Web Oriented Architectural Styles for Integrating Service e-Marketplace Systems

Robert Kutera and Wiesława Gryncewicz
Department of Information Systems, Wroclaw University of Economics, Wroclaw, Poland
{robert.kutera, wieslawa.gryncewicz}@ue.wroc.pl

Keywords: Service e-Marketplace, Web-Oriented Architecture, Web Services, REST, Integration.

Abstract: e-Marketplaces for services have faced many challenges in the last few years. Many factors, including user expectations, service specificity or technological development, influence the scope of requirements for integrated solutions on that market. Such an integration can help in creating the place in the cyberspace, where any user obtain an access to any service he or she actually needs. However, the integration isn't an easy task, especially in such a diversified environment. That is why there is a need to find a suitable way of integrating the service e-marketplaces. Using loose coupling and WOA principles can make this task easier, what was proved during the real-life experiment, using the "consortium research" method. In the paper authors proposed the concept of integration of service e-marketplaces that bases on three foundations: user interface, data & business logic and security. In that context authors described the integration procedure which takes into account not only the classic software lifecycle, but also agile development principles. Having such foundations, verification of this procedure on the example from ActGo-Gate research project was performed. The high-level architecture of the project IT solution was presented and the way of integration of the offer synchronization process was described.

1 INTRODUCTION

The technological progress and, most of all, the IT revolution have significantly affected the operation of the service markets. Many services have gained their electronic form, while others – which are difficult or impossible to provide remotely – have been given a broad range of visual and interactive solutions (including communication and transaction ones). Thanks to them the consumer can become familiar with the description of a given service, engage in a discussion with the service provider's representative or other customers of the company, place an order and pay online (Łysik et al., 2015).

e-Marketplaces are more and more fragmented, customized and personalized (Maciaszek et al., 2017). There are three most important factors determining the development of service e-marketplaces:

- user expectations and behaviours,
- service specificity,
- technological development.

Nowadays users have very simple access to different information. Thanks to the Internet and ICT technologies they have very good market recognition. On the other hand people don't have much time, so they expect to receive personalized offers in one place, effortless and immediately.

Services are very differentiated and varied. Some of them require coordination of appointments (one-time services), the others use recruitment and evaluation process (recurrent or long-term services) (Maciaszek at al., 2017).

There are many service providers which offer different services. They need efficient, quick and effective technology solutions, especially in case of increased traffic on web pages. That is why design patterns are implemented. The main elements of design patterns are typical and effective development solutions which relate to one or more regularly occurring problems (Fowler, 2003). The key feature of design patterns is that they are rooted in practical experience. They provide advices for developers how to proceed in such cases to work more efficient, transparent and orderly. Design patterns are often not original ideas, but rather an attempt to systematize observations of what is routinely used in practice. They consist of many implemented functions that can be called with one command (Gamma et al., 2008).

In order to streamline and optimize the web applications development process, design patterns are also reflected in web frameworks like Laravel, Symphony (PHP), Spring (Java), Django, Flask (Python) or AngularJS, ReactJS (JavaScript). The web framework consists of a collection of software components that help developers create and execute web-based user interfaces (Vosloo and Kourie, 2008). The framework manages the content displayed on the web interface, the pages that are to be displayed, and what actions are available to the user of the page. They also standardize the ways of communication between web applications.

Taking into consideration all these three factors: very personalized and precisely defined user expectations; different kinds of services and various types of available programming patterns and technologies - there is a need of a specific kind of integration in the service e-marketplace area.

This integration should take into account a set of requirements:

- viewing and managing of data in one place,
- ease of access to data (Single Sign On),
- safety,
- consistency of data presentation,
- a user-friendly interface,
- keeping current information and reminding about any user events.

The above-presented list of requirements has been established on the basis of analysis related to the feedback from scientific and research projects conducted by the authors and the development of web applications.

The aim of this paper is to present a concept of integration of service e-marketplace systems using web oriented architectural styles. The paper is a part of the ActGo-Gate project funded by the AAL Agency awarded on the basis of the agreement number AAL6/1/2015.

2 RELATED WORK

There are different types of integration. Among them the most common are (Hohpe and Woolf, 2004):

- information portals,
- data replication,
- shared business functions,
- service-oriented architectures,
- distributed business processes,
- business-to-business integration.

It is not a complete classification, it illustrates the kind of solutions that integration architects build. *Information portals* aggregate information from different systems into a single display. User may happen when many business systems require access to the same data. When a user change something in one system, all the other systems need to change this element. To avoid redundant functionalities developers can use a *shared business function* that is implemented once and available as a service to other systems. *Service-oriented architecture (SOA)* is applied when enterprise gathers a collection of useful services and managing them becomes a critical function. Remote services provided by other applications ae integrated as *distributed business processes*. A *business-to-business integration* manages the execution of a business function across multiple existing systems.

The paper is based on the experience from ActGo-Gate project. In this project developers have used a few types of integration simultaneously: information portals, shared business functions, but mostly SOA. It provides the basis of distributed application frameworks in which software components are delivered as modular and reusable services. The benefits of a SOA approach are evident in the flexibility of business processes involving loosely coupled services and resulting potential cost decrease, reduced complexity, reusability, and high flexibility (Thies and Vossen, 2009).

A software architecture style which extends SOA to web-based applications is called Web-oriented architecture (WOA). This architecture emphasizes generality of user interfaces and Application Programming Interfaces (APIs) to achieve global network effects.

Pautasso (2014) draws attention also to the Representational State Transfer (REST) architecture. This architectural style emphasizes the scalability of component interactions and promotes the reuse and generality of interfaces. It decreases also coupling between components. The basic principle of loose coupling is to reduce the assumption that two parties (components, applications, services, programs, users) exchange information with one another (Fowler, 2003). Although REST is usually chosen to build simple CRUD (create, retrieve, update and delete) services, there is a possibility to develop REST web services offering complex services and stateful behavior (Rauf, 2013). REST comes with 4 basic principles like: using HTTP methods explicitly, being stateless, exposing directory structure-like URIs and transferring XML, JavaScript Object Notation (JSON), or both (Fielding, 2000).

In practice, the web service typically provides an object-oriented web-based interface to a database server, utilized for example by another web server, or by a mobile application, that provides a user interface to the end user.

3 RESEARCH METHOD

During the research process two main research methods were used. First of them was literature review, that included both research and professional books and articles from the area of software engineering and project management that helps to build a scientific foundation for the paper.

The applied part of the article is based on the Consortium Research method (Österle and Otto, 2010a) (Österle and Otto, 2010b). It is the method of developing research artefacts which applicable in real-life environments. It is an extension of the Design Science research method (Österle et al, 2011) (Myers and Venable, 2014), which is widely used in IT & business-related research projects. It bases on the cooperation of practitioners and researchers in order to develop artefacts possible to be applied in real-life use cases.

The obtained results are coming from such a cooperation within a multinational consortium consisted of research and business partners.

Business partners are supporting the research and testing the developed research artefacts in the real-life use cases of service e-marketplace within local communities, while researchers, beside the creation process, analyse the feedback and improve the previous results. The result from the research are real-life pilot implementations in three cities in Germany and Switzerland, which support their users by offering them different occupational opportunities.

4 THE CONCEPT OF WOA INTEGRATION FOR SERVICE E-MARKETPLACE

With the evolution of the consumer behaviours in the Internet, service e-marketplaces gain more and more popularity. Analyzing the previously identified requirements there seems to be a strong need for contributing solutions, which are integrating different service providers at one place, with convenient and secure access by the user. Such a solution should take into account the differentiation of the essence of particular services and their processing. It influences on the extent to which the particular processes could be integrated. The conclusions obtained during the realization of the research project help in determining the foundations of the integration of service e-marketplace.

The conceptual illustration is presented in Fig.1.

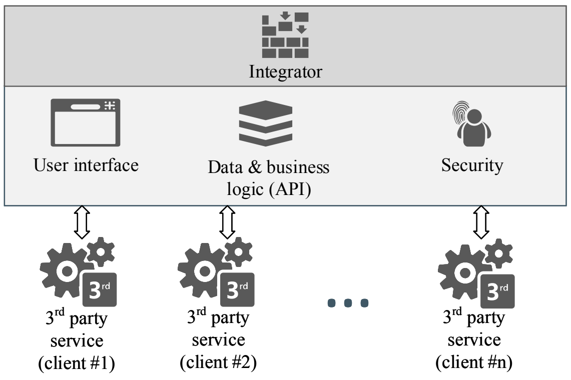
Integrator
User interface
Data & business logic (API)
Security
3rd party service (client #1)
3rd party service (client #2)
3rd party service (client #n)

Figure 1: The high-level concept of integration of the service e-marketplace.

That concept indicates three main foundations for integration of service e-marketplaces (clients) under one system – the integrator (Maciaszek et al, 2017):

- User interface – assuring, by the frontend part of the application, the rich and seamless user experience when using an integrator. It should make the impression of using single and compact application, not only the gate for different 3rd party services / clients.
- Data & business logic – utilizing basic data flows, like messaging or asset (category, offer) data retrieved from connected 3rd party services / clients in common data views.
- Security – securing the communication (e.g. with reliable token-based solutions and using safe SSL channel) and managing user identity centrally, where a user can log in once and have access to all services (incl. clients) of the platform (Single Sign On). It should utilize popular and reliable mechanisms of user authentication and manage permissions within the whole ecosystem.

While the first foundation is related only to front-end adaptations within a particular system in compliance with the given standard, the two deeper ones depend strongly on data exchange and communication between clients and the integrator (Maciaszek et al, 2017).

The communication integrator – 3rd party service within data and security integration should be based on WOA principles and loose coupling paradigm, which meet the marketplace requirements in the most complete way. In fact, dependent on the function analyzed, the communication could be initiated with a particular request by both sides.

The basic communication schema is presented in Figure 2:

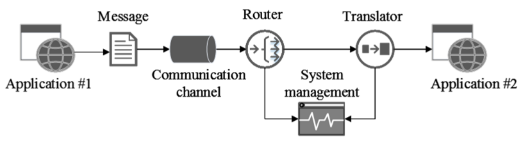

Figure 2: The communication between application within the integrated solution (Hohpe, 2004).

The communication process that occurs between two applications within the integrated system, begins with construction of the message. It should be possible to understand by both sides in the given context and needs to be transported via common communication channel, which is HTTP in case of REST web services. REST uses HTTP standard methods (GET, POST, PUT, DELETE, OPTIONS) to call for resources. Combination of the method, the Uniform Resource Identifier (URI) and some additional parameters (required or optional) sent via HTTP causes that the particular operation (explicitly indicated by the chosen method) is performed and the response message is sent with requested resource in various formats: Text, JSON or XML (JSON is the most widely used). Router in that process could be responsible for redirecting messages to the correct places. On the other hand, the translator's role is to decode the message, validate it and convert it to the form acceptable by the other application. System monitoring is a supportive actor in the process and is responsible for overseeing the data flow, verifying the status of all participants of the process, logging and reporting errors.

Service e-marketplaces require a set of endpoints within two last foundations previously mentioned: data & business logic as well as security. The first one involves mostly endpoints for each objective entity (like category or offer, where operations are defined by particular HTTP methods), while the second one – user endpoint and all supportive entities which help to authenticate him and grant proper privileges. Among those entities common structural elements should be defined and they should be treated as the basis for the integration. When, for example, the integrator is allowed to call directly the 3rd party's web service of creating a new instance of the entity (a concrete object) with the set of common data, this web service will be responsible for filling the missing specific data for that 3rd party application. It can do it automatically, or ask user for necessary additional information. The data & business logic foundation in more homogenous environments could also handle with integration of transactional entities (like appointment, order, recruitment, event, project etc.), however in general those entities are fundamentally diversified (as mentioned in the introduction) and there is a real challenge to find the commonly accepted solution for that problem. That is why in such systems it is allowed to let some processes to be performed only in 3rd party services (and use the information portal integration type for displaying 3rd party's view within the integrator view), which are fully specialized in such processes.

5 THE CONCEPT OF THE INTEGRATION PROCEDURE

The process of software integration is very similar to every other software development process, however usually involves more actors as well as requires more arrangements and iterations caused by unexpected problems on 3rd party side. That is why common understanding on all sides is very important factor for the final success of the integration. It can be achieved via preparing good documentation and using popular web standards and design patterns in the implementation.

Classic software development lifecycles are usually performed in compliance with architecture-first design paradigm (Booch, 2007). Initial architecture should provide fundamental concepts or properties of a system and usually consists of elicited elements, relationships, and the principles of its design and evolution (IEEE/AWG, 2017): However, the rapid expansion of agile software development techniques in recent years caused that the initial concepts are changing during the project. The short development cycles in agile (Agilemanifesto.org, 2001) and lean techniques (Poppendieck and Poppendieck, 2003) are bringing new functions / new versions of the application which can be quickly implemented, deployed and validated each time by real users or project collaborators. The input from real-world tests provides architects and developers with new requirements (remarks to the way of satisfying already known needs as well as new needs which are emerging). Those new requirements can change the initial concept significantly from the original version. That is why some agile techniques diminish the importance of the rich architectural documentation and limit it to the minimum. However, still the architecture is the most important way of getting the common clear understanding how the system works for all stakeholders, even non-technical ones.

Some agile techniques, like Agile Model-Driven Development (Eliasson et al., 2014) (Kulkarni et al (2011) has successfully combined agile with the approach of defining the abstract specification, which finally will be transformed into the implementation in compliance with the given architecture. Taking their experiences as well as the concept of WOA-oriented architecture modelling proposed by Thies and Vossen (2009) a procedure of developing an integrated system is proposed (Fig. 3).

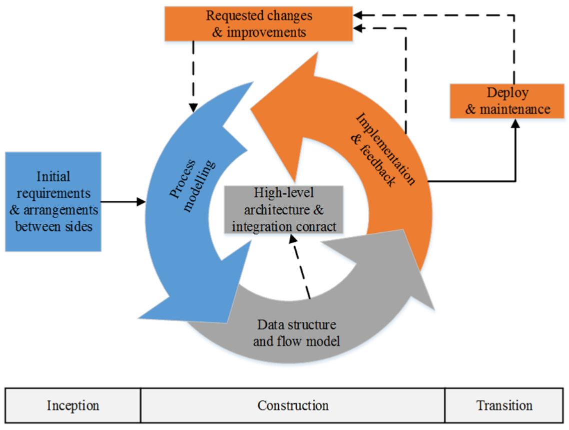
Requested changes & improvements
Deploy & maintenance
Initial requirements & arrangements between sides
Process modelling
Implementation & feedback
High-level architecture & integration conract
Data structure and flow model
Inception
Construction
Transition

Figure 3: The procedure of developing an integrated system in Agile approach.

The proposed concept is based on Disciplined Agile Delivery (DAD) framework (DAD, 2014) It is a three-phase lifecycle, where a consumable solution is built over time. The process of integration begins with the inception phase, where the first requirements are gathered. In the meantime first arrangements between integration sides are made. Usually those arrangements are the basis for the integration contract, which finally will include basic parameters and uniquely identifies the access interface.

Having such a starting point the first iteration of construction phase can be started where the integrated business processes are modelled, usually in commonly understandable notation like BPMN (Business Process Modelling Notation). The involved systems and business roles should be identified and the detailed processes should be constructed. That step will show the complexity of the particular processes and help to identify all parties involved.

In the second step the basic data structures and flows are designed. The achievements of this step, together with the high-concept architecture schema, constitute the architectural documentation of the integrated system. All tasks within the flows, which require communication of at least two parties, should be described in a way that was arranged in the inception phase by all parties. The WOA concept is promoting an easy way of documenting APIs and therefore the WADL (Web Application Describing Language) gains popularity. The well-structured documentation in commonly-understood notation is an important factor for the better performance of the whole project. There are also many tools that enable the automation of the API documentation (like Swagger, Apiary, Apigee, Sphinx and so on), which could be useful especially in large-scale projects..

Third step is the implementation of the previously documented APIs by all sides involved and consequently bringing the WOA to life. The API owner implements proper web services and binds them to the process while other parties are implementing the necessary changes on their side, which make it possible to consume API. When the process is completed a series of tests should be performed to indicate possible errors in the communication between parties or in the data processing. The results from tests should be gathered and requested changes and improvements in particular processes should be reported. That will constitute the foundation for performing the next iteration within the construction phase. First iteration usually covers the most important processes and structures, while further ones introduce next features.

The third phase, the transition, includes the deployment and maintenance of the developed system. In a result the integrated system is available to its stakeholders who are allowed to use it in practice. According to lean software development technique, this phase should be initiated as soon as possible with a minimum viable product (MVP). MVP usually offers basic processes and is usable in its narrow scope. Then users can provide feedback from real-life usage. Their remarks could lead to important decisions about changes. New requirements are then provided to the construction phase which run the iteration with taking into account the new setup. If there are any changes affecting the high-level architecture, it also should be adapted to the new requirements to keep the architectural documentation up-to-date, what is extremely important for assuring the quality of the continuous development.

6 IMPLEMENTATION – AN INTEGRATED SERVICE E-MARKETPLACE FOR OCCUPATIONAL OPPORTUNITIES

Service e-marketplaces and their integrations generate many challenges for software architects and developers. This chapter contains the demonstration how the practical problem was solved in the particular service e-marketplace, using WOA architectural style. The essence of the e-marketplace, which is taken into consideration, is to provide occupational opportunities to elderly people. Specific offers are provided to them by separate specialized services: appointment coordination system (processing such types of offers as services and demands, developed in Python) and recruiting service (processing voluntary jobs, developed in Java). Users should be able to browse occupational offers as well as create their own offers and manage them. They should also be able to perform transactions (request for appointment, apply for job) and receive notifications about particular actions within the system. The application provides Single Sign On feature to users. More about this topic was discussed in (Kutera, 2016). The integrating application is built mainly on top of Laravel (PHP) and AngularJS (JavaScript) frameworks. The high-level architecture of the integrated solution is presented in Figure 4.

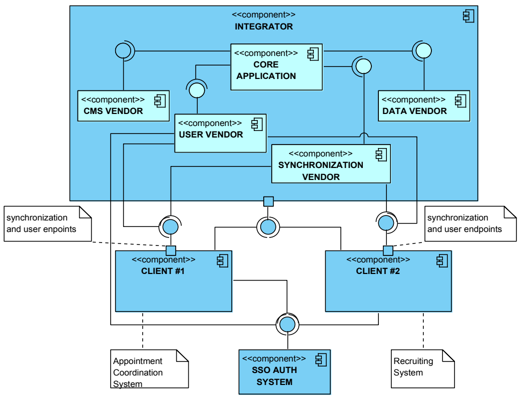

Figure 4: The high-level architecture of the integrated solution.

Because of the scale of the project for demonstration purposes one representative endpoint was chosen. The endpoint for offer synchronization is most important from utilitarian point of view. It allows for gathering offers from different sources and enables them to be displayed in a convenient form for users. It also allows for keeping the offer data structure consistent.

The previously described procedure for building the integrated solution for service e-marketplace begins with eliciting initial requirements for the integration process in the inception phase. For demonstration purposes they have been limited only to those connected with offer endpoints:

- Client applications repository,
- Adding an entity of neighborhood with its mapping with client neighborhoods,
- Central categorization of offers with category mapping from client categories,
- Support for registration of new offering,
- Support for management/editing of offerings via the Gate application,
- Search for offers with advanced tools,
- Browsing the catalogue with available filters,
- Presence of the common map view,
- Availability of news feed.

Not all of them are affecting the direct process of offers synchronization so they will be omitted in further discussion.

The most important business rules for the offer entity are following:

- offer has to be assigned to at least one neighborhood and category,
- offer has to be mapped with client offer index,
- offer has to be owned by one or more users;
- offer is obliged to have geographical coordinates set for displaying it on map,
- offer which don’t have any active owner or assignment to an active category or neighbourhood has to be softly deleted.

Next, there is a basis for running the first iteration of the construction process. First, the business process is being modelled (Fig. 5 and Fig. 6). The most important is the fact, that before the direct request for offer is sent, neighborhoods and categories are synchronized (Figure 5). The synchronization process is invoked automatically with using CRON mechanism at a certain frequency. One of the assumptions, arranged in the integration contract, is the focus on the transmission of the small pieces of information frequently to keep the consistency of the data within all systems. That is why the data flows contain only information about recent changes (added as well as modified or deleted offers).

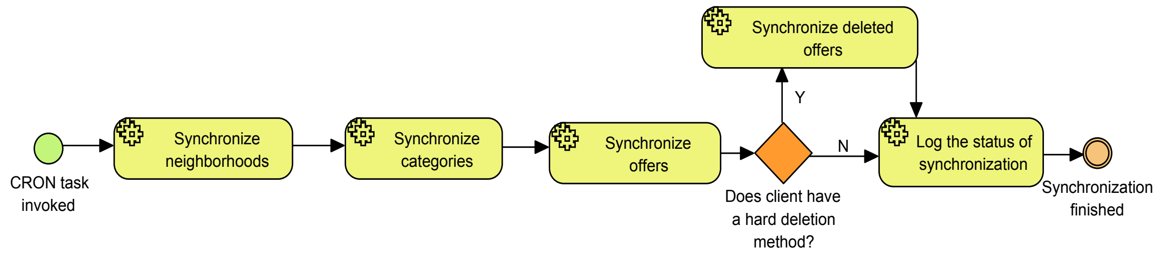
CRON task invoked
Synchronize neighborhoods
Synchronize categories
Synchronize offers
Does client have a hard deletion method?
Y
Synchronize deleted offers
N
Log the status of synchronization
Synchronization finished

Figure 5: The high-level process of overall synchronization task.

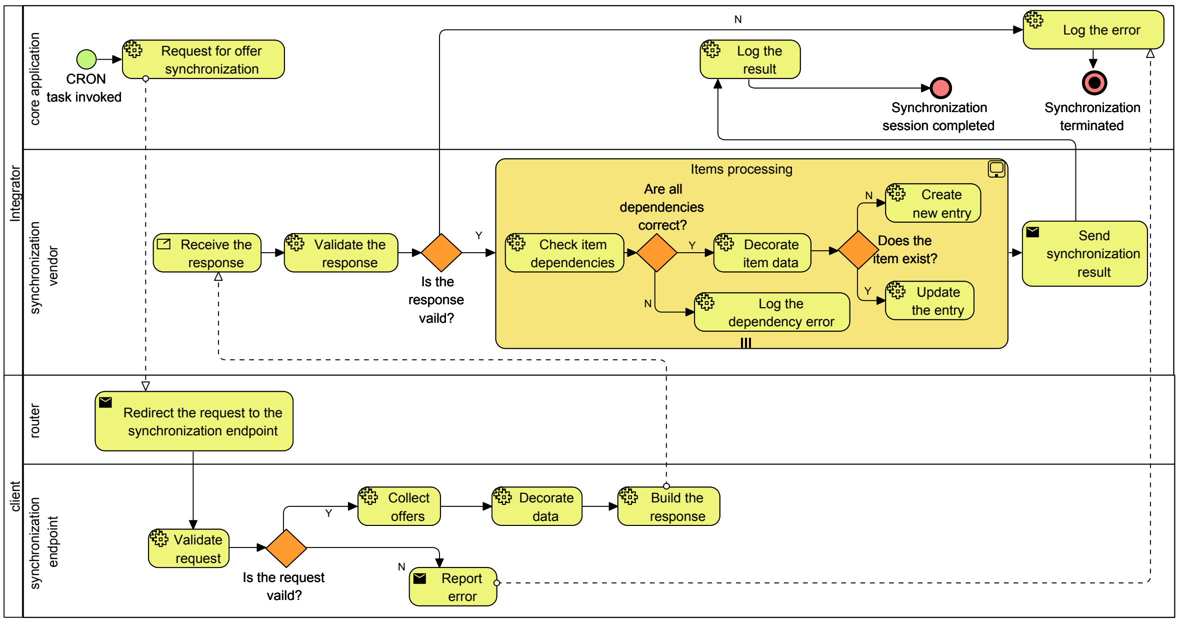

Figure 6: The low-level process of offer synchronization task.

The request for the offer synchronization (Fig. 6) is invoked automatically by CRON server tool (from CRONtab entry). The request is sent to the router (discovery API). It redirects it to the proper URI and the procedure of building the collection of offers and converting it to the JSON format is performed. As the web service get the parameter "FROM::" each offer is checked whether timestamp of last update operation fits the requested period (if the parameter is missing, all offers are added to the collection – this path is allowed for special maintenance purposes). If the collection is ready, it is being sent to the synchronization endpoint of the integrator. It validates the request for syntax (expected structure and data types) and verifies dependencies (neighborhood, category, user). If any dependency is returning errors (e.g. doesn't exist, aren't mapped), the offer is omitted and it is recorded in log files. If validation is successful the offer is being processed: the offer is being subjected to decoration (conversion to the form, in which data could be stored into the integrator's database). Decorator is also adjusting the timestamps and checking the availability status of the offer. If the offer already exists in the database (was only updated in the client), it is updated in the integrator's database. If the offer doesn't exist, it is added. Log counters are also updated. All offers from the collections are checked within a multiinstance loop and if it ends the synchronization result is logged. If any error occurs the respective status in logger database is set and the counter is updated. If the counter reach a limit value, the problem with synchronization is reported to the administrator.

In the next step of the process the four services (that were recognized before as necessary to be implemented in clients) were designed and described in REST/JSON notation. They are: neighborhood endpoint, category endpoint, offer endpoint and offer IDs endpoint. Due to the capacity limitations of the paper, only the offer endpoint will be discussed. The request and response are taking the form of:

The request:

```
    GET http://clientapplication.com/
    <AGG_API>/<API_VERSION>/offers[?fr
    om=<TIMESTAMP>]
```

The response:

```
    Content-Type: application/json [{
    "client_offer_id": 455,
    "client_users": [33],
    "client_categories": [12,15],
    "client_neighborhoods" : [5,6],
    "is_recommended" : 0,
    "is_active": 1,
    "name": "Lorem ipsum dolor sit",
    "type": "JOB",
    "description": "Etiam malesua … ",
    "image_url": "image_url",
    "localization": {
         "latitude": 47.427438,
         "longitude": 9.376254,
    },
    "apartment_no": "33/5",
    "street": "Kolorowa",
    "city": "Warszawa",
    "zip_code": "22-001",
    "country": "Poland",
    "language": "en-GB",
    "created_at": 1456733752,
    "updated_at": 1456733752,
    "expired_at": 1456733752,
    "deleted_at": 1456733752,
}]
```

The integrator is waiting for HTTP response and only the code 200 is treated as a success. All other answers are logged. Also the detailed restrictions for the particular fields are described (i.e. the expected data types, the policy of using null value, enum. values etc.).

Both sides (development teams of the integrator and the client) owning complete documentation, have to implement the code to handle these operations on such structures. The integration is performed using Laravel framework and vendor-based architecture. So the synchronization middleware takes the form of a vendor which collaborates directly with integrator's database via Eloquent ORM. The previously mentioned mapping of neighborhoods and categories could be performed in the administration panel using simple drag&drop feature. The mapping of users is being done automatically after the registration.

It is very important to setup of all participants (the integrator as well as clients) in the proper way.

The integrator offers a client repository where basic configuration data is stored (i.e. the URI of discovery API, URL's dynamic patterns (parametrized) for CRUD operations on particular types of offers, information about deletion policy, etc). That all makes the communication process easy to configure and helps in avoiding problems in synchronization.

7 CONCLUSIONS

The integration using WOA principles is very efficient and lightweight. Clear structure of requests and responses, explicit usage of HTTP methods and the independence from any programming language make this concept very pragmatic for developments teams.

The paper discussed process of building the integrated solution for service e-marketplace. The applied consortium research method has shown that there are many challenges that need to be overcome. The proposed loosely coupled integration based on three foundations: UI, data&business logic and security can provide users with rich and seamless experiences. It makes feeling of using one application instead of working with several systems. Thanks to Single Sign On authentication mechanism (OpenIDConnect) they do not have to put their credentials several times. The offers are gathered in one place, may be compared, added or managed on one window. The users can be informed about any operation on their account immediately via three parallel channels: the application notifications, SMS and e-mail. From the business point of view integration of service e-marketplaces can also bring profits to professional service providers, as they are able to reach much more potential customers.

Further research in the area of using WOA-based integration will concentrate on ways of securing the communication as well as sharing permissions between the integrated systems. Authors will also verify the current concept in the next real-life implementations and will adapt it in order to reach a satisfying level of its universality.

REFERENCES

Agilemanifesto.org, 2001. Principles behind the Agile Manifesto, http://agilemanifesto.org/principles.html.

Booch, G., 2007. The Economics of Architecture-First, IEEE Software, Sept./Oct., pp.18-20.

DAD, 2014. Disciplined Agile 2.X, http://www.disciplinedagiledelivery.com/lifecycle/.

Eliasson U., Heldal R., Lantz J., Berger C.,2014. Agile Model-Driven Engineering in Mechatronic Systems - An Industrial Case Study. In: Dingel J., Schulte W., Ramos I., Abrahão S., Insfran E. (eds) *Model-Driven Engineering Languages and Systems. MODELS 2014*. LNCS, vol 8767. Springer.

Fielding, R. T.,2000. *Architectural styles and the design of network-based software architectures* (Doctoral dissertation), University of California, Irvine.

Fowler, M., 2003. *Patterns of Enterprise Application Architecture*, Pearson Education Inc.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 2008. *Inżynieria oprogramowania: Wzorce projektowe*. Sec. ed. Warszawa, WNT.

Hohpe, G., Woolf, B., 2004. *Enterprise Integration Patterns*, Addison-Wesley.

IEEE/AWG, 2017. ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description http://www.iso-architecture.org/ieee-1471/defining-architecture.html

Kulkarni, V., Barat, S., Ramteerthkar, U., 2011. Early experience with agile methodology in a model-driven approach. In: Whittle, J., Clark, T., Kuhne, T. (eds.). *MODELS 2011*. LNCS, vol. 6981, pp. 578–590. Springer, Heidelberg.

Kutera R., Gryncewicz W., 2016. Single sign on as an effective way of managing user identity in distributed web systems. The ActGo-Gate project case study. In: *Business Informatics* (in print).

Łysik, Ł., Kutera, R., Machura, P., 2015. Social Collaboration Solutions as a Catalyst of Consumer Trust in Service Management Platforms - A Research Study. In: Abramowicz, W.(ed.). *Business Information Systems*, LNBIP, Springer International Publishing Switzerland, pp. 220-232

Maciaszek, L., Gryncewicz, W., Kutera, R., 2017. Integrated Service E-Marketplace for Independent and Assisted Living – Business and Software Engineering Challenges. In: Shishkov B. (ed.): *Business Modeling and Software Design.* Revised Selected Papers, LNBIP, vol. 275, 2017, Springer.

Myers, M. D., Venable, J. R., 2014. A set of ethical principles for design science research in information systems. Information & Management, 51(6), pp. 801-809.

Österle, H., Otto, B., 2010a. Consortium Research. Business & Information Systems Engineering. 2(5) 283-293.

Österle, H., Otto, B., 2010b. Relevance through Consortium Research? Findings from an Expert Interview Study. In: Winter, R., Zhao, J.L., Aier, S. (eds.). Global Perspectives on Design Science Research pp. 16-30.

Österle, H., Becker, J., Frank, U., Hess, T., Karagiannis, D., Krcmar, H., Loos, P., Mertens, P., Oberweis, A., Sinz, E. J., 2011. Memorandum on design-oriented information systems research. European Journal of Information Systems. 20(1), pp. 7-10.

Pautasso, C., 2014. RESTfulWeb Services: Principles, Patterns, Emerging Technologies. In Bouguettaya, A. et al. (eds.) *Web Services Foundations*, Springer Science+Business Media, pp. 31-50.

Poppendieck, M.; Poppendieck, T., 2003. *Lean Software Development: An Agile Toolkit*. Addison-Wesley.

Rauf, I., Siavashi, F., Truscan, D., Porres, I., 2013. *An Integrated Approach to Design and Validate REST Web Service Compositions*. Technical Report 1097.

Thies, G., Vossen, G., 2009. Modelling web-oriented architectures. In: *Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling-Volume 96* (pp. 97-106). Australian Computer Society, Inc..

Vosloo, I., Kourie, D. G., 2008. Server-centric Web frameworks: An overview. Computing Surveys (CSUR), 40(2).