Interface-based software integration

Aziz Ahmad Rais

Department of Information Technology
Faculty of Informatics and Statistics
University of Economics, Prague,
Czech Republic

aziz.rais@vse.cz, aziz.ahmad.rais@gmail.com

DOI: 10.20470/jsi.v7i3.261

Abstract: *Enterprise architecture frameworks define the goals of enterprise architecture in order to make business processes and IT operations more effective, and to reduce the risk of future investments. These enterprise architecture frameworks offer different architecture development methods that help in building enterprise architecture. In practice, the larger organizations become, the larger their enterprise architecture and IT become. This leads to an increasingly complex system of enterprise architecture development and maintenance. Application software architecture is one type of architecture that, along with business architecture, data architecture and technology architecture, composes enterprise architecture. From the perspective of integration, enterprise architecture can be considered a system of interaction between multiple examples of application software. Therefore, effective software integration is a very important basis for the future success of the enterprise architecture in question. This article will provide interface-based integration practice in order to help simplify the process of building such a software integration system. The main goal of interface-based software integration is to solve problems that may arise with software integration requirements and developing software integration architecture.*

Key words: Enterprise architecture, enterprise application integration (EAI), service-oriented architecture (SOA), system integration, software integration, interface-based integration.

1. Introduction

Application software architecture is one of the several types of architecture that together form enterprise architecture (TOGAF, 2011). According to TOGAF (2011), one of the goals of enterprise architecture is to "*lower software development, support, and maintenance costs*". The achievement of such a goal can be accomplished in many ways, one of them being the reuse of the application software already in place in the organization in question. Furthermore, this reuse of existing software can help to reduce the duplication of business processes and data, and thus will aid in achieving the other goal of enterprise architecture, namely to "*lower business operation costs*". Integrating two or more examples of application software from the data and business processes perspective, including their reusability and distribution, is called enterprise application integration (EAI) (LINTHICUM, 1999).

Existing application software can be reused in two ways:

1. Software service: The new application software consumes the service of the previously existing application software by using its remotely available or published service interface. This interface may be at the user interface level, shared data level or business function level (see the EAI levels described below).
2. Embedded library: In this case, the new application software embeds the existing software and consumes the latter's service by calling its application programming interface (API). The application programming interface is a set of components that simplifies the complexity of all the features provided by the software in order for them to be reused or invoked by other software. This type of integration is similar to the concept of component-based development, where software is decomposed into many sub-components, all of which communicate with each other via their provided interfaces (GROSS, 2005).

Both types of software reuse require software integration that can be designed and developed in different ways and from different perspectives. Equally, we must bear in mind that there are problems associated with any kind of software integration, some of which will be described below. The goal of this article is to cover the first types of reuse.

It is critical to differentiate between defined integration aspects such as software integration concepts, software integration layers, software integration architecture styles, and software integration technologies. Integration layers and architecture styles are important aspects to discuss here.

Software integration layers or, in other words, EAI levels (LINTHICUM, 1999):

1. User interface: This is graphical user interface (GUI) level integration; for example, it embeds the page or page section of one software GUI into another.
2. Shared data: This is database and file level, or any other data storage level integration. Examples of such integration occur when multiple software applications access the same data store, or when data is replicated in several storage locations, data population, data federation, and so on.
3. Business function (service layer): This type of integration is used when software requires the functionality of other software for its own runtime functionalities. In multi-layer software architecture, this is integration in the middle layer, or service layer, where the business logic is implemented.

Software integration architecture styles as defined in (Schmutz, Liebhart and Welkenbach , 2010):

1. Point-to-point architecture: This integration is fast, but its complexity grows by adding more and more integration endpoints.
2. Hub-and-spoke architecture: This integration uses an integration mediator (middleware) that transforms the messages of one example of application software to another, and routes them to the appropriate application software endpoint.
3. Pipeline architecture: This is an extension of hub-and-spoke architecture, and is built on top of messages. However, rather than using middleware, the software service provider and software service consumer are decoupled.
4. Service-oriented architecture (SOA): SOA can be achieved in many ways, using an integration mediator like an enterprise service bus (ESB). This means further extending pipeline architecture by adding more principles and governance to the integration.
5. Event-driven architecture (EDA): This architecture is a special example of SOA. The latter does not distinguish between events, or between synchronous and asynchronous communication among systems, but instead provides principles and layers. EDA, however, is integration based on events, and each event can have one or more processors.
6. Grid computing architecture: This is an extension of EDA, and is the parallelization of processes in order to use multiple computer hardware resources.

The six architecture styles listed above can be grouped into two types. Point-to-point integration architecture requires an integration component for communication. The other software integration architecture styles require additional software such as an integration mediator. In relation to the goals of enterprise architecture, the integration software requirements can be deduced from the enterprise architecture in question and its business and information system processes. For this purpose, ISO/IEC/IEEE 29148 (2011) provides an example of the requirements relationship between different parts of an enterprise, including its relation to the external environment, as follows:

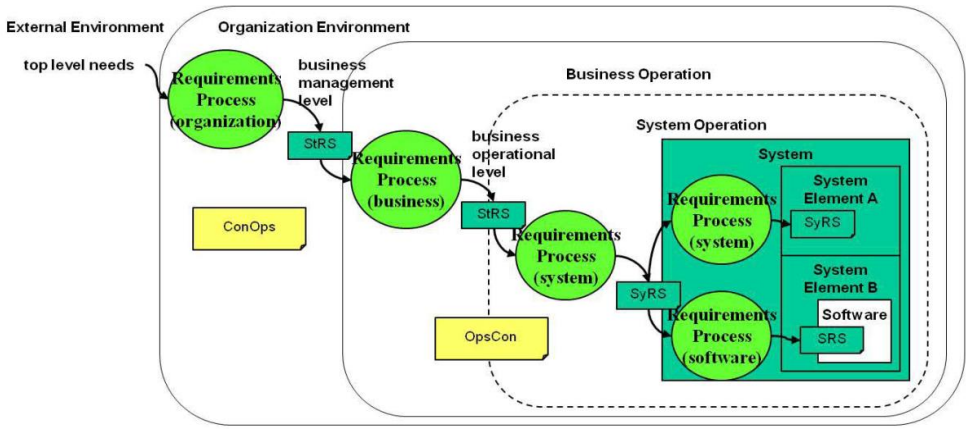

Figure 1 - Example of the sequence of requirements processes and specifications (ISO/IEC/IEEE 29148, 2011)

Because software is a system element, software integration forms a very important part of system integration as a whole (ISO/IEC/IEEE 29148, 2011), and we can apply the latter's principles, concepts, goals and architecture to the former. As a consequence, the following goal of system integration is equally valid for software integration: in terms of enterprise architecture, its goal is the creation of a maintainable and integrated information system that makes optimal usage of available information technology (IT) in order to maximize support for business objectives (Voříšek, 1999).

2. Software integration problems

Development issues may, of course, arise in software integration, as in any other software development project. It is, therefore, necessary to analyze the common problems that occur, and their roots. This can be done by examining, for example, the success/failure factors of IT projects based on the Standish Group reports and summaries. These reports, summaries and results are based on the analysis of 70 000 IT projects over sixteen years of research (Standish, 2010). Aside from another 100 practices necessary for an IT project to succeed, this research has also identified ten key success/failure factors that have been defined as "The 10 Laws of CHAOS". These ten laws clarify the meaning of the success/failure factor, and provide solutions as rules to be followed in order to deliver a project successfully. For example, the user involvement success/failure factor means that if the IT project in question does not have a skilled and authorized person able to provide the requirements, review them, and verify them, then these requirements will be difficult to define, describe, review or validate against business needs. We can conclude from this law that *understanding* the requirements is the key factor, and user involvement is simply one way of providing the requirements in the first instance. All ten success/failure factors are analyzed in 0 below:

Table 1 - Analysis of the common failure factors of software development projects
(source: author)

The 10 Laws of CHAOS (Standish, 2009)	Description of the 10 Laws (Standish, 2009) and (Standish, 2010)	Success/failure factors (Standish, 2015)
1. Law of the Two Faces	*Users are both your best friend and worst enemy.* This law concerns user involvement in the project, with the aim of providing feedback, requirements and requirements review, and a prototype of the user's needs. This law means that if a user is skilled, his or her involvement can help a project succeed, while if not, the project is doomed to failure. This law can also be interpreted as stating that if a project is missing requirements, or if those requirements are incorrect, misunderstood or badly described, a project can fail. The form and format of how the requirements are provided are also important. Requirements can be provided by user involvement, documentation, prototype, and so on.	User Involvement
2. Cheetah's Law	*Swift decisions are typically better than long, drawn-out analysis.* This law concerns support from executive management when it comes to making critical decisions in the project. At some stage, every project needs to choose between the following options: • *Long meetings, discussion and analysis that can introduce other open questions to answer and agree upon before taking the next step* • *Quick decisions based on the information available and continuous supervision in order to give appropriate step-by-step direction in the given circumstances.* *The right decisions can be made quickly by utilizing available strategies, goals, and the correct vision and architecture.*	Executive Support
3. Law of the Roads	*It does not matter which road everyone comes from as long as they end up in the same place.* This law supports the Success Factor of Clear Business Objectives. The aim is to build one ultimate goal from all the stakeholders' needs and wishes. On the basis of this goal, clear objectives can be defined for the project.	Clear Business Objectives

The 10 Laws of CHAOS (Standish, 2009)	Description of the 10 Laws (Standish, 2009) and (Standish, 2010)	Success/failure factors (Standish, 2015)
	The right tool for achieving these clear business objectives is architecture that maps business goals and requirements to the enterprise, system and software architecture. This means that missing or unclear goals, objectives and **architecture** will result in **quality** issues with the project deliverables (e.g. software product and integration software), and complicate the delivery of the project. In the worst-case scenario, this can cause the entire project to fail.	
4. Law of the Five Deadly Sins	*You will encounter the Five Deadly Sins in all projects*. This law supports the Emotional Maturity Success Factor. The five sins are over-ambition (asking for too much in too short a time), arrogance (hubris with a lack of knowledge, interest, and humility), ignorance (being uninformed about the project's goals, direction, details, issues, and opportunities), abstinence (not contributing to the project) and fraudulence (deliberately gaining advantages and avoiding confrontation or lying to oneself) (Standish, 2010). According to the CHAOS Report (Standish, 2015), this law is a collection of basic behaviors among people working together. As a result, the goals of the project have to be to have skills, and to assess, manage and direct the emotions in play. In other words, this law groups communication and behavioral skills and problems, and Standish provides HealthCheck methodology to identify and isolate the source of any frustration. The five sins can be regarded as human errors or **human factors**, because: *The primary focus of any human factors initiative is to improve safety and efficiency by reducing and managing human error made by individuals and organizations. Human factors are about understanding humans—our behavior and performance* (CASA, 2012).	Emotional Maturity
5. Law of the Long-Tailed Monster	*You will always build too much of what you don't need and not enough of what you do need.* This law is about the optimization of **requirements**. In other words, it states that most of the requirements are not needed, and those that are needed are not delivered. Thus, optimization starts with managing scope based on relative business value (Standish, 2015).	Optimization
6. Law of the Edible Elephant	*The only way to eat an elephant is one bite at a time. Complexity causes confusion and cost.* This law is about delivering work in iterations with a small team. In other words, to accomplish a task with small bites or stepping-stones. The law supports the agile **process**, which encourages learning lessons, adopting processes, teamwork, self-organization and self-accountability.	Agile Process
7. Law of the Mad Hatter	*Complexity causes confusion and cost.* This law concerns simplifying the project management **processes** and activities, in order for the team to be able to meet stakeholder expectations and produce value for the organization.	Project Management Expertise
8. Law of the Empty Chair	*Your best person will leave at the worst possible time.* This law is about maintaining skills inside the project. If a skilled or key person leaves, this can have a negative impact on the project's success. According to the CHAOS Summary report (Standish, 2010), over 52% of a project's success depends upon the quality of its technical staff. In other words, the **technical quality** of the software is dependent on the technical skills of the people on the project.	Skilled Resources

The 10 Laws of CHAOS (Standish, 2009)	Description of the 10 Laws (Standish, 2009) and (Standish, 2010)	Success/failure factors (Standish, 2015)
9. Panda's Law	*Inaction is the purest form of failure*. This law supports the Execution Success Factor. It is about balancing risk and reward, like features and functions in an application. The aim is to assign every **requirement** a value of cost, a return rate, possible gains and potential risks.	(Modest) Execution
10. Law of the Fools	*A fool with a tool is still a fool.* This law concerns integrating practices, services, and products for developing, implementing, and operating software applications. In other words, it is about using the right tools (for development, infrastructure, operations, and so on), and having the skills to use them. This law supports both technical skills and **technical quality** to help an IT project succeed.	Tools and Infrastructure/ Standard Architectural Management Environment

We can conclude from an analysis of the success factors in 0 that **clear requirements, quality and especially technical quality, architecture design, development processes, and human factors** are all instrumental in determining whether or not a software development project will succeed. Furthermore, problems in any of these areas are common among such projects. Of course, the last of these, human factors, is more of a psychological issue, and, as such, can be at least partially solved by clearly setting the requirements scope, and by providing well-defined quality criteria and appropriate decision-making processes. Because 15% are technical, 20% are related to business processes, and 14% to user functions (Standish, 2010), most decisions can easily be made by designing proper architecture (business, system and software). Architecture can, therefore, be considered a tool for defining quality attributes and offering appropriate solutions.

Overall, it can be seen that interface-based software integration provides a solution for simplifying software integration requirements and helps improve and accelerate the process of designing software integration architecture.

3. Interface-based software integration requirements

In order to specify the software integration requirements, it is good practice to define their relation to the system and business requirements in play*. Business requirements are higher-level statements of the goals, objectives, or needs of the enterprise. They describe the reasons why a project has been initiated, the objectives that the project will achieve, and the metrics that will be used to measure its success. Business requirements describe the needs of the organization as a whole, and not of the groups or stakeholders within it. They are developed and defined through enterprise analysis* (IIBA, 2009). According to ISO/IEC/IEEE 29148 (2011), business requirements are the business operations in an enterprise environment with the business goals, objectives, models and information providing a service to the external environment. The system is a subset of the enterprise environment; the system requirements are the business requirements in the organization (enterprise) environment that build and support service to that environment. Because software is a system element, the software requirements will be those business requirements that need to be performed by software. We can conclude from the relation, described above, between different levels of requirements in different environments that the software integration requirements are those business requirements that need the software service of two or more examples of application software.

Business processes, along with business policies, rules, constraints, quality, mode and structure, together compose the business operation or business service (ISO/IEC/IEEE 29148, 2011). The business process language BPMN is OMG standard and provides all the necessary activity, events and gateways to support business processes, rules and constraints. In addition, it also supports the orchestration and integration processes of conversation (collaboration), and choreography (OMG BPMN, 2011).

Any BPMN process activities can be mapped to software component interfaces in order to describe the functionalities with which the software is provided or expected to integrate. 0, below, describes an example of business process collaboration and mapping the activities necessary for integration.

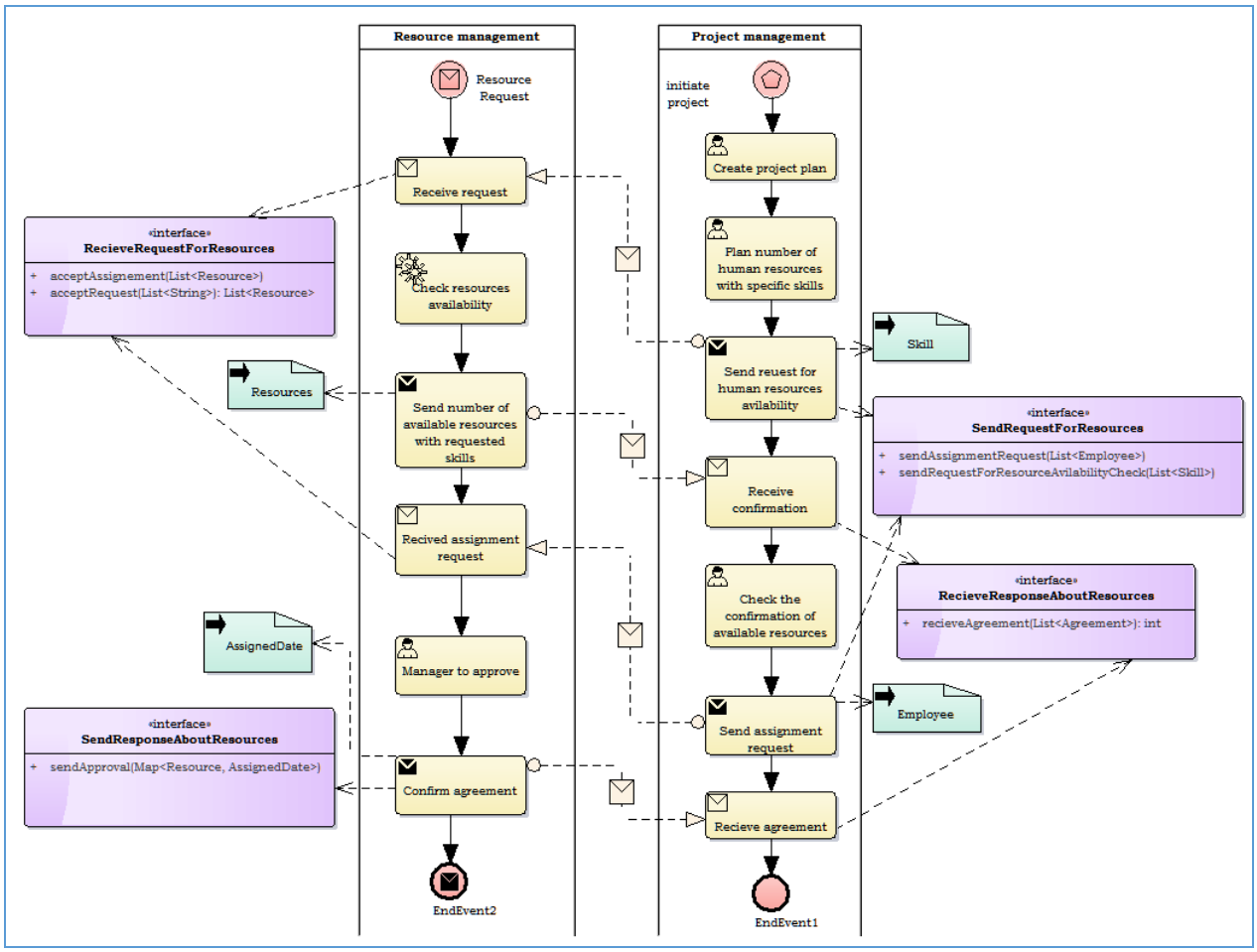

Figure 2 - Business process mapping to software interfaces (source: author)

Based on the integration software requirements, the example describes the two business processes of two different business services in an enterprise environment. The assumption is that both will be supported by software, and, thus, the conversation points of the business processes will be mapped to the interfaces to be implemented by both application software types.

The example of business process collaboration illustrated in 0 shows that there are two business services available in the organization, as the project management service needs the business functions of the resource management service. The integration points are logically mapped to the interfaces with their expected operations to support the business activities.

The business process shown in 0 also indicates that both processes produce and expect different data objects. In the software architecture, these data objects are represented by domain objects, and thus represent the business domain in the business process. The project management has two data objects, Skill and Employee, and both objects can be called **domain-1**. The resource management also has two data objects, Resource and AssignedDate, that can be called **domain-2**.

This chapter has demonstrated how to specify software integration requirements and thereby solve one of the key or root problems of software development projects mentioned in the chapter titled “Software integration problems.”

4. Interface-based software integration design

Enterprise architecture can be simplified by mapping business processes to interfaces, while the latter in turn map business functions to each software component and layer (RAIS, 2016). This can be carried further to map the interfaces identified in the previous chapter to software integration

architecture in order to simplify integration logic and domain transformations, and to help identify software integration components.

The software integration requirements described in the chapter titled “Interface-based software integration requirements” specified the **integration points** in the business process, as well as the **interfaces** necessary for software integration. In addition to the integration points and integration interfaces of the two business services, each of which can be implemented by separate application software, two business domains, **domain-1** and **domain-2**, were also identified. There are two types of application software relevant here: the first implements the resource management process and provides it as a software service to the second, and can be called **service provider software** (provider); the second type of software implements the project management process, and can be called **service consumer software** (consumer).

Another root issue for software integration is architecture design that better defines and measures technical quality, which is a basis for other qualities, such as, for example, the quality of functionality, usage, and so on. First, it is necessary to define the technical problems of software integration to be solved by software integration architecture.

1. **Domain usage**: The problem that arises most frequently in software integration is that the service consumer software uses the domain objects of the service provider software without transforming them to its own domain. This causes tightly-coupled dependencies between both levels of software.
2. **Domain optimization**: This problem is connected to integration message optimization and can arise if the service provider software returns many unnecessary parameters in the response and requires a lot of superfluous information in the reintegration request.
3. **Integration component and layer**: Another technical problem occurs when the service consumer software fails to designate an integration component to specify the integration logic for consuming the service of the service provider software. In cases where the service consumer software integrates with more than one example of service provider software, it is good practice to group all integration components into one layer.
4. **Service interface**: Complications can also ensue when the service provider software lacks proper service definition or design. Problems here are connected to the granularity of the service, meaning that the service consumer software is forced to make multiple calls to receive or provide information necessary for the business process, rather than just one.

Enterprise application integration (EAI) can be done on three levels (see the Introduction chapter). Shared data and user interface level integration between the two types of application software can be excluded. The shared data integration level for our sample requirements means that the service consumer software (the project management service) uses the data given by the service provider software (the resource management service). A problem arises, however, in the fact that our example also needs user interaction in the resource management process, so the service provider software is required to furnish not only data, but also functionality and workflow. Integration at the user interface level would cause difficulties in shared data integration, making it problematic for the service consumer software to involve or initiate the resource management process.

Thus, the only appropriate type of integration remaining is business function (service layer) EAI. Integration at this level is that most frequently used for business processes.

The next stage is to integrate the two types of application software, and to decide between the software integration architecture styles that were grouped into two varieties in the Introduction chapter: point-to-point integration architecture, and architecture that uses integration mediator software.

4.1 Point-to-point integration architecture

In this case, the service consumer software will implement all the integration activities in the integration layer in order to prevent problems of **domain usage** and **integration components and layers**.

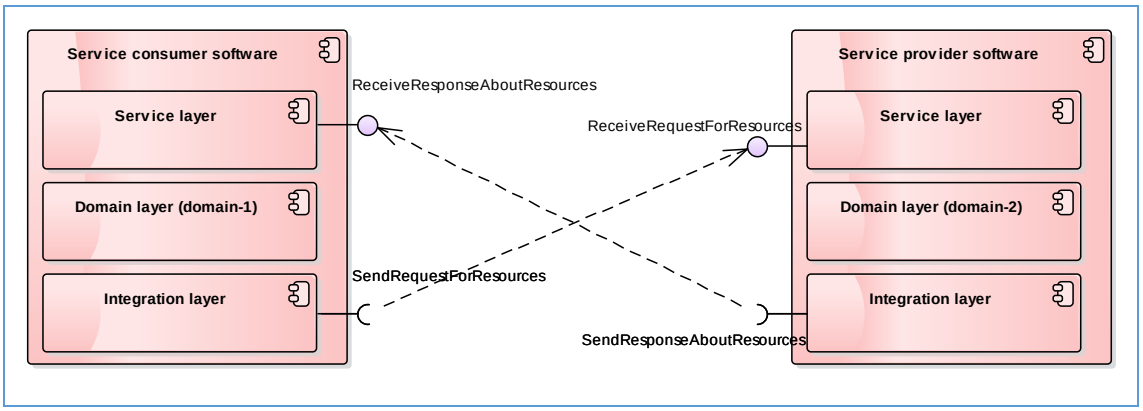

Figure 3 - Interface-based software integration (source: author)

To solve problems of **domain optimization and service interfaces,** a **façade layer** can be added on top of the service provider's service layer to act as a screen between it and those issues. .

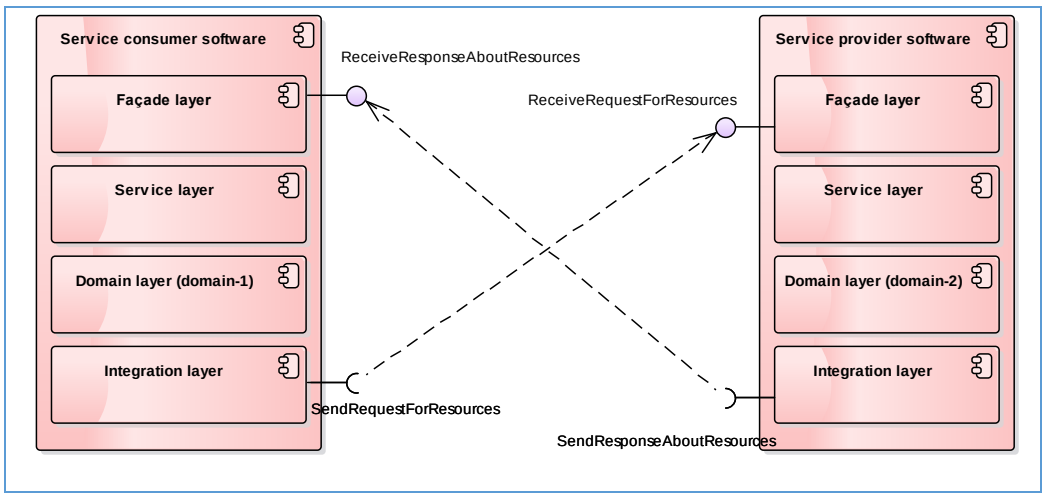

Figure 4 - Interface-based software integration (source: author)

4.2 Integration architecture that uses integration mediator software

Integration complexity can increase if a software service is utilized by multiple examples of service consumer software simultaneously (one service-to-multiple consumer), or if many services are utilized by one service consumer software program (multiple services-to-one consumer).

In both cases, most of the issues dealt with in the integration layer and façade layer can be moved to integration mediator software in order to reduce the dependencies and the impact of changes, and to increase the reusability of the integration system. An example of this kind of integration can be illustrated with the following diagram:

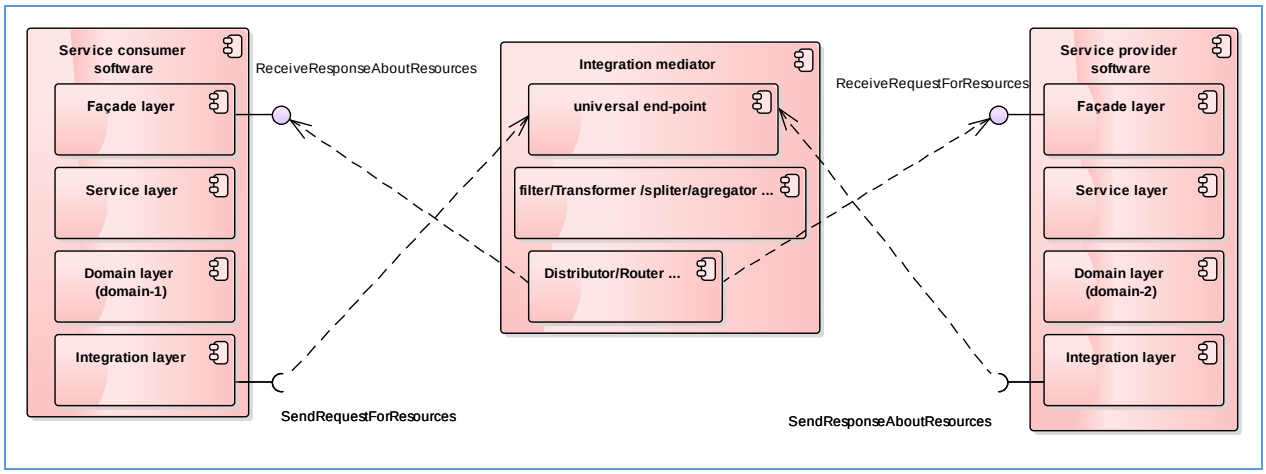

Figure 5 - Example of integration using integration mediator software (source: author)

5. Conclusion

The different levels of integration and the different architecture styles used to achieve the goals of interface-based software integration were defined in the Introduction chapter. The common problems associated with software development projects were analyzed based on the Standish Group identified success and failure factors of IT projects, and the foundational determinant of whether a project ultimately succeeds or fails was found to be the requirements. Further, other complications and obstacles were recognized and examined, and poor technical quality was singled out as the root cause of low quality in other areas. This issue can, however, be cleared up with proper architecture design. To summarize, requirements are the basis of IT projects' problems, and architecture is the solution to quality issues.. This article has provided an example of a system that simplifies software integration requirements by using interfaces. In the chapter titled "Interface-based software integration design", four key technical problems related to software integration were detailed and grouped, and were solved with architecture styles that clearly defined and mapped interfaces to appropriate layers of software architecture.

Using another example, we have also seen that interfaces can help not only in understanding the requirements, the business needs, and the use of enterprise architecture, but also in lucidly delineating the components of software integration and in preventing the technical problems associated with them.

References

Civil Aviation Safety Authority (CASA), 2012. [online], Available at: https://www.casa.gov.au/sites/g/files/net351/f/_assets/main/sms/download/2012-sms-book6-human-factors.pdf [Accessed 24 May 2016]

GROSS Hans-Gerhard, 2005: *Component-based software testing with UML*. Researchgate, DOI: 10.1007/b138012, pp. 2-9

International Institute of Business Analysis (IIBA), 2009: *A Guide to the Business Analysis Body of Knowledge (BABOK Guide)*, Version 2.0. Toronto: IIBA, ISBN-13: 978-0-9811292-2-8. pp. 176-177

International Organization for Standardization (ISO/IEC/IEEE 29148), 2011. *Systems and software engineering —Life cycle processes — Requirements engineering*

LINTHICUM David S., 1999: *Enterprise application integration*. Addison Wesley. ISBN: 0-201-61583-5

Object Management Group (OMG BPMN), 2011: *Business Process Model and Notation*.[online], Available at: http://www.omg.org/spec/BPMN/2.0/ [Accessed 24 May 2016]

RAIS AZIZ AHMAD, 2016: Interface-based enterprise and software architecture mapping. *Journal of systems integration*, 7 (2): . DOI: 10.20470/jsi.v7i2.253

Rikard Land, Ivica Crnkovic. Existing Approaches to Software Integration – and a Challenge for the Future. Mälardalen University, Department of Computer Science and Engineering http://www.es.mdh.se/pdf_publications/642.pdf [Accessed 24 May 2016]

SCHMUTZ Guido, LIEBHART Daniel, WELKENBACH Peter, 2010: Service-Oriented Architecture: An Integration Blueprint. Birmingham:Packt Publishing Ltd. ISBN 978-1-849681-04-9. pp. 30-31

Standish Group, 2009: *CHAOS Summary 2009, the 10 Laws of CHAOS*. [online] Available at: https://www.classes.cs.uchicago.edu/archive/2014/fall/51210-1/required.reading/Standish.Group.Chaos.2009.pdf [Accessed 24 May 2016]

Standish Group, 2010: *CHAOS Summary 2010.* [online] Available at: https://cours.etsmtl.ca/mti515/Notes/Cours01/CHAOS%20Summary%202010.pdf [Accessed 24 May 2016]

Standish Group, 2015: *CHAOS Report 2015 [online], Available at:* http://www.infoq.com/articles/standish-chaos-2015

The Open Group Standard (TOGAF), 2011: *The open group architecture framework*. Version 9.1, Document Number: G116. ISBN: 978-90-8753-679-4

VOŘÍŠEK Jiří, 1999: *Systémová integrace na prahu nového tisíciletí - čtyři základní koncepty systémové integrace*. Available at: http://nb.vse.cz/~vorisek/FILES/Clanky/1999_Ctyri_koncepty_SI.htm [Access

JEL Classification: M15