Articles

Analysis and Classification of Business Software Systems Integration Approaches

Smilen Kouzmanov[*]

Summary:

The modern business environment becomes increasingly globalized and tightly packed and this exerts pressure on the organisations to react to events much faster than before. In such dynamic conditions, sharing up-to-date information, synchronizing data and ensuring access to strategic and decision-supportive knowledge are a top priority of critical importance. As it is known, these activities are carried out by the enterprise application and integration infrastructure, which is facing a large set of different and serious issues due to modern organizations' size and characteristics. These issues, combined with the various business context and models, related to different organizations and economic sectors led to development of a wide variety of software integration approaches and techniques, which applicability depends on the context of the business case. Knowledge of different software integration approaches and their business applicability can reduce software research, development and integration costs in large organizations and can give them access to more up-to-date and detailed business and market information for gaining competitive advantage.

Key words: Software systems integration, software integration middleware, software integration topologies, service-oriented architectures, enterprise service bus

JEL Classification: *C61, C63, O33*

1. Introduction

In order to store, access, process and share up-to-date, valuable and decision-supportive information about business context and market, modern organizations use very large number of software systems at the same time. The number of application programs in a large organization or company can reach hundreds, including custom, purpose-built software, software systems from acquired companies, legacy systems, web-applications and portals and on-premises deployed instances of software packages like ERP, CRM, HRP, BPM, etc. systems. There is a large number of simultaneously used systems, which can be based on different implementation technologies, execution platforms (operation systems, virtual machines and application servers), etc. This leads to a very high level of heterogeneity of the enterprise information and application environment, which brings difficulties to

[*] PhD Student, Department of Information Technologies and Communications, Faculty of Applied Informatics and Statistics, University of World and National Economy, e-mail: skouzmanov@gmail.com

the orchestration of multiple systems in a single business process. This major problem, which is sooner or later faced by every large enterprise, undoubtedly makes software integration and building effective enterprise integration infrastructure, one of the key milestones on the road of business development and success.

2. Problems in the Field of Software Systems Integration

It is not realistic to admit that the integration of stand-alone software systems is a non-trivial, complex task that generally involves serious challenges which have no unified solution. Usually in an enterprise application integration environment a lot of different systems should be orchestrated and they usually execute on various platforms, communicating in a distributed environment. These factors make the "easy integration" term quite controversial. Big independent software vendors offer integration middleware platforms, which automate some tasks and make building of the integration infrastructure easier, but since real integration challenges are not only technological, but also have their architectural, design, business and political aspects, these software packages cannot solve all the issues. That is why, when classifying the integration approaches, we have to consider a quick overview of main problem categories in the field of software integration:

- ***Wide variety of systems***. In the process of growth of every business organization, nodes are added in vertical and sometimes horizontal direction in its hierarchy. These new organizational units are usually functionally divided from others, have different tasks and independent business model, which also leads to different information systems. Adding new units with their own software systems usually result in a lot of non-interconnected systems, which prevents the information flow in the organization (Binildas C. A. 2008).
- ***Lack of common data format.*** If different business units have different information systems, it is not surprizing that they will have different data models and data access approaches as well. These differences in the data format obviously prevent data transfer and sharing between systems and on the other hand it can lead to data duplication scenarios, which will provide more than one data source for the same entity (Binildas C. A. 2008).
- ***Integration and autonomy.*** If we are working on software integration, in no time we will start wondering how long stand-alone systems will stay divided and whether they can be united under centralized control. One aspect of the problem is that every system will need data transfer from one or more of the others systems, but on the other hand every stand-alone system is responsible for different line of business so it will need some autonomy. This leads to the conclusion that an integrated application environment consisting of stand-alone application is a better choice than a mega system responsible for all.
- ***Global scale integration and transparency.*** Modern enterprises are global, they are not based in a single geographical location or region and they do not limit their scale – they have operations around the world, in different countries, dependent on various

administrative and technical factors. This means that integrated systems will be deployed in different networks and will be connected through the cloud, which can be obstructed by the network route, firewalls, etc.

- ***Partner systems.*** Enterprises have their partners as well as their own information systems. It is obvious that business operations suggest data transfers between participants. These systems cannot be fully integrated because they are under different organizations control, but they have to have a certain level of interoperability. This demands business information systems to have communication and B2B integration by design (Binildas C. A. 2008).
- ***Legacy systems.*** One of the biggest integration challenges is the limited control over stand-alone systems. Many of the systems participating in the integration scenarios are already implemented and deployed and cannot be changed with the only aim to be included in an integrated environment. These applications are called with the term *legacy systems*, no matter of their age, and in fact, from this perspective almost every stand-alone system which is a subject of integration can be qualified as a legacy system at some point. In these cases integration engineers are forced to overcome these limitations at the integration middleware level (Laszewski T. 2008) (Chowdhury M. W. 2004) (Saran C. 2004) (Sutor B. 2004).
- ***System coupling level.*** This is another one of the most discussed challenges in the field, since every integration solution is required to keep the coupling level at the minimum, or in other words the systems should be loosely coupled. The main way to achieve a loose coupled integration is to make minimum level of assumptions about another system or software agent. Doing a lot of assumptions and sharing a lot of knowledge about systems internal operation can make the communication more effective but the solution will be more sensitive to change (Hohpe G. 2003).
- ***Methodological and management issues.*** One of the most up-to-date approaches in integration is related to the adoption of service-oriented architectures. These types of software and application architectures have significant advantages in software integration, since they use open standards and technologies, but despite their wide adoption there is a lack of common integration approach. This leads to experimenting with different service-oriented integration approaches, which has negative impact on systems interoperability, project costs, etc.
- ***Other issues.*** Except for the above issues, there are some additional challenges, like political and business model change after implementing an integration environment, defect management in the already deployed integration middleware, different vendors and implementations of integration technologies, maintainability of the large scale integration solutions, etc.

3. Research and Classification of Integration Approaches

After we made an overview of the problems in the field of software integration, we can proceed with considering different approaches, techniques and technologies which are applied in the area and are used to tackle the main challenges in practical integration projects.

It is obvious that the main aim of integration solutions is to provide communication between stand-alone software systems in an organization. Of course, different integration solutions use various communication channels and endpoint configurations, various data transfer modes, etc. These characteristics are logically called *integration topology*.

On the other hand, because of the complexity and heterogeneity of modern application environments, integration middleware usually provide more advanced functionality and different level of integration tasks automation. This is usually related to adding new layers in the integration architecture, making it more complex and can be called a *level of abstraction*.

4. Software Integration Approaches from the Perspective of Middleware Level of Abstraction

As we already mentioned, the level of abstraction is an integration solution characteristics, which describes its architectural complexity, the level of out of the box functionality provided and the cost of implementation work.

Basically, it can be said that integration solutions with lower level of abstraction do not add, or add less, layers of interaction in the software architecture, which means they do not add performance drawback also and have higher computational performance. These solutions do not provide a lot of out of the box functions, which also makes the implementation more resource-intensive.

On the other hand, integration solutions with higher levels of abstraction and automation offer a lot of ready-to-use functionality and maintenance mechanisms, which reduce implementation costs and enhance scalability. The drawback is that they add new layers of interaction in the software architecture, which introduces additional complexity and has negative impact on performance and leads to additional compute resources needs.

In order to classify the integration approaches based on their level of abstraction we can propose the following series: *file transfer, shared repository, remote procedure call, message transfer, and service-oriented integration, sorted in ascending order of their level of abstraction*.

File Transfer. Since a large number of systems are used in modern organizations, there are a lot of factors which determine serious differences between stand-alone applications. This includes different the age of the applications, different technologies and development approaches and even lack of application programming and functional integration interfaces. This raises the need for the integration of almost incompatible software systems running on different platforms and based on different assumptions about the business context.

Integrating such applications can be a complex task requiring technology knowledge and business understanding. However, in some cases the integration scenario does not require a very high level of abstraction, automation and out-of-the-box features. In such cases we need only a simple, shared and accessible mechanism for data transfer, which can be used with different technologies and platforms. Due to the high heterogeneity of the environment this approach should have minimal limitations to the hardware and software.

Files are such mechanism. They are a universal way for data storage, which is embedded and can be found in every operating system used within the organization, so we can say that the simpler way to integrate applications is the usage and exchange of files.

Basically, this approach consists of providing an opportunity for each of the applications to export files with data which will be needed by external systems. The export can be run manually or automatically and on different time intervals, depending on the business context and the integration scenario (Hohpe G. 2003). Since systems are often deployed on different physical locations and machines, files should be transferred over the network, which incorporates network protocols for file sharing and transfer (like *File Transfer Protocol – FTP* and *Server Message Block – SMB*). These traditional approaches can have bad performance and reliability in a large and complex application environment, so some authors propose distributed protocols for parallel file transfer between applications in large scale environments (Kolano 2012) (Lin W. C. 2013) (Anastasiadis S. V. 2009).

Another issue is the choice of time interval for the data exports. Since file export and processing is computational resources consuming, usually the longest possible interval which will not leave the systems too unsynchronized is chosen. The interval depends on the business context and can be daily, weekly, monthly and so on. This makes this technique appropriate for batch processing, but not for on-line synchronisation, which on the other hand is always related with some risk of data inconsistency between systems.

The greatest advantages of this approach are the low coupling between systems and the simplicity of implementation. The systems are well decoupled since they do not know any technical details of each other, the only agreement is about the file format. The other advantage is that no additional packages are needed for installation, since the file system is part of the operating system infrastructure. The only thing that should be implemented are modules for file export, if needed.

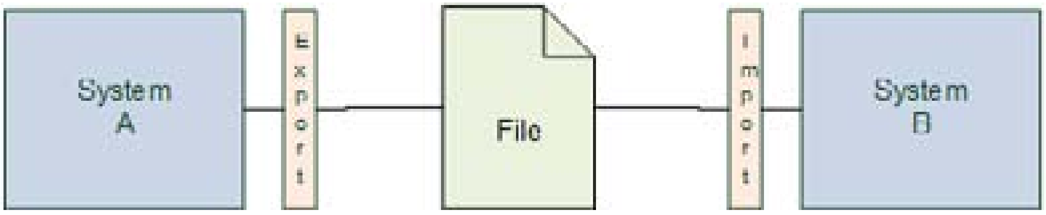

Fig. 1. File Transfer Integration

One of the main problems in this approach is how to choose a data and file format. Usually files exported from different applications differ in their format, which requires additional file processing and conversion. This formed the need for creating common, wide adopted data formats for usage in files. Nowadays the most widely used such format is the so called *eXtensible Markup Language (XML)*.

Shared repository. When using file transfer to synchronize data, there can be a lot of drawbacks, like differences in data format, different entity ownership, different data representation, semantic dissonance, data scheme and model differences, data structure inconsistency and mutual data inconsistency (Dayal U. 1984) (Batini C. 1986).

The most straightforward solution to such problems is using a common, centralized database, which is shared between applications, and every stand-alone system has the ability to use and update the data (Hohpe G. 2003). There are also distributed variations of this approach, which are based on database replication and/or real-time synchronization between more than one repositories which are kept up to date (Tzaneva M. 2013).

With modern database management systems using *SQL (Structured Query Language)* Shared databases are straightforward, easy to implement approach. Very large portion of nowadays software uses SQL or SQL based technologies to communicate with databases so we do not have to worry about different communication format and protocol. Another advantage of shared databases is that every system uses similar data representation and entity schema, which eliminates the risk of semantic dissonance and the need for entity aggregation. The engineers need only a careful design of the data model and scope in order to cover all applications needs.

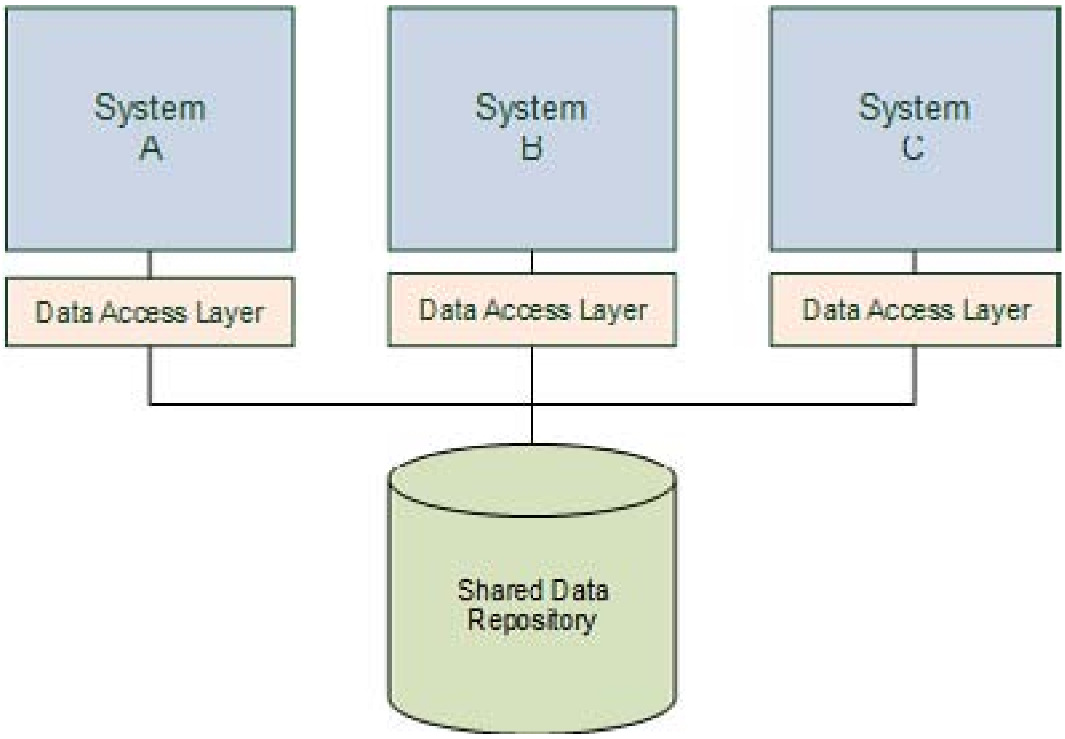

Fig. 1. Shared Data Repository Integration

If all the systems in an integration scenario have access to a shared database, this fully eliminates the risk of data inconsistency between applications. At the same time modern database management systems have mechanisms for transactional access, which manages concurrent operations and lowers the risk of data errors due to simultaneous updates.

The concurrent access can be stated as a limitation of this approach. When a lot of applications use a shared database, multiple access to a single record often occurs. Every one of the application locks the record while accessing it and the other cannot use it until it is unlocked and this strategy can lead to serious performance drawbacks for all applications in the scenario.

Remote Procedures Call. Shared databases and file transfer are data integration methods, but sometimes this is not enough, since in some integration scenarios applications require to call an operation which is part of another system functional layer. An example for such approach is the so called RPC – Remote Procedure Call, also known as RPI – Remote Procedure Invocation and RMI – Remote Method Invocation. Such integration approaches apply the principle of data encapsulation. This means that if an application needs data from another application, it can send request calling a relevant function, procedure or method through an application programming interface, which is a form of functional integration interface. Since data can be passed to the API as parameters, this mechanism is applicable for both input and output operations and this guarantees that every application takes care of its own data integrity and consistency, without preventing other systems operations (Hohpe G. 2003).

Since procedure calls are one of the best known methods for flow control and data passing in computer programming, the same concept can be leveraged to a multi-machine environment if the needed network infrastructure is supplied. When a remote procedure is called, the calling system is blocked and the parameters are passed over the network to the remote executing system. After the remote system finishes the procedure execution, the results are passed back over the network, and calling system's flow continues, like this will be done in a single-machine call (Birrell A. D. 1984).

An advantage of this approach is that the functional procedures or methods which wrap the data and control access to it can lower the risk of data semantic issues. Since one system can provide more than one method for accessing the same data entity, we can implement a couple of different representation styles of the same data, which can provide more than one view to an entity for different external systems. Another advantage is that the concept of procedure and method invocation is well known for software engineers, which makes the approach straightforward and easy to implement.

A disadvantage of this approach is the high level of coupling between systems, due to a lot of shared knowledge about programming interfaces and internal implementations like functions and procedures. Another drawback of the

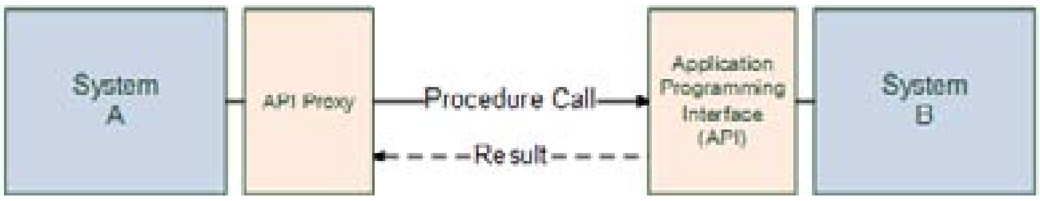

Fig. 3. Remote Procedure Call Integration

approach is the serious difference in performance between local and remote procedure calls – remote calls are slower and this can lead to long blockings in the calling system flow.

Messaging. One of the main challenges in the field of integration is providing system interoperability in a real time data synchronization fashion and without adding tight system coupling. An integration approach which is pretending to address all these issues is the message passing or simply messaging. Basically this is a mechanism for transferring data packets between systems in a reliable, asynchronous way with usage of configurable data formats (Hohpe G. 2003).

transfer is a slow operation and this stimulates the development loosely coupled integration solutions.

The message passing environments usually have a common mechanism for data transformation. This means that every system can use its own format for communication and the message will be transformed in the appropriate format in the central mediator, which additionally loosens the coupling. Since such functions are held by the central element of these environments, the presence of a central mediation element means that messaging integration solutions should be designed with care about scalability (Eugster P.Th. 2003).

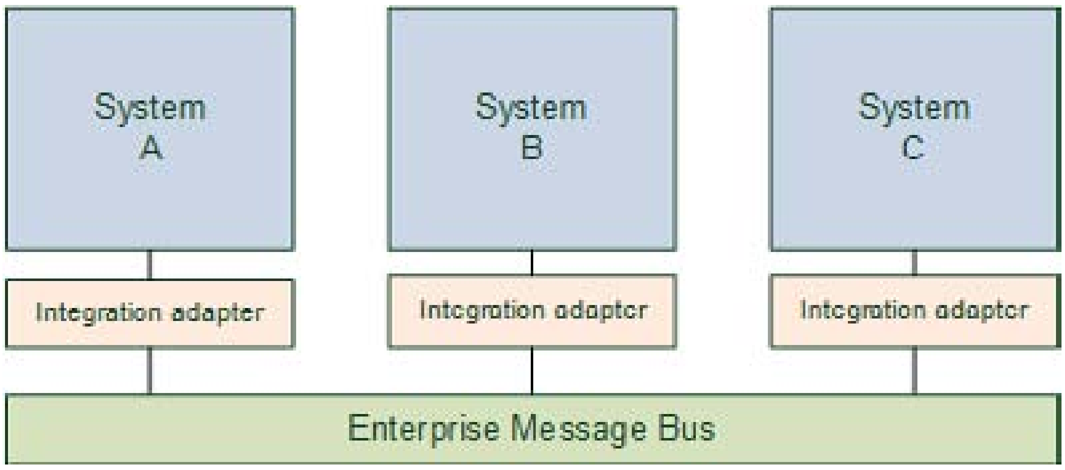

Fig. 4. Message Passing Integration

Asynchronous messaging is a good approach for tackling the issues in the field of distributed systems, because the process of sending a message itself does not require both systems to be in operational state in the same time. Apart from that asynchronous style of communication reminds software engineers that the intersystem data

Another advantage of the asynchronous message passing is the lack of blocking operations and if one of the system needs to pass a message to another it can just post it to integration environment and continue working without waiting for result. After the message is processed the sender system will receive a call back from the mediator, containing the result.

A drawback of this approach is that asynchronous communications are not the usual way of thinking of software engineers and this makes the implementation harder, slower and consuming more resources. Also asynchronous calls make testing, troubleshooting and maintenance harder and time-consuming. Another disadvantage of this type of integration solutions is that a way for connecting to the mediator element is not always provided and in such cases some integration adapters should be implemented for the systems which need them.

Service Oriented Integration. In a modern enterprise the most current integration approaches are those which wrap the stand-alone systems as software services. Usually information environments have a large scope and integration and orchestration of a very large number of systems is needed. The solution which solves these issues with the use of software services are usually called Service Oriented Architectures (SOA). These solutions enable different organizational units to build their own software services that satisfy their own needs, while at the same time can be reused by a higher integration layer (Juric M. B. 2007). Such software services should keep the level of coupling low and enable interoperability despite the differences in quality and scope (Menasce D. 2010).

The service itself is a discrete unit of business logic and functionality, which is provided to the user through the service contract. The contract defines all interactions between the service and the client, including the service interface, which is a set of operations and parameters descriptions, the transferred documents or business objects and the quality of services. The business logic and processing are held by the service implementation. An important characteristic of the services is their granularity, which describes the quantity and scope of business logic and functions included in a single service. The coarser granularity means implementing more functions in one service, which reduces the business processes complexity and network latency related with the usage of the service. On the other hand finer granularity means implementing fewer functions in one service, which usually pressupposes better reuse ability. The reuse of services is one of the main advantages of this approach.

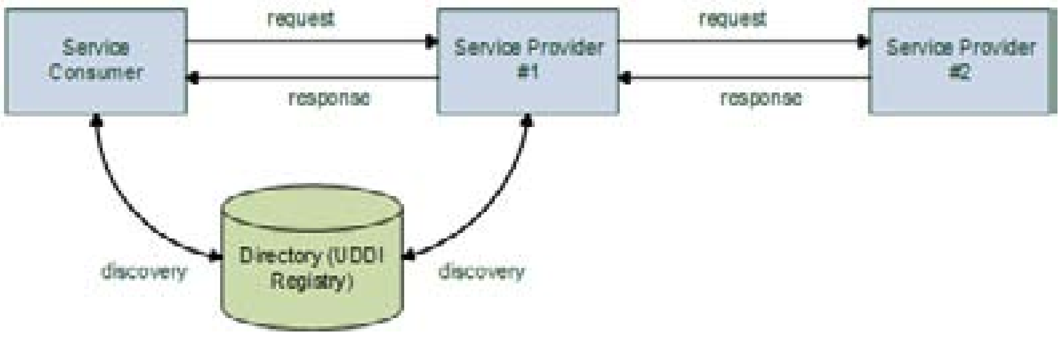

Fig. 5. Service Oriented Integration

The integration mechanism in service-oriented architectures consists of interactions between three main components: the service

Table 1. Software integration approaches from the perspective of level of abstraction

Approach	Advantages	Disadvantages	Applicability
File Transfer	• Loose coupling • Simplicity; No additional tools required	• Transfer of large data volumes • No common data format • Concerns about the time slot of file export and transfer	• Systems with no real time sync needs and higher tolerance to data delay • Systems that rely on batch jobs for data transfer
Shared Data Repository	• Easy to implement, straightforward • All systems have common data model, no risk of semantic dissonance	• Lower performance of business critical applications • Higher complexity in data access concurrency management.	• Systems which need common data format • Systems where immediate, real time data synchronization is needed
Remote Procedure Call	• Easy portability and good code readability • Data type checking and control • Escaping semantic dissonance with different data views with getter methods. • Well known foe engineers, easy implementation	• Harder support of doubled data methods • Low performance of remote calls • A lot of knowledge about the business layers of system, which leads to tight coupling	• Systems where immediate, real time data synchronization is needed • Systems where integration engineers have access to source code and functional layer.
Messaging	• Loose coupling due to a central mediator element • Asynchronous calls are not blocking, so calling application can continue work • Fast transfers and real-time actual data	• Harder and complex • Management of central element activities like addressing, routing, etc.	• Systems which aim for loose coupling • Fast message passing and on line transfer.
Software Service	• Control over granularity, business process optimization • Loose coupling between systems • Usage of open standards and wide adopted protocols • Service registry and discovery mechanisms • Interface self-definition • Wrapping legacy systems	• Risk of network or communication issues • Security risks, higher need of security control	• Systems which need loose coupling • Systems with real time data synchronization • Environments with high level of heterogeneity • Useful for systems wrapping and legacy systems • Useful in business process management environments

provider, the service consumer or client and the service registry. Usually the consumer calls the registry to find the appropriate service and examine the information about their interface, then calls the service provider with appropriate request and listens for response.

An important advantage of service-oriented integration is the usage of modern web-services. The main advantage of web-based services is that they usually communicate with XML and JSON messages, which are open and widely used standards that can be parsed by almost every client. They are also almost transparent for firewalls. Another advantage of this approach is the good applicability to legacy systems, because we can wrap a legacy system as a service without investing a lot of time in development. We just have to implement a service-like adapter which communicates with the original system over a known protocol.

As a disadvantage of this approach we can state the higher security related risk. When implementing such solutions we have to be careful about message security, since while in transmission cross-service messages can be intercepted, faked, replaced or intentionally corrupted. This leads to the need to use secure protocols like Hypertext Transfer Protocol Secure (HTTPS).

In the table below we can see a short summary of the advantages, disadvantages and application of the integration approaches by their level of abstraction.

5. Software Integration Approaches from the Perspective of the Integration Topology

As we already said, the integration topology is a logical abstraction which describes the communication characteristics and configuration of the integration environment. In this abstraction we usually include communication channels types, data transfer strategies, the usage of mediator software agents, the type and number of integration adapters and interfaces and so on. Usually these components form a common logical structure, which we can consider a topology.

Integration topology is an important characteristic of the integration environment, since it describes the system interconnection, which is the most basic functions of the environment. This makes the topology coherent with the architecture characteristics of the environment and gives us the opportunity to research important aspects of the integration solution.

In real life projects a lot of topologies are used, but generally they can be classified as such which use a mediator software agent component and such which do not use one for data transfer. The most common types we can use to classify the integration topologies are point-to-point integration, message broker (hub), message bus and service bus.

Point-to-Point Integration. A lot of integration projects start to interconnect two systems and the easiest way to implement this is the direct connection between them. The point-to-point integration approach considers the systems that need to transfer data as pairs only and solves every pair integration problems separately. To make such communication possible we have to implement integration adapters on both ends of the channel in order to transform data in a proper format. Such integration adapters are purposely built for every system pair in a point-to-point integration environment. This integration approach is considered one of the earliest and traditional ones, but nowadays it is not modern enough and cannot address the issues in large and complex integration environments (Linssen 2011) (O'Brien 2008).

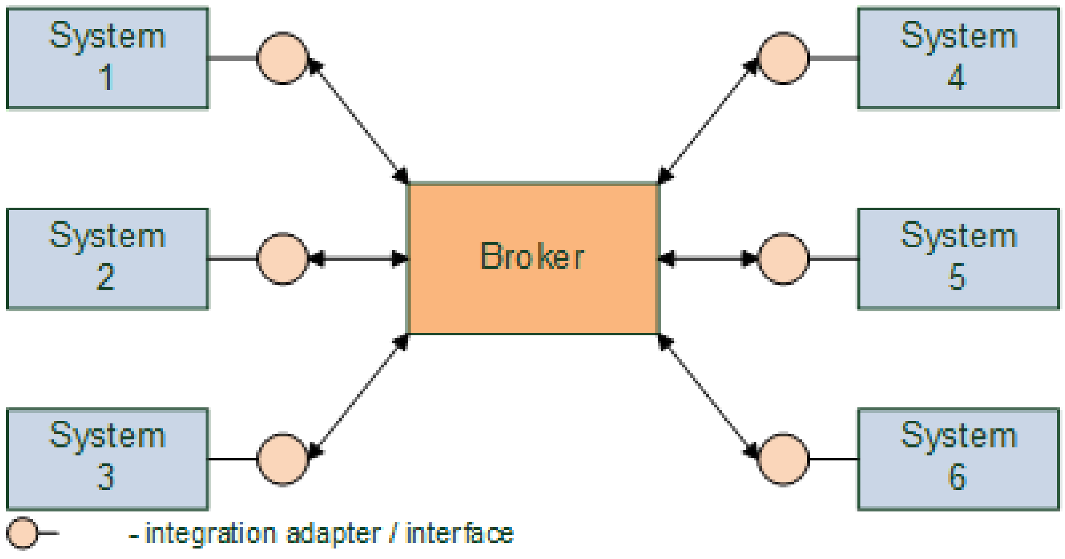

Fig. 7. Message Broker Integration

The main advantage of point-to-point connection is its easy implementation for environments with a small number of systems and the straightforward approach. For interconnecting two systems only the implementation of the integration adapters is sufficient and systems application programming interfaces, remote call technologies, etc. can be used, which eliminates the need of installing mediator elements.

This, on the other hand, is a reason for one of the main disadvantages of the approach. Since the integration adapters' implementation is specific for every systems pair, this increases the level of shared knowledge for systems internal operations and the increased level of cohesion brings problems to the system's maintainability and development and can lead to future system interconnection issues.

Another drawback of this approach is the scalability and the management of the environment when the number of systems is increasing. Since there is no central mediation element, the number of integration channels and adapter grows rapidly when new systems are added to the environment (Schmidt n.d.) (Clifford 2005). When the number of systems is high, this makes the topology flawed, hard to manage and increases the risk of further issues, since an application or communication channel outage will be hard to troubleshoot.

Broker. Another interesting type of topology is the one using a central mediation element for data transfer between systems (Goel 2006), which is called a message broker or hub-and-spoke. This approach is used in a wide range of different integration scenarios and such mediators are used for remote procedure call, message passing, service based communication, etc. (Trowbridge D. 2004).

The main aim of this integration topology is to achieve lower level of coupling between systems. In such configuration the broker is the only element which is achieving direct communication with the stand-alone systems and it carries all the integration tasks like routing, data transformation, keeping a register with system endpoints and so on. This generally means that the broker is the only element that has the technical detail for the systems, like application programming interfaces, communication channels and protocols and so on and so the broker can serve as an isolation level between systems.

Another advantage of this approach is the lower number of communication channels and adapters between systems, which mitigates the complexity of the environment and makes the maintainability easier (Johannesson P. 2001). Since the systems are connected directly only with the broker, adding a new system means adding only one new communication channel and configuring only one adapter. This makes the environment more scalable and adding new systems adds less complexity to it.

The main disadvantage of this type of topology is that the central mediator element is a single point of failure, and an outage of the broker lead to a breakdown in the whole integration environment. The risk of such failures can be mitigated with a clustering solution for the central broker.

Message Bus. The term bus is adopted from hardware and electrical engineering, where a common bus is used for interconnecting a lot of components and every bus has its own data protocol. In integration the *message bus* topologies are used to additionally lower the coupling between systems by offering a common communication mechanism which is not dependent on systems technical details and environment heterogeneity (Ott J. 2000). The main characteristics of such mechanism are a *common data format,* a *bus control command set* and a *shared infrastructure for message passing*. A lot of solutions to integration of intelligent agents are based on the idea of a message bus. Such systems are usually applied in large, geographically distributed enterprises.

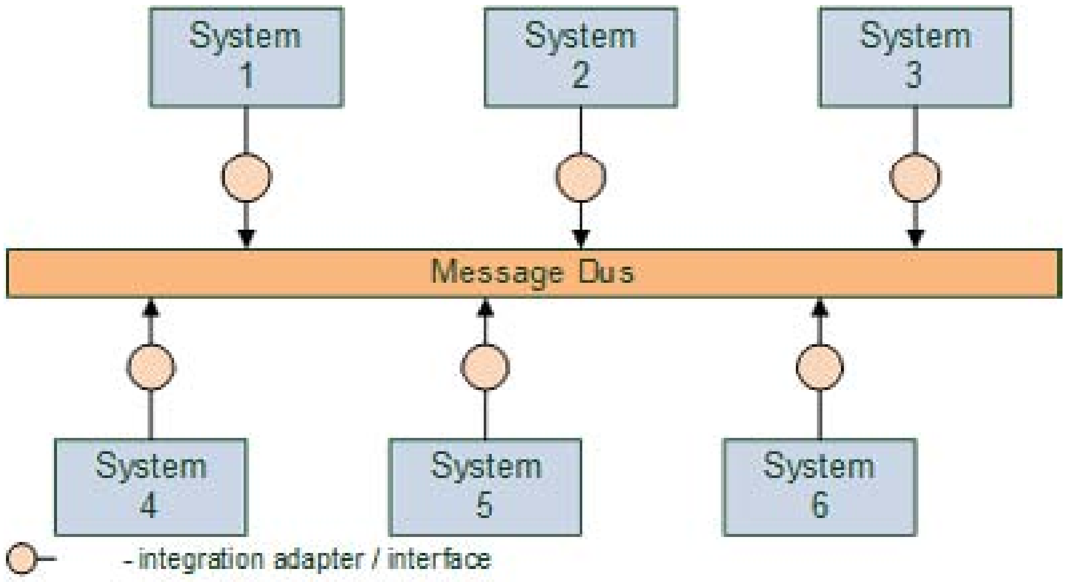

Fig. 8. Message Bus Integration

Usually large and powerful software packages which offer a lot of out of the box functionality are used to implement message bus solutions. Such product are *IBM WebSphere MQ (WMQ), Microsoft MQ (MSMQ)*, etc. These products has the base functionality of the bus, like primitive control command, rousting, addressing, media access, data transformation and some more advanced features like a lot of predefined integration adapters. This is one of the main advantages of such integration topologies, since the implementation of the integration is easy and straightforward and usually it consists only of bus installation and configuration tasks and integration scenarios that need specific integration adapters' development are relatively rare.

Another advantage of this type of integration is that messaging products are large software packages developed and maintained by large independent software vendors and their user can rely on high performance and scalability and enterprise-level support.

Disadvantages of this topology can be considered almost similar with the broker based solutions, with the difference that message buses have more advanced functionality which can make the installation, configuration, support and troubleshooting more resource consuming than in a hub and spoke solution.

Service Bus. In recent years one of the most widely used integration approaches are the so called *enterprise service bus*. Usually this concept is considered an enterprise integration design pattern or integration topology, which offers a flexible approach to integration challenges. The concept of a service bus can be defined as a new type of architecture which incorporates a web service based middleware for message passing, intelligent routing and transformation (Schulte 2002). Usually this type of buses are applied in service-oriented architectures and their role is to enable the environment for the needed service level, interoperability and

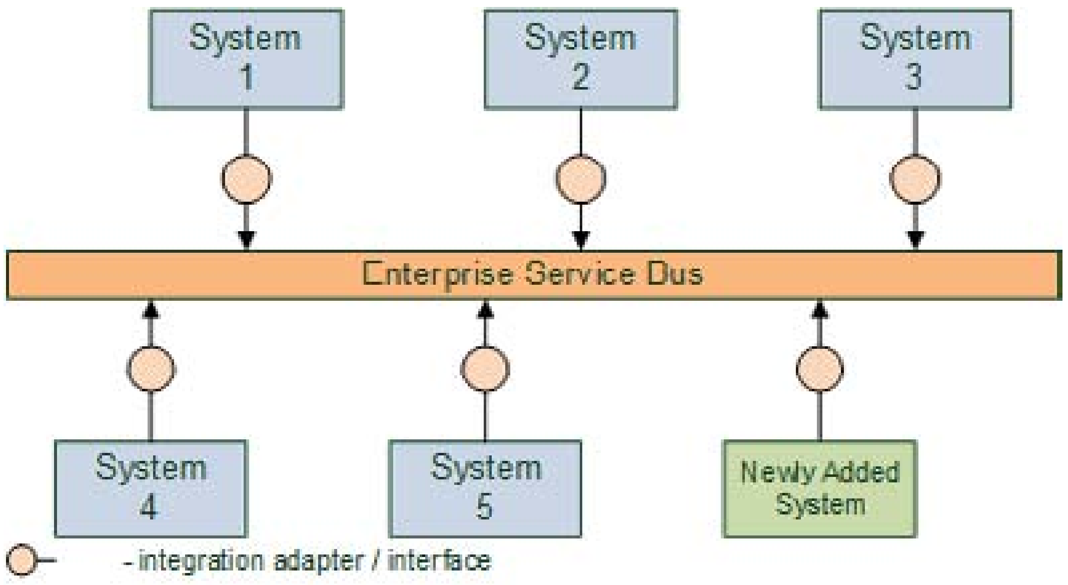

Fig. 9. Enterprise Service Bus Integration

manageability (Keen M. 2004). As other integration topologies the service bus relies on a central mediation component, but in the case of service buses this functionality can be distributed over different physical locations (Rademakers T. 2009). Another characteristic of this integration approach is the usage of open, non-proprietary standards like XML, JMS, JCA, web services, etc. (Chappell 2004).

Table 2. Software integration approaches from the perspective of the integration topology.

Approach	Advantages	Disadvantages	Applicability
Point-to-Point Integration	• Straightforward approach, easy to implement for small number of systems	• Tight coupling • Dependency of the adapters on every system • Low scalability • Hard to manage and maintain for a larger number of systems	• Environments with small number of systems with no explicit need for low cohesion
Broker (Hub-and-Spokoe)	• Lower level of cohesion because of the usage of mediation component • Lower number of communication channels and adapters • Easier to maintain with higher number of systems • Better scalability	• The central broker is a single point of failure	• Environments which need a lower level of cohesion • Environment where a redundant or clustered broker can be used
Message Bus	• Lower level of cohesion because of the usage of mediation component • Lower number of communication channels and adapters • Easier addition of new systems and better scalability • High performance and reliability • Wide range of out of the box features and adapters	• The bus is a single point of failure • Harder to install and configure due to the wide range of out of the box features	• Environments which need a lower level of cohesion • Environments which need real time data transfer
Service Bus	• Lower level of cohesion because of the usage of mediation component • Lower number of communication channels and adapters • Better scalability and extensibility, easy addition of new systems • Wide range of out of the box features and adapters • Adoption of open, widely used standards • Better applicability for heterogeneous environments • Better total cost of ownership	• The bus is a single point of failure • Harder to install and configure due to the wide range of out of the box features	• Environments which need a lower level of cohesion • Environments which need real time data transfer • Heterogeneous environments which use wide number of protocols and technologies • Environments which include legacy systems which are to be wrapped as services

As we can see on the figure, one of the main advantages of this topology is the high level of scalability, since newly added systems should be connected only with the service bus with a proper web-service based integration adapter and all other integration tasks like routing, data transformation and so on can be carried by the integration middleware.

Another important advantage is the ability to adopt a service bus in mixed, heterogenic environments with different systems running on different platforms and technologies. Usually, if we need to build an environment based on a lot of different protocols like for example JMS, FTP, HTTP, SOAP, etc. this increases the complexity of the integration solution, while enterprise service buses offer a lot of open standard based adapters which enables their usage in heterogenic conditions (Robinson, 2004).

These environments also offer a better total cost of ownership (TCO), because of the lower number of adapters end endpoints. Usually, in mesh or point-to-point topologies the communication channels are a lot more than in a mediated topology which makes the maintenance more resource consuming, while a bus environment keeps this number low and consume less time and resources for keeping the integration environment operational. Also, open standard integration adapters are easy to troubleshoot in case of communication issues.

In the table below we can see a short summary of the advantages, disadvantages and applications of the integration approaches by their logical topology.

Conclusion

Integration of software systems is one of the most complex problems in the scope of the modern management of company information infrastructure. Business organisations nowadays use a very large number of software systems to support their business operations and usually these software applications are used in complex heterogeneous environments. As a result a lot of different integration technologies, approaches and styles have been developed, and if they are not used appropriately the integration issues can deepen and an integration scenario can face tougher challenges. Hence the need arises to study and classify the integration approaches and identify their advantages, disadvantages and applicability. Such classification can support business organizations in choosing an integration approach which suits their needs and can also be used as a basis on which a common integration methodology can be identified.

References

Anastasiadis S. V., Wickremesinghe R. G., Chase J. S. 2009. Rethinking FTP: Aggressive block reordering for large file transfers. (ACM) 4 (4).

Batini C., Lenzerini M., Navathe S. B. 1986. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys* (ACM) 18 (4): 323-364.

Binildas C. A. 2008. Service Oriented Java Business Integration. Packt Publishing.

Birrell A. D., Nelson B. J. 1984. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems* (ACM) 2 (1): 39-59.

Chappell, D. 2004. Enterprise Service Bus. O'Reilly.

Chowdhury M. W., Iqbal M. Z. 2004. Integration of Legacy Systems in Software Architecture. *Specification and Verification of Component-Based Systems, Workshop at ACM SIGSOFT 2004/FSE-12.* Newport Beach, CA USA.

Clifford, A. 2005. Minimal integration 7: point-to-point, hub, or bus? Ziff Davis, LLC (Toolbox.com). http://it.toolbox.com/blogs/minimalit/minimal-integration-7-pointtopoint-hub-or-bus-6527. [Accessed December 2014]

Dayal U., Hwang H.Y. 1984. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Transactions on Software Engineering* (IEEE) 10 (6): 628 - 645 .

Eugster P.Th., Felber P. A., Guerraoui R., Kermarrec A.-M. 2003. The Many Faces of Publish/Subscribe. *ACM Computing Surveys* (ACM) 35 (2): 114-131.

Goel, A. 2006. Enterprise Integration EAI vs. SOA vs. ESB. White Paper. Infosys Technologies.

Hohpe G., Woolf B. 2003. Enterprise Integration Patterns - Designing, Building and Deploying Messaging Solutions. Addison-Wesley.

Johannesson P., Perjons E. 2001. Design principles for process modelling in enterprise application integration. *Information Systems (12th International Conference on Advanced Systems Engineering)* (Elsevier Science Ltd.) 26 (3): 165–184.

Juric M. B., Loganathan R., Sarang P., Jennings F. 2007. *SOA* Approach to Integration. Birmingham: Packt Publishing.

Keen M., Acharya A., Bishop S., Hopkins A., Milinski S., Nott C., Robinson R. 2004. Patterns: Implementing an SOA using an Enterprise Service Bus. New York: IBM Redbooks.

Kolano, P. 2012. High Performance Reliable File Transfers Using Automatic Many-to-Many Parallelization. Proceedings of the 5th Workshop on Resiliency in High Performance Computing. Rhodes Island.

Laszewski T., Williamson J. 2008. SOA Integration in the Legacy Environment. Oracle Technology Network. Oracle Corporation. October. http://www.oracle.com/technetwork/articles/laszewski-williamson-soa-legacy-082027.html. [Accessed December 2014]

Lin W. C., Liaw J. J., We C. T. 2013. The Modified parallelized file transfer protocol for multi-users. Proceedings of the 2013 International Joint Conference on Awareness Science and Technology and Ubi-Media Computing (iCAST-UMEDIA). Aizuwakamatsu: IEEE.

Linssen, M. 2011. Perfect Integration 8 - history of the last decades. http://www.martijnlinssen.com/2011/03/perfect-integration-8-history-of-last.html. [Accessed December 2014]

Menasce D., Casalicchio E., Dubey V. 2010. On optimal service selection in Service Oriented Architectures. *Performance Evaluation* (Elsevier B.V.) (67): 659-675.

O'Brien, R. 2008. Integration Architecture Explained. Hubpages Inc. http://russellobrien.hubpages.com/hub/Integration-Architecture-Explained. [Accessed December 2014]

Ott J., Kutscher D., Perkins C. 2000. The Message Bus: A Platform for Component-based Conferencing Applications. Proceedings of CBG2000: The CSCW2000 workshop on Component-based Groupware.

Rademakers T., Dirksen J. 2009. Open Source ESBs in Action. Greenwich, CT: Manning Publications Co.

Robinson, R. 2004. Understand Enterprise Service Bus scenarios and solutions in Service-Oriented Architecture, Part 1. IBM Developerworks. IBM Corp. June. http://www.ibm.com/developerworks/. [Accessed December 2014]

Saran C. 2004. Integration of legacy systems is vital to effective customer service. ComputerWeekly.com. TechTarget. May. http://www.computerweekly.com/feature/Integration-of-legacy-systems-is-vital-to-effective-customer-service. [Accessed December 2014]

Schmidt, J. n.d. Can canonical design solve your point-to-point integration complexity? informatica.com. http://www.informatica.com/us/potential-at-work/architects/can-canonical-design-solve-your-pointtopoint-integration-complexity.aspx#fbid=hkbz9MgA7t6. [Accessed December 2014]

Schulte, R. W. 2002. Predicts 2003: Enterprise Service Buses Emerge. Stamford, CT: Gartner Inc.

Sutor B. 2004. Something Old, Something New: Integrating Legacy Systems. eBizQ.net. TechTarget. October. http://www.ebizq.net/topics/legacy_integration/features/5229.html. [Accessed December 2014]

Trowbridge D., Roxburgh U., Hohpe G., Manolescu D., Nadhan E. G. 2004. Integration Patterns. Microsoft Developer Network. Microsoft. June. http://msdn.microsoft.com/en-us/library/ms978729.aspx. [Accessed December 2014]

Tzaneva M., Kouzmanov S. 2013. Impact of Business Problem Characteristics on the Architecture and Specification of Integration Framework. Proceedings of the Third International Conference on Application of Information and Communication Technology and Statistics in Economy and Education. Sofia. 628-635.