Computing. Asmus et al. focus on the migration of enterprise applications to the cloud [43]. Integration is considered a key factor influencing cloud deployment. Several migration patterns are described as a basis for enabling enterprise cloud solutions. The following challenges are named in the paper: data volume, network latency, identity and data security management, interoperability (i. e., supporting the trends big data, security, and variety as in multimedia). Asmus et al. state that "integration pattern can be a starting point in deciding integration options" [43]. The key areas addressed in the approach include on premise, off-premise private cloud, cloud integration, cloud service provider, and external users. The integration patterns refer to process to process and data integration. Overall, the proposed solutions are "patterns and processes-based" methods for an initial evaluation of the risk and effort required to move new and existing applications to a cloud service. In Ritter and Rinderle-Ma, a collection of integration patterns derived from requirements of hybrid and cloud applications is presented [44], thus propose a solution for cloud and patterns. The main challenge described by Merkel et al. is a secure integration [45]. The approach proceeds in a top-down manner by deriving integration patterns from scenarios and in a bottom-up fashion by deriving patterns from case study requirements. It identifies the need for security (access control, integrity, confidentiality) as well as security constraints (e. g., EU Data Protection Directive) and presents an evaluation based on an architecture with major focus on hybrid and multi-cloud setups. The described patterns are cross-cloud ESB, usage of ESBs, as well as security patterns as architecture components such as LDAP. The approach only works in private clouds. Merkel et al. propose future work on public cloud that involves content encryption, key management, data splitting, computing with encryption functions, anonymization, data masking, and encrypted virtual machines. They mention Cross-Cloud Balancer, Cross-Cloud Data Distributor, and replication patterns as further future work. Other challenges mentioned are cross-cloud monitoring and cloud management. In summary, the proposed solution are new patterns for SaaS integration and centralized as well as decentralized multi-cloud integration.

Business Network. Ritter provides mappings of EIP integration semantics and patterns to BPMN-based models as well as an implementation of a business network scenario example [46,47]. Both works do not directly propose a solution to the trends depicted in Fig. 4, but introduce *modeling* as a possible solution in the context of EIP, thus added as category to Table 1.

Hybrid Applications. A major challenge in hybrid applications is the decision where to host parts of the application. In this regard, Mansor recommends to bear in mind the patterns in the envisioned process [48]. The work addresses technical challenges when implementing a hybrid architecture. The proposed solution refers to architectural patterns. A holistic approach for the development of a service-oriented enterprise architecture with custom and standard software packages is presented by Buckow et al. [49]. The system architecture to be developed is often based on integration patterns for the physical integration of systems. No solution is provided in the context of this work.

2.2.3. Internet of things and big data

With affordable and widespread mobile sensors and devices comes the Internet of Things and together with the immense amount of data from cloud and mobile computing comes Big data.

Internet of Things (IoT). Heiss et al. collect challenges in cyber-physical systems such as communication quality, interoperability, and massive amounts of data [50]. As interesting requirements they state "placement" (of integration scenarios), e. g., cloud or on-device, the demand for global optimization, more intelligent devices, networking and cloud and security including data security and privacy etc., decoupling of layers vs. direct data access for on-top applications. Rather than proposing a solution, the industrial and business perspectives on such envisioned platforms are described.

Big data. Ritter and Bross suggest moving-up relational logic programming for implementing the integration semantics within a standard integration system [51]. For this EIP semantics is translated to relational logic. For declarative and more efficient middleware pipeline processing (e. g., parallel execution, set-operations), the patterns are combined with Datalog. The expressiveness of the approach is discussed, and a practical realization by example is provided. Although no direct solution to the trends is provided the approach directs to "data-aware" integration patterns.

2.2.4. General EAI approaches

From practical EIP implementations to ideas for new patterns, formalization approaches, enabling techniques and domain-specific work, this section rounds off the literature analysis with further EAI challenges.

Practical Aspects. Scheibler and Leymann present a framework for configuration capabilities of EIP, specifically for code generation based on a model-driven architecture [52]. In [53], EIP are implemented in IBM WebSphere. Again no solution for the trends is provided, but a solution to the EIP through *implementation*. Thullner et al. analyse EIP coverage in open source tools and implement a sample scenario in Apache Camel and Mule [54]. No solution is provided.

EAI Patterns. [55] presents a pattern language for conversations between loosely coupled services, i. e., *patterns* are suggested as solution. Gonzales and Ruggia deal with response time and service saturation issues (more requests than can be handled) using an adaptive ESB infrastructure [56]. They propose solutions in the form of *strategies*, i. e., Delayer, Defer Requests, Load Balancing, and Cache.

Formalization and Verification. Fahland and Gierds present a conceptual translation of EIP into Colored Petri nets, hence providing a formal model based on a system specification using EIP [57,58]. The Petri net based formalizations can be used to simulate and conduct model checking of pattern compositions. Though the

formalization can be understood as solution, it does not address any new trends beyond EIP, thus this approach is not contained in Table 1. A semantic representation of EIP for automatic management of messaging resources (e. g., Channels, Filters, Routers) is presented by Patri et al. [59]. The application is to connect mobile customers to Smart Power Grid companies. Data is accessed in form of alerts from a complex event processing engine using SPARQL queries. The proposed solution is a formalization for resource management of integration patterns. Basu and Bultan focus on the interaction behavior in asynchronously communicating systems resulting in decidable verification for a class of these systems [60]. As the proposed solution (formalization) is not in the context of the trends, it is not included in Table 1. Mederly et al. generate a sequence of processing steps needed to transform input message flow(s) to specified output message flow(s) [61,62]. The work takes into account requirements such as throughput, availability, service monitoring, message ordering, and message content and format conversions. Additionally, it uses a set of conditions, input and output messages, and a set of configuration options. Control flow ordering is formalized. The work is excluded from Table 1 because it provides no solution, but rather creates parts of integration solutions from the description of what has to be achieved, not how it should be done.

EAI enabling techniques. The following approaches address different enabling technologies. However, neither are the presented approaches related to the trends, nor do they propose concrete solutions. Hence they are not included in Table 1. Architectural patterns (e. g., Remote Process Invocation, Batch Data Synchronization, SOA, Pub/Sub, P2P, Broker, Pipes and Filters, Canonical Data model, Dynamic Router) are contributed by Kazman et al. [63]. This work constitutes a guideline for IT architects that combines existing patterns. Land et al. integrate the existing software after restructuring or merger, i. e., address the question of how to carry out the integration process [64]. Multiple case studies and recurring patterns for vision process and an integration process are provided as well. Basic concepts of enterprise architectures including integration and interoperability are summarized by Chen et al. [65].

Domain-specific Approaches. Cranefield and Ranathunga integrate agents with a variety of external resources and services using Apache Camel and the EIP endpoint concept [71]. e-Learning as a growing and expanding area with huge number of disparate applications and services is addressed by Rajam et al. [68]. The approach redefines the Model-View-Controller pattern. It can be further enriched to encapsulate certain non-functional and integration activities such as security, reliability, scalability, and routing of request. As all these approaches do not propose a solution directly connected to EIP and the trends, and hence they are not included in Table 1.

EAI Challenges. A survey to motivate some more challenges in the area of enterprise application integration and to link back to the trends is presented by He and Xu [69]. Further this work examines the architectures and technologies for integrating distributed enterprise applications, illustrates their strengths and weaknesses, and identifies research trends and opportunities for horizontal and vertical integration. Though no solution is proposed, the discovered trends are strengthened, for example, SOA, personal, mobile, and IoT. The survey also addresses NFA, e. g., security, which are collected and serve as input for Table 1. Another survey by Panetto et al. discusses trends and NFA in enterprise integration [66]. Moreover, *modeling* and *formalization* (formal methods such as verification) are proposed as challenges, but no concrete solution provided.

2.3. Synthesis and discussion of non-functional aspects

The second aspect of our analysis of trends are topics that were named by Gartner [6] and Zimmermann et al. [5] as relevant or that were identified during the literature review. However, these topics have a more cross-cutting quality (i. e., relevant for several trends). We call them non-functional aspects or requirements (NFA), which we appended to Table 1 together with the references that supported them as challenges (as evidence). They are set into context to important aspects, when working with integration scenarios, namely patterns, formalization and modeling. The focus on patterns comes from the EIP [3] and supported by many related domains, that capture knowledge and best practices in form of patterns (e. g., SOA, Cloud Computing). Panetto et al. [66] bring up the formalization (supported by [5]) and modeling (supported by [46,47]) as additional relevant topics. We now set these topics into context with the references from the literature analysis in Table 1.

For the EIP, we added asynchronous message processing as Asynch to cover the solutions in this space, e. g., by [3,56]. For the NFA, solutions in the area of formalizations are proposed by [57,58] for the validation of pattern composition and business processes. Another NFA is *Security*, which was seen as challenge at least by Gartner [6] and in the literature by [31,50,68] (in general), by [69] (performance concerns, real-time integration), and by [45] (e. g., safe integration, indications that content encryption, key management and more is missing). Autili et al. [15] mention the need for security integration patterns. The solutions for patterns are limited to SOA with patterns like Secure Service Consumption or Security Handler Information Exchange [30,31].

According to Gartner, multimedia format handling and processing can be seen as a non-functional requirement [6]. This includes image, video and text image formats, which are increasingly produced through mobile devices and, e. g., interacted on social media, becoming of increasing interest for (business) applications.

In the context of the big data challenges of integration systems (from Gartner; e. g., volume, velocity, stability), (synchronous) streaming protocols are seen as one possible solution. The authors of [5] mention that patterns as well protocols are currently missing in EAI.

With more and more communication partners that result from the trends in Section 1.1, (stateful) conversational protocols might be required, according to [5] and also Gartner (e. g., device meshes). First ideas have been sketched by Hohpe [55] with an initial collection of conversation patterns, which should be extended [5]. For SOA web service conversation policies [19] and interaction patterns [37] solutions were provided. Formalizations have been proposed in [34] for the SOA domain with focus on the controllability of a process. The proposed solutions for SOA might be transfered to integration processes as starting point for more general conversation patterns.

To handle erroneous situations during message processing, escalate them and make systems more fault-tolerant, error handling is seen as a major aspect [5,45]. Hohpe et al. [3,70] do only cover Dead Letter Channel as solution and sketch some ideas about the topic. Overall, in the literature, the topic is neither addressed from a pattern, formalization, nor modeling perspective. While [5] mentions missing patterns and formalization, Merkel et al. [45] lists Balancing and Distribution, as well as [69] mentions Fault-tolerance and Message Scheduling as missing aspects. Similarly, the insight into the current state of affairs, called monitoring, for services and cross-cloud are seen as important topics in [45,61,62].

The monitoring of integration processes as well as cross platform monitoring were only mentioned, however, no solution was provided. The Control Bus, a Wire Tap and the Message History patterns in Hohpe et al. [3] denote partial solutions, which can be used to build a monitoring solution on integration process level.

Table 2
System review - horizontal search.

Category	hits	selected	Selection criteria	Selected Systems
Commercial	12	7	Gartner and Forrester IPaaS Quadrants	Dell Boomi [72], IBM Cast Iron [73], Informatica [74], Jitterbit [75], MS BizTalk [76], SAP Cloud Integration [77], Oracle [78]
Startup	20	2	cloud/data integration, B2B, API, #followers	Tray.io [79], Zapier [80]
Open Source	13	2	application integration, data ingestion	Apache Flume [81], Apache Nifi [82]
Wikipedia	34	1	enterprise application integration; non-duplicates	Apache Camel [83]
Added Systems	n/a	3	expert knowledge	Cloudpipes [84] (startup), Tibco [85], WebMethods [86] (commercial)
Removed Systems	–	–		
Overall	74	15		

3. System review

This section reports on the results of a system review to answer **hypotheses H2** *the EIP of the 2004 book are all widely used in praxis*, and **H3** *current system implementations do support more patterns* as set out in Fig. 3. The system review is based on the guidelines described in [9] for a horizontal search including "well-established" commercial application integration systems, more experimental systems from startups, open source systems and public knowledge in form of a Wikipedia search. The selection of systems was conducted on 2016-10-04, and the results of the horizontal search are summarized in Table 2. The NFA are used to focus the search in those systems.

First, seven commercial systems were collected by taking the systems listed in both, the Gartner (Leaders, Visionaries, Challengers) [87] and the Forrester (Application Integration) IPaaS list [88] – out of 12 systems, leading to the following systems: Dell Boomi [72], IBM Cast Iron [73], Informatica [74], Jitterbit Harmony Cloud Integration [75], Microsoft BizTalk [76], SAP Cloud Integration [77], and Oracle Cloud Integration [78]. We scratched MuleSoft due to its similarity to Apache Camel [83], which we selected as expert addition from Wikipedia (discussed later).

In addition, two startup systems from the top 20 overall systems were selected due to their number of followers on `angel.co`,[1] namely Tray.io [79] and Zapier [80]. While the former is striving to build an "Integration Marketplace" for enterprise applications, Zapier is a cloud integration startup.

Out of 13 open source systems of the Github Hadoop Ecosystem,[2] we selected Apache Flume [81] and Nifi [82] as data ingestion systems according to the selection criteria (cf. Table 2). We scratched the application integration systems Talend (also listed as commercial system), Spring Integration and MuleESB for their similarity to Apache Camel as well as Apache Beam, Apache Sqoop and Spring XD for their similarity to Apache Flume.

The open source integration system Apache Camel [83] does not appear in the open source list, however, it was the only non-duplicate from the other lists that has to be selected, since it implements the existing EIP from [70] and is a role model for many systems like Spring Integration, or Red Hat's FuseESB.

The software systems of Tibco [85] and Software AG [86] are wide-spread and influencial integration systems for on-premise with a cloud integration offering and are listed among the top for wide integration and deep integration for traditional on-premise by Forrester.[3] Hence we add them as expert selected additions. We add Cloudpipes [84] from the startup list as cloud integration system.

That leaves us in total with 15 systems with a good mix of well-established commercial and startup products, as well as community projects. Since the main focus lies on commercial systems that are known to be less well accessible for a systematic analysis of their features, we focus on the publicly available material (i. e., without registration or login) and try to get more information by possibly underlying open source systems, where possible.

3.1. Processing of selected systems

3.1.1. EIP Solutions used in system implementations

We start our system review with an analysis of all selected systems with respect to their implementation of EIP solutions. The EIP describe six pattern categories, namely, Messaging Channels, Message Construction, Message Routing, Message Transformation, Messaging Endpoints and System Management. We focused the analysis on the two pattern categories of message routing and transformation, since they represent the core aspects of integration systems. Furthermore we left out composed patterns (e. g., composed message processor, scatter-gather), when their single parts were already in the selection. Table 3 (from Boomi to Oracle) and Table 4 (from Flume to Webmethods) depict the solutions found in the system implementations that could be associated to the routing and transformation patterns.

The Apache Camel system seems to be specifically designed around the EIP, thus supports nearly all EIP and sticks to the original EIP naming for the respective solutions, which makes it a benchmark for the others. Most notable deviations are the Envelope Wrapper (i. e., wrap application data inside an envelope, compliant with the messaging infrastructure) and Message Translator patterns (i. e., translate one data format into another one; not in transformation patterns). None of them is directly represented in Camel, however, can be implemented using UDFs (i. e., user-defined functions like Camel Processor) or scripting (e. g., Camel Script), therefore marked as partially covered by parentheses.

The most common routing pattern solutions are the Content-based Router, the Splitter and the Aggregator. Since the Message Filter is a special case of the content-based router and filter can be used to construct the latter, not all systems provide implementations for both of them. The splitter is sometimes implemented according to the description in the EIP, however, some vendors decomposed it to its iterative core functionality (e. g., For Each in IBM, Oracle, Cloudpipes). The aggregator shows many partial solutions that require user-defined functions (e. g., Informatica, Oracle, Tray.io), while only few provide its EIP functionality (e. g., Aggregator in BizTalk, SAP Cloud Integration, Tibco or ContentMerge in Apache Nifi).

The transformation patterns seem to play a major role in the analyzed systems, since most of them are broadly supported. However, there seems to be a tendency to provide UDF capabilities and leave the burden to the user to deal with the semantics.

Finally, the dynamic routing patterns (e. g., Dynamic Router), those patterns that contain the recipient in their content (e. g., Recipient List, Routing Slip), and the Message Resequencer, e. g., used for the exactly-once-in-order service quality [89], were

[1] Angel.co, visited 02/2017: https://angel.co/data-integration

[2] Hadoop Ecosystem on Github, visited 02/2017: https://hadoopecosystemtable.github.io/

[3] The Forrester Wave: Hybrid2Integration, Q1 2014

Table 3
Original EIP used in systems; supported √, partial (√), unknown/not supported.

Pattern	Boomi	IBM	Informatica	Jitterbit	BizTalk	SAP	Oracle
Content-based Router	√	√	–	–	√	√	√
Message Filter	–	√	–	–	√	–	–
Dynamic Router	–	–	–	–	√	–	–
Recipient List	–	–	–	–	–	–	–
Splitter	√	√	(√)	√	√	√	√
Resequencer	–	–	–	–	–	–	√
Routing Slip	–	–	–	–	–	–	–
Aggregator	–	–	(√)	–	√	√	(√)
Envelope Wrapper	(√)	(√)	(√)	(√)	–	–	–
Content Enricher	(√)	–	–	(√)	(√)	√	–
Content Filter	(√)	–	–	(√)	(√)	√	–
Claim Check	(√)	–	–	(√)	–	(√)	-
Normalizer	(√)	–	(√)	√	(√)	(√)	–
Message Translator	√	–	(√)	√	√	√	–

Table 4
Original EIP used in systems; supported √, partial (√), unknown/not supported.

Pattern	Flume	Nifi	Camel	Tray.io	Zapier	Cloudpipes	Tibco	Webmethods
Content-based Router	(√)	√	√	√	–	√	√	√
Message Filter	√	–	√	–	√	√	–	–
Dynamic Router	(√)	–	√	–	–	–	–	–
Recipient List	–	–	√	–	–	–	–	–
Splitter	√	√	√	–	√	√	√	–
Resequencer	–	–	√	–	–	–	–	–
Routing Slip	–	–	√	–	–	–	–	–
Aggregator	(√)	√	√	(√)	–	–	√	–
Envelope Wrapper	–	–	(√)	–	–	–	–	–
Content Enricher	(√)	√	√	√	(√)	(√)	√	√
Content Filter	(√)	√	√	(√)	(√)	(√)	√	√
Claim Check	–	–	√	–	–	–	(√)	–
Normalizer	(√)	–	√	(√)	(√)	(√)	√	√
Message Translator	(√)	√	(√)	√	(√)	(√)	√	√

sparsely implemented. This leaves the question on their relevance or other components that take over their function.

Summary. While some of the EIP like Content-based Routing or Message Filter, Splitter and Content Enricher can be found in most of the systems, others are rarely implemented (e. g., Resequencer, Routing Slip). The analysis of these patterns and their relevance are left for future work, and thus not analyzed further.

3.1.2. New solutions not covered by system implementations

We now analyzed the collected systems with respect to their functionalities according to the harvested NFA from Section 2.3), namely security, media, streaming or more abstract "processing", conversations, error handling, and monitoring. Comparing the NFA with the collected system functionalities, while neglecting functionalities covered by the EIP, gives an answer to the question which topics are required and used in addition. Hereby, the system functionalities represent an implemented solution as part of an actual integration system.

Fig. 5 depicts found solutions not covered by the EIP by NFA and system vendor. During the analysis new NFA were identified – not mentioned by Gartner, the EIP authors, or the literature – that seem to play a role in practical terms: stateful integration processes using storage, (pattern) composition (mentioned in Zimmermann et al. [5]), and system operations. These three new NFA were included into the analysis of the other systems as well. All non-related topics are collected as miscellaneous (Misc).

Notably, all identified NFA are at least partially covered by system implementations, indicating that solutions in form of conceptual definitions are required (e. g., as patterns). According to Mulesoft [90], the major challenges in cloud integration systems are security and management. The management includes error handling and monitoring, which allow to control the behaviour of the integration scenarios.

The classic application integration addresses the variety problem for textual message formats [2]. With the availability of integration systems for "everybody" (e. g., in form of a cloud integration system) non-textual formats gain importance.

The trade-off between stateful and stateless message processing is represented by storage capabilities in integration systems, for which nearly all vendors propose a solution and conversations. The stateful approaches could be represented by conversational protocols, which allow to move the state from the integration systems to the communication partners (idea sketched in [55]). Most of the service qualities (e. g., at-least-once, exactly-once processing) [3,89] require stateful integration processes. Consequently, this would require changes in the applications. Current systems provide only rudimentary support, if at all.

Finally, a broad variety of miscellaneous topics was collected, e. g., sentiment analysis, natural language translators, but also more general functions like sort, loops, as well as explicit format handling, i. e., marshalling and type conversion.

Summary. Notably, security and error handling (and monitoring) capabilities are predominantly found. They address the challenges of security and management. Furthermore, solutions for the increasing variety of message formats (cf. media) as well as the volume and velocity handling can be found in the systems are part of new processing types. The storage of data and message semantics like quality of service are relevant for integration scenarios. This leads to the trade-off between stateful vs. stateless integration processes, which briefly address in Section 5. The (stateful) conversations, which could be part of a solution, are currently under-represented.

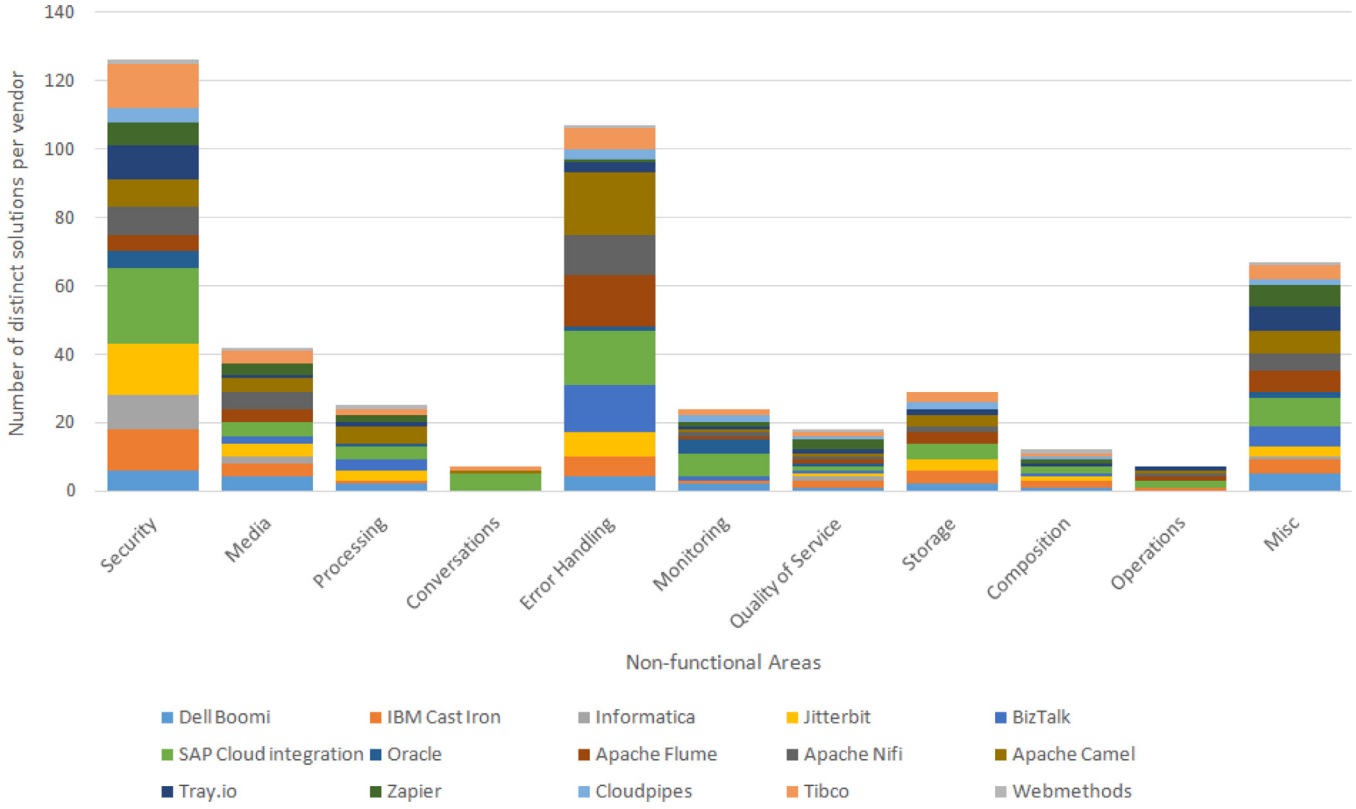


Fig. 5. Solutions for NFA not covered by the EIP by system vendor.

3.2. System summaries along NFA

3.2.1. Security

The aspect of confidentiality or message privacy is solved on transport, message and storage levels. The transport level channel encryption can mostly be specified in the inbound and outbound adapters in form of the transport protocol (e. g., HTTPS, SFTP) and guarantees that the message cannot be read during transmission (e. g., Jitterbit's Transmission Protection). Once, the message is received by the inbound adapter and handed to the subsequent operation in the integration process, message privacy can be applied or reversed. Therefore many vendors provide explicit message encryptors and decryptors (e. g., PGP Encrypt and Decrypt from Dell Boomi, AES_ENCRYPT from Informatica or Encrypt / Decrypt in Apache Nifi), or encrypting adapters (e. g., FileProcessorConnector in Informatica, FileChannel in Apache Flume, WSSProvider in Tibco). The encrypted storage of messages helps to protect the message's privacy in the store, e. g., can be configured in SAP's DBStorage and Persist operations. The configuration of the message privacy solutions mostly include encryption algorithms, key lengths and certificates. Similarly, the integrity and authenticity of a message can be ensured on the different levels. Most of the vendors provide configurations for safe and authenticated transport (e. g., using user and password, certificate or token-based authentication). The transport is considered safe if changes of the message can be recognized by the receiver and the authenticity guarantees that the sender is the expected one. For instance, most of social media endpoints like Twitter and Facebook use token-based OAuth authentication. In addition, many vendors provide explicit message signers and signature verifiers (e. g., Digest/Hash function in IBM, Signer and Verifier in SAP Cloud Integration) as well as safe message storage is provided, e. g., by Jitterbit or SAP Cloud Integration. For the storage, the authenticity seems to be implied, since the cloud platform message or data store is used. The availability of integration scenarios is not only a stability, but also a security concern. Therefore some vendors like IBM, SAP Cloud Integration provide implicit countermeasures, e. g., redundant message stores with high availability and disaster recovery, as well as Apache Flume with explicit MorphlineSolrSinks and Kafka Channel configurations. Finally, changes to the message are tracked for auditing purposes. This is made explicit as Audit Trails in Jitterbit and Oracle or Service Auditing in WebMethods.

3.2.2. Media

The literature review in Table 1 shows that there are no solutions for multimedia processing in application integration or related domains (e. g., SOA, EDA). The system analysis does only provide few, superficial solutions. For instance, textual representation of binary content is explicitly configurable in most of the systems (e. g., Base64 Encode / Decode in Dell Boomi and IBM, Encoder / Decoder in SAP Cloud Integration). These encodings play a major role when communicating with remote applications, but also when calling services (e. g., user-defined operations) using textual message protocols. In addition, most of the vendors allow user-defined operations in form of scripting capabilities (e. g., Script, Processor in Apache Camel, Expressions in Informatica). With that, more complex operations can be performed like the compression of – usually bigger – multimedia messages. Despite that, pre-defined compression operators can be found in, e. g., Dell Boomi, Jitterbit, Apache Nifi, which allow to configure the compression type (e. g., zip). The explicit support of scripting seems to be a general trend, when representing transformation patterns (cf. Section 3.1.1). This could either mean that the implementations are too diverse to formulate a general solution or indicate that the topic was not considered yet. The support of explicit image processing operations seems to be limited to Nifi's ResizeImage and ExtractImageMetadata functions as well as IBM's Read MIME activity. The only real multimedia operation is the image resizing,

since the metadata simply provide a format specific capture of the defined image's meta tags.

3.2.3. Processing

While the literature review does not show solutions for message processing, especially not for "data-aware" or Big Data processing, the systems implement solutions. The canonical solution for processing larger amounts of data is to scale-out to multiple processing units, constituting parallel subprocesses. The parallel processing of one message in subprocesses using a broadcast can be done, e. g., in BizTalk with Create concurrent flows, SAP Cloud Integration Gateways, or Apache Camel Multicast. Furthermore, the explicit configuration of parallel processing within an integration process (i. e., not process parallelization) is supported by, e. g., Dell Boomi using the Flow Control properties, Jitterbit Parallel Processing, Tibco Non-inline subprocesses and Critical Section, BizTalk Scope batch property. Alternatively, a more data-centric approach, however, impacting the latency of the process, is micro-batching [91]. Vendors like Dell Boomi and Jitterbit also support batch processing of messages using the Flow Control properties or Chunking configurations. The processing of message streams allows the system to handle larger amounts of data than the integration system resources would allow. This more connection oriented approach was identified by Zimmermann et al. [5] as missing pattern category in the context of synchronous message processing. An explicit streaming support is provided, e. g., by Jitterbit Streaming Transformation and Apache Camel. However, not all integration operations or adapters are (conceptually) capable of streaming.

3.2.4. Conversations

Gartner [6] as well as Zimmermann et al. [5] mention the importance of conversations for messaging. These conversations are similar, however, stand in contrast to the choreography (e. g., [15,92]) and interaction patterns for services [37] in SOA because they denote more complex tasks than sending and receiving data or messages. They target complex (stateful) conversations as partially covered in [55]. Some of the systems allow a timed redelivery of messages in a non-error case (e. g., SAP Cloud Integration, Apache Camel). This feature is similar to the Contingent Requests pattern in [37]. For a conversation, an acknowledgement mechanism would be required similar to [55]. One technique of reducing the number of requests to an endpoint is request caching. In Tibco, request caching can be configured by specifying time slices and operations in the Caching Stage. The SAP Cloud Integration system allows to map synchronous to asynchronous communications and vice versa. This becomes necessary when the endpoints' message exchange mechanisms do not fit.

3.2.5. Error handling

Error handling is a crucial aspect of integration scenarios [5] for the control and fault tolerance aspects. In the literature we found solution attempts [3,70] like the Dead Letter Channel pattern for the collection of failing messages, while the systems implement various, more sophisticated solutions. The fundamental topic for dealing with errors in integration scenarios is the handling of exceptions. Therefore, most of the systems provide a "catch-all" capability (e. g., Catch All in IBM), which sometimes even come with an exception subprocess for more advanced handling (e. g., Exception Subprocess in SAP Cloud Integration). In addition, vendors like Dell Boomi, IBM, SAP Cloud Integration and Tibco provide more fine-granular scoping of exception handlers, e. g., down to the single operation. More advanced topics include escalation, fault-tolerance and eventually prevention techniques. Most notably, the systems support escalation mechanism like (partial) abortion of complex processes (e. g., incl. parallel processing) and raising indicators for alerting, as well as message redelivery on exception for tolerance, and message validation, load balancing (cf. [56]) and flow control to prevent errors. More recent work [93,94] – not found in the literature review – covers all of the found system solutions as patterns and shows their composition. Furthermore, it introduces the concept of compensations (e. g., for undo operations), which was not found within the reviewed systems.

3.2.6. Monitoring

The monitoring of integration scenarios gains importance especially within integration platforms hosted by a third party and across those platforms in cloud and mobile computing. The major monitoring aspects found in the systems can be distinguished into UI components that show important aspects of the system, called monitors, and a rather event-based registration on instance level. For example, Dell Boomi supports message change events by Find Changes, which can be extended to Field Tracking. Oracle supports the latter with Business Identifiers. That means, user-defined events on technical and business level can be tracked via conditional events. Examples for monitors can be found in most of the systems across all parts of an integration scenario (i. e., from adapter or channel, over component, down to message monitors). The monitors can be fed by built-in and user-defined message interceptors (e. g., in Apache Flume and Camel), which allow scenario specific monitoring. When integrating hybrid applications, most systems provide central, cloud monitors instead of local ones.

3.2.7. Storage

An integration system requires persistent stores and queues to be operable, e. g., for system management and monitoring. In addition, message delivery semantics (e. g., reliable messaging like "exactly-once-in-order") [89], secure messaging, and legal aspects (e. g., "Which messages were received and processed?") must be ensured. In the literature only simple messaging related storage are mentioned like the *Message Store* [3], for storing messages during processing, and the *Claim Check* [3] to store (parts of) the message during processing and re-claim them later. Consequently, several system vendors identified the need for additional storage capabilities, summarized to data stores and their access (e. g., DB Update in Jitterbit, DBStorage in SAP Cloud Integration), as well as memoization and caching during one instance of a scenario or between them (e. g., Add to Cache in Dell Boomi, Shared Variables in Tibco, Global Variables in Jitterbit). For secure messaging, security related storage like the *Key Store* (e. g., in Apache Flume, Camel and SAP Cloud Integration), for storing certificates and secure key material, and the *Secure Store* (e. g., in SAP Cloud Integration), for storing secure tokens, users, and passwords, can be found.

3.2.8. Composition

In Zimmermann et al. [5] the composition of EIP is mentioned as one of the missing pieces. Many system vendors, e. g., Dell Boomi, IBM, BizTalk, SAP Cloud Integration, allow subprocess modeling as well as delegation from the main integration process. One important solution are integration process templates, which are configurable re-use processes. Many of the vendors support them, however, under different names, e. g., Template integration process in IBM, Snapshot of Jitterpack in Jitterbit, Blueprint in Cloudpipes.

3.2.9. Miscellaneous

The most notable, specific features are explicit or implicit loops, keyword search and replace as well as content sort and format handling. The implicit loop configurations include While Loop activity, e. g., in IBM, Looping in BizTalk, and the Loop Collection connector in Tray.io. Explicit loops are possible in most of the systems by back-references in the process. Dedicated search and replace functionality is provided, e. g., in Dell Boomi, Jitterbit, and Apache Nifi. While most type converters are implicit in most

Table 5
New integration patterns for NFA in the context of system implementations from *security* to *processing* without pattern solutions already covered by literature.

Category	Scope	Pattern name	System examples
Security	Confidentiality, Privacy	Message Encryptor, Message Decryptor, Encrypted Message	PGP Encrypt / Decrypt [72], AES_ENCRYPT [74], Encrypt / Decrypt [82]
		Encrypting Endpoint / Adapter	FileProcessorConnector [74], FileChannel [81], WSSProvider [85]
	Authenticity, Identity	Message Signer, Signature Verifier, Signed / Verified Message	Digest/Hash [73], Signer, Verifier [77]
	Storage	Encrypted / Encrypting Store	DBStorage, Persist [77]
		Safe Store	Most of the vendors
		Redundant Store	MorphlineSolrSinks [81], implicit [73,77]
	Transmission	Encrypted Channel	Transmission Protection [75]
		Safe, Authenticated Channel	Password, certificate, token-based (Most of the vendors)
		Audit Log	Audit Trails [75,78], Service Auditing [86]
Media	Format	Type Converter	Type Converter [79,80,83]
		Encoder, Decoder	Base64 Encode / Decode [72,73], Encoder / Decoder [77]
		Marshaller, Unmarshaller	"Data Format"[83], "ConvertJSONToSQL" [82], "JsonXMLConverter" [77]
		Compress Content, Decompress Content	implicit [72,75], Compress Content [82]
		Custom Script	Script, Processor [83], Expression [74]
		Metadata Extractor	Read MIME activity [73], ExtractImageMetadata, ExtractMediaMetadata [82]
	Unstructured	Image Resizer	Image Resizer [82]
Processing	Synch / Streaming	–	Streaming transformations [75], partially [77], streaming [83]
	Parallel	Parallel Multicast, Sequential Multicast	[76,77,86]
		Join Router	implicit [83], join [77]
	Other	Delegate	Process Call [77], Direct-VM [83]
		Loop	Loop Activity [73], Looping [76]
		Find and Replace	Search/Replace [72], Control Character Replacer [75], Scan Content [82]
		Content Sort	Sort [83]

systems, marshalling support is made excplicit, e. g., in Jitterbit, SAP Cloud Integration, and Apache Nifi. More "exotic" functions are text sentiment analysis in Cloudpipes, an Archiving activity in IBM, and a Yandex language translator in Apache Nifi.

4. Design of pattern catalog

This section summarizes the findings of the literature and system reviews in form of a pattern catalog, capturing and describing the found ad-hoc solutions and functionalities as new patterns. These patterns can be seen as the starting point of a continuation of the EIP, but also recent trends to express domain knowledge as patterns [95]. In doing so, **hypothesis H3** "Some trends are handled in an (yet) immature and ad-hoc fashion" is targeted. The design goals for the pattern catalog are:

1. Comprehensiveness, i. e., coverage of system implementations that are not in the literature
2. Novelty, i. e., literature coverage of the missing or only partial pattern definitions for NFA

The proposed pattern catalog is summarized in Tables 5 and 6 categorizing the patterns by NFA as *Category*. The patterns in column *Pattern Name* are further grouped by sub-categories as *Scope*. Due to lack of space, the descriptions of all patterns contained in the catalog are provided as supplementary material [44] and two of them are introduced in detail in Section 5. While in this section we focus on patterns, the supplementary material illustrates the modeling of the new patterns for two integration scenarios from the quantitative analysis in Section 6.

4.1. Goal 1 (comprehensiveness): system implementation coverage

In detail, comprehensiveness is evaluated by comparing the coverage of patterns with the NFA that are not or only partly covered by patterns in the literature, represented by the combination of columns *Category* and *Scope*. The coverage of system implementations reflected by column *System Examples* was chosen in order to provide pattern definitions referring to examples (but not all) of the corresponding system implementations (if at least one vendor supported them). Subsequently, we refer only to the categories that have a special relevance for the comprehensiveness of our analysis.

Security. Take, for example, NFA *Security* in combination with scope *Confentiality* (cf. Table 5), for which no comprehensive pattern is provided in the literature on the one side, but system implementations by, for example, Dell Boomi [72], Informatica [74], or Apache Nifi [82] exist. Addressing design goals 1) and 2) led to the set of suggested patterns *Message Encryptor, Message Decryptor* and *Encrypting Endpoint. Message Encryptor*, for example, covers the system implementations *PGP Encrypt / Decrypt, AES_ENCRYPT, and Encrypt / Decrypt*.

Media. Besides formatting patterns for structured message content, the media specific patterns for unstructured content are under-represented in current system implementations, since there is only one pattern with direct relation to multimedia processing (e. g., Image Resizer [82]). Although there are functionalities for the work on the structured multimedia metadata (e. g., Metadata Extractor), further research should target the unstructured multimedia data and processing (e. g., in the context of synchronous, streaming protocols).

Summary – Comprehensiveness. With the pattern catalog we address 94.74% of the NFA scopes or subcategories (i. e., all but 1 out of 19) derived from system implementations. A synch / streaming pattern elicitation – as also mentioned by [5] – was not conducted in the context of this work, since the system review did not lead to pattern changes or new patterns, but only adds an additional processing style. However, we consider this an interesting topic especially in the context of the Media and Big Data trends, and propose a separate study for this current gap.

Table 6
New integration patterns for NFA in the context of system implementations from *conversations* to *composition* without pattern solutions already covered by literature.

Category	Scope	Pattern Name	System Examples
Conversations	Endpoint	Commutative Receiver	–
		Timed Redelivery until Acknowledge	–
	Fault tolerant	Timeout synchronous request	–
		Failover Request Handler	Failover Client [81]
	Resources	Request Collapsing	–
		Request Partitioning	–
Monitoring	Processing	Message Cancellation	[76,82]
		Usage Statistics	[77,78]
	Immediate Insights	Raise Indicator	[72,75–77]
		Detect	[76,82]
		Message Interceptor	[81,83], implicit [77]
	Monitors	Component Monitor	[77,84]
		Channel Monitor	[77,78,80,84,85]
		Message Monitor	[77–79]
		Resource Monitor	[77,85]
		Circuit Breaker	[83]
		Hybrid Monitor	[77]
Storage	Data, Variable	Data Store	[73,75,77]
		Store Accessor	DB Update [75], DBStorage [77]
		Transient Store	Add to Cache [72], Shared Variables [85], Global Variables [75]
	Security	Key Store, Trust Store, Secure Store	[77,81,83]
Composition		Integration Subprocess	[72,73,76,77]
		Integration Process Template	Template Integration Process [73], Snapshot [75], Blueprint [84]

4.2. Goal 2 (novelty): literature coverage

Now, we set the pattern findings from the literature review for the NFA – summarized in Table 1 – into context with the new pattern proposals derived from the system analysis. We exclude the solutions from the original EIP [3], and Media, Synch / Streaming and Composition (not in NFA, however, came up during system analysis and mentioned in [5]), for which no solutions were found in the literature. In addition, we excluded Error Handling, since it is comprehensively covered from a pattern perspective and compared to system implementations in prior work [93,94] (not found in the literature review).

Security. Although some security patterns were proposed in the SOA domain [30,31], they only provide partial solutions with respect to the NFA and no solution in the context of the system review.

Conversations, Processing. In terms of conversation patterns, the system implementations only showed basic support (cf. Table 6), however, some more can be found in the literature, showing that this is an area for integration systems to add more features. Although, Barros et al. [37] mostly reiterate the original EIP, there are few patterns that are new in the context of the system implementations. In the category of multi-lateral communication, the One-from-many pattern [37] is a special case of our more general Join Router that we found in the system implementations (e. g., Apache Camel, SAP Cloud Integration). The One-to-many send pattern [37] is similar to the (parallel) Multicast – found in the systems (e. g., Apache Camel, SAP Cloud Integration), however, some systems have variants that we captured as Sequential Multicast, which routes messages of the same type to multiple receivers in sequence to guarantee the successful delivery to all recipients.

Summary – Novelty. From the functionality required by system implementations, 59 distinct, new patterns are derived that were not found in the analyzed literature. However, for 5 out of 7 NFA (compare to Table 1), the literature indicates missing patterns as research challenge (cf. Section 2.3), thus supporting the extension of the integration pattern catalog for security [15,31,45,50,68,69], multimedia [6], synch / streaming [5], conversations [5,6], monitoring [45,61,62], and pattern composition (from system review Section 3, [5]). In addition, the system review raises a demand for storage patterns that was not mentioned in the literature.

4.3. Solutions for future challenges

We propose several new conversation patterns, of which none was found in the system implementations. The proposed endpoint-specific patterns Commutative Receiver and Timed Redelivery Until Acknowledgement (similar to the Contingent requests pattern in [37,55]) – that together denote a solution for a critical trade-off for scalability inspired by [95] – are discussed in detail in Section 5. The other patterns are further discussed in the supplementary material [44], and target the additional conversation scopes: Fault tolerance and Resources. The multi-tenant processing, conversation patterns (e. g., Cross Scenario and Cross Tenant) patterns that are mostly required in hybrid and cloud computing setups, are already covered by prior work [89], thus not shown.

Toward a more stable system, the Timeout Synchronous Request and Failover Request Handler patterns are improving the fault-tolerance of the messaging. Especially in the Big Data context, the resources of an integration scenario or platform become crucial for their stability. Therefore, the Request Collapsing pattern reduces the number of requests within a conversation. In addition, the Request Caching reduces the amount of duplicate requests to the same endpoint, while Request Partitioning optimizes requests to endpoints and confines errors to one request aspect. Together with other patterns from the literature review and the proposed patterns in this work, we see a clear evidence for further research required. Since none of the patterns was found during the system review this indicates potential for current integration system and application endpoint implementations.

The monitoring of integration scenarios reaches from real-time, scenario-specific processing to near-real time monitors. One further challenge – also identified by [45] and partially covered by the systems with mixed on-premise and cloud integration – is the monitoring across different platforms (e. g., cross-cloud, across on-premise and cloud).

5. Example integration pattern realization

As example from the catalog, we selected patterns related to the non-trivial trade-off between stateful and stateless integration processes (inspired by cloud computing challenges [95]). Especially the system review shows that all vendors provide extensive storage capabilities – beyond the EIP, leading to stateful processes. However, the under-represented conversation patterns could offer an alternative, thus allowing for stateless processes. While stateless processes have scalability benefits, they come with some drawbacks that have to be considered. We selected this trade-off due to its relevance in the context of Big Data, its relevance for Cloud Computing, and because it addresses one well-represented (i. e., storage) and the currently under-represented, but important area of (stateful) conversations. Subsequently, we describe the trade-off as problem description, discuss a suitable pattern format and conclude with two pattern descriptions and their realization.

5.1. Problem description: stateful vs. stateless integration processes

Operating an integration system requires persistent stores and queues, e. g., monitoring, key or secure store to achive security, or auditing for legal reasons. In addition, transactional message processing (e. g., aggregator pattern) as well as message delivery semantics (e. g., reliable messaging like “exactly-once-in-order”) [89] require some persistent state. While the system operability avoids influencing the message processing by not using shared states between integration scenario instances, the transactional processing and message delivery semantics of the stateful message processors (i. e., patterns) usually require shared states. For example, when a stateful aggregator – as part of a scenario instance – processes a sequence of messages, a second scenario instance could be used to distribute the load. However, in the absence of “process stickiness” (i. e., messages of one sequence are only sent to one instance), the stateful aggregator in the second instance has to be able to complete a sequence the other instance started, thus shared state. Hence, the shared states imply complex state handling across integration processes in compute clusters or cloud environments, and this may have a negative impact on their scalability. Alternatively – following the ideas on (stateful) conversation patterns from Hohpe [55] – some of the discussed messaging related storage and message delivery semantics could be moved to “smart” message endpoints (i. e., applications), which already have a persistent state, thus making the integration processes stateless.

For example, Fig. 6 illustrates the trade-off between Exactly-once In Order (EOIO) delivery semantics within the integration scenario (i. e., requires a stateful Message Store, Resequencer and Idempotent Receiver [3], and transactional Message Redelivery on Exception [94]) in Fig. 6(a) and as a (stateful) conversational approach in Fig. 6(b). The integration processes are represented in BPMN 2.0 similar to [46]. An EOIO delivery requires a transactional redelivery in case of an exception, a message ordering step according to a sequence of messages in form of a Resequencer and an Idempotent Receiver, which is able to deduplicate the messages. Fig. 6(a) depicts the instance spanning state for the retry and the resequencing. To avoid stateful integration process, both capabilities can be moved to the endpoints (cf. Fig. 6(b)). While this will not work for legacy, packaged applications, it results into an improved scalability within the integration process and moves the resequencing decision to the receiver. To eventually stop sending, the sender – redelivering the message periodically – requires a stop event (i. e., an acknowledgement) from the receiver.

The solution’s trade-off are the several messages that are sent by the receiver until an acknowledgement is received, while being able to process all messages in parallel using stateless integration process instances. In other words, the performance improvements gained through better scalability and lower latency of the conversational approach – by not waiting for the failure of a sent message – is contrasted by the more resources overall required in case of many failures.

5.2. Patterns and pattern formats

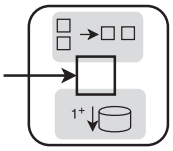
To formalize the new challenges and the resulting, required capabilities within an integration system, thus coming to less immature and ad-hoc solutions (cf. H3), we propose to express them as patterns. Similarly, expert knowledge and best practices were already collected for software design by Gamma et al. [96], EIP by Hohpe et al. [3], and recently for cloud computing patterns by Fehling et al. [95]. For a suitable pattern representation, we compare their pattern formats in Table 7, and select common categories for our proposal.

From the analysis of several known pattern formats in Table 7, we selected: name, icon, driving question, context, solution, results, and known uses to round-off the description. We leave out the separate categories of forces (i. e., problem constraints) and implementation (i. e., pattern variants), which we include into the selected context and known uses categories, respectively. The pattern descriptions in the supplementary material [44], add a *Data Aspects* category (not further discussed here), which gives even more insight into the configuration of the pattern solutions.

5.3. Pattern examples and realization

From the problem description we take three capabilities that are required to represent an EOIO, while keeping the integration processes stateless. We summarize the capabilities to the following two patterns, for which we explain the realization: Commutative Receiver and Timed Redelivery until Acknowledge. In addition, we require the Quick Acknowledgement pattern from [55].

Commutative Receiver. The commutative receiver accounts for two tasks: message deduplication and out-of-order handling. Therefore, the application’s state is re-used, hence no additional state in the integration process is required.



How to ensure idempotent, in-order message processing without intermediate state in form of persistent integration scenarios?

(**Icon:** the icon uses the icon notation from [3], combining the in-order sequencing as well as the idempotent storage.)

Context: Out-of-order communication with endpoints/applications.

Solution: Guarantee that endpoint/application handle arriving out-of-order messages will be stored within their sequence and only then processed, if the sequence is (partially) complete and in the correct order.

Result: This solution handles out of order messages and applies them in-order within the application endpoint.

Relations to other patterns: This pattern is an extension of the Idempotent Receiver from [3] with additional Message Sequence handling.

Known uses: not found in literature or system review, however, Microsoft advices developers to implement commutative endpoints in the context of micro services[4].

[4] Designing Services, visited 02/2017: https://msdn.microsoft.com/en-us/library/ee658114.aspx.

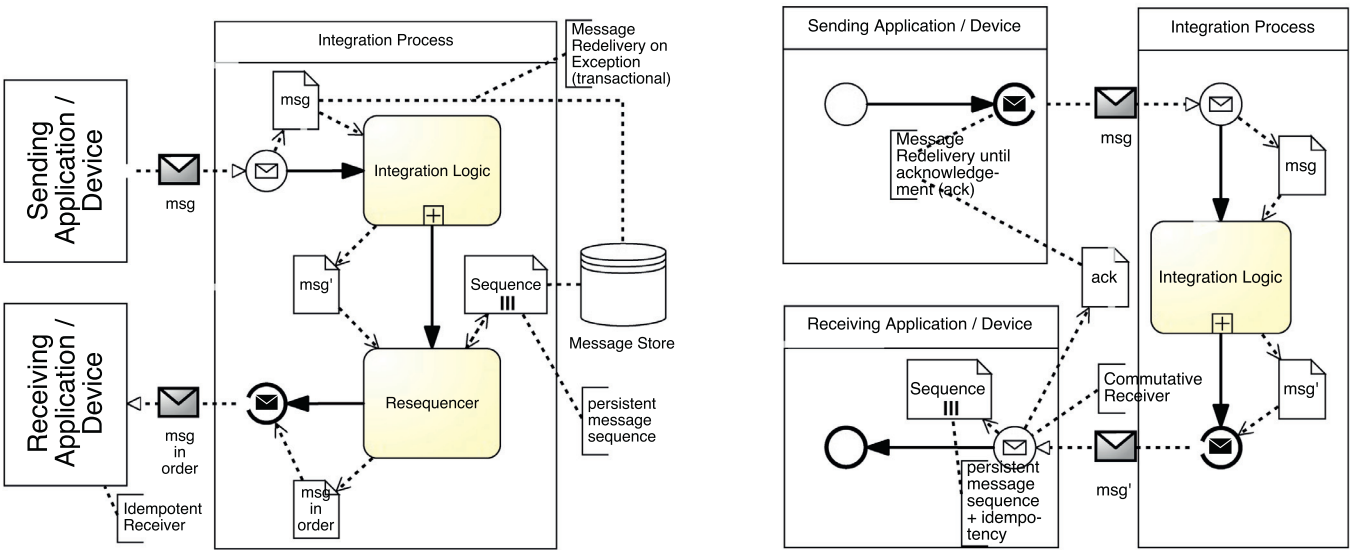
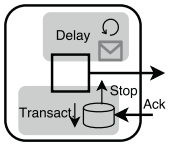


Fig. 6. Conversational approach for Exactly Once in Order (EOIO).

Table 7
Common pattern formats: Enterprise Integration Patterns (EIP) [3], Cloud Computing Patterns (CCP) [95], Design Patterns (DP) [96].

Categories	Description	EIP	CCP	DP
Name	pattern identifier	√	√	√
Icon	visual representation	√	√	–
Problem / Driving Question / Motivation	difficulty as question	√	√	√
Intent	statement about design issue	–	–	√
Also known as	other pattern names	–	–	√
Context / Motivation	introduces problem domain	√	√	√
Forces, Appilcability	problem constraints	√	–	√
Solution	how to solve the problem	√	√	–
Sketch, Structure	illustrate solution	√	–	√
Participants, Collaborations	participants, responsibilities	–	–	√
Results / Consequences	how to apply the solution	√	√	√
Next / Related Patterns	related patterns, differences	√	–	√
Sidebars / Implementation / Code	pattern variations	√	–	√
Examples / Known Uses	real system examples	√	√	√

Timed Redelivery until Acknowledge. The commutative receiver moves the message redelivery on exception from the integration process to the sender application, while conducting an asynchronous communication. Hence, no exceptions are propagated back to the sender, however, the redeliveries are stopped by asynchronously received Acknowledgements from the receiver (via the integration system). Until then, the messages are resent with increasing delay to reduce the load of duplicate messages.



How to ensure that a message will be received without intermediate storage, e. g., in form of Redelivery on Exception [94]?

(**Icon:** the icon uses the icon notation from [3], combining delayed message send with asynchronous reception of acknowledgments using a transactional store.)

Context: This pattern is used for 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 [55]) reach the sender. Requires an Idempotent [3] or Commutative Receiver for certain message delivery semantics [89]

Result: Send copies of the same message asynchronously until the receiver's acknowledgement reaches the sender.

Relations to other patterns: This pattern is an extension of the Retry pattern in [55], and related to the Redelivery on Exception pattern in [94].

Known uses: - (not found in literature or system review).

Solution Summary. As an extension a *Timed Redelivery until Acknowledge* pattern would be required that makes multiple attempts to deliver a message (potentially with exponential back-off delay). That might result to duplicate message instances, sent to the receiver. Assuming a stateless integration process, an idempotent receiver [3] is required to detect and handle the duplicates. The sketched conversation works fine for exactly-once processing semantics [89]. However, for ensuring in-order message processing (e. g., create sales order, before update), it would not be sufficient. A stateless integration process cannot reliably re-order the incoming messages, delegating this task to the receiver application. The receiving application has to handle incoming messages in an associative and commutative way (e. g., handle update, before cre-

ate). An implementation of this *Commutative Receiver* pattern can be found in the microservice context (e. g., service applications). When the receiver got all messages belonging to one message sequence (i. e., without duplicates), it sends an Acknowledgement message that is asynchronously processed by the sender, which stops redelivering corresponding messages immediately.

6. Quantitative analysis

In this section we conduct a quantitative analysis of integration scenarios to study the usage of original EIP and the new patterns from the pattern catalog in Section 4. We selected the SAP Cloud Integration system from the review (cf. Section 3), for which we found "real-world" examples of all scenarios from Fig. 2. With over 1,000 customers and several hundred integration scenarios delivered as standard content SAP Cloud Integration represents a cloud integration system for application and data integration. The analysis targets the hypotheses "The original EIP of the 2004 book are all widely used in praxis" and "H4: Solutions not in EIP can be found in real-world integration scenarios for the trends". Therefore, we firstly select several scenarios along the identified trends and briefly describe them. Secondly, we evaluate found original EIP and new solutions.

6.1. Integration scenarios

The new trends – set into context denoted by edges via the integration system node in Fig. 2 – can be summarized to integration the scenario domains:

- On-Premise to Cloud: Most organizations productively run on packaged, on-premise applications. They need to connect these applications with cloud applications for various reasons, e. g., extensions for legal reasons or new functionality, to connect with business partners. This integration domain is called hybrid integration [4].
- Cloud to Cloud or Business Network (including social): Native cloud applications or cloud integrated on-premise applications interact with business partners in business networks (e. g., payment, financial, supplier relationships), connect to social networks (e. g., social marketing) or consume cloud services (e. g., machine learning).
- Device to Cloud (including mobile and personal computing): What starts with business applications on "bring your own device" for mobility, extends to intelligence brought to machines (e. g., sensors and actuators in smart logistics or production) and eventually comes down to sensors and devices for personal computing.

We left out the conventional On-Premise to On-Premise application integration and other variations due to our focus on new trends. For the quantitative analysis, we selected one application integration solution for each of the new scenario domains, and we added one for cloud to cloud and business networks due to the slight focus on these domains. The solutions can be visited as SAP Cloud Integration standard content [97].

SAP Cloud for Customer (C4C). The C4C solutions for the communication with on-premise Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) applications, abbreviated *c4c-erp* and *c4c-crm*, can be considered typical hybrid corporate to cloud application integration [97]. The dominant integration styles – according to the classification in [2] – are process invocation and data movement. The state changes (e. g., create, update) of business objects (e. g., business partner, opportunity, activity) as well as master data in the cloud or corporate applications (e. g., ERP, CRM) are exchanged with each other.

SAP Financial Services Network (FSN). In contrast to C4C, FSN [98], abbreviated *fsn*, is a cloud-based, business network that connects banks and other financial institutes with their corporate customers (e. g., for payments, bank account management). The integration style is mostly process invocation [2]. Besides reliability, the major focus lies on secure message exchange.

SAP Cloud Integration eDocument/Electronic Invoicing (eDocument). The SAP Cloud Integration eDocument/Electronic Invoicing is a solution for country-specific electronic document management [97]. The *edocuments* solution helps cooperations to interact with legal authorities (e. g., implement the new *EU Data Protection Regulation*[5]).

SAP Predictive Maintenance and Service (PdMS). In PdMS [97], machine data is collected using an Internet of Things (IoT) platform and enriched with business information coming from, e. g., SAP Business Suite. This allows real time monitoring of the machine that triggers alerts resulting in service tickets in SAP CRM or ERP. Based on that any unusual trends or behavior becomes visible and appropriate action, potentially avoiding unnecessary service costs, can be taken.

6.2. Scenario analysis

For the analysis of the SAP delivered standard content in this paper, we prototypically implemented data discovery and mining capabilities into the SAP Cloud Integration system, which identified a total of 154 distinct integration scenarios (c4c-erp: 42, c4c-crm: 37, fsn: 56, edocument: 13, pdms: 6).

The total number of patterns for all scenarios is 1501 (w/o adapters, endpoints). For the more "classical" integration scenarios in c4c-erp and c4c-crm nearly all integration patterns could be covered by original EIP from [3] (apart from second level configuration on monitoring and exception handling). For the cloud integration scenarios fsn and eDocument as well as for the pdms IoT scenario, 466 new requirements (and 597 complementing, second level configurations) were needed in total, out of which 66% are covered by existing EIP (i. e., 1025 capabilities in total). This means that for these integration scenarios approximately $\frac{1}{3}$ of the required patterns are not covered by the original EIP.

Pattern Solutions covered by the EIP. The distribution of patterns covered by original EIP is depicted in Fig. 7. Not all EIP were required in the scenarios of the integration solutions, however, nearly all of them facilitate the tasks of (i) Message Construction: solutions *Document Message* and *Request-Reply*; (ii) Messaging Channels: solution *P2P Channel*; (iii) Message Routing: solutions *Content-based Router, Splitter*, and *Aggregator*; (iv) Message Transformation: solutions *Content Enricher, Content Filter*, and *Message Translator*.

New Pattern Solutions. Additional pattern solutions are covered by integration capabilities, depicted in Fig. 8. We grouped the new solutions according to the pattern catalog from Section 4 and added the corresponding pattern proposals for each of them. While approximately half of the new patterns from the catalog are used in the real-world scenarios, the new conversation patterns, are not yet used in any of the scenarios. For example, the confidentiality requirements covered message confidentiality or privacy patterns (Message Encryptor, Message Decryptor, Encrypted Message, Encrypting Endpoint, Encrypting Adapter) are called *Msg. Privacy*, and the authenticity and integrity requirements (Message Signer, Signature Verifier, Signed / Verified Message) are summarized to *Msg.*

[5] EU – General Data Protection Regulation: http://goo.gl/Ru0slz.

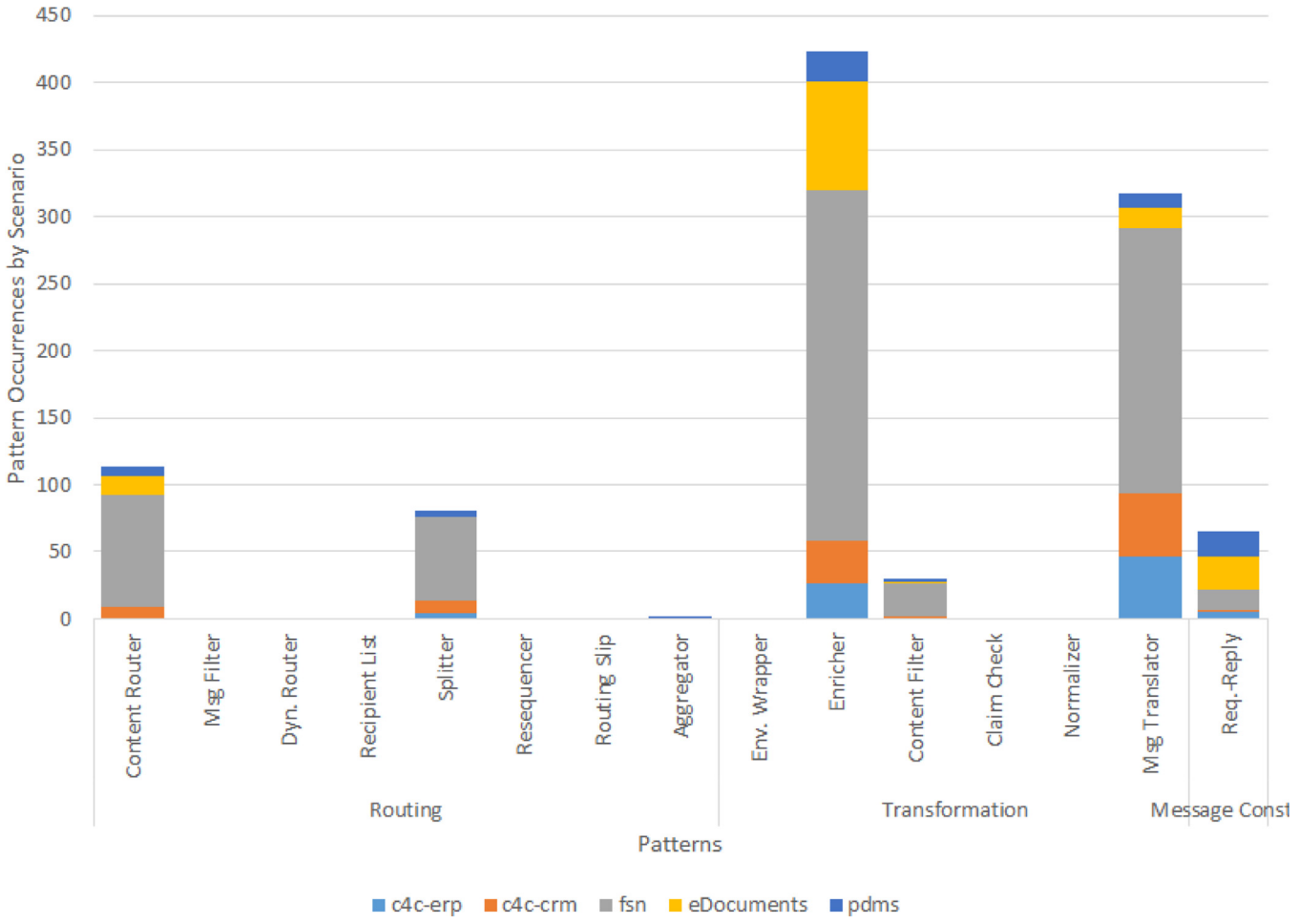


Fig. 7. Scenarios using original EIP.

Auth.. Thereby the message confidentiality is exclusively required for the communication within the FSN business network, while message authenticity is also used for exchanging eDocuments with the legal authorities and for PdMS.

In the category multimedia, currently no real media format handlers (Type Converter, Encoder, Decoder) are used (e. g., image, video). However, we grouped the used functionality into three patterns. The Encoder and Decoder patterns represent the handling of binary message content, exclusively used in FSN and eDocument scenarios. This is due to the various communication partners, using different encodings, as well as third party services (e. g., financial document mapping engines), which requires special encodings. The Custom Script allows to add versatile User-defined Functions, which are mostly used as auxiliary in FSN scenarios. The compression algorithms (Compress Content, Decompress Content), which would be immensely relevant in real multimedia scenarios, are used in FSN scenarios due to larger messages sizes (e. g., aggregated FSN payment details). Finally, marshalling (Marshaller, Unmarshaller) support is required in FSN scenarios, since some communication partners require JSON to XML conversion and vice versa.

The processing of messages is mostly done by moving messages through the pipeline. However, especially the FSN and CRM integration require streaming and parallel processing. This is again due to scenarios with larger messages to be processed (e. g., IDoc segments in CRM) and stream-based splitting in PdMS. The Multicast pattern is used as Sequential Multicast in FSN to allow guaranteed rollback for all branches in case of an error and as Parallel Multicast in PdMS for parallel processing purpose.

This behaviour is complemented by a Stop All setting in the FSN, PdMS and partially CRM scenarios, consequently stopping the message processing in all parallel instances of the integration scenario. Another error handling functionality is the Rethrow, which allows to (re-) throw exceptions. The Rethrow is mainly used in eDocument, FSN, PdMS and CRM scenarios to indicate that a situation is still unresolved. Especially in FSN, PdMS and eDocument scenarios, it is important to inform a business expert or administrator about erroneous situations. The Raise Indicator is used for this purpose. To prevent from uncontrollable behaviour and to customize the information returned in case of an error in synchronous scenarios, a Catch-all exception process (with several steps) is used in FSN and eDocument scenarios.

To remember parts of a message or additional information generated by adapters or message processors, a Transient Store (cf. [56]) is used in FSN. The Store Accessor, used by FSN and eDocument, is mostly used for stateful pattern compositions and for legal reasons (Data Store, Audit Log). Especially in FSN, most of the message stores are encrypting (Encrypting Store), which means that the messages are stored confidentially.

The composition in terms of the Integration Subprocess pattern (excluding the exception sub-processes) indicates complex processing logic, which can mostly be found in FSN, PdMS and eDocument scenarios. Sometimes composition is used for re-usable process parts.

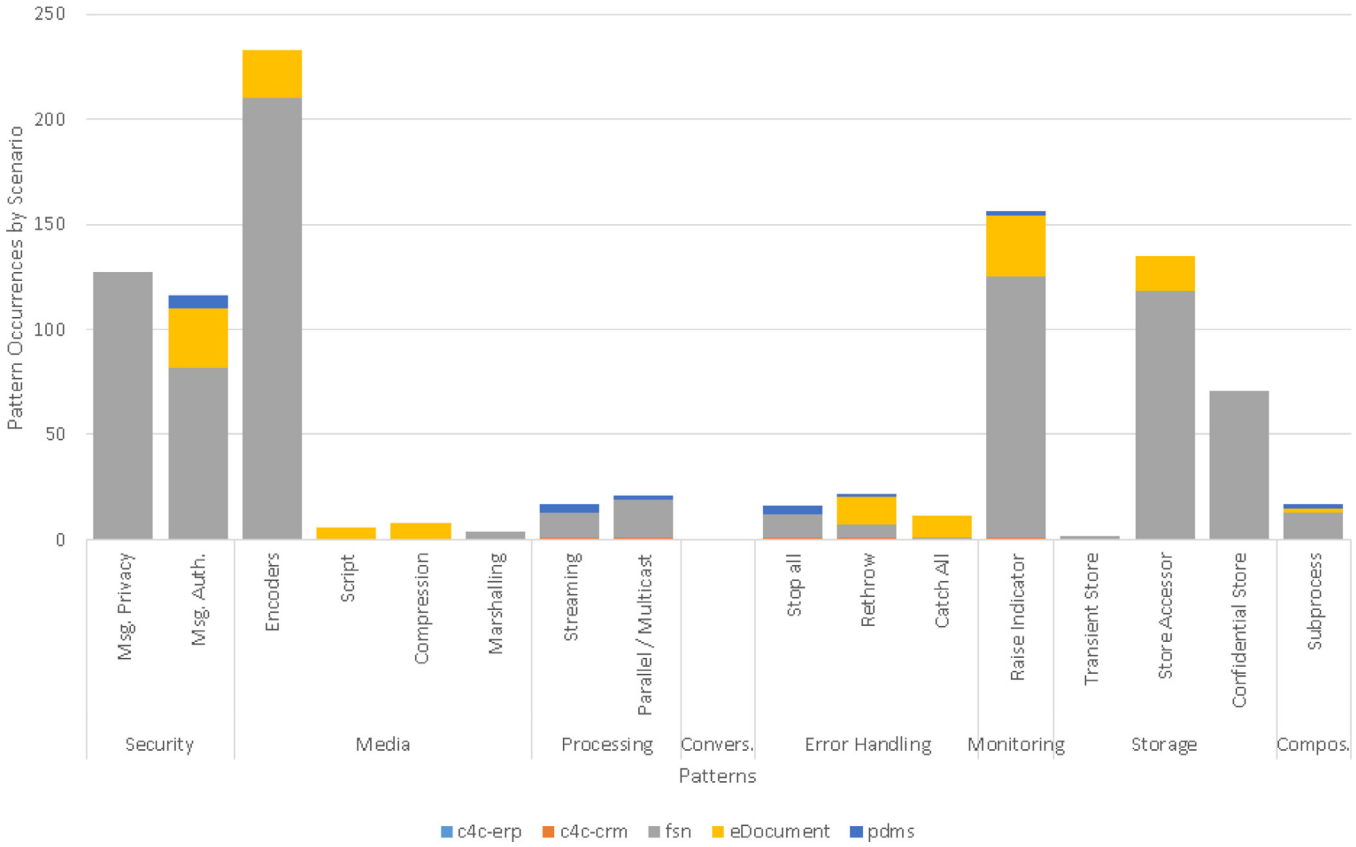


Fig. 8. New capability categorization.

In summary, the analysis shows the need for new patterns and solutions as seen in the system review. While the hybrid integration scenarios simply extend the on-premise integration into the cloud relying on transport level security, all other new integration scenario domains require more on security and control over the message processing in form of error handling. This becomes more obvious, the more exclusively the integration scenarios are running in the cloud. For example, the small amount of device integration scenarios still relies on integration logic on the devices, thus show only few security, error handling and processing capabilities. The scenarios are less complex – compared to the cloud to cloud cases, hence require limited composition options. Furthermore, with increasing cloud focus, the trade-off between more complex, but expressive, stateful and simpler, better scalable, stateless message processing seems to lean towards the interaction with storage currently. The conversation patterns – including stateful conversations – are still mostly unexplored. The same is true for the increasingly important topic of multimedia processing, which will give a new edge to the variety and interoperability problems in EAI [2].

7. Challenges, limitations, impact

The literature (cf. [5]) and the quantitative analysis of real-world integration scenarios (cf. Fig. 7) show that some of the enterprise integration patterns (EIP) described by Hohpe and Woolf [3] in 2004 are still widely used, thus denote best-practices in the area of application integration. This supports the assumption of the EIP authors that the patterns are still practically relevant [5]. Literature and system reviews also reveal that since 2005 several new trends and non-functional aspects (NFA) for enterprise application integration have appeared that are not covered by the EIP from 2004. Literature as well as systems partly offer solutions to these new trends and NFA where the systems provide a more comprehensive support. Solutions mentioned in the literature comprise patterns, formalization, and modeling, covering the spectrum from a more abstract description as patterns (cf. Section 5) to the implementation and execution as well as towards the user's point of view. For this reason, patterns are regarded as the "glue" for which a comprehensive description is required first. Hence, this work (together with the supplementary material [44]) aimed at filling the gaps in existing patterns for new integration trends and NFA (cf. Section 5): security, (ideas on) conversation, monitoring, storage. Nonetheless, the different reviews and analyses conducted in this work indicate open research challenges. These are subsequently summarized and discussed.

7.1. Research challenges

The literature review shows that for the trends and NFA different solutions have been proposed, mainly patterns, formalization, and modeling.

7.1.1. Patterns

Patterns are the predominant solution proposed in literature (cf. Table 1). Moreover, this work has closed gaps by providing patterns for NFA not present so far. Still pattern descriptions would be necessary in the context of the following trends and NFA. At first, *multimedia* functions are under-represented. Due to the access to the end user, multimedia becomes more and more interesting for all kinds of applications (e. g., sentiment analysis, monitoring in different domains like medicine or agriculture). For application integration, this targets the volume, variety and interoperability problems. The resulting increase of heterogeneity of media formats and communication partners (e. g., cloud applications, mobile devices, camera phones) demands for revisiting the EIP in

the context of multimedia operations and their semantic aspects. Consequently, the increasing message sizes require the evaluation of optimization techniques (e. g., message indexing), and more efficient processing styles like *streaming*, which the EIP authors also acknowledge [5], or data-aware integration pattern solutions (e. g., [91]). In general, to address the *Big Data* challenges of volume and velocity requires corresponding benchmarks for pattern (e. g., EIPBench [99]) as well as for end-to-end system implementations, which are currently missing. As additional NFA, only few of the *conversation patterns* are supported. For instance, Section 5 showed that conversation patterns can provide alternatives to improve the current processing and might become useful in more complex application or device interactions (e. g., device mesh [6]). The *monitoring* of integration scenarios across multiple platforms (e. g., mobile, on-premise, cloud) – including aspects like raising indicators in case of an event – remains a challenge. This also hints on further work required for *Mobile Computing* and *Internet of Things*, e. g., standardized protocols, conversation or interaction patterns (incl. data collection, device reconfiguration), energy efficiency. Finally, as new trends and NFA might constantly arise, their analysis with respect to pattern support becomes a continuous task.

7.1.2. Formalization

Starting from the pattern view, formalization is an important step to precisely specify the semantics of the pattern realizations, i. e., formalization constitutes an important step towards the implementation and execution of the patterns in integration scenarios. As shown in Table 1 formalization approaches have been predominantly proposed in the context of *service oriented architectures* (SOA) for validating and optimizing compositions by, for example, mapping them to Petri nets. Notably, a more formal definition of integration pattern composition (also suggested by the EIP authors [5]) is required not only for structural validations using Petri nets – as in the literature review, but also semantic, runtime validations and optimizations on static scenario as well as dynamic, workload data is missing. First work on the latter was conducted by [100], however, has to be revisited in the context of the trends and NFA as well as new technical capabilities (e. g., machine learning of / for workload patterns, routing conditions, condition orderings). Furthermore, cloud, mobile and device computing raise new questions about optimal runtime placements of integration processes. In general, there is still an enormous potential for elaborating formalizations for both, trends and NFA, specifically, as a more or less comprehensive set of patterns has been proposed by now. A follow-up research question is how to implement patterns and pattern compositions in solutions using formal models.

7.1.3. Modeling

Except few works in the *SOA* domain providing modeling support for compositions, no attention has been paid to model integration-specific aspects so far. For compositions, business process modeling notations such as business process model and notation (BPMN) can be used, however, the integration-specific aspects exceed the modeling capabilities. However, conveying information on the integration scenarios to users is of utmost importance for, e. g., maintenance and adaptations of these scenarios. Also here patterns might help to form the basis for different modeling and visualization proposals. It could be envisioned to base integration flows on existing business process modeling languages (e. g., BPMN) in order to keep the mental map of users, but to enhance them with integration-specific icons. In [44], a first idea of equipping BPMN with integration icons is depicted for the SAP Cloud Integration eDocument use case (due to lack of space we refer to the technical report). In general, NFA like *security* and *multimedia* have to be further analyzed. Therefore, new visual configuration editors, e. g., allowing to "query by sketch" conditions, for integration scenarios would provide a more adequate, non-textual configuration. In addition, editors and visual data science (incl. machine learning) tools for scenario-based, runtime monitoring, which are capable of dealing with large amounts of data, could lead to smarter (cross-) integration platform administration of integration scenarios. In this context, the development of visual modeling notations, new editors together with extensive user studies become necessary.

7.2. Limitations

Limitations of the work concern the literature and the system review. For both the searches were led by the selection of keywords and criteria due to the vast amount of existing work and in order to not loose focus of this study. Nonetheless, conducting further vertical searches and expert additions that were not found based on the keywords could be included in the analysis. The selection of representative systems is supposed to reflect the current system offering. Different types of systems (open source, commercial, and startup) were considered. In summary, both reviews were envisioned to be comprehensive, but not complete. The quantitative analysis aims at covering a broad range of applications based on five use cases. One might argue that the use cases are all provided by the same organization. However, this provides the chance to analyze real-world scenarios instead of toy examples.

7.3. Impact

The impact of a continuous analysis of integration trends and NFA on research and practice is enormous. The impact on research is reflected in the open research challenges stated in Section 7.1. In order to address these challenges, a plethora of new approaches is necessary. The importance of the topic from a practical perspective is already paramount as the system and scenario analyses of this paper show. Facing new trends that often stem from practice will perpetuate the importance of this work in the future. Putting the focus on the human aspect in addition to a more technical treatment of the topic will also lead to a multitude of new research questions and practical implications. While the original EIP from 2004 are still relevant for many of the new trends in 2016 and beyond, new capabilities are required to address requirements (e. g., non-functional aspects) resulting from these trends (cf. hypothesis H1).

References

[1] S. Conrad, W. Hasselbring, A. Koschel, R. Tritsch, Enterprise application integration, Spektrum Akademischer Verlag, 2005.

[2] D.S. Linthicum, Enterprise application integration, Addison-Wesley Longman Ltd., 2000.

[3] G. Hohpe, B. Woolf, Enterprise integration patterns: designing, building, and deploying messaging solutions, Addison-Wesley Professional, 2004.

[4] Forrester ResearchInc., The Forrester Wave: Hybrid Integration For Enterprises, Q4 2016, 2016. (https://www.forrester.com/report/The+Forrester+Wave+Hybrid+Integration+For+Enterprises+Q4+2016/-/E-RES131101).

[5] O. Zimmermann, C. Pautasso, G. Hohpe, B. Woolf, A decade of enterprise integration patterns: a conversation with the authors, IEEE Softw. 33 (1) (2016) 13–19.

[6] GartnerInc., Gartner Newsroom Emerging Technologies from 2005 to 2017, 2017. (http://www.gartner.com/newsroom/id/492152, http://www.gartner.com/newsroom/id/495475, http://www.slideshare.net/dinhledat/dinh-ledat-top-10-technology-trends-20072014-gartner, http://www.gartner.com/newsroom/id/530109, http://www.gartner.com/newsroom/id/777212, http://www.gartner.com/newsroom/id/1210613, http://www.gartner.com/newsroom/id/1454221, http://www.gartner.com/newsroom/id/1826214, http://www.gartner.com/newsroom/id/2209615, http://www.gartner.com/newsroom/id/2603623, http://www.gartner.com/newsroom/id/2867917, http://www.gartner.com/newsroom/id/3143521, http://www.gartner.com/newsroom/id/3482617).

[7] Forrester ResearchInc., The Top 10 Technology Trends To Watch: 2016 To 2018, 2016. (https://www.forrester.com/report/The+Top+10+Technology+Trends+To+Watch+2016+To+2018/-/E-RES120075).

[8] R. Wieringa, Design science methodology for information systems and software engineering, Springer, 2014.
[9] B. Kitchenham, Procedures for performing systematic reviews, Technical Report TR/SE-0401, Keele University, Keele, UK, 2004.
[10] C. Hentrich, U. Zdun, Patterns for business object model integration in process-driven and service-oriented architectures, in: Proceedings of the 2006 conference on Pattern languages of programs, 2006, p. 23.
[11] C. Hentrich, U. Zdun, Service integration patterns for invoking services from business processes., in: EuroPLoP, 2007, pp. 235–278.
[12] C. Hentrich, U. Zdun, A pattern language for process execution and integration design in service-oriented architectures, in: Transactions on Pattern Languages of Programming I, Springer, 2009, pp. 136–191.
[13] U. Zdun, C. Hentrich, W.M. Van Der Aalst, A survey of patterns for service-oriented architectures, Int. J. Internet Protoc. Technol. 1 (3) (2006) 132–143.
[14] U. Zdun, Pattern-based design of a service-oriented middleware for remote object federations, ACM Trans. Internet Technol. (TOIT) 8 (3) (2008) 15.
[15] M. Autili, A. Di Salle, A. Perucci, M. Tivoli, On the automated synthesis of enterprise integration patterns to adapt choreography-based distributed systems, ArXiv e-prints (2015).
[16] V. Gacitua-Decar, C. Pahl, Ontology-based patterns for the integration of business processes and enterprise application architectures, Semantic Enterprise Application Integration for Business Processes: Service-Oriented Frameworks, IGI Publishers, Hershey, PA (2009) 36–60.
[17] M. Heller, M. Allgaier, Model-based service integration for extensible enterprise systems with adaptation patterns, in: e-Business (ICE-B), Proceedings of the 2010 International Conference on, 2010, pp. 1–6.
[18] E. Kaneshima, R.T.V. Braga, Patterns for enterprise application integration, in: Proceedings of the 9th Latin-American Conference on Pattern Languages of Programming, 2012, p. 2.
[19] K. Umapathy, S. Purao, Designing enterprise solutions with web services and integration patterns, in: IEEE International Conference on Services Computing (SCC'06), 2006, pp. 111–118.
[20] Y. Zheng, H. Cai, L. Jiang, Application integration patterns based on open resource-based integrated process platform, in: International Conference on Information Computing and Applications, 2011, pp. 577–584.
[21] C. Gierds, A.J. Mooij, K. Wolf, Reducing adapter synthesis to controller synthesis, IEEE Trans. Serv. Comput. 5 (1) (2012) 72–85.
[22] R. Seguel, R. Eshuis, P. Grefen, An Overview on Protocol Adaptors for Service Component Integration, Technical Report, Technische Universiteit Eindhoven, 2008.
[23] V.N. Gudivada, J. Nandigam, Enterprise application integration using extensible web services, in: IEEE International Conference on Web Services (ICWS'05), 2005, pp. 41–48.
[24] W. Deng, X. Yang, H. Zhao, D. Lei, H. Li, Study on EAI based on web services and SOA, in: International Symposium on Electronic Commerce and Security, 2008, pp. 95–98.
[25] C. Mauro, J.M. Leimeister, H. Krcmar, Service oriented device integration-an analysis of SOA design patterns, in: 43rd Hawaii International Conference on System Sciences (HICSS), 2010, pp. 1–10.
[26] Y. Liu, X. Liang, L. Xu, M. Staples, L. Zhu, Using architecture integration patterns to compose enterprise mashups, in: Software Architecture & European Conference on Software Architecture, 2009, pp. 111–120.
[27] Y. Liu, X. Liang, L. Xu, M. Staples, L. Zhu, Composing enterprise mashup components and services using architecture integration patterns, J. Syst. Softw. 84 (9) (2011) 1436–1446.
[28] D. Braga, S. Ceri, F. Daniel, D. Martinenghi, Mashing up search services, IEEE Internet Comput. 12 (5) (2008) 16–23.
[29] S. Cetin, N.I. Altintas, H. Oguztüzün, A.H. Dogru, O. Tufekci, S. Suloglu, A mashup-based strategy for migration to service-oriented computing., in: International Conference on Pervasive Service, 2007, pp. 169–172.
[30] X. Qu, X. Yang, J. Zhong, X. Lv, Integration patterns of grid security service, in: Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05), 2005, pp. 524–528.
[31] D. Shah, D. Patel, Dynamic and ubiquitous security architecture for global SOA, in: The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, UBICOMM'08., 2008, pp. 482–487.
[32] M. Fisher, S. Sharma, R. Lai, L. Moroney, Java EE and. net interoperability: integration strategies, patterns, and best practices, Prentice Hall Professional, 2006.
[33] C. Ouyang, E. Verbeek, W.M. Van Der Aalst, S. Breutel, M. Dumas, A.H. Ter Hofstede, Formal semantics and analysis of control flow in WS-BPEL, Sci. Comput. Program 67 (2) (2007) 162–198.
[34] N. Lohmann, P. Massuthe, C. Stahl, D. Weinberg, Analyzing interacting WS-BPEL processes using flexible model generation, Data Knowl. Eng. 64 (1) (2008) 38–54.
[35] A. Kumar, Z. Shan, Algorithms based on pattern analysis for verification and adapter creation for business process composition, in: OTM Confederated International Conferences, 2008, pp. 120–138.
[36] J.-m. Jiang, S. Zhang, P. Gong, Z. Hong, Message dependency-based adaptation of services, in: IEEE Asia-Pacific Services Computing Conference (APSCC), 2011, pp. 442–449.
[37] A. Barros, M. Dumas, A.H. Ter Hofstede, Service interaction patterns, in: International Conference on Business Process Management, 2005, pp. 302–318.
[38] T. Köllmann, C. Hentrich, Synchronization patterns for process-driven and service-oriented architectures., in: EuroPLoP, 2006, pp. 199–228.
[39] F.B. Vernadat, Interoperable enterprise systems: principles, concepts, and methods, Annu. Rev. Control 31 (1) (2007) 137–145.
[40] G. Grossmann, M. Schrefl, M. Stumptner, Exploiting semantics of inter-process dependencies to instantiate predefined integration patterns, in: Proc. of the 26th international conference on Conceptual modeling, 2007, pp. 155–160.
[41] H. Taylor, A. Yochem, L. Phillips, F. Martinez, Event-driven architecture: how SOA enables the real-time enterprise, Pearson Education, 2009.
[42] O.P. Patri, V.S. Sorathia, A.V. Panangadan, V.K. Prasanna, The process-oriented event model (PoEM): a conceptual model for industrial events, in: International Conference on Distributed Event-Based Systems, 2014, pp. 154–165.
[43] S. Asmus, A. Fattah, C. Pavlovski, Enterprise cloud deployment: integration patterns and assessment model, IEEE Cloud Comput. 3 (1) (2016) 32–41.
[44] D. Ritter, S. Rinderle-Ma, Toward a collection of cloud integration patterns, arXiv preprint arXiv:1511.09250 (2015).
[45] D. Merkel, F. Santas, A. Heberle, T. Ploom, Cloud integration patterns, in: European Conference on Service-Oriented and Cloud Computing, 2015, pp. 199–213.
[46] D. Ritter, Experiences with business process model and notation for modeling integration patterns, in: European Conference on Modelling Foundations and Applications, 2014, pp. 254–266.
[47] D. Ritter, Towards more data-aware application integration (extended version), CoRR abs/1504.05707 (2015).
[48] D. Mansor, Moving to the cloud: patterns, integration challenges and opportunities, in: Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, 2009. 9–9.
[49] H. Buckow, H.-J. Groß, G. Piller, K. Prott, J. Willkomm, A. Zimmermann, Integration strategies and patterns for SOA and standard platforms, in: GI Jahrestagung (1), 2010, pp. 398–403.
[50] M. Heiss, A. Oertl, M. Sturm, P. Palensky, S. Vielguth, F. Nadler, Platforms for industrial cyber-physical systems integration: contradicting requirements as drivers for innovation, in: Modeling and Simulation of Cyber-Physical Energy Systems, 2015, pp. 1–8.
[51] D. Ritter, J. Bross, Datalogblocks: relational logic integration patterns, in: International Conference on Database and Expert Systems Applications, 2014, pp. 318–325.
[52] T. Scheibler, F. Leymann, A framework for executable enterprise application integration patterns, in: Enterprise Interoperability III, Springer, 2008, pp. 485–497.
[53] T. Scheibler, F. Leymann, Realizing enterprise integration patterns in WebSphere, Technical Report, University of Stuttgart, 2005.
[54] R. Thullner, A. Schatten, J. Schiefer, Implementing enterprise integration patterns using open source frameworks, Softw. Eng. Tech. Prog. (2008) 111–124.
[55] G. Hohpe, Conversation patterns, in: The Role of Business Processes in Service Oriented Architectures, in: number 06291 in Dagstuhl Seminar Proceedings, 2006, p. 7.
[56] L. González, R. Ruggia, Addressing QoS issues in service based systems through an adaptive ESB infrastructure, in: Proceedings of the 6th Workshop on Middleware for Service Oriented Computing, 2011, p. 4.
[57] D. Fahland, C. Gierds, Using Petri nets for modeling enterprise integration patterns, Technical Report, bpmcenter.org, 2012.
[58] D. Fahland, C. Gierds, Analyzing and completing middleware designs for enterprise integration using coloured petri nets, in: International Conference on Advanced Information Systems Engineering, 2013, pp. 400–416.
[59] O.P. Patri, A.V. Panangadan, V.S. Sorathia, V.K. Prasanna, Semantic management of enterprise integration patterns: a use case in smart grids, in: Data Engineering Workshops (ICDEW), 2014, pp. 50–55.
[60] S. Basu, T. Bultan, Automatic verification of interactions in asynchronous systems with unbounded buffers, in: International conference on Automated software engineering, 2014, pp. 743–754.
[61] P. Mederly, M. Lekavỳ, M. Závodskỳ, P. Návrat, Construction of messaging-based enterprise integration solutions using AI planning, in: IFIP Central and East European Conference on Software Engineering Techniques, 2009, pp. 16–29.
[62] P. Mederly, P. Návrat, Construction of messaging-based integration solutions using constraint programming, in: East European Conference on Advances in Databases and Information Systems, 2010, pp. 579–582.
[63] R. Kazman, K. Schmid, C.B. Nielsen, J. Klein, Understanding patterns for system of systems integration, in: System of Systems Engineering, 2013, pp. 141–146.
[64] R. Land, I. Crnkovic, S. Larsson, Process patterns for software systems in-house integration and merge-experiences from industry, in: Conference on Software Engineering and Advanced Applications, 2005, pp. 180–187.
[65] D. Chen, G. Doumeingts, F. Vernadat, Architectures for enterprise integration and interoperability: past, present and future, Comput. Ind. 59 (7) (2008) 647–659.
[66] H. Panetto, R. Jardim-Goncalves, A. Molina, Enterprise integration and networking: theory and practice, Annu. Rev., Control 36 (2) (2012) 284–290.
[67] K. Wang, M. Dumas, C. Ouyang, J. Vayssière, The service adaptation machine, in: European Conference on Web Services, 2008, pp. 145–154.
[68] S. Rajam, R. Cortez, A. Vazhenin, S. Bhalla, Design patterns in enterprise application integration for e-learning arena, in: International Conference on Humans and Computers, 2010, pp. 81–88.
[69] W. He, L. Da Xu, Integration of distributed enterprise applications: a survey, IEEE Trans. Ind. Inf. 10 (1) (2014) 35–42.
[70] G. Hohpe, Your coffee shop doesn't use two-phase commit, IEEE Softw. 22 (2) (2005) 64–66.

[71] S. Cranefield, S. Ranathunga, Embedding agents in business processes using enterprise integration patterns, in: International Workshop on Engineering Multi-Agent Systems, 2013, pp. 97–116.
[72] DELL Boomi, AtomSphere User Guide, 2017. (http://help.boomi.com/atomsphere/GUID-A98714FA-9EAB-4B69-BCC8-7D8984F0B0EC.html).
[73] IBM, WebSphere Cast Iron Cloud integration, 2017. (https://www.ibm.com/support/knowledgecenter/SSGR73).
[74] Informatica, Cloud-Integration, 2017. (https://www.informatica.com/de/products/cloud-integration.html).
[75] Jitterbit, Harmony Cloud Integration, 2017. (https://www.jitterbit.com/harmony/).
[76] Microsoft, BizTalk Server, 2017. (https://msdn.microsoft.com/en-us/library/dd547397(v=bts.10).aspx).
[77] SAP SE, SAP HANA Cloud Integration, 2017. (http://www.sap.com/product/technology-platform/hana-cloud-integration.html).
[78] Oracle, BEA WebLogic Integration, 2017. (https://docs.oracle.com/cd/E13214_01/wli/docs81/index.html).
[79] Tray.io, Tray.io Docs, 2017. (http://docs.tray.io/).
[80] Zapier, Zapier v2 Documentation, 2017. (https://zapier.com/developer/documentation/v2/reference/).
[81] Apache Foundation, Apache Flume, 2017. (https://flume.apache.org/).
[82] Apache Foundation, Apache Nifi, 2017. (https://nifi.apache.org/).
[83] C. Ibsen, J. Anstey, Camel in action, 1st, Manning Publications Co., 2010.
[84] Cloudpipes, Cloudpipes Documentation, 2017. (https://docs.cloudpipes.com/).
[85] Tibco, Tibco Cloud Integration Documentation, 2017. (https://integration.cloud.tibco.com/docs/index.html).
[86] Software AG, Webmethods Integration Cloud, 2017. (http://www.softwareag.com/corporate/products/cloud/integration/default.asp).
[87] GartnerInc., Magic Quadrant for Enterprise Integration Platform as a Service, Worldwide, 2016. (https://www.gartner.com/doc/3263719/magic-quadrant-enterprise-integration-platform).
[88] Forrester ResearchInc., The Forrester Wave: iPaaS For Dynamic Integration, Q3 2016, 2016. (https://www.forrester.com/report/The+Forrester+Wave+iPaaS+For+Dynamic+Integration+Q3+2016/-/E-RES115619).
[89] D. Ritter, M. Holzleitner, Integration adapter modeling, in: Conference on Advanced Information Systems Engineering, 2015, pp. 468–482.
[90] MuleSoft, Integration: The Cloud's Big Challenge, 2017. Accessed: 01/2017.
[91] D. Ritter, Towards more data-aware application integration, in: British International Conference on Databases, 2015, pp. 16–28.
[92] C. Peltz, Web services orchestration and choreography, IEEE Comput. 36 (10) (2003) 46–52.
[93] D. Ritter, J. Sosulski, Modeling exception flows in integration systems, in: Enterprise Distributed Object Computing Conference, 2014, pp. 12–21.
[94] D. Ritter, J. Sosulski, Exception handling in message-based integration systems and modeling using BPMN, Int. J. Cooperative Inf. Syst. 25 (2) (2016) 1–38.
[95] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter, Cloud computing patterns - Fundamentals to design, build, and manage cloud applications, Springer, 2014.
[96] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley Longman Publishing Co., Inc., 1995.
[97] SAP SE, SAP HANA Cloud Integration - Prepackaged Content, 2017. (https://cloudintegration.hana.ondemand.com/)
[98] SAP SE, SAP Financial Services Network, 2017. (http://www.sap.com/product/financial-mgmt/financial-services-network.html)
[99] D. Ritter, N. May, K. Sachs, S. Rinderle-Ma, Benchmarking integration pattern implementations, in: International Conference on Distributed and Event-Based Systems, 2016, pp. 125–136.
[100] M. Böhm, D. Habich, S. Preissler, W. Lehner, U. Wloka, Cost-based vectorization of instance-based integration processes, Inf. Syst. 36 (1) (2011) 3–29.