Inf Syst E-Bus Manage
DOI 10.1007/s10257-015-0291-6

ORIGINAL ARTICLE

A pluggable service platform architecture for e-commerce

Fabian Aulkemeier[1] · Mohammad Anggasta Paramartha[1] · Maria-Eugenia Iacob[1] · Jos van Hillegersberg[1]

Received: 17 January 2015 / Revised: 31 August 2015 / Accepted: 4 September 2015

Abstract In the beginning of the e-commerce era, retailers mostly adopted vertically integrated solutions to control the entire e-commerce value chain. However, they began to realize that to achieve agility, a better approach would be to focus on certain core capabilities and then create a partner ecosystem around them. From a technical point of view, this means it is advised to have a lightweight platform architecture with small core e-commerce functionality which can be extended by additional services from third party providers. In a typical e-commerce ecosystem with diverse information systems of network partners, integration and interoperability become critical factors to enable seamless coordination among the partners. Furthermore an increasing adoption of cloud computing technology could be observed resulting in more challenging integration scenarios involving cloud services. Thus, an e-commerce platform is required that suites the advanced needs for flexible and agile service integration. Therefore, this paper aims to present a reference architecture of a novel pluggable service platform for e-commerce. We investigate on currently available online shop platform solutions and integration platforms in the market. Based on the findings and motivated by literature on service-oriented design, we develop an architecture of a service-based pluggable platform for online retailers. This design is then instantiated by means of a prototype

✉ Fabian Aulkemeier
f.m.aulkemeier@utwente.nl

Mohammad Anggasta Paramartha
mohammadanggastaparamartha@alumnus.utwente.nl

Maria-Eugenia Iacob
m.e.iacob@utwente.nl

Jos van Hillegersberg
j.vanhillegersberg@utwente.nl

[1] Centre for Telematics and Information Technology, University of Twente, Enschede, The Netherlands

Published online: 25 September 2015

for an e-commerce returns handling scenario to demonstrate the feasibility of our architecture design.

Keywords E-commerce platform · SOA · Cloud integration · Reference architecture · Pluggability

1 Introduction

In the beginning of the e-commerce era, retailers mostly adopted vertically integrated solutions to control the entire e-commerce value chain. However, they began to realize that to achieve agility, a better approach would be to focus on certain core capabilities and then create a partner ecosystem around them. According to (Chu et al. 2007) vertical collaborative platforms based on web service technologies are the next breakthrough in e-commerce systems. Thus, retailers should aim for a modular and flexible architecture, with small core e-commerce functionality, which can be extended by additional services from third party providers. Furthermore, looking at the current e-commerce landscape, an increasing adoption of cloud computing technology can be observed, supporting the distributed and modular service concept.

However, for the modular approach to function, the increasing number of services have to stay manageable. Current IT services are mostly based on packaged applications and require significant resources to make them ready for the business needs of the user (O'Leary 2000). The required efforts during each phase of adopting an IT service has been reflected in the concept of pluggability. More precisely, pluggability is a software quality characteristic consisting of a number of quality aspects to measure the resources required throughout all phases of service adoption (Aulkemeier et al. 2015). The lack of current e-commerce architectures to support pluggable services is a potential obstacle for more flexible and scalable collaboration in e-commerce. Thus, it is important to gain insights into the capabilities of the state of the art in e-commerce platforms to support pluggable services, in order to pave the way for a pluggable service platform architecture.

Accordingly, the research goal of this paper was to *assess the capability of state of the art services in e-commerce and integration platforms to support a pluggable service architecture*. Following a design science research methodology (Peffers et al. 2007) the research was carried out in four steps:

1. *Defining the objects and benefits* of a service based e-commerce architecture and a*nalyze* the state of the art to identify its current building blocks (Sect. 2).
2. *Design and development* of an architectural reference model reflecting the state of the art (Sect. 3)
3. *Demonstration* of the architecture through implementation of a prototype, based on the architectural model (Sect. 4)
4. *Evaluation* of the prototype with regards to the criteria for pluggability (Sect. 5)

Thus, the contributions of the presented work can be summarized as follows: First of all, the state of the art study provides an overview about the current practice in e-commerce. Furthermore, the presented architecture can be considered as a reference model of a service based pluggable platform architecture for e-commerce. More precisely, we present the initial version of the model as well as a number of enhancements resulting from prototype evaluation. Finally, the prototype is a product of the design research and thus accommodating design knowledge as pointed out by (Cross 2006). The evaluation of the prototype delivers insights into the pluggability of state of the art services for e-commerce.

2 Service based platform architectures

The goal of the reference architecture proposed in this paper is to complement the existing e-commerce reference models in order to reflect recent developments in service based architectures. By taking the state of the art in e-commerce platforms as a starting point we were able to assess the current practice and point out shortcomings. In the following we discuss the objectives and benefits of the service based approach and outline the state of the art in e-commerce platforms.

2.1 Objectives and benefits

Existing reference models in the domain of e-commerce and retailing are generally business layer models (Frank and Lange 2004; Becker and Schutte 2007). They describe the entirety of functions and processes of the business model. Systems that are built based on these reference models tend to encompass all the primary business activities, often resulting in monolithic solutions. As mentioned earlier companies increasingly focus on single activities within the value chain. Providing end-to-end systems in the highly disaggregated business environment, with individual organizations that only cover parts of the value chain, leads to inefficient use of IT resources. Furthermore, monolithic systems are not built for collaboration with external systems, making the exchange of individual IT functionality and external business partners cumbersome.

The main objective of a service based, modular architecture is therefore *to support the construction of systems that go beyond what current, monolithic systems achieve with regards to flexibility in IT and business service adoption*. It enables companies to integrate innovative IT services faster and also helps them to connect with business partners with less efforts. The so called quick connect capability (QCC) has been proposed by a number of authors (van Heck and Vervest 2007; Koppius and van de Laak 2009) and describes the capability of network partners to setup business collaboration with less efforts in less time. The authors claim that the decomposition of the system and modularity are required to achieve versatility. Heck et al. suggest that digital platforms improve the interaction and the QCC. The platform based approach differentiates between the stable component in a system and the other, evolving components which evolve around it (Baldwin and Woodard 2009). A major goal in constructing the platform based architecture can therefore be

expressed as *identifying the stable components of the system, to maximize pluggability of the remaining components*.

The expected benefits of the increased pluggability through the service platform can be summarized as the ability of the platform user to *source external and innovative IT services* as well as to *collaborate with external business partners* more easily.

2.2 State of the art

To analyze the state of the art in pluggable service platforms for e-commerce we started by investigating the available products and cloud services in the market. Based on our findings we extended the market research by looking at cloud integration platforms, which complement the functional components with the aspects of connectivity. Even vendors which are in favor of a lightweight, pluggable e-commerce platform differentiate between core business functionality and connectivity components (e.g. spreecommerce).

Similar with other enterprise application systems the e-commerce platform solution landscape has evolved from custom-made components to pre-packaged solutions. Pre-packaged e-commerce solutions provide the e-commerce specific functionality such as shopping cart, product catalogue management, marketing tools, and payment (Humeau and Jung 2013). By implementing a pre-packaged solution, it becomes easier for business owners to set up and launch their online store, resulting in a faster time to market. Despite the ease and functionality that e-commerce solutions offer, some challenges remain, that ought to be solved by specialized pieces of functionality.

In a preceding study, a systematic literature review was carried out, investigating on processes and architectures for online retailing. The results were validated with the online retail practice in the Netherlands (Aulkemeier et al. 2016). According to the findings, the online retail process is comparable to existing reference processes in retailing. Furthermore, the IT landscape of online retailers is characterized by five major components, namely procurement, sales, service, logistic, and finance. In practice, most of these components are not covered by the e-commerce system, so that additional components such as an ERP system and a warehouse management system (WMS) are in use.

We conclude that the state of the art e-commerce platform are modular to some extent. However the granularity of the services is limited to a small number large application systems that cover an extensive set of functionality. Furthermore the interoperability of the functional components relies on middleware components which are supposed to increase the pluggability.

The most widely adopted solution by the e-commerce platforms under study for solving the enterprise integration issue is to rely on hard-wired web service based integration. In this approach, each external service is connected to each online shop platform through the so-called “connectors” or “adaptors”. If a connector is not available, some platforms also provide toolkits for users or service providers to develop their own application connectors. While this approach seems to work just fine, it will produce an inefficient point-to-point integration topology in the end.

When the number of systems to integrate increases, the entire integration schema will become highly complex, with a negative impact on scalability.

Besides, in near future it is expected that cloud computing will gain more popularity with companies and organizations migrating their existing local systems to the cloud. Because of this situation, new integration scenarios emerge that involve both on-premise and cloud based applications. It might become cumbersome to integrate systems of different nature like SaaS systems and legacy systems. Connections between SaaS applications are also challenging due to diversity of data models and lack of standardization (Potočnik and Juric 2012). The increasing adoption of cloud computing brings novel ways to solve integration challenges. Traditional middleware and integration platforms could obtain benefits from cloud computing technology by leveraging themselves as cloud-based integration platform (Kleeberg and Holger 2014). Commonly referred to as Integration Platform as a Service (iPaaS), a term coined by (Pezzini 2011). A recent study by Gartner evaluated and compared iPaaS providers (Pezzini et al. 2014). Another research by (Ried 2014) assesses 14 vendors providing hybrid integration solutions, which in their description, comprise of four integration scenarios: on-premise integration, cloud-based integration, iPaaS and API Management.

2.3 Common platform services

According to the earlier mentioned objectives, we extracted the common features that iPaaS vendors typically offer in their platform. They incorporate API Management capabilities into their platform in addition to SOA Governance to deliver a complete solution to take care of both SOA and REST web services. We regard both SOA Governance and API Management as essential components to enable pluggability of services. SOA Governance and API Management basically share the same underlying architectural design principle, which is service oriented design. Both aim to govern and manage the service lifecycle including design, implementation, publication, operation, maintenance and retirement of services and APIs (Malinverno et al. 2013). SOA governance technologies, however, have been around for several years and almost reached maturity. SOA governance covers a wide range of functions including but not limited to policy enforcement, security, service contract, compliance, service level agreement (SLA), lifecycle management, service registry and repository (Schepers et al. 2008). On the other hand, although API Management comprises of similar building blocks as SOA Governance, it involves some distinct capabilities (Maler and Hammond 2013). It can be said that the fundamental difference of API and SOA lies in their orientation of service consumption. In general terms, SOA is geared towards service consumption within an organization while APIs, due to their openness, can be used both internally and externally. As a consequence, some additional components, such as enterprise gateway, security, developer portal, and service billing need to be incorporated in API Management.

We grouped the services offered by those platforms in two categories. Meta-services on the one hand facilitate the access and use of provided services. Process services on the other hand offer additional features to enable process execution

Table 1 Service framework meta-services

Service framework meta-service	Features
Developer portal	In the developer portal, companies should provide relevant and comprehensive aspects of their APIs such as API documentation, policy, terms and agreement, testing environment (sandbox or real), or API versioning
Enterprise gateway	Management of the interaction between the API and external API consumers
Policy enforcement and management	Management of both, design time and runtime policies of services. Design time policies are concerned with aspects such as design guidelines or security mechanism while run time policies are concerned with operational environment and requirements that have to be met by the service at runtime
Security	The difference between security in SOA Governance and API Management is that in SOA Governance, the organization administers internal and known users while API Management handles external and unknown users. API Security manages additional aspects like authorization and authentication, API Key management, as well as Identity and Credential Management
Service analytic and reporting	Exploration of insightful traffic analytics and reports of API activities with respect to developers account, application, or services as well as observation of the overall API usage and trends
Service level agreement	Management of service levels as stated in SLA contract, service evaluation as well as fees for consuming the service and fines in case of contract violation
Service lifecycle management	Managing the design, development and delivery of individual services in a SOA. The tasks include change management procedure, service registration and even deciding on service granularity
Service metering and billing	Monitor and measure service usage as the basis for billing and calculation for the service consumers. Also the service performance can be monitored regularly
Service registry and repository	The catalogue of services and management of their publication. Definition of taxonomies of the published services allowing consumers to find suitable services to their needs. While the Service Registry only contains service references, the Service Repository is the actual holder of documentation, policies and metadata about the versioning of the service

across the integrated services. The *service framework* meta-services are presented in Table 1 and *process framework* services are presented in Table 2.

3 A reference architecture for e-commerce service platforms

As mentioned in the previous sections, a popular way to source functionality in modern enterprise architectures are outsourced cloud applications, offered by third party service providers. Retailers that want to add or replace such services have to be able to integrate them into their current system landscape. The idea behind the

Table 2 Process framework services

Process framework service	Features
Development and lifecycle management platform services	Manages service integration process flows throughout their lifecycle including modeling, development, configuration, testing and deployment
Integration platform services	Consists of aspects that ensure seamless integration flow both at design time (service orchestration) and runtime (process execution). These aspects include but are not limited to: Message transformation and routing, an Integrated Development Environment (IDE), adapters, flow management, protocol conversion, service virtualization, and security federation
Monitoring, management, and administration platform services	Takes care of deployment and administration of integration flows, monitor their execution and manage their behavior. Covers several aspects such as technical and business activity monitoring, logging and tracking. as well error resolution

pluggable platform architectures is to give users the possibility to integrate e-commerce services into the existing environment with a minimal effort in terms of sourcing and implementation. The platform should allow supply-chain partners to share their services, execute inter-organizational processes and work on resources collaboratively, eventually resulting in a an open and agile e-commerce business network. Such inter-organizational integration platforms have some distinct requirements compared to systems internally deployed and used within one organization or only available to a closed business consortium (van Hillegersberg et al. 2012), especially if the platform aims to act as a one stop shop to source IT services. In this section we are going to present a reference architecture for a pluggable service platform, which incorporates the findings on state of the art e-commerce and cloud integration platforms from the previous section.

3.1 Framework

According to (Baldwin and Currie 2000) a platform can be considered as "a set of stable components that supports variety and evolveability in a system by constraining the linkages among the other components". In our case the components are e-commerce services that together with the platform compose a working information system for e-commerce businesses. The goal of such a platform is to improve the pluggability of the services to support variety and evolvability of the used services and the overall system.

The service platform has three stakeholders, namely the service provider, the platform provider and the service consumer which is the company running an e-commerce business. To illustrate our architecture the ArchiMate modelling language is used (Lankhorst et al. 2009). It provides concepts on business, application and technology layer to model enterprise architectures. As we are dealing with an inter-organizational architecture we choose to model the three business actors as high level concept to structure the model. All further concepts are assigned to either of these actors. We focus on the application layer components that

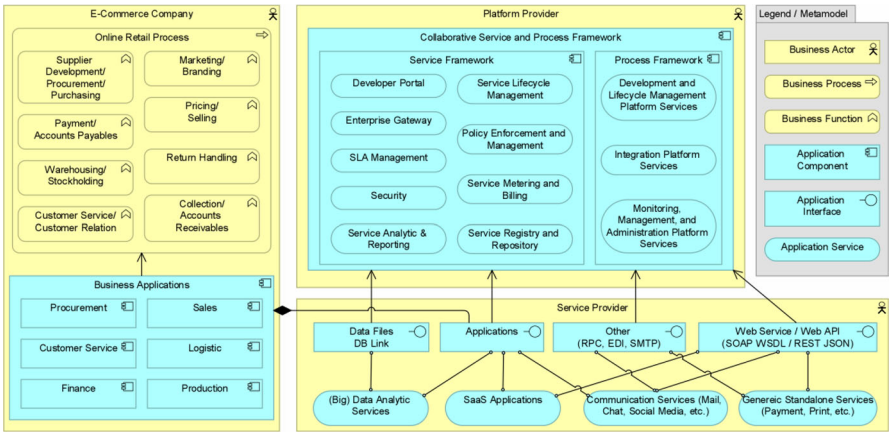

Fig. 1 ArchiMate model of the pluggable service platform

support the e-commerce process at hand. Figure 1 illustrates the view of the architecture including a meta-model of the relevant concepts in the top right. The details of each actor are discussed in the following.

3.2 E-commerce company

The e-commerce company is the actor selling goods partially or exclusively over the online channel. On the business layer the online retail process consists of pre-trade, trade and post trade activities (Liu and Hwang 2004). Internally the actor implements eight different business functions which have been identified by different authors in (Gunasekaran et al. 2002; Burt and Sparks 2003; Becker and Schutte 2007; Frank and Lange 2007) and have been consolidated in (Aulkemeier et al. 2016). The same study presents six application layer components implemented by most online retailers. Depending on the business model of the e-commerce company, different legacy components will be implemented on the application layer. A retailer coming from an offline channel business with a number of brick and mortar stores will have an ERP system to manage its operations. In that case components will be bundled into the ERP system. When introducing an online channel the retailer will add an online shop component to the landscape that allows customers to browse and order goods online. The order fulfillment and other back office activities will be carried out by the ERP system. Thus the e-commerce platform in this case consists of a lightweight online shop and the ERP system. A pure online retailer on the other hand might implement a more comprehensive e-commerce platform as discussed in Sect. 2.2. Those platforms not only provide a online shop but also a rich set of back office functionality. Depending on the complexity and size of the business an ERP component might not be present at all. All these application components can be either operated on-premise or as SaaS solutions provided in form of web applications by a service provider.

The presented architecture for the retailer actor can be considered as the current state of the art and does not introduce any new concepts in itself. In that sense it is a starting point for the use of a pluggable service platform. The architecture should allow for a gradual transition from the current, monolithic landscape to a cloud service based architecture. It should be possible to add services to that landscape and successively shift new and existing functionality from internal systems to the cloud.

3.3 Service provider

The service provider can either issue pure IT services or be a supply chain partner that provides business services (B2B e-commerce). Both service types have to be integrated on information system level for seamless process execution.

The actual service provided can contain either additional components that internal systems or business functions do not cover, or functions that should be outsourced for strategic reasons. The actual services as well as their granularity are too diverse to provide a comprehensive list. Effectually, it should be possible to integrate any kind of service through the pluggable architecture. However, a more important aspect is to obtain a comprehensive picture of potential service interfaces the platform needs to support. Four different services interfaces have been identified.

- Message based integration can be realized through modern web services or web APIs that communicate over HTTP and can be consumed with state of the art integration tools and techniques. This kind of interface is suitable for standalone services such as payment services, address verifications, customer or credit enquiries but also to access or populate resources of SaaS applications and social media services in a programmable manner.
- Another interface type is based on more specialized protocols that can be considered as an older technique to integrate services. Despite their higher complexity and technical dependencies, those protocols are still widely used to integrate legacy systems or communication services such as mail and chat but will not be implemented by modern SaaS applications.
- Web applications are generally used as user interfaces in SaaS or social media services as well as in analytical services and reporting in form of dashboards.
- On the backend analytical services will be integrated through an interface type that allows exchange of large amounts of data. Message based integration would produce too much overhead and is therefore not suitable for such scenarios which involves large to big data sets. Instead the integration will rather be based on data extraction, transfer and loading (ETL) or through database links.

3.4 Platform provider

The pluggable service platform acts as an intermediary between the retailer and the service provider. The goal of the platform is twofold: it should allow retailers to

source IT services and to collaborate with supply chain partners. It provides a service framework that allows provisioning and consumption of services and a process framework to implement service based process flows. Both, the meta-services of the service framework, and the services of the process framework are based on the findings in Sect. 2.3.

4 A service platform based return registration process

Based on the propositions in the previous sections we demonstrate the implementation of a service based process by realization of a prototype for a specific e-commerce case. The goal of the prototypical implementation was to assess the state of the art in e-commerce services and integration platforms. The prototype development gave insights into the feasibility of a process, based on a loose set of e-commerce relevant services and integration platforms. In a subsequent step the level of pluggability of the services in the resulting system was determined.

4.1 Business case and solution design

As the existing reference models for retail do not cover return handling processes, it can be assumed that retail ERP systems based on such models are not designed to handle return shipments efficiently. This might cause problems for multi-channel retailers that are facing high volumes of returns especially in the fashion sector (Banjo 2013). In the following we first describe a business case which covers a part of the overall return handling process. We then discuss the services used to implement the process and present the architecture of the solution.

In the scenario an end customer should be able to register a return online. Through a web page integrated into or referenced by the online shop, the user can select his order and retrieve information on the items contained in that order. The customer chooses the items and amount he wishes to return, specify the reason for the return and optionally add comments. The return request is transferred to the retailer who then authorizes the return of the material (RMA). Afterwards the return is planned by registering the expected goods and assignment of the appropriate return center. Finally the shipper is contacted by e-mail to inform him about the approval providing the link to a return label for print and possible drop of points based on the customer address. With that information the end customer can prepare the goods for shipment and bring it to the drop-off point.

Four application components are required in the scenario, each provided by individual service providers. Those include a SaaS solution, that allows customers to register their return shipments, (such as provided by 12return.nl), a generic standalone web service from a logistic service provider (LSP) that allows to register and pay shipments and to obtain the required documents (such as intraship.de). Furthermore a workflow task list that allows back office staff to approve and reject requests and finally an e-mail service provider (ESP) that delivers high volume customer communication services (such as tripolis.com) are used in the process (Table 3).

Table 3 Services used in the business case

Service	Description
Return registration	A web application that handles return shipments. It allows end customers to request a return of goods through the web interface. The user has the possibility to look up recent orders and select individual goods for return
Parcel registration	The parcel delivery registration service allows to register parcel shipments, print parcel badges and schedule parcel pick up. In the case study scenario it is used to issue parcel label to the end customer
Task list	A task list application assigns a lists of tasks to each user which they have to act upon. This can be approvals, responses or other action items. Those task list are often integrated into workflow systems and have advances features such as task forwarding, task escalation or holiday calendar integration. In this case the task list is used to assign approval notifications for the requested returns
E-mail transmission	The e-mail service is used to send outgoing e-mails to the customer including the approval and parcel label for the return shipment

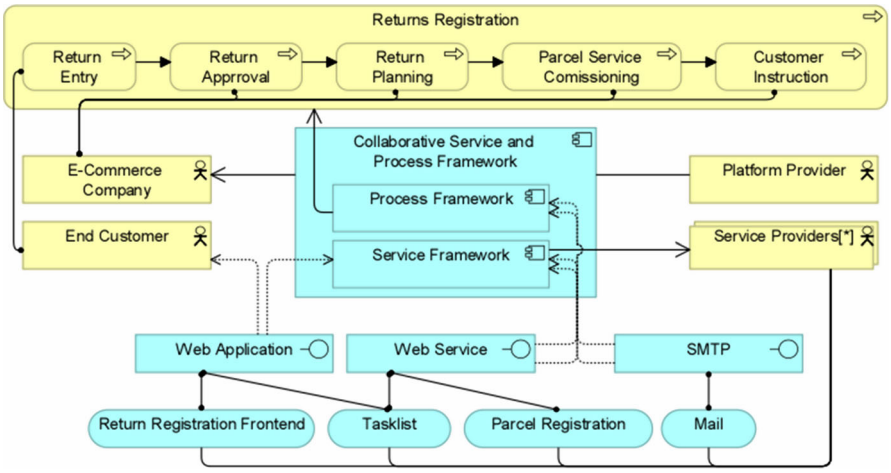

Fig. 2 Architecture of the returns registration process

Figure 2 shows the overall architecture of the solution. The pluggable platform executes the collaborative flows that make use of the various services to provide the business functions with the required functionality. The model shows that each and every service relies on the collaborative data resources required to fulfill the service. The return registration SaaS solution requires information about the orders made by the customer in the past. The same applies for the LSP and the ESP that require information on the customer. Having these resources in the platform allows adding and exchanging services to the overall process in a more flexible way.

4.2 Prototype

The services in our example are implemented using diverse technologies, are distributed among different environments, and use various protocols to

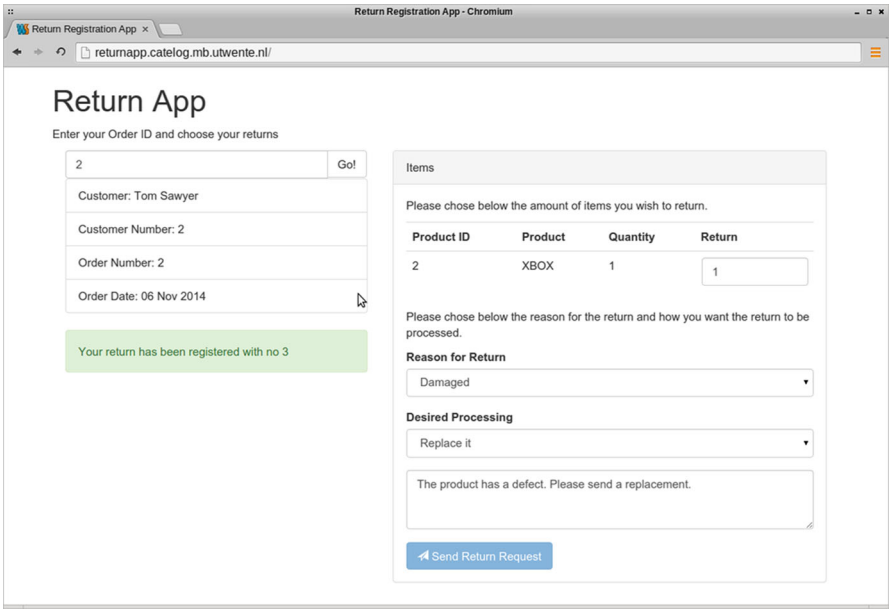

Fig. 3 Return registration app frontend

communicate with other systems. In the following we give an overview over the different services and platforms that have been used for prototype construction.

For parcel delivery registration many carriers offer their own web services to register shipments and generate parcel labels. Among those, DHL seems to be among the leaders when it comes to easy adoption of their web services, offering a developer portal and well documented services. However the interface to register parcels seems very large with 225 required data field and another 173 optional data fields exposing a lot of internals of the system which the user has to comprehend before getting the service to work. Other services such as shipcloud.io or postmaster.io evolve and facilitate the integration of various logistic service providers. Their REST APIs only have 16 data fields to achieve the same shipment label generation. The time to integrate and exchange parcel services could be reduced from 2 days to half an hour by using the interfaces of these broker services. Also switching between different carriers during runtime by requesting quotes and selecting the cheapest offer is becoming easier as the brokers cover a wide range of parcel services through the same interface.

The web application frontend for return registration is a custom made lightweight single page application (SPA) realized with common web application technologies, namely HTML, JS and CSS. Figure 3 shows the browser fronted of the application. It allows the user to enter his order number and to choose the returns from the contained items. He can further specify the return reason and type of processing. All resources including orders, customers and metadata is retrieved from the cloud database service running on a remote backend.

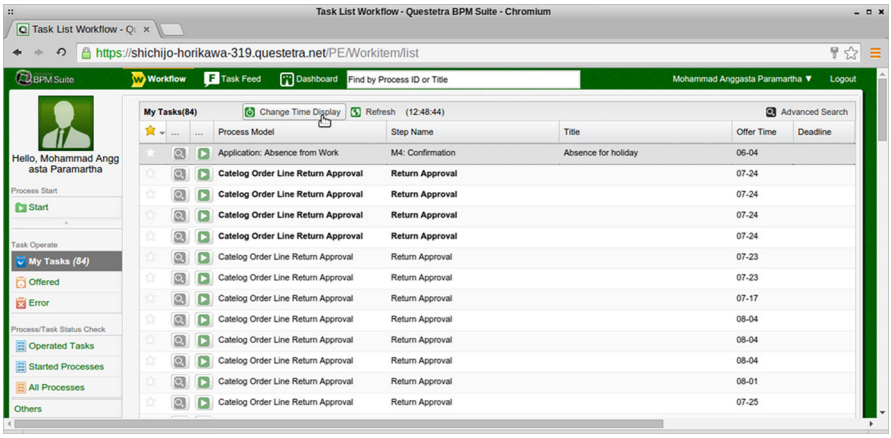

Fig. 4 Task list application with pending approvals

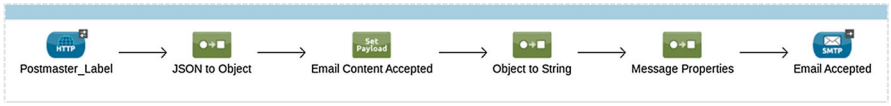

Fig. 5 Service composition using the example of the integration of parcel and e-mail services

As e-mail service the prototype uses Gmail which has an SMTP interface like any common e-mail server. In the domain of marketing communication, more specialized B2B e-mail service providers exist, such as mailchimp.com or tripolis.com which offer business specific services like e-mail templates, analytics, and different integration endpoints. However, these advanced features were not relevant for the business case and we did not find the pluggability changing significantly by using a different interface.

The prototype uses the Questetra BPM Suite for the task list as it is one of the few BPM suites we found, that is cloud based during development and run time and offers an API to integrate with other services. The screenshot in Fig. 4 shows the task list of a user, listing the assigned tasks that are awaiting action, including the pending returns waiting for approval.

The key component in the solution is the integration platform containing a service framework to plug the different services together and a process framework to execute the business processes. According to (Pezzini et al. 2014) three of those services stand out in the market with regards to their completeness. For the prototype we choose the Mulesoft CloudHub platform as it is was the most accessible in terms of documentation and subscription, which is one criteria for pluggability (cf. 'ease of service provisioning' in Sect. 5.1). Figure 5 shows how messages are routed and transformed between endpoints using the example of parcel label generation response and outgoing e-mail.

During prototype development we had to introduce another component that contains data about order, customer and product information in a cloud database. Those resources have to be available throughout the different services used in the process. In a real world scenario these information will be scattered across order management, customer relationship and product catalog systems or, in case of a more sophisticated architecture, be stored in appropriate master data management (MDM) systems. The cloud integration platform could access the database through the build in adapters. However we decided to introduce a layer of business logic build into the component which exposes the data through a REST interface. For the database and business logic we choose the cloud application platform heroku and the lightweight web application framework flask, however the same could be achieved with any other platform and web framework on the market.

5 Validation

While the prototyped process at hand offers limited complexity compared to a real world scenario and cannot be considered as a reference solution for practitioners, the goal is to test the feasibility of implementing solutions based on a set of distinct IT services using the reference architecture and state of the art cloud based integration platforms. Furthermore the main purpose of the prototype is to evaluate the pluggability of the resulting solution. In this section we are going to validate the practicability and utility of the architecture based on the prototypal implementation as well as the support of available cloud integration platforms towards the goal of a pluggable system.

5.1 Pluggability

The concept of pluggability can be considered as a quality characteristic of information systems, equivalent to reliability, efficiency, or maintainability, specified in software quality standards such as the ISO/IEC 9126. We deduced the software characteristic of pluggability from the lack of the traditional models to reflect the external quality criteria of IT services (Aulkemeier et al. 2015). It was formally defined as ‘a quality characteristic that describes the external criteria of a service which facilitate its adoption in a specific context’. Its criteria are guided by the life cycle of service adoption. The life cycle consists of six phases, namely provisioning, deployment, adoption, integration, operation, and exchange of the service. The criteria are described in Table 4. Based on the six criteria an instrument was proposed that allows to assess the pluggability of a service. Appendix 1 contains the predefined levels for each criteria which is used in this study. The criteria have been evaluated in cooperation with practitioners in the field of IT service integration. The application of the predefined levels to assess the prototype presented in this paper is also a means to further validate the instrument.

Table 4 Criteria for pluggability

Criteria	Description
Ease of service provisioning (EOP)	The goal of service provisioning is to discover potential services, compare the various available services, to assess individual services with regards to the business needs, and to enter into a contract with the service provider. The service provider can facilitate these task by listing the service in various service marketplaces, disclose all the relevant information publicly including the terms of use, pricing, service levels, and documentation. Furthermore, the access to demo environments and self-service subscription can further facilitate the assessment and comparison of services
Ease of service deployment (EOD)	Individual services should be easy to install, learn and test. By default cloud based services do not require technical testing and installation and thus, have an inherent advantage over traditional software components with regards to deployment. In any case the service should support learning and functional testing through high quality and accessible documentation
Ease of service adaptation (EOA)	The service should be easy to adapt to the functional needs of the consumer. This includes the ability to configure and customize the service. Customizations have a higher level of technical complexity as additional or deviating logic has to be implemented. Configuration in contrast leverages existing logic through setup. A pluggable service maximizes configurability while reducing the need for customizations
Ease of service integration (EOI)	Services should be able to communicate and share data mutually in order fulfill the overall business process. The construction of dedicated interfaces between services is labor intensive and should be supported by the service provider, for example through adapters or service platforms. Services should be able to share and exchange resources without other service quality criteria being affected, especially EOD, EOA and EOE
Ease of service operation (EOO)	Service operation encompasses the long-term tasks to enable the continuous use of a service, namely maintenance, monitoring and customer support. Service providers can facilitate service operation by providing service level agreements for availability, bug fixing and change requests, as well as a suitable infrastructure such as call centres, a bug tracking systems, and support portals. In order to provide a single point of contact across services, a joint service infrastructure can further improve the EOO
Ease of service exchange (EOE)	Loose coupling is a fundamental principle of service-oriented architectures and requires services to act as independent units of computing. The benefit is to facilitate the exchange of individual services. However, loose coupling of services requires dedicated service orchestration which affects the EOI

5.2 Observations

In order to measure the pluggability of the five services the instrument was applied individually by the developer of the prototype, two scholars and two practitioners from a Dutch iPaaS provider. The prototype was presented to the participants as well as evaluated against the predefined levels for pluggability in Appendix 1. The graph in Fig. 6 shows the average score for each pluggability criteria and for each of the five services as well as the average overall pluggability of each service.

The service with the lowest overall pluggability is the return registration frontend. It is a custom-built service which is hosted on-premise. The cloud

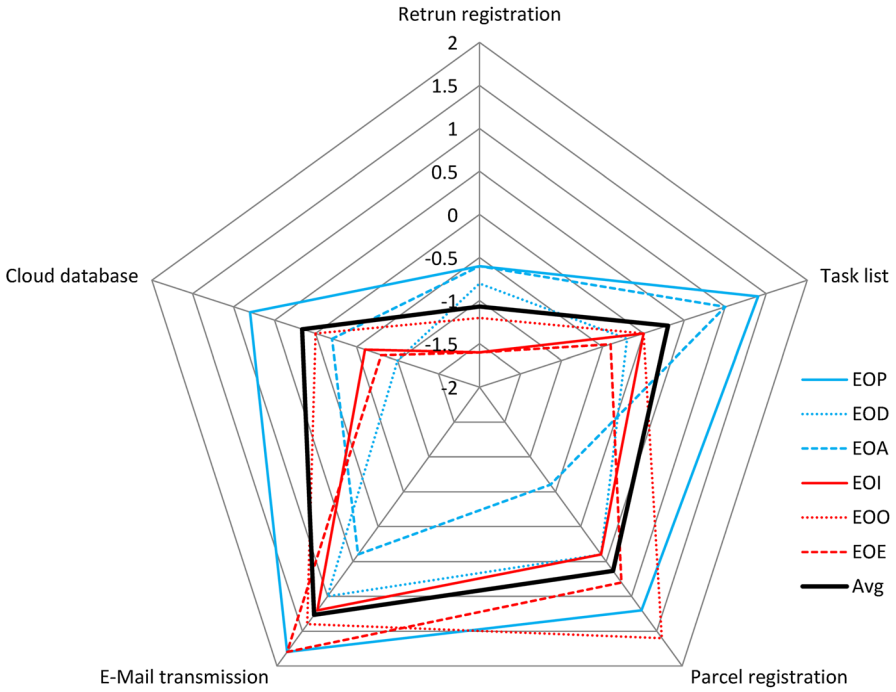

Fig. 6 Average score per pluggability criteria and average overall pluggability

database is a custom-built solution but deployed on a PaaS environment which seems to be beneficial for the overall pluggability. In the mid-level of the overall pluggability is the tasklist solution which is a PaaS that follows a model driven approach to process implementation. Its non-coding approach seems to be beneficial for the overall pluggability. Finally the e-mail transmission and parcel registration can be considered as SaaS solutions and exhibit the highest level of pluggability.

The EOD is high for the parcel delivery service and the e-mail service as those services are installed and maintained entirely by the service providers. The EOD is low in contrast for the return registration frontend and the cloud database as the deployment and management is carried out by the retailer. The workflow solution is operated by the service provider but the actual processes running on the platform have to be developed, tested and monitored by the service user.

For the workflow task list application the EOA is high because the used BPM service provides a model driven approach to implement business logic. The user has very high flexibility to adopt the service to his needs without having to customize the system. The same applies for the e-mail service which is flexible to the extent that the content of the e-mail is concerned. The return registration frontend and the cloud database are adaptable but need technical expertise to implement customizations. The parcel registration service also has a low level for EOA because it delivers specialized services which are not adaptable.

All four services of the prototype which are based on cloud platforms have a high EOP. This indicated that cloud service providers in general are doing a good job in documenting and providing potential user with resources to assess their services. The low EOP for the custom developed return registration service however is difficult to explain. We think that the concept of provisioning might not be applicable to custom developments as the task of a fit gap analysis does not have to be carried out in that case. The provisioning is then happening on the technical level which might explain the low EOP.

The EOO is low for the return registration fronted because it requires monitoring of the application and the underlying infrastructure as well as support and maintenance of the application. The use of PaaS for the cloud database and workflow increases the EOO as basic infrastructure is handled by the service provider. However application maintenance and support is still in responsibility of the service user. The other two services have a high EOO as the entire service operation is outsourced.

The parcel delivery and e-mail service have a high EOE both for distinct reasons. The protocol used by the e-mail service provider is highly standardized and used by any other service provider. Therefore the migration to another services has minimal impact. The exchange of the parcel delivery service with another service can be achieved easily too. We have carried out this exercise by changing the postmaster.io service with shipcloud.io which is only a matter of changing a couple of fields and the service endpoint. The return registration frontend can be exchanged without impact on any other service in the architecture. However it is relying on other services such as the cloud database to retrieve order and customer information. Introducing another service for this purpose require it to be adopted to the interface of these services or the service bus respectively. Exchanging the cloud database or the workflow system however impacts the entire architecture which also leads to a low EOE.

The EOI is the criteria with the lowest average score across all services. The return registration service, the task list application, as well as the cloud database have a very low EOI due to the complexity of their interfaces. Each of the services requires throughout design, build and maintenance of custom interfaces to connect to the other services. While the parcel delivery and e-mail service require the same, their interfaces are rather simplistic so that the EOI is a bit higher.

5.3 Improvements on the state of the art cloud integration platforms

In the previous section we have investigated to what extend the services and the architecture of the prototype adhere to the criteria of pluggability. We discovered that the ease of integration is the criteria that the current services and integration platforms lack of. We see the reason for this shortcoming in the complexity of the task of developing and maintaining the interfaces between the various software services.

Service implementers can easily gain a good understanding of the reference processes, data models and use cases required to implement a certain application component. It allows them to deliver services that can be used in a wide variety of

organizations. However, service providers have no insights into the environments in which the services eventually operate, which may be the explanation of the lack in delivering the appropriate integration artifacts. Furthermore those system landscapes vary across the different potential service users which poses another obstacle to delivering those artefacts.

To address this issue we propose to adjust the architecture of the current integration platforms to facilitate the task of delivering pre-integrated services. As with cloud based software services that release the user from struggling with the underlying technology a suitable integration platform allows the users to reduce their workload in service integration from customization to configuration.

(Baldwin and Woodard 2009) describe the goal of a platform to provide 'a set of *stable components* that supports variety and evolvability in a system by constraining the linkages among the other components'. In an application landscape with evolving functionality the most obvious stable component is the data used throughout the system. This data is the same throughout the different types (integration can be seen as the task of transferring data from one system to another) and different generations (migration can be seen as the task of transferring data from one system generation to its successor) of services. However this aspect is ignored by all of the investigated integration platforms.

As a result to the lack of current platforms and the indicated integration challenges we claim that the business data should be part of the platform rather than

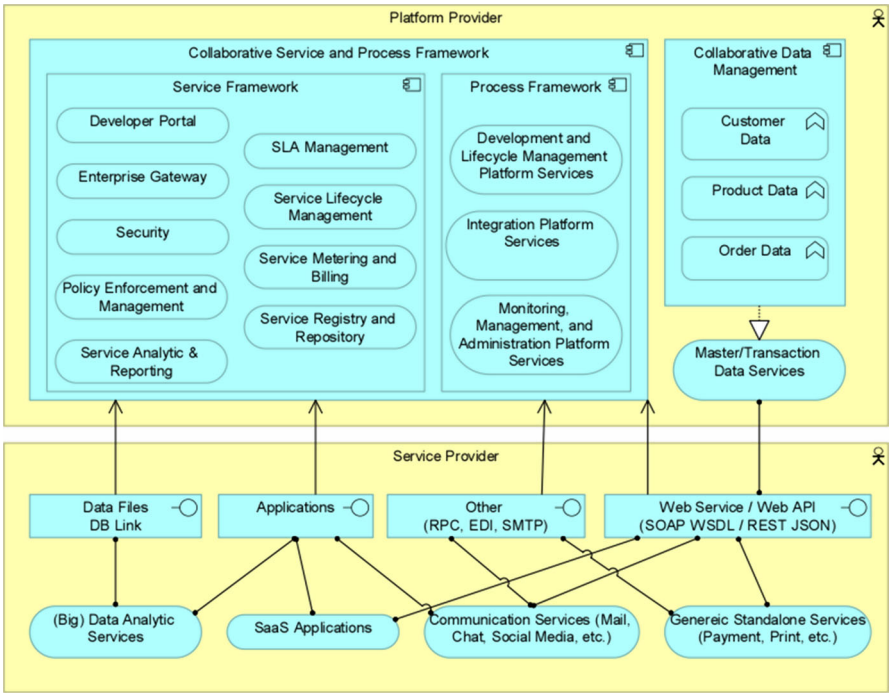

Fig. 7 Evolution of the pluggable service platform model (cf. Fig. 1)

the individual services. The integration platform becomes a domain specific artefact, including a canonical data model (Hohpe and Woolf 2003) and the required services to help service providers to ship pre-integrated services. In Fig. 7 we show and extended version of the state of the art model and add the collaborative data management component to the platform provider. By introducing this concept we shift the component that is most critical for the integration of systems from the service to the platform. This component handles the canonical data that is used, and provides an interface to these resources. In the context of this study individual e-commerce services evolve around the platform. They can operate by default on the data service and thus, allow their users to skip the integration efforts. Furthermore it is possible for the processes to access the canonical data through the same interface, which allows to integrate legacy services that are not participating in the use of the data service.

6 Conclusion and future research

In this paper we have shown how the IT service industry currently approaches the issue of plugging software services into existing environments. The presented reference model was implemented based on a simplified real world scenario. Using the pluggability assessment model we found the ease of integration one of the main challenges service users face today. The proposed assessment model for pluggability can be considered as an ad-hoc version of a quality model for software services, which we will be subject to future research.

Furthermore, we propose an extended reference model that can improve the ease of integration and is in line with the platform concept. The extension consist of a canonical data management component containing the data that has to be shared among the various services. However, the introduction of such a component will have consequences with regards to the handling of the shared data. At this point various scenarios are possible how existing and new services deal with the centralized data repository. While new services can interact directly with the data services, the link leads to a strong dependency between service and platform. Furthermore the availability of platform compatible e-commerce services will be limited unless the platform is gaining strong support from service providers. Furthermore the adoption of the platform requires integration of existing services and thus a strong commitment and initial investment from the e-commerce company.

The construction of the collaborative data management component itself should be subject to further research. First, the canonical data model can be further specified based on the various existing reference models in the field. Furthermore the data access level has to be defined to allow the various partners to work collaboratively and assure protection of sensitive information at the same time.

The goal of future research is to design these components that will undergo prototypical implementation and evaluation. Finally other shortcomings of the state of the art integration platforms such as ease of deployment and ease of operation have to be addressed in the extended reference model.

Appendix 1: Predefined levels for pluggability criteria

	Low	Medium	High
EOP	Detailed information on the service is only available through individual contact with the service provider	Documentation and information about the service can be requested and is made available according to a transparent process	All required information including documentation, pricing and demos are openly available
EOD	Technical expertise such as development or scripting is needed to make the service operable	The deployment does not require any technical expertise but complex setup and configuration	The service can be used straight away through subscription
EOA	The service can hardly be adapted for use cases that have not been specified by the service provider	The service can be adapted to any use case scenario but needs technical expertise to do so	The service can be adapted to any possible use case through configuration or setup
EOI	The integration of the service into the landscape requires coding or scripting	The integration of the service into the landscape requires configuration or setup	The service is automatically integrated into the landscape and requires no further action after deployment
EOO	Monitoring, maintenance, and customer service have to be carried out by the service user	Monitoring, maintenance, and customer service are partly handled by the service provider	Monitoring. maintenance, and customer service are entirely handled by the service provider
EOE	Exchanging the service impacts other services and requires development or scripting	Exchanging the service impacts other services but can be handled through reconfiguration or setup	Exchanging the service does not impact any other service

References

Aulkemeier F, Iacob M-E, van Hillegersberg J (2015) Pluggable SaaS integration: quality characteristics for cloud based application services. In: Enterprise Systems Conference (ES), Basel

Aulkemeier F, Schramm M, Iacob M-E, van Hillegersberg J (2016) A service-oriented e-commerce reference architecture. J Theor Appl Electron Commer Res 11:1–20

Baldwin LP, Currie WL (2000) Key issues in electronic commerce in today's global information infrastructure. Cogn Technol Work 2:27–34. doi:10.1007/s101110050004

Baldwin CY, Woodard CJ (2009) The architecture of platforms: a unified view. In: Gawer A (ed) Platforms, markets and innovation. Edward Elgar, UK, pp 19–44

Banjo S (2013) Rampant returns plague e-retailers. Wall Str J

Becker J, Schutte R (2007) A reference model for retail enterprises. In: Fettke P, Loos P (eds) Reference modeling for business systems analysis. IGI Global, pp 182–205

Burt S, Sparks L (2003) E-commerce and the retail process: a review. J Retail Consum Serv 10:275–286. doi:10.1016/S0969-6989(02)00062-0

Chu S-C, Leung LC, Hui YV, Cheung W (2007) Evolution of e-commerce Web sites: a conceptual framework and a longitudinal study. Inf Manage 44:154–164. doi:10.1016/j.im.2006.11.003

Cross N (2006) Designerly ways of knowing. Springer, London, pp 95–103

Frank U, Lange C (2004) ECOMOD—reference business processes and strategies for e-commerce. Research Group Enterprise Modelling, University Duisburg-Essen

Frank U, Lange C (2007) E-MEMO: a method to support the development of customized electronic commerce systems. Inf Syst E-Bus Manag 5:93–116. doi:10.1007/s10257-006-0034-9

Gunasekaran A, Marri HB, McGaughey RE, Nebhwani MD (2002) E-commerce and its impact on operations management. Int J Prod Econ 75:185–197. doi:10.1016/S0925-5273(01)00191-8

Hohpe G, Woolf B (2003) Enterprise integration patterns: designing, building, and deploying messaging solutions. Addison-Wesley Longman Publishing Co, Boston, pp 355–360

Humeau P, Jung M (2013) Benchmark of e-commerce solutions. NBS System. https://www.nbs-system.co.uk/blog-2/benchmark-of-e-commerce-solutions.html. Accessed 13 Sept 2015

Kleeberg M, Holger K (2014) Information systems integration in the cloud: scenarios, challenges and technology trends. In: Brunetti G, Feld T, Heuser L et al (eds) Future business software. Springer, Cham

Koppius OR, van de Laak AJ (2009) The quick-connect capability and its antecedents. In: Vervest PHM, van Liere DW, Zheng (eds) The network experience. Springer, Berlin, Heidelberg, pp 267–284

Lankhorst MM, Proper HA, Jonkers H (2009) The architecture of the ArchiMate language. In: Halpin T, Krogstie J, Nurcan S et al (eds) Enterprise, business-process and information systems modeling. Springer, Berlin, pp 367–380

Liu D-R, Hwang T-F (2004) An agent-based approach to flexible commerce in intermediary–centric electronic markets. J Netw Comput Appl 27:33–48. doi:10.1016/S1084-8045(03)00039-0

Maler E, Hammond JS (2013) API management platforms, Q1 2013. In: The forrester wave. Forrester research. https://www.forrester.com/fulltext/fulltext/-/E-RES81441. Accessed 13 Sept 2015

Malinverno P, Plummer DC, Van Huizen G (2013) Gartner magic quadrant for application services governance. Gartner Inc. https://www.gartner.com/doc/2571325. Accessed 13 Sept 2015

O'Leary DE (2000) Enterprise resource planning systems: systems, life cycle, electronic commerce, and risk. Cambridge University Press, Cambridge

Peffers K, Tuunanen T, Rothenberger MA, Chatterjee S (2007) A design science research methodology for information systems research. J Manag Inf Syst 24:45–77. doi:10.2753/MIS0742-1222240302

Pezzini M (2011) Integration PaaS: enabling the global integrated enterprise. Paper presented at the Gartner Application Architecture, Development & Integration Summit, Las Vegas, 29 Nov–1 Dec 2011

Pezzini M, Natis YV, Malinverno P et al (2014) Magic quadrant for enterprise integration platform as a service. Gartner Inc. https://www.gartner.com/doc/2657018/. Accessed 13 Sept 2015

Potočnik M, Juric MB (2012) Integration of SaaS using IPaaS. In: Proceedings of the 1st international conference on cloud assisted services. Bled

Ried S (2014) Hybrid integration, Q1 2014. In: The forrester wave. Forrester research. https://www.forrester.com/fulltext/fulltext/-/E-res111881. Accessed 13 Sept 2015

Schepers TGJ, Iacob ME, Van Eck PAT (2008) A lifecycle approach to SOA governance. ACM Press, New York, p 105

van Heck E, Vervest P (2007) Smart business networks: how the network wins. Commun ACM 50:28–37

van Hillegersberg J, Moonen H, Dalmolen S (2012) Coordination as a service to enable agile business networks. In: Kotlarsky J, Oshri I, Willcocks LP (eds) The dynamics of global sourcing. Perspectives and practices. Springer, Berlin, pp 164–174