Software Architectures – Present and Visions

Cătălin STRÎMBEI, Octavian DOSPINESCU, Roxana-Marina STRAINU,
Alexandra NISTOR
Faculty of Economics and Business Administration, AL.I.Cuza University, Iasi
linus@uaic.ro, doctav@uaic.ro,
roxana.strainu@gmail.com, alexandra.anichitoaei@yahoo.com

Nowadays, architectural software systems are increasingly important because they can determine the success of the entire system. In this article we intend to rigorously analyze the most common types of systems architectures and present a personal opinion about the specifics of the university architecture. After analyzing monolithic architectures, SOA architecture and those of the micro- based services, we present specific issues and specific criteria for the university software systems. Each type of architecture is rundown and analyzed according to specific academic challenges. During the analysis, we took into account the factors that determine the success of each architecture and also the common causes of failure. At the end of the article, we objectively decide which architecture is best suited to be implemented in the university area.
Keywords: *System Architectures, Monolithic Architecture, SOA, Micro-Services Architecture, University System Architecture*

1 Introduction

The software systems currently face a multitude of challenges resulting from different factors such as high complexity, changing technology and the need for interoperability of heterogeneous data. All these factors have led to the emergence of various architectural models that have tried to address these challenges. Chronologically speaking, we believe that the most important steps are those focused on monolithic architectures and service oriented architectures. We estimate that the new wave will have in the forefront the micro-services based architectures in response to the challenges of interoperability. For example, in the last 10 years the market of the solutions dedicated to the interoperability [1] has grown from $3.4 billion (2004) to $11 billion (2008) and over $20 billion in 2015. The monolithic architectures focused on a traditional approach; although some authors consider them outdated, many current systems are designed in this style. Their integration is achieved mostly through SOA, but SOA limitations and challenges have prompted the emergence of a new vision: micro-services. In this paper we will try to make a comparison of these approaches and highlight their specific in the academic field.

2 Monolithic Architecture

When referring to monolithic architecture, the literature in this field avoids defining the term. According to Aoyama [2] this type of architecture is considered to be conventional and belongs to the older styles adopted during the development of new software. Having as starting point the idea promoted by Aoyama, Lake [3] considers that the monolithic name refers to the organizing of the fundamental application elements together in a single component or unit. In other words, the monolithic approach can be viewed as an integrated architecture design in comparison to a modular one. In general, the expression "monolithic" is used for lack of a better term, to indicate that all different types of the foundational architectural elements of the application can be used together in one block.

In fact, the subject has been approached by the researchers especially when referring to other categories of architecture. Arguing the importance of the codebase architecture in the development of an open source program, Baldwin & Clark [4] underline the differences between monolithic and modular architectures. While in the first case the participants have no access to other co-workers codes, in the second case the developers can join their efforts when writing the code. A similar idea

DOI: 10.12948/issn14531305/19.4.2015.02

has been stated by Carbonell et al. [5] when discussing the solving capabilities of PRODIGY architecture, which is modular. As a result, in comparison with SOAR (a monolithic architecture), the authors consider that PRODIGY is superior in terms of engineering principles.

Starting from some relevant articles concerning monolithic architecture, we notice that this concept is considered to be obsolete, the need of architectural restructuration being suggested by other researchers of the field. As an example, Mens et al. [6] propose the architectural restructuration of a monolithic model into a client-server or a three-tiered one, through which the user interface, business logic, and data layer can be clearly separated.

Using a monolithic architecture, the user interface elements can be mixed with the business logic, and the data management code. It doesn't mean that all elements will always be present in all software classes of the code, but the design allows them to be mixed together into one unit. *The positive aspect* of a monolithic approach regards a lower complexity of interaction between parts when multiple components or modules can be gathered into a single unit. Another positive is the ease of seeing a whole process in one place. For example, the user interface code can be seen along with the processing of data from the interface and the persisting of it to a database in a single class and file. The ability to use logic to manipulate the user interface, which is typically less dynamic, is another benefit of using a monolithic approach to the architecture [7].

A short characterization of the concept is made by David A. Penny [8], summarizing the main *features* of a monolithic architecture as:

- Normally, the programming language used is single;
- The code is compiled and linked through a unique (monolithic) program.
- When is operating, it can be both in: batch mode and GUI mode.
- Data used can be load into memory and to write *all* back on explicit save. No simultaneous data sharing.

Concerning the data used in this kind of architecture, we noticed that it is manipulated and read directly into the memory of the program. In order to save it, there are two option, first to use the same source and, secondly, to select a different one.

Regarding the changes which can be made into the programs developed using this architecture, the issue of visibility occurs. In this case, there are two options for seeing the changes done by a user by others. Because a multi-user access is not possible in this type of application, only sequential access is granted. As a result, changes made cannot be seen if the data is read into the memory by each copy of the program. In our opinion, the improvement of a monolithic architecture requires extra cost in order to: offer access to multiple users, connect to relational databases, update simultaneously volatile data etc.

Some example of applications using monolithic architectures could be some office and communication (e-mail) applications, accounting packages, business reporting, payrolls.

Table 1. Monolithic – advantages and disadvantages [8]

Advantages	Disadvantages
• Performance improvements	• Impossibility to be accessed by multiples users
• Simplicity in writing and other tasks	• Problems when processing large data

In terms of *advantages*, the *performance* is the first feature evoked by specialists when referring to this type of architecture. Concerning specifically the capability to data access from the applications developed using the monolithic model, this kind of operations are highly optimized considering that: (1) data is read directly from disk through the file system and (2) the user has also the possibility to cache and pre-fetch built-in data. When taking into account data update operations, in-memory is

DOI: 10.12948/issn14531305/19.4.2015.02

massively quicker, while caching is not an option for shared data systems because of the delays encountered while committing changes to a record.

Simplicity, the second advantage of this kind of architecture, is due to the fact that there is less code to write and also by the fewer issues to deal with, such as: locking, integrity, performance, transactions, geographic distribution etc.

Regarding *disadvantages*, the most obvious is the *impossibility of using the system by multiple users in the same time* (multiuser is not quite an option). As a solution, several measures can be enforced, such as: (1) allowing datasets to merge multiple files or (2) a hybrid approach, using complex monolithic analysis software and a simple data client/server update software.

The second biggest concern refers to the *impossibility of processing a large amount of data*, which could led to increase the time amount required to load data into memory or to increase virtual memory amount used. Solutions to this kind of problems could consider sequential or selective access can be granted into the application.

Likewise, other disadvantages could me mentioned, related to other specific needs of the users. In this regard, Roschelle et al. [9] discuss the negative aspects of monolithic architecture systems when referring to the *educational software*. According to the same authors, the monolithic architecture used to develop various software applications could affect five types of organization roles as:

1. Funders, through the money wasted on redundant coding;
2. Developers, whose wide purpose of required functions lead to a decrease of quality;
3. Researchers, which cannot perform a viable systematic comparison between programs;
4. Authors, which are incapable of customizing, integrating or extending different products; and, finally,
5. Schools, which develop and / or use incompatible, expensive or fragmentary software.

Considering the increased disadvantages in comparison to the benefits of monolithic architectures, the research trends in the field are justified as are the main opinions formulated in the detriment of this type of model.

3 Service Oriented Architecture

Service Oriented Architecture is a way to build a technology-independent architecture. According to [10] Service Oriented Architecture is an architectural style for building systems based on interactions of loosely coupled, coarse-grained and autonomous components called services. Each service exposes processes and behavior through contracts, which are composed of messages and discoverable addresses called endpoints. A service's behavior is governed by policies that are external to the service itself. The contracts and messages are used by external components called service consumers.

The SOA basic architecture contains 3 main components [11]:

- Service provider;
- Service consumer;
- Service registry.

These components interact in order to publish, find, bind and invoke specific services, as we can see in the following figure.

DOI: 10.12948/issn14531305/19.4.2015.02

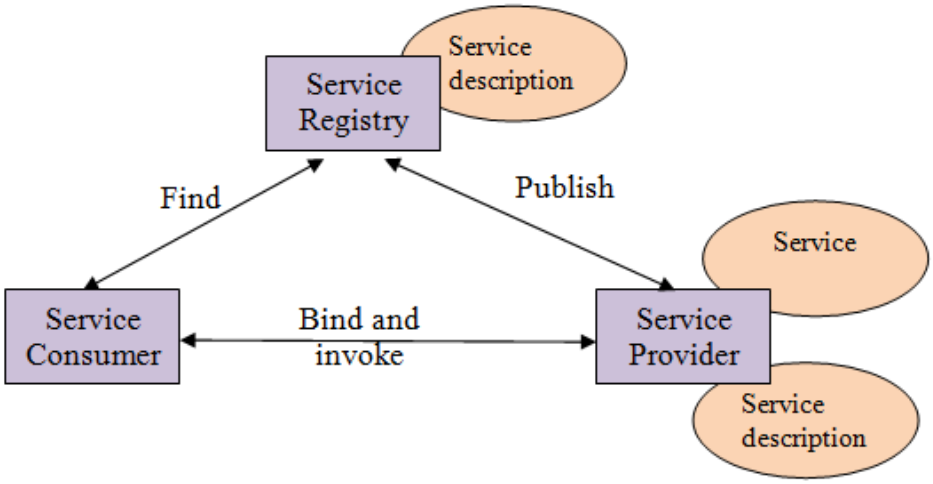

Fig. 1. The general components of SOA [11]

Some authors [12] consider that SOA is not just an architecture of services seen from a technological perspective, but the policies, practices, and frameworks by which we ensure the right services are provided and consumed.

There are some characteristics of SOA [13]:

- Services communicate with messages that are defined by XML schemas. The messages go through heterogeneous environments and they have the information needed in order to run an action.
- Web Services Description Languages (WSDL) is the language for describing the interfaces of SOA services.
- The services in SOA are managed by a registry that acts as a directory listing. The applications must "read" the registry, find the specific service and invoke it.

The advantages of using SOA are described by the specialized literature [14]:

- Reduction in development time and cost;
- Lower maintenance cost;
- High-quality services;
- Lower integration costs;
- Reduced risk.

A model of a general business SOA is presented in the following image.

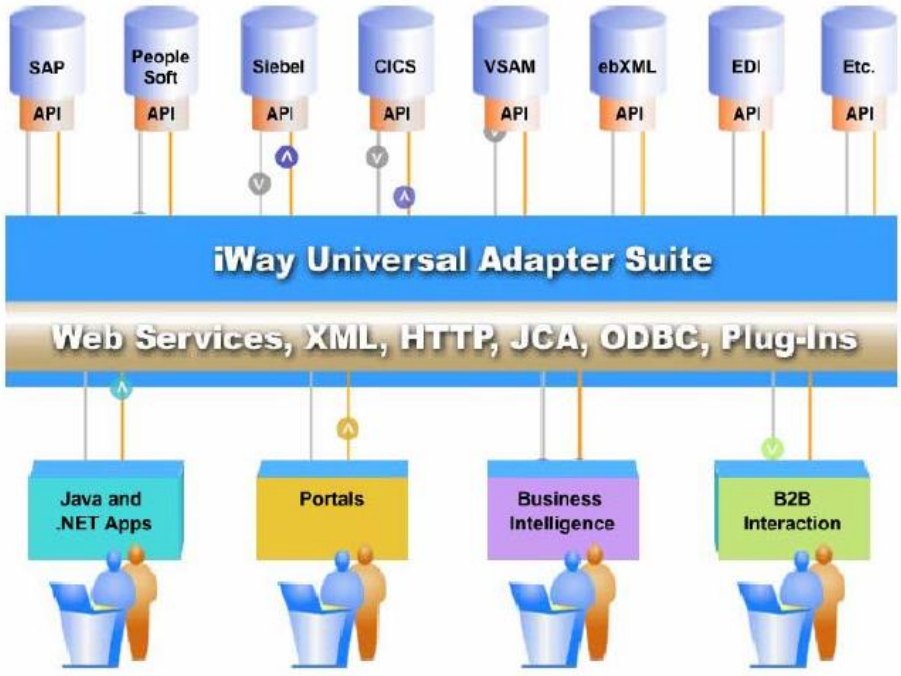

Fig. 2. A model of SOA integration [15]

DOI: 10.12948/issn14531305/19.4.2015.02

In the SOA model we have one or more adapters that manage the interaction between the consumer, the services and the provider, using standards like Remote Procedure Call or Representational State Transfer as described in [16].
From different points of view, SOA has many challenges in real business [17]: lack of experts, service identification roadmap, SOA delivery strategy, business IT alignment, economic issues, lack of long-term planning and strategy, stability, complexity, service boundaries, interoperability.

4 Micro-Services Architecture

Micro-services came out as a relative new approach from some practitioners which were looking for an architectural style even "more" democratic than the one traditional SOA could provide. Being in its infancy, micro-services architecture is still in search for a widely-adopted definition, still evolving and unproven over the long term.

J. Thönes sees micro-services as software apps (how small, how big?) independently developed, managed and maintained: "deployed independently, scaled independently, and tested independently and that has a *single responsibility*" [18].
Galen Gruman and Alan Morrison see micro-services architecture (MSA) as those service-components with "greater modularity, loose coupling, and reduced dependencies all hold promise in simplifying the integration task" [19]. One of the most compelling points of view comes from Martin Fowler that considers micro-service architecture [20] as a method "to describe a particular way of designing software applications as suites of independently deployable services". Fowler define micro-service architectural style as "an approach to developing a single application as a suite of small services, each running in its own process and communicating with light-weight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery."

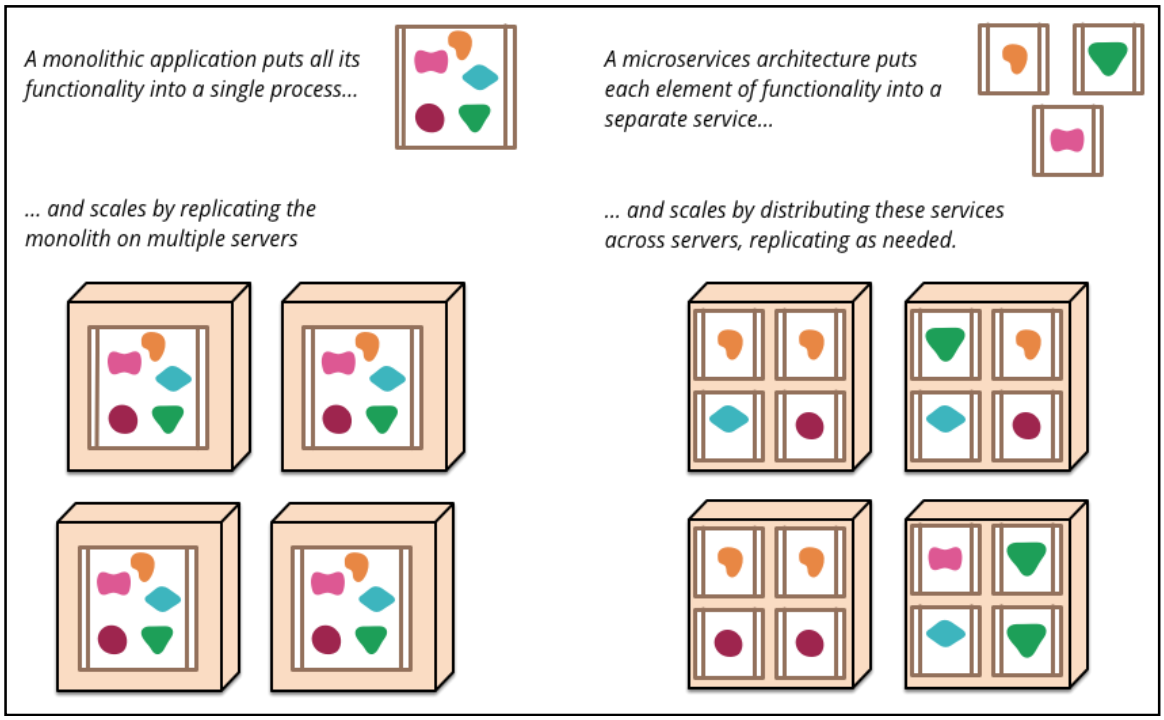

Fig. 3. Monoliths vs. micro-services

Sam Newman wrote some of the few books about MSA [21] for practitioners where he identified micro-service emerging base coming from Domain-Driven Design, Continuous

DOI: 10.12948/issn14531305/19.4.2015.02

delivery, On-demand virtualization, Infrastructure automation, Small autonomous teams, Systems at scale. Sam Newman sees micro-services as "small, autonomous services that work together. Small, and focused on doing one thing well" and brings out "The Single Responsibility Principle" defined by Robert C. Martin: "Gather together those things that change for the same reason, and separate those things that change for different reasons".

The last point of view in defining micro-services architecture that we will expose is the one of H. Kurhinen: "Usually micro-services are quite standalone, they have their own process, manage their own dependencies and possibly manage their own database connection. Micro-services may also have their own API for communication" and "it will be better to wrap micro-services inside a lightweight communication layer" [22].

J.Thönes characterizes the micro-services architecture as a lightweight stack ready to be deployed using lightweight container runtimes as [23]: embedded Jetty, embedded Tomcat, SimpleWeb or WebIt. To picture this statement, this author opposes them to the heavyweight category of centralized ESB. He remarks that another defining feature of micro-services is the movement of the complexity from the monolith into the networking layer. To model micro-services architecture it is proposed the domain-driven design approach (of Eric Evans) with a "service" label. Also, Galen Gruman, Alan Morrison in their effort to characterize micro-services approach underlines the following features [24]:

- web-scale development: software that must evolve quickly, whose functionality is subject to change or obsolescence in a couple of years—even months—and where the level of effort must fit a compressed and reactive schedule;
- dependencies: pre-SOA tight coupling, traditional SOA loose coupling, MSA decoupled;
- simple parts with clean, messaging-style interfaces;
- simpler messaging systems such as Apache Kafka;
- fine-grained, stateless, self-contained nature: easy to update, replace, remove, or augment.

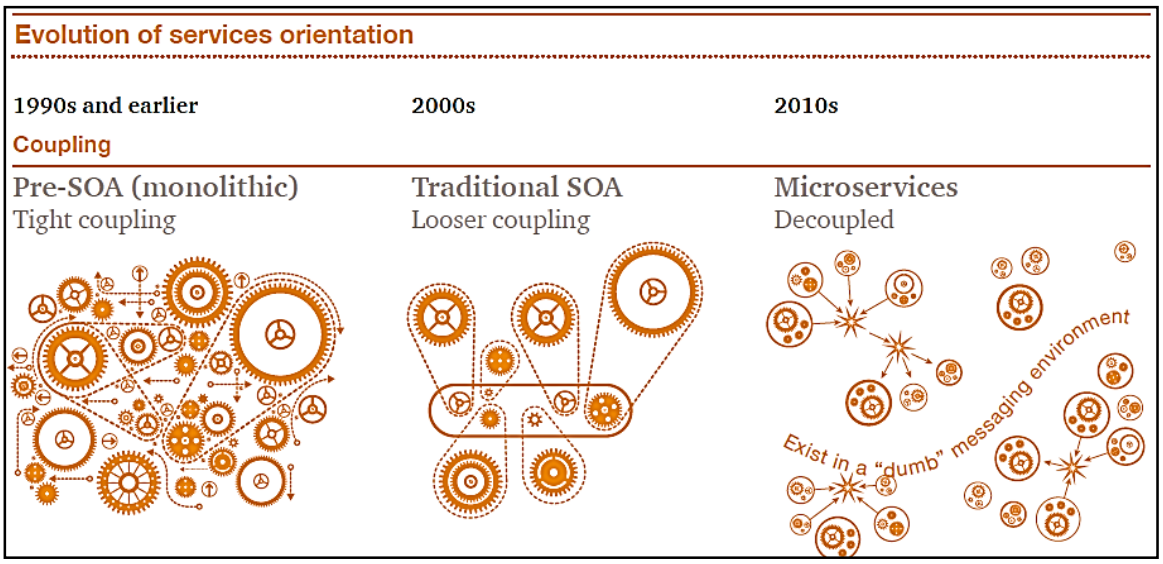

Fig. 4. The evolution of services orientation [24]

Galen Gruman and Alan Morrison proposed a comparison between Service-Oriented-Architectures and Micro-services-Architecture taking into account some technical criteria like messaging type, programming style, application session state and databases.

DOI: 10.12948/issn14531305/19.4.2015.02

Table 2. SOA vs. MSA

Criteria	SOA	MSA
Messaging style	Smart, but dependency-laden ESB	Dumb, fast messaging (Apache Kafka)
Programming style	Imperative model	Reactive actor programming model that echoes agent-based systems
State	Stateful	Stateless
Messaging type	Synchronous: wait to connect	Asynchronous: publish and subscribe
Databases	Large relational databases	NoSQL or micro-SQL databases blended with conventional databases

Martin Fowler - one of the most respected author in the architecture patterns domain - reviews an extensive set of characteristics of a Micro-service Architecture emphasizing the following aspects [25]:

- componentization via Services (rather than libraries): services meaning out-of-process components that communicate through mechanisms as web-requests or RPC using explicit component-published-interface (interface outside the code-base where is defined, published vs. public and not public vs. private);
- organized around Business Capabilities: implementation around business area, from UIX to persistent storage to external service integration;
- products not projects: the development team owns the product during its full lifetime (quoting Amazon: "you built it you run it");
- smart endpoints and dumb pipes: "micro-services aim to be as decoupled and as cohesive as possible"; cohesive meaning that they encapsulate their own (complete) business logic, decoupled meaning to communicate through simple messaging or lightweight messaging bus (no inter-process communication);
- decentralized governance: avoid standardization and overhead, use patterns like tolerant reader and consumer-driven contracts (service evolution pattern);
- decentralized data management: decentralized conceptual level supposing "conceptual model of the world will differ between systems" involving concept of Bounded Context Domain-Driven Design (Evans); decentralized data storage taking into account polyglot persistence: each service - each database using different database systems - and distributed transactions problem, eventually consistency could be tolerated;
- infrastructure automation covering continuous delivery, continuous integration, automated deployments, automated tests and service versioning management (especially in production);
- design for failure: tolerate the failure of services; manage failures: detect and restore faulty services;
- evolutionary design: service decomposition (from SOA design principles) seen as a tool to enable control of changes in software applications at the pace of business changes. The micro-services architecture must have the property of service independent replacement and upgradeability.

Sam Newman also made a compelling analysis of the most defining feature list [26] to consistently describe software architecture as being micro-services-based. He starts with "small enough and no smaller" and small service for small teams (agile-scrum approach) then continues with:

- *autonomy*: "The golden rule: can you make a change to a service and deploy it by itself without changing anything else?";
- technology *heterogeneity*;

DOI: 10.12948/issn14531305/19.4.2015.02

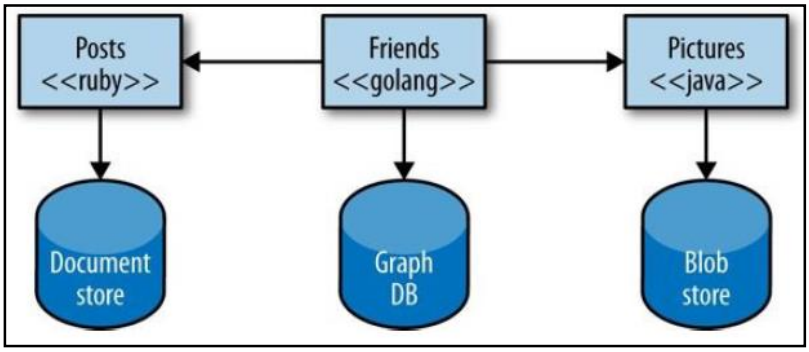

Fig. 5. Technology heterogeneity [26]

- *resilience*: single-component failure does not break down (tear apart) the whole system;
- (fine-grained) service *scaling*: scaling individual services by node-sharing;
- *ease of deployment*: deploy each service independently by the rest of the system;
- *organizational alignment*: smaller teams to better alignment to a distributed architecture;
- *composability*: as SOA attribute for reuse of functionality;
- optimizing for *replaceability*: "small in size, the cost to replace them with a better implementation, or even delete them altogether, is much easier to manage".

Micro-service architectural style is not a universal panacea, as monolithic and SOA are not. Also, being an emerging approach, micro-services assume some inherent risks in development, implementation and production process. In this regard, Galen Gruman and Alan Morrison summarized some key attributes that favor MSA [27]:

- fast applications and not quite elegant in the first place;
- (very) frequent changes in business functionality;
- functional isolation and simple integration are more important than module cohesiveness;
- functionality could be easily separated into simple, isolatable components;

They note that in MSA, integration could be the problem, and not the final solution: if you need complex integration, you shouldn't use MSA for that part of your software development. Instead, their opinion is to use MSA where broad integration is not a key need.

MSA implementations already took place, so micro-services exceeded the theoretical or conceptual discourse. In this regard, Matthias Vianden, Horst Lichter, Andreas Steffens described their experience with one practical micro-service-based reference architecture in an actual enterprise [28]. They approached the core problems of Enterprise Measurement Infrastructures (EMIs) using dedicated micro-services to implement business features like measurement, calculation and visualization. They managed to build and reuse micro-services that worked fluently without any major issue in an architecture that allowed very easy to add additional functionality by simply adding new augmented services.

5 Challenges for the Integration of Software Systems in the University Environment

Universities are special types of institutions which can have different structures depending on country or geographical area. According to the specialized literature [29], the competition that exists among the higher education institutions involves great efforts to adapt to the new requirements of the modern society. The educational offers must face the new challenges that require flexibility, rapidity, complexity and provide students both with specific habits and efficient work tools. Starting from this, there can be identified some *common features* inside these types of institutions, but also some *differentiations*. The identified common features that any university worldwide share are listed below:

- university employees and assets;
- students;
- study cycles;

DOI: 10.12948/issn14531305/19.4.2015.02

- curricula;
- specializations;
- evaluations;
- scholarship programs.

Those common features also share some different features depending on the *country, language or geographical area*. Regarding the software and hardware implementations, the challenges become more obvious. These challenges concern the software implementations on more levels, delimited by the scope they serve and the usage frequency in the following categories:

- learning management systems;
- evaluation and admission systems;
- administrative oriented systems.

The most used and well known tools are the Learning Management Systems (LMS). The Learning Management System (LMS) is a set of software tools (toolbox of programs) intended to support teaching, learning and course administration in order to deliver, to track and to managing education and online training [30]. These types of systems are oriented on delivering learning materials. By educational software we understand a "software designed for educational purposes" [31], so we can consider a LMS as an educational software. These types of software require a lot of attention, because the design and architecture *can differ by more aspects* like: the curricula, the number of students, the language, the technology, the connectivity, the hardware. If a LMS intends to encompass and an evaluation system, its design and architecture will *grow in complexity* and a poor designing can lead to a jam.

An important factor and problem in designing software architecture for this type of system is the *scalability* as a solution to the given question "How well a solution to some problem will work when the size of the problem increases" [32]. Of course this leads us to one of the conclusion that the storage space and storage location are very important.

Evaluation and admission systems are special types of processes that take place inside any university. They are used with a low frequency but have a higher *workload* than the ones special designed for learning, because the number of connections or users increases in and for a short period of time. Admissions take place twice a year, while the evaluation can happen from four times per year, depending on the rules and laws each university has adopted.

The challenge in designing a software architecture for those two processes stands in predict a maximum workload, using a hardware architecture suited for this types of operations. The performance and the response time are the main important factors in this case. The *response time* defined as "a measure of the *latency* an application exhibits in processing a business transaction. Response time is most often (but not exclusively) associated with the time an application takes to respond to some input" [33] is very important when we associate it with the evaluation process, because all evaluations are timed.

The evaluation process is much more complex than it looks. The second concern about evaluation is the equalization process. This can differ from each university, country or area and this can raise a problem when scholarship programs are involved in evaluation and equalization.

The performance factor "defines a metric that states the amount of work an application must perform in a given time, and/or deadlines that must be met for correct operation" [34]. When an admission deadline is on the field, the crucial factor concerns the *performance* of the system, along with the number of *simultaneous connections*.

Administrative oriented systems, or systems associated with administrative processes like financial and economical operations, the staff, organizational activities are dedicated software tools used by any type of institution. There can be met some big differences designing this type of software architecture and implementation. Each administrative or financial software is dedicated or associated with financial or economical laws each country has adopted. There can be a lot of *discrepancies in each system implementation and architecture*, depending on this main differentiation factor. The globalization is an important factor in eliminating this problem, but in a little

DOI: 10.12948/issn14531305/19.4.2015.02

measure. A solution to eliminate this problem is adapting the software to accept different types of indicators, depending on the type of required operations.

In this context the biggest challenge is to build a system that can accept and convert data from different sources and to transform it into usable data into the existing or new system. Of course every university *already have implemented software solutions* for almost all the processes identified before. One of the biggest challenges leaving from this point of view will be *adapting or improving the old software solutions*.

The main asset in the university is the data. Data must reside in a *secured* place. Data confidentiality is very important for each student. Any system must forbid unauthorized access to confidential data about students. This goes to the importance of hardware improvements, as long as technology evolves rapidly. In the same time, the world wide technological improvements must be taken very serious and embedded into the system. This should be an opportunity and not a problem, for each system involved in the universities work.

Another important aspect is the internet. Each country and continent has *different speeds of the Internet*. The software solutions designed for campuses can be used with an intranet solution to avoid bad Internet connections, but not every university has the funds to implement this, so internet dependence must be taken very serious when the software has the purpose to share the data externally via Internet. The *connections via mobile devices and mobile networks* can create issues in sending and receiving the data.

As discussed in the LMS, other important limitation or thing should be carefully analyzed when implementing a software solution is the language. Every application has versions in more languages; every international used website allows reading an interface in a desired language. This is a concern when the old solutions already use data designed in specific language, which isn't globally used, or when the alphabet is different than the one used in designing the software solution. Evaluation and admission systems can also be affected by the language and alphabet barrier if we talk about grades validations between universities in scholarships programs.

Another challenge regarding any implementation is the *internal regulation* of each university. While the specializations differ by curricula, and each discipline differs by university, also evaluation criteria are different. This challenges the designers to consider a solution to convert different evaluation criteria and scoring systems, between them and inside the same system. This requires a strong knowledge and documentation about grading systems worldwide. The grading system is again important in an evaluation system when data validation between universities is necessary in scholarships programs, highlighting the importance of data conversion in this type of system.

Different admission systems are also a challenge, because each university has its own admission system. When trying to implement a software solution to automate this process, the task is difficult, because this software solution can change from year to year, depending on many predictable or unpredictable factors. In an educational institution "the admission process is [either] centrally coordinated (i.e., examination scores are submitted to a central entity, which determines student placements) or the university system as a whole is centrally planned (i.e., the number of spaces available in each institution is determined by the national government)." [35]

The curricula is important in designing a software solution specialized on delivering learning material, because each discipline differs by the other in information or type of learning material. In the same time, another important factor is again data storage. Each university must choose the way data is stored, where it resides, who can access it. Different levels of data access are a challenge when the university or the country regulation allows only *specialized server solutions* which are not owned by the university. The access to existing data can also represent a problem and a challenge.

The number of students influences the number of concurrent connections which can be simultaneously active to the server, and this is

DOI: 10.12948/issn14531305/19.4.2015.02

why a hardware architecture suited to the size of the university is important. Private universities with a limited number of students don't necessary have this problem. But a solution which is scalable to any size or any number of connections is preferred regarding the course access, the admission and evaluation.
The technological challenge is the main factor that influences the whole system. First of all the *technological evolution from desktop to mobile devices* is a big step and an important shift in any software architecture. The software solutions for any area of interest must take into account and are almost dependent on mobile devices, as long as the number of mobile devices continues to grow. Hardware improvements are a key point to the success of any type of software implementation.
The classifications and issues identified above suppose that the system can be designed from the beginning. A big challenge is migrating from monoliths to micro services, or redesigning the structure, using heterogeneous data sources and converting existing data into valid data for the new system. If each process identified in the first part would have a special software solution which only allows data exports or owns an API to allow connection to other software solution the problem would become a great opportunity. Because each process has different behavior, each process *shares information* with the other, an *efficient way to communicate data* between those processes could be the key factor to improve the workflow into activities that take place inside each university and to avoid gaps or bottlenecks. The degree of *technical interdependencies* involved can be a significant break on innovation [36], so this is one of the main aspects that should receive more attention, also for the future existence and adaptation of the system.

6 Conclusions on Architectural Styles and Challenges of University Environment

Summarizing the critical challenges and problems from within an actual business context (as of university environment) that has to be addressed by nowadays evolutionary architectures, we could divide them in two categories:

- business challenges as:
  - education processes with common features but also with specific implementations;
  - specific educational processes having functional differentiations;
  - regional settings: country, language or geographical area;
  - financial or economical laws differentiations;
  - internal regulation differentiations;
- technical challenges:
  - growing in complexity;
  - scalability;
  - workload management;
  - short response time;
  - minimizing latency;
  - simultaneous connections;
  - legacy implemented solutions;
  - adapting or improving the old software solutions;
  - different speeds of the internet;
  - connections via mobile devices and mobile networks;
  - specialized server solutions;
  - technological evolution from desktop to mobile devices;
  - sharing information;
  - efficient way to communicate data.

Our analysis on monolithic, service-oriented and micro-services architectures, from previous sections, emphasizes some critical differences between those three architectural styles. In short, taking into consideration their layering approaches, monoliths include and closely integrate all levels and functional modules: UIX, business logic and database access. Also, service oriented architecture focuses on integration and loose-coupling of components within three or more vertical layers and on multiple functional horizontal layers. These horizontal layers could share infrastructure components (e.g. for database access) so they could not be entirely autonomous. Autonomy and extreme functional flexibility (as *replaceability*) are the distinguished features of micro-services architecture. These kinds of architectures have to support a set of criteria (as those from Table 3) based on those different business and technical problems previously

DOI: 10.12948/issn14531305/19.4.2015.02

outlined.

Table 3. Architectures and criteria

Criteria	Monolithic Architecture	Service Oriented Architecture	Micro-services Architecture
Business Criteria Support			
Adaptive business (educational) process	Yes*	Yes	Yes
Regional Settings	Yes	Yes	Yes
Integration: regional differentiations - heterogeneity	No	Yes*	Yes
Orchestration: global business features - homogeneity	No	Yes*	Yes*
Technical Criteria Support			
Integration : legacy autonomous systems	No	Yes	Yes*
Integration: platform heterogeneity	No	Yes	Yes
Integration: maximize functional flexibility	No	Yes*	Yes
Controlled integration complexity	No	Yes	Yes*
Functional autonomy	Yes	Yes*	Yes
Scalability	Yes*	Yes	Yes
Multiuser access	Yes*	Yes	Yes*
Web and mobile access	No	Yes	Yes
Data Integration and intercommunication	No	Yes	Yes*
Adaptive software solutions	No	Yes*	Yes
Performance: minimize response time	Yes	Yes	Yes*
Performance: minimize latency	Yes	Yes	Yes
Offline and online processing (specific internet availability)	No	No	Yes

(Yes* means that the current criteria is not quite fully supported, but could be but under special circumstances or taking into consideration some exceptions)

Our opinion is that none of the above architecture could be entirely avoided in a scenario where a development team tries to build a global software system (as in an university environment in the European context) where legacy systems (mostly being irreplaceable and having monolithic architecture) has to be integrated within global business processes, and where new adaptive features have to be added and accommodated on a constant (or even growing) change rate.

Consequently, our "best fit" approach assumes:

- treating already in place business systems as a set of services that could participate into a micro-services architecture: each

DOI: 10.12948/issn14531305/19.4.2015.02

external system might be considered as a monolith that could be integrated by using a proxy-autonomous service (supporting functional autonomy and flexibility);

- building additional functional and infrastructure services using an evolutionary SOA architecture where the developing team could control complexity and coupling aspects. If the development team is homogeneous and is already committed to an infrastructure of services (as persistence or database access) that will guarantee a certain level of performance, then a proven SOA architecture could take place, otherwise, in order to offer a more democratic environment, micro-services architecture could be at least an exploratory solution.

Acknowledgments:
„This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS – UEFISCDI, project number PN-II-RU-TE-2014-4-0748".

References

[1] O. Dospinescu and D. Popescul, „The Adoption of Electronic Banking Services in Developing Countries – the Romanian Case", *Future of Banking after the Year 2000 in the World and in the Czech Republic*, 2005, pp. 1609-1616.

[2] M. Aoyama, „New age of software development: How component-based software engineering changes the way of software development", in *1998 International Workshop on CBSE*, pp. 1-5.

[3] B. Lake, „An empirical evaluation of an agile modular software development approach: A case study with Ericsson", Independent Thesis, Stockholm University, 2012.

[4] C. Y. Baldwin and K. B. Clark. "The architecture of participation: Does code architecture mitigate free riding in the open source development model?", *Management Science*, 2006, pp. 1116-1127.

[5] J. Carbonell, O. Etzioni, Y. Gil, R. Joseph, C. Knoblock, S. Minton and M. Veloso, „Prodigy: An integrated architecture for planning and learning", *ACM SIGART Bulletin*, 1991, *2*(4), pp. 51-55.

[6] T. Mens, J. Magee and B. Rumpe, „Evolving software architecture descriptions of critical systems", IEEE Computer Society, 2010, vol. 43, issue 5, pp.42-48.

[7] B. Lake, „An empirical evaluation of an agile modular software development approach: A case study with Ericsson", Independent Thesis, Stockholm University, 2012.

[8] D. A. Penny, „Software Architecture & Design Course Notes", University of Toronto, available at: http://www.cs.toronto.edu/~penny/teaching/csc407/lectures/13monolithic.pdf.

[9] J. Roschelle and J. Kaput, „Educational software architecture and systemic impact: The promise of component software", *Journal of Educational Computing Research*, 1996, *14*(3), pp. 217-228.

[10] R. Arnon, „SOA Patterns", Manning Shelter Island, USA, 2012, pp. 4-6.

[11] A. M., Karande, V. Chunekar and B. B. Meshram, „Working of Web Services using SOA", *International Journal of Advanced Research in Computer Science*, Vol. 1, No. 4, nov-dec. 2010, pp. 292-296.

[12] D. Sprot and L. Wilkes, „Understanding Service Oriented Architecture", *Microsoft Developer Network*, 2004, available at: https://msdn.microsoft.com/en-us/library/aa480021.aspx#aj1soa_topic5

[13] R.R. Kodali, „What is Service Oriented Architecture?", *Javaworld*, june 2005, available at: http://www.javaworld.com/article/2071889/soa/what-is-service-oriented-architecture.html

[14] A. M., Karande, V. Chunekar and B. B. Meshram, „Working of Web Services using SOA", *International Journal of Advanced Research in Computer Science*, Vol. 1, No. 4, nov-dec. 2010, pp. 292-296.

[15] L. Hurbean, D. Fotache, V. D. Pavaloaia and O. Dospinescu, „Platforme integrate pentru afaceri. ERP", Editura Economica, Bucuresti, 2013, p. 258.

[16] O. Dospinescu and M. Perca, „Web Services in Mobile Applications",

DOI: 10.12948/issn14531305/19.4.2015.02

Informatica Economica Journal, Vol. 17, No. 2, pp. 17-26, 2013.

[17] A. T. Zadeh, S. Sahranb and M. Mukhtar, „Service Identification in SMEs: Appropiate Elements and Methods", *International Journal of Machine Learning and Computing*, vol. 3, no. 3, pp. 279-283, June 2013.

[18] J. Thönes, „Micro-services", *IEEE Software*, ISSN 0740-7459/15, 2015, pp. 113-116.

[19] G. Gruman and A. Morrison, „Micro-services: The resurgence of SOA principles and an alternative to the monolith", *Technology Forecast: Rethinking integration*, Issue 1, 2014, www.pwc.com/technologyforecast

[20] M. Fowler, „Micro-services", available at: http://martinfowler.com/articles/micro-services.html, 25 March 2014.

[21] S. Newman, „Building micro-services", O'Reilly Media, 2015.

[22] H. Kurhinen, "Developing micro-service-based distributed workflow engine", Bachelor's Thesis, Mamk University of Applied Sciences, 2014.

[23] J. Thönes, „Micro-services", *IEEE Software*, ISSN 0740-7459/15, 2015, pp. 113-116.

[24] G. Gruman and A. Morrison, „Micro-services: The resurgence of SOA principles and an alternative to the monolith", *Technology Forecast: Rethinking integration*, Issue 1, 2014, www.pwc.com/technologyforecast

[25] M. Fowler, „Micro-services", available at: http://martinfowler.com/articles/micro-services.html, 25 March 2014.

[26] Sam S. Newman, „Building micro-services", O'Reilly Media, 2015.

[27] G. Gruman and A. Morrison, „Micro-services: The resurgence of SOA principles and an alternative to the monolith", *Technology Forecast: Rethinking integration*, Issue 1, 2014, www.pwc.com/technologyforecast

[28] M. Vianden, H. Lichter and A. Steffens, „Experience on a Micro-service-based Reference Architecture for Measurement Systems", 2014, *21st Asia-Pacific Software Engineering Conference*, ISSN 1530-1362/14, IEEE 2014

[29] N. Dospinescu, M. Tatarusanu, G. I. Butnaru and L. Berechet, „The Perception of Students from the Economic Area on the New Learning Methods in the Knowledge Society", *The AMFITEATRU ECONOMIC journal*, vol. 13, no. 30, Academy of Economic Studies-Bucharest, Romania, pp. 527-543.

[30] Z. Gulzar, „An exploratory Analysis of Learning Management System as an Emergin ICT tool in India", Jun. 2015, *Bonfring International Journal of Industrial Engineering and Management Science*, Vol. 5, No. 2

[31] P. Tchounikine, „Computer Science and Educational Software Design: A Resource for Multidisciplinary Work in Technology Enhanced Learning", *Springer Science & Business Media*, p. 4, June 2011.

[32] I. Gordon, „Essential Software Architecture", Second Edition, Springer Science & Business Media, 2011, p. 24.

[33] I. Gordon, „Essential Software Architecture", Second Edition, Springer Science & Business Media, 2011, p. 25.

[34] I. Gordon, „Essential Software Architecture", Second Edition, Springer Science & Business Media, 2011, p. 24.

[35] R. M. Helm, „University Admission Worldwide", World Bank, p. 7, Jul. 2008.

[36] W. D. Hutton et al, „The Social Shaping of a Virtual Learning Environment: The Case of a University-wide Course Management System", *Electronic Journal of e-Learning*, Vol. 2, Issue 1 (February 2004), pp. 69-80.

DOI: 10.12948/issn14531305/19.4.2015.02

Cătălin STRÎMBEI has graduated the Faculty of Economics and Business Administration of Al.I.Cuza University of Iaşi in 1997. He holds a PhD diploma in Cybernetics, Statistics and Business Informatics from 2006 and he has joined the staff of the Faculty of Economics and Business Administration as teaching assistant in 1998 and as associate professor in 2013. Currently he is teaching *Object Oriented Programming*, *Multi-Tier Software Application Development* and *Database Design and Administration* within the Department of Business Information Systems, Faculty of Economics and Business Administration, Al.I.Cuza University of Iaşi. He is the author and co-author of four books and over 30 journal articles in the field of object oriented development of business applications, databases and object oriented software engineering.

Octavian DOSPINESCU graduated the Faculty of Economics and Business Administration in 2000 and the Faculty of Informatics in 2001. He achieved the PhD in 2009 and he has published as author or co-author over 30 articles. He is author and co-author of 10 books and teaches as an associate professor in the Department of Information Systems of the Faculty of Economics and Business Administration, University Alexandru Ioan Cuza, Iasi. Since 2010 he has been a Microsoft Certified Professional, Dynamics Navision, Trade & Inventory Module. In 2014 he successfully completed the course "Programming Mobile Applications for Android Handheld Systems" authorized by Maryland University. He is interested in mobile devices software, computer programming and decision support systems.

Roxana-Marina STRAINU graduated in 2014 the Master of Business Information Systems at the Faculty of Economics and Business Administration, Alexandru Ioan Cuza University of Iasi. She also graduated the Faculty of Mathematics in the year 2005. She is interested in developing smart systems and mobile applications on Android platform. Now she is a PhD student in the business information systems area.

Alexandra NISTOR graduated the Faculty of Economics and Business Administration in 2011 and the Master of Business Information Systems at the Faculty of Economics and Business Administration in 2013. Her research interests include the use of automated testing in small and medium companies. Now she is a PhD student in the business information systems area.

DOI: 10.12948/issn14531305/19.4.2015.02