Integrate Enterprise Systems to our Hyperconnected World: Anything, Anywhere, Anytime through architectural design

Completed Research Paper

Eldar Sultanow
University of Potsdam
Eldar.Sultanow@wi.uni-potsdam.de

Carsten Brockmann
University of Potsdam
Carsten.Brockmann@wi.uni-potsdam.de

Renée M. E. Pratt
Washington and Lee University
PrattR@wlu.edu

Katja Andresen
Berlin School of Economics and Law
katja.andresen@hwr-berlin.de

ABSTRACT

The architectures of currently sold Enterprise Systems were developed in a time when the amount of data to be processed was limited. Since then the necessity to capture and process real-time data from multiple sources has surged and needs to be considered in a world where everything must be exchanged and available anywhere, anytime and in any format. Yet the abstinence of novel approaches on the architectures of Enterprise Systems creates a gap between the increasing requirements and existing information systems. In this paper, we suggest a new architectural design approach, which will close the gap between increasing requirements and existing information systems. In order to determine a future-proof architecture, the authors conducted a Delphi survey where technology providers and users were inquired on the business needs and technical requirements. The result of the Delphi survey has been used to create a proposal for a different approach towards ES architectures.

KEYWORDS

Enterprise System, Hyperconnected World, Software Architecture, Big Data

INTRODUCTION

Momentarily, companies using an Enterprise System (ES) reach the limits of their ability of processing and analyzing data. One of the most important tasks of an Enterprise System as a company-wide information system is support changes, e.g. in altered business processes or in the use of new technology. An increasingly important requirement is the ability to deal with "Big Data". In order to meet the needs of analyzing Big Data, it is necessary for the ES to capture all information. In particular information that is not always maintained in the conventional databases (Hopkins, Leganza, Cullen and Cahill 2011).

Currently there is a call to enterprise architects to assist their organization in creating more value for their business by incorporating Big Data techniques in their enterprise architecture (Hopkins et al, 2011). In particular, the Forrester report (Hopkins et al, 2011) expresses the need to maintain a holistic approach to information architecture and the evaluation of where BI and PA integrate to create a SOA capability. To meet these requirements and others is a major challenge for ES providers. As a result of this situation, providers of Enterprise Systems and ES-technologies recognize an emerging market for Big Data solutions.

This paper examines the current state of EA to find the gaps that currently hinder Enterprise Systems from supporting a hyperconnected world. We develop a framework that addresses the question - *How do we integrate an Enterprise System that incorporates the anywhere, anytime, and anything aspects of data through architectural design?* To gather the answer to this question we asked, *What are the requirements for an EA in a hyperconnected world and how does the current EA address*

these needs? Through the creation of requirements and identification of architectural imperfections, we develop a future-proof architecture of an Enterprise System.

In the remaining sections of the paper, we depict a historical account of the architecture of ES, describe the current state of ES architecture and define layers and dimensions of software architecture. Following this discussion, we present the methodology and Delphi study used to develop the proposed architecture, including a requirements section for future-proof architectures. Finally, we present our proposed architecture and explain how the findings from the Delphi study establish the new architecture and future research that will enhance the needs of ES providers. The goal of this paper is to propose a new architectural design approach, derived from a comprehensive Delphi study, that supports any type, anywhere, and anytime accessibility of data.

BACKGROUND:

The architecture of an ES determines the possibility of future use when environmental turbulences induce change (Andresen 2006). Environmental turbulences comprehend regulatory, organizational and operational changes leading to new technical and process related requirements, which also affect types of data and relations. Hence in the following paragraphs, we explore the history and trends of the architecture of an ES to see the gaps and areas of importance for future-proof architectures.

The architecture of ES shall not be confused with the enterprise architecture (EA) which explicitly documents and describes current and desired relationships among management and business processes as well as information technology (Kaisler, Armour and Valivullah 2005). The EA provides a companywide blueprint of the information systems (Armour, Kaisler and Liu 1999). The EA can be split into two components: The first component is the corporate architecture which contains the business processes and hierarchical information. The second component is the information systems, which consists of procedural models and solutions to be applied in order to reach the desired state of the corporate architecture (Gronau 2000).

The term software architecture is used on an ever increasing basis (Gorton 2006). According to Bass, Clements and Kazman (2003), software architecture of a computing system or program is the structure or structures of the systems, comprehending software elements, the externally visible properties of the elements and the relationships among them. Since the software architecture is an intangible conceptual entity, its elements should be illustrated by means of the software visualization discipline (Ghani, Rimal and Jeong 2012).

Enterprise Systems are software packages containing applications for different business areas (e.g. marketing, finance, etc.), storing and accessing information from a database management system, used to effectively and efficiently plan and manage the resources of companies (Sultanow, Brockmann and Gronau 2010). Enterprise Systems should be capable of planning and managing at least three resources (Gronau 2010). Exemplary resources are personnel, finances and manufacturing planning. Enterprise Systems must be developed applying an appropriate architectural approach so that the resources can be reflected within the system. There are multiple database management models to address these resources. In one approach, the architecture only consists of the presentation and a combined database and application layer (Giachetti 2010). A more common approach was taken by Gronau (2010), for whom the Enterprise System consisted of seven layers: (1)Infrastructure – where hardware is located, (2) Data – where the data is saved, (3) Application and (4) service – where the application's core resides, (5) Presentation – where the interaction with the user takes place, (6) Control – where the business processes are modelled, and (7) Adaption – where the customizing takes place. The three most cited layers are the presentation, application and data layers (Kogent Learning Solutions 2010). The introduction of layers into the software architecture is conceived to be part of the evolution of Enterprise Systems.

Continuing with the aim of investigating the history of ES architecture, we review dimensions of architectures to further classify, interpret, and analyse the differences among software architectures. Based on several scholarly publications (Gull 2011; Jamshidi, Ghafari, Ahmad and Pahl 2013; Rico, Sayani and Field 2008; Senthilvel 2008), we derived six dimensions to classify and compare software architectures (Figure 1). One dimension characterizes interrelated architectural concepts that have evolved. Specifically, (1) Programming Language Type, (2) Popularity of Languages, (3) Layers of Abstraction, (4) Software Technology, (5) Database Technology, and (6) Requirements-driven Architecture Trends. Programming languages as well as their popularity have a significant impact on software architectures, since languages are the basis for architectural frameworks. For example *Spring* (a DI-Framework), *Hibernate* (an OR-Framework) and the *Eclipse* (a OSGi-Platform) fashioned the architectural trend and are conducive to the popularity of the Programming Language Java (and also C#), in which these frameworks are implemented and thus available specifically for them. Thus it becomes clear why the most common dimension is the *programming language* along with the *language popularity* and their use in Information Systems.

Within the *layers of abstraction*, the Model-View-Controller (MVC) is one of the most fundamental concepts that identifies an architectural pattern as the de facto standard for the design of complex software systems (Buschmann, Meunier, Rohnert, Sommerlad and Stal 1999): "The MVC architectural pattern divides an interactive application into three components: Data, presentation and control." The separation of these components allows independent development and use. Therefore we can create different views, which give us multiple representations of one and the same model from reality. An abstract principle, which is also popular, is called Dependency Injection: An IoC (Inversion of Control)-Container generates objects externally, initializing them and injecting them into dependent objects (Mak 2008). In this way objects no longer define their dependencies. Dependencies are thus declared outside the objects.

The challenge of combining relational data structures and object-oriented programming concepts led the authors to introduce the *database technology* dimension. Object-Oriented database systems have been created in order to close this gap. Since relational database systems store and provide data in rows and columns only, the Object-Relational Mapping (ORM)-Frameworks raised, forming their own architectural layer. This concept returns to Scott (1996), according to whom, objects (in terms of object oriented programming) will be mapped to relational database records and vice versa. Thus, reviewing the historical trends of an ES architecture provides us with a foundation to determine the needs and requirements of experts who expect an ES with the capabilities of any type, anywhere, and anytime accessibility of data.

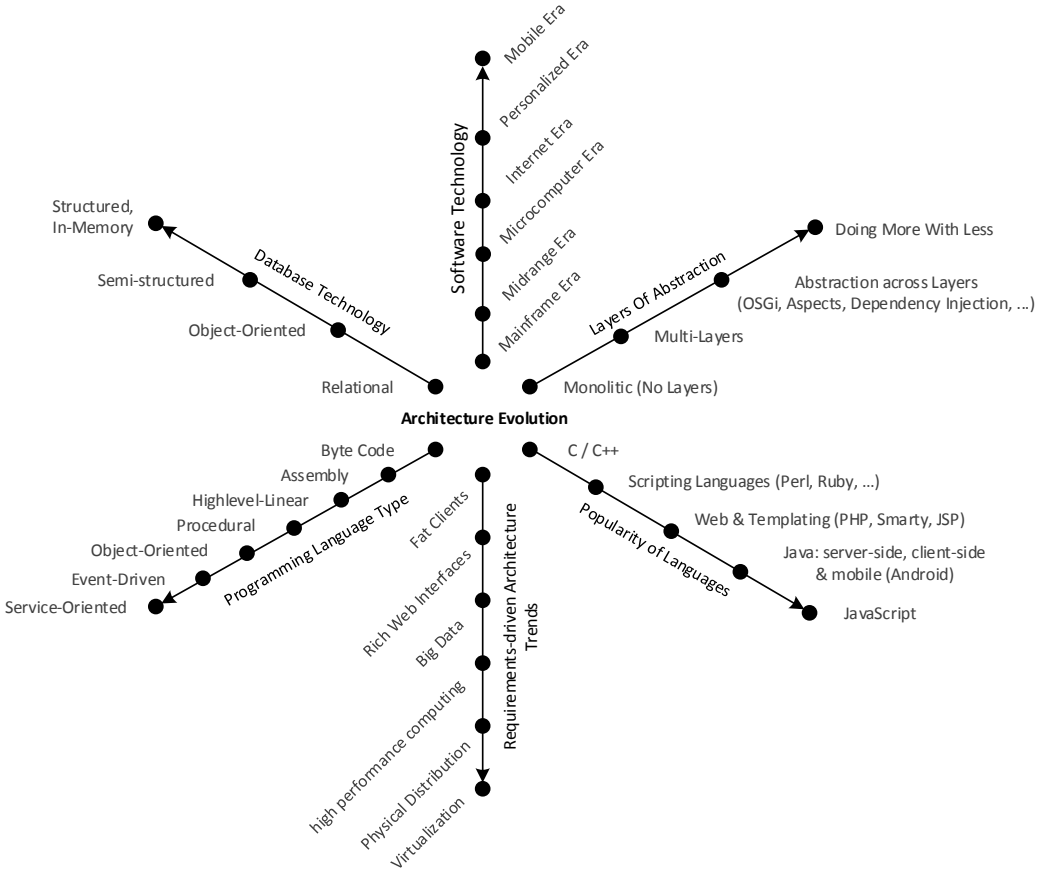

Figure 1: Exemplary values for dimensions for characterizing and classifying software architectures

METHODOLOGY

In order to determine the appropriate architecture for Enterprise Systems, the qualitative Delphi approach was used. Using the Delphi technique, a group of persons considered to be experts in the field are gathered to respond to a series of questionnaires (Gerald and Cluskey 2008). The method consists of five stages (Bradford 2010): First, experts are identified. Second, experts are contacted and asked to cooperate. Third, questionnaires are sent to panellists. Fourth, the results of the questionnaires are analyzed and selected panellists are invited to provide their opinion on topics considered to be important. The fifth and final stage consists of circulating the newly generated views on the topic as well as subjects of mutual disagreement (Jost, Nilakanta and Willis 2002), asking the experts for their opinion on the cases. For this contribution, 21 experts participated in the Delphi Survey (See Table 1). For this survey, the experts were obtained from the formal and informal network of the authors. The Delphi technique has been applied within this contribution to obtain the essential layers of the ES as well as the requirements towards the layers of an ES.

Company	Experts involved	Location
Software consultancy with a focus on user centric design	2	Potsdam, Germany
Automobile manufacturer	1	Wolfsburg, Germany
IT-Outsourcing company for software development, digital medical products	3	Dalian, China
Operator and developer of social networks	4	Potsdam, Germany
Pharmaceutical wholesaler, focussing on oncology	3	Hof, Germany Asch, Czech Republic
Service provider and value added seller for pharmaceuticals, and medical equipment	3	Hongkong, China
Automobile Manufacturer (Headquarter in Munich)	1	Tokyo, Japan
Chinese technology company (Headquarter in Shenzhen)	4	Düsseldorf, Germany

Table 1: Origin and amount of experts participating in the Delphi study

Figure 2 shows the procedure deployed to determine the architecture of the Enterprise System. By adding the results of the Delphi study to the current delta in literature, a new model is proposed which is presented further on (Figure 5).

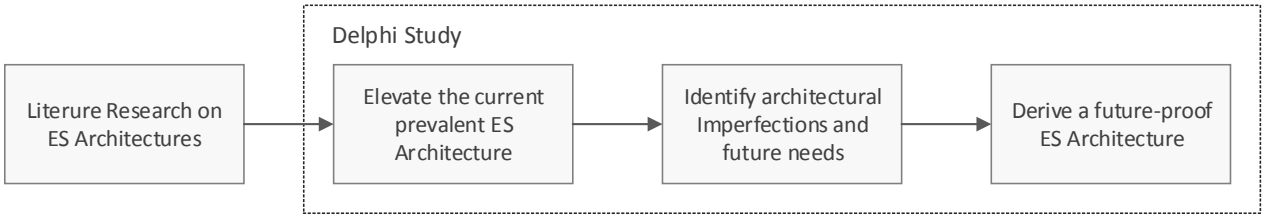

Figure 2: Procedure to analytically derive the ES architecture

THE CURRENT ARCHITECTURE OF ES

In the past, multi-layer architectures have been popular – even for small and simple web applications. From the interviews with the CIOs, we indicated that a significant majority of IT decision-makers clearly preferred Web-based multi-layer architectures (as shown by Figure 3) in recent years. With these architectures, CIOs and software engineers have tried to abstract as much as possible, in order to make purposive implementations (system components) interchangeable. The large number of frameworks that appeared in the market was also another reason for this increase in abstraction. And following the trend of SOA, enterprise portals and B2B automation the architectures of ES turned more and more into web architectures.

The interchangeability (and even simultaneous applicability) of sub-systems vary from database systems in the backend to ORM frameworks and also template engines in the (web) front-end. This was of course at the expense of performance, since adapters between each layer had to take care of the cross-layer communication of framework-specific implementations. This inevitably led to so called "Glue Code".

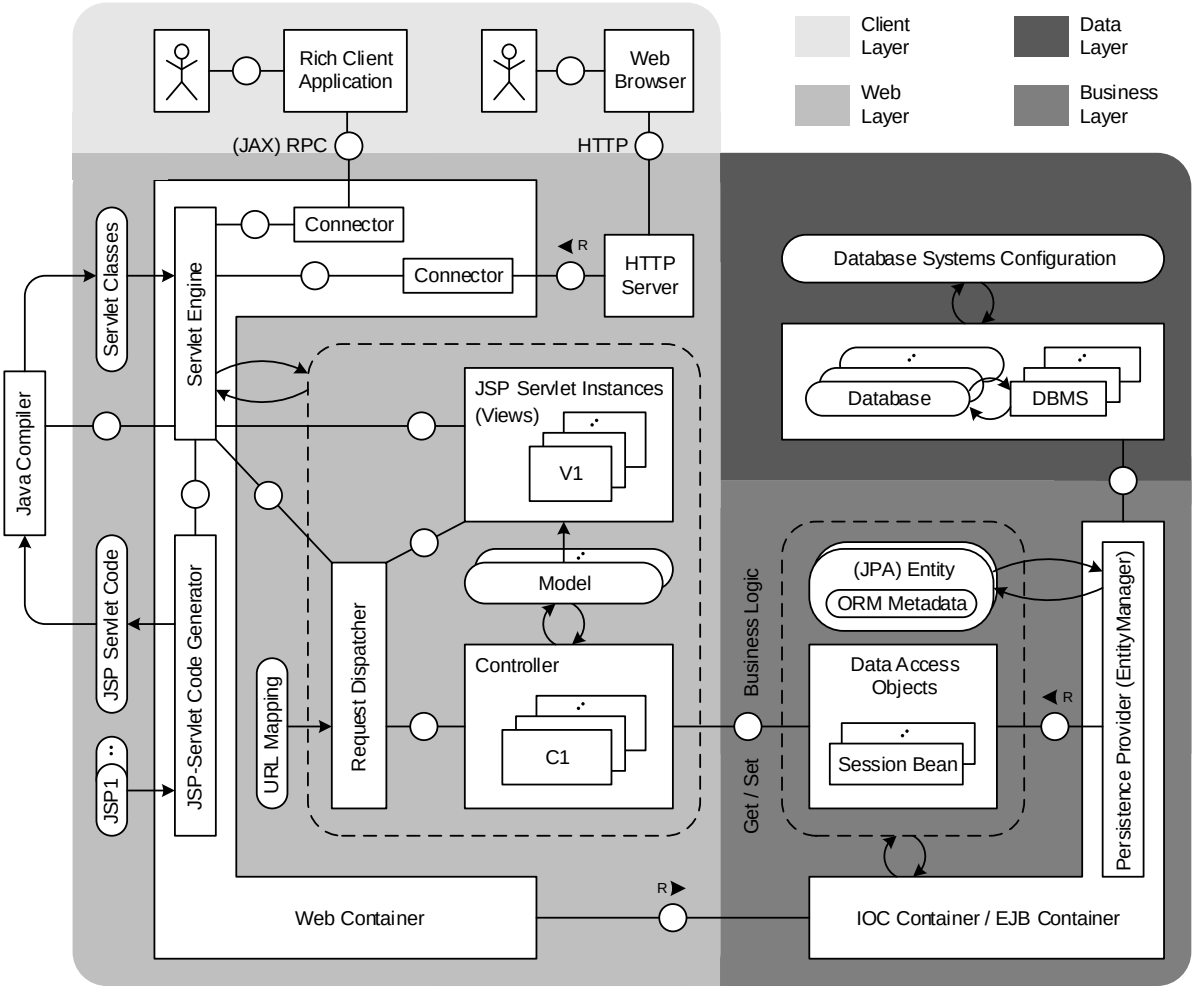

Figure 3: Typical multi-layer architecture of enterprise web applications (Sultanow 2010)

Identification of Architectural Imperfections

The Delphi study conducted for this contribution indicated as well that currently a lower degree of abstraction is preferred. The *requirements-driven* dimension leads towards scalability and big-data. The software technology dimension comprehends the evolution from mainframe application over web applications to mobile apps. Web application architectures use four main layers (Bengel 2004; Jablonski 2004): the (1)client, (2) web, (3) business and (4) data layer. The client layer displays the processing results of the application system. In addition to HTML clients, this layer may also include rich client applications and applets. The web layer generates the logic for a web-based presentation, receiving the requests of different clients and generating the corresponding responses. In the business layer, objects of the business model and the implementation of business logic are located. A data access object (DAO) encapsulates database queries and query results are returned in the form of objects. In the data layer relational data is accessed and modified through the used database systems, ES and legacy systems. These layers are useful for especially complex enterprise applications, which are developed by a distributed team.

The multi-layered approach provides benefits, as greater efficiency as a result of better portability more flexibility is provided (TheOpenGroup 2003), (Vasconcelos, Sousa and Tribolet 2007). However, software systems still not efficiently answer the continuously changing demand of business needs (Andresen 2006), (Vasconcelos et al. 2007). The successful management of these architectural issues is clearly seen as foundation for successful investment in resources and its efficient usage (Pessi, Magoulas and Hugoson 2011). Therefore the need for the manipulation of the concept arises as enterprise software systems and business needs are still gaped.

A major drawback of these architectures is their ability to be multi-layered and abstracted. This initial benefit becomes a drawback due to its ability to eventually end in themselves and its continuous facilitation to use and/or change any part of the system easily. However, such appearances are deceptive, because exchanging (a framework-specific) component with another meant to understand the entire abstraction overloaded architecture and to write new "Glue Code". Cross-section requirements such as performance and big data capability were not grantable anymore. According to our Delphi study, many experts complained of this disadvantage. In addition, software developers have been forced to familiarize themselves with numerous frameworks, design workarounds to fix errors and make allowances for the strong version-dependent compatibility.

REQUIREMENTS FOR FUTURE-PROOF ARCHITECTURES

On one hand the survey confirmed the trend towards increased business party interaction and ad-hoc third-parties connectivity as an essential element of the architecture. Interaction can take place on any level of the infrastructure, platform, services, software, partners or communities (Stephane, Nabelsi, Passerini and Cakici 2011).

However the Delphi study also revealed the following trends which were taken into account when defining the requirements:

1. Rise of Server-Side JavaScript (SSJS): For many years it was established that a programming language was used for interactivity in the browser (JavaScript) and other languages were implemented for server-side logic and request processing (PHP, Java, Ruby ...). The JavaScript language is increasingly on the rise for use on the server side (also in the mobile area). The trend is towards using only one language – JavaScript. Code can be reused between browser and server implementations and APIs no longer need to be mapped between different languages. JSON can be used as a language-independent serialization standard. In recent years there has been considerable investment made in making JavaScript interpreters as fast as possible to meet the rising complexity of web applications. The result of which, for example, has given rise to the exceptionally fast V8 JavaScript engine used by Google's Chrome browser.

2. Middleware use event-driven, non-blocking I/O models that make it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. Such systems can handle more than 6000 requests / sec per CPU core and thousands of concurrent connections which are moderately active.

3. Websockets supplement Ajax and enables Streaming: The WebSocket specification, developed as part of the HTML5 initiative, introduced the WebSocket JavaScript interface, which defines a full-duplex single socket connection over which messages can be sent between client and server. The WebSocket standard simplifies much of the complexity around bi-directional web communication and connection management.

As a result of the Delphi study, requirements towards an Enterprise System were identified and separated into functional components (See Table 2); namely, logistics, manufacturing and finance. Other industry independent functions such as archiving and reporting can be found within various layers. The functional components extend vertically through all the architectural layers. The architectural layers are distributed horizontally with the client layer at the top followed by other layers. The data layer is the final layer on the list. Like all information systems that implement layered architecture, Enterprise Systems also encounter challenges. Horizontally, a clean architectural separation can be found; however, vertically they usually do not provide a clean separation in functional areas nor do they contain a simple way to extend the system by new functional components (for example by plug-ins - that would be the supreme discipline). Additional requirements are grouped by interfaces and cross-sectional requirements.

Logistics	Inventory Control	Asset Accounting	Material Management
Finance	Sales	Invoicing	Purchasing
Manufacturing	Production Planning	Warehouse Management	Reporting

Table 2: Core functional requirement areas (derived from the Delphi study)

Through the integration layer, the Enterprise System communicates with external systems or devices, such as barcode scanners. Modern Enterprise Systems comprehend a middleware which ideally is service oriented. Table 3 displays interface requirements.

To devices	Barcode-Scanner, mobile devices
To third-party applications	Enterprise Systems of partners
To the government	Fiscal authorities, health insurances, building authorities

Table 3: Interface-related requirement areas (derived from the Delphi study)

The layer-specific and cross-functional requirements, which were derived through the Delphi study, are depicted in Figure 4. Layer-specific requirements reflect necessities that refer strongly to multi-layered software architecture as shown in Figure 3. In contrast to layer-specific requirements, the cross-sectional requirements pervade all layers.

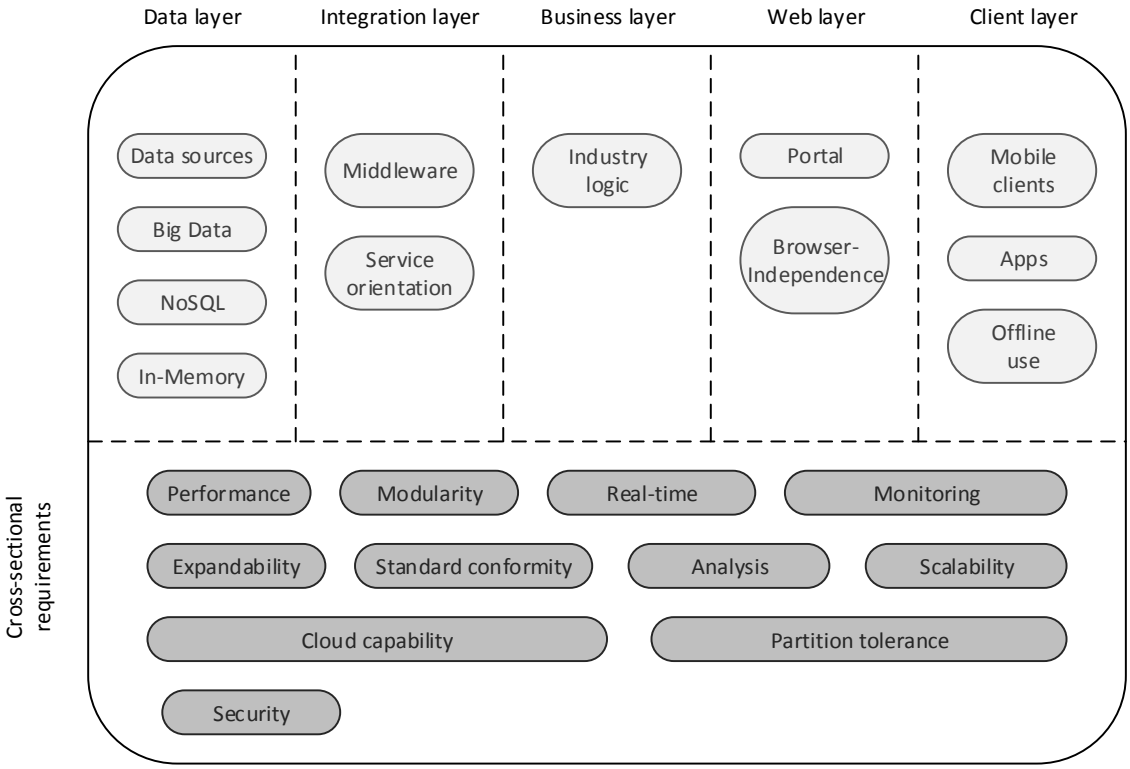

Figure 4: Architectural requirements of ES (derived from the Delphi study)

As mentioned earlier, requirements can be based on functionality or they can be interface-related. Furthermore, cross-sectional requirements exist for each layer, which can be subdivided as follows:

- The performance must be optimal in every layer. This is an architectural design task, considering the interdependencies between the layers. For example, it would not be useful to optimize the data layer via In-Memory Data Management if the Web or Client layer has not been developed in a way that it can process such a huge amount of data.
- The Modularity of the Enterprise System ensures a great maintainability, configurability and expandability.
- Expandability of the Enterprise System is not only derived due to its modular structure but also because external system integrators are able to add functionality. These external integrators could be from the customer's IT-department or Value Added Reseller. The architecture should allow the use of a tool (ES-developing tool).
- Real-time functionality allows immediate access to information. Information is automatically updated by the system. Every layer should fulfil this requirement and should also allow streaming real-time data as well.
- The architecture must be capable of monitoring, in order to detect errors and optimize the whole Enterprise System. Instrumentalization of classes and logging via AOP are two possible solutions.
- Standard conformity is especially relevant for the client and web layer. The use of HTML 5-standards allows mobile devices with varying operating systems (e.g. iOS, Android, etc.) to access the system. The HTML 5 standard even allows access to graphical resources through the WebSocket-protocol streaming of real-time data.
- Analytical capabilities are provided when the architecture has a Business Intelligence (BI) layer that meets the analysis tasks. This BI layer is stretched over all architectural layers - from client layer (where analytics are displayed in the form of statistics) to data layer (where the analysis is being performed). The architecture within the Integration-layer (and data layer) provides the ability to connect statistical Software (e.g. R). Further on, in the client layer, specific components shall be present which enable cognitive access to the data.
- Scalability of the architecture is given if its performance remains the same with an increasing amount of data within the data layer and an increasing amount of accesses through the client layer. The middleware is also considered since it must be able to handle a huge amount of data to many clients (eventually even stream it). Highly scalable middleware sometimes means that the OS-Kernel (Linux) is modified in a way that the whole OS is loaded in the memory and allows various thousands of parallel connections.
- Physical distribution allows a distributed architecture like those that can be found in cloud systems. Therefore, middle-ware is of great use.
- Partition tolerance means that distributed ES continues to work normally even if messages are lost. It can be split on geographically different sites and still works even if the connection is interrupted.
- Security within an architecture refers to the communication of the components within the architecture of the Enterprise System as well as the secure data exchange of the ES with third-party systems (also over the internet).

While cross-layer requirements (e.g. performance) affect all layers, layer-specific requirements target only one specific layer (e.g. "big-data-ability", which is related to the data layer).

PROPOSED ARCHITECTURE

In order to propose a new architecture, current architectures have been validated against the fulfillment of the proposed requirements. Since traditional architectural approaches do not fulfill the requirements, a new architecture has been developed and tested as a prototype in a German software SME. The Delphi study revealed five layers which are identified as essential for an Enterprise System:

1. Client layer: In this layer, applications are located through which users access the ES. These applications use the inputted data to return the desired information.
2. Web layer: This layer contains the logic for the web based presentation.
3. Business layer: In this layer, the business logic and business processes can be found. The layer could be further separated in industry-specific sub-layers.
4. Integration layer: This layer contains all interfaces required to exchange data with other systems (e.g. an MES or CRM-System).
5. Data layer: Through this layer the modified access to structured data occurs through the database systems.

Based on the aforementioned essential layers and requirements from the previous section, the authors propose a new architecture of an Enterprise System, which can be found in Figure 5.

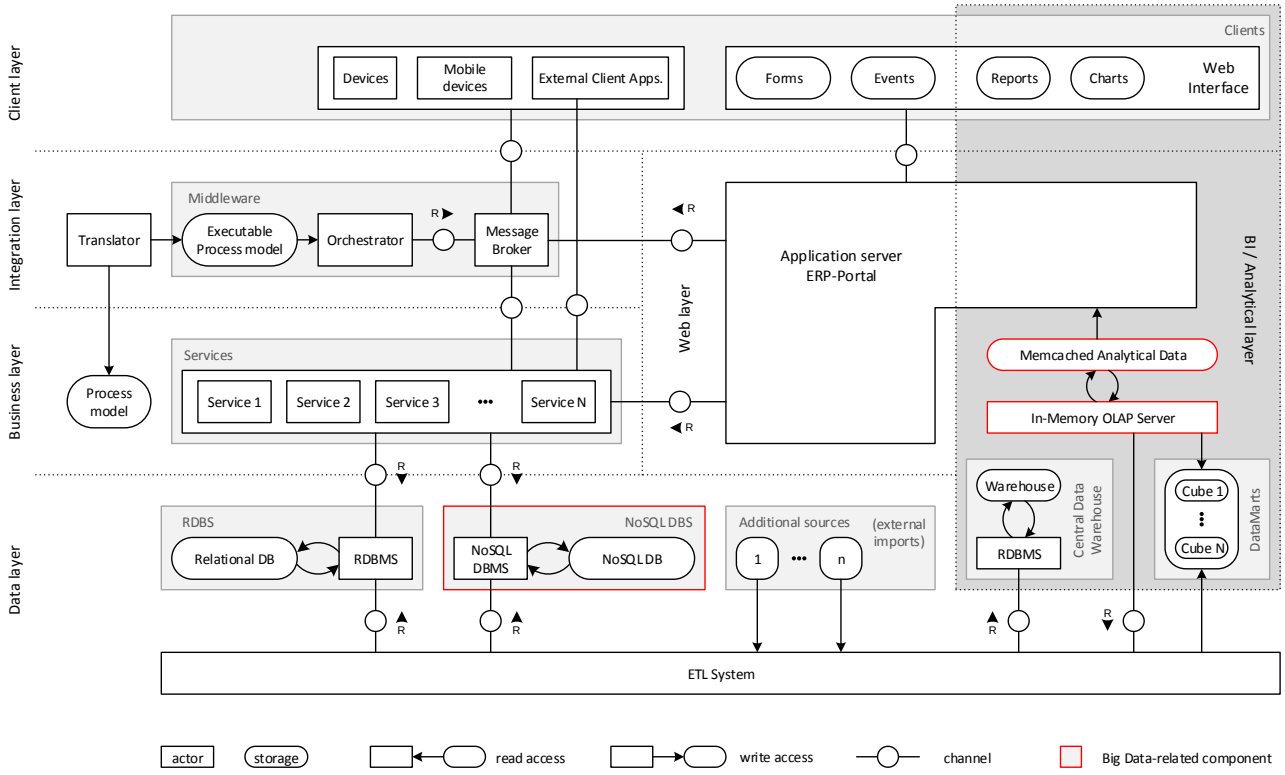

Figure 5: Architecture of an Enterprise System

The following layers can be found in the proposed architecture:

- **Data layer (horizontal):** The data sources located in this layer can be of varying types. In order to exchange data between sources, ETL processes are applied. A Big Data-capable architecture needs a physical distribution and partition tolerance. Certain data which is often found in large amounts are not saved in conventional databases. They are saved in a distributed file system (e.g. HDFS). A NoSQL-DB allows inquiries to the data located in the distributed file system. It can absorb and save a huge amount of events, for example, devices within manufacturing generation. The respective service is using a queuing and prioritization system to provide offline capability. In-Memory-Architectures maintain huge amounts of data in the memory and allow it to be persistently available. When inquiries are done via SQL, results are delivered quicker than with conventional databases. The ETL system transforms data to the warehouse. Data marts are loaded in an OLAP-server within the main memory. It is important that data warehousing and in-memory databases are located in two different machines. Data warehousing and in-memory computing utilizes a great deal of hardware resources. Running a data warehouse and an in-memory database on the same machine is not recommended since both simultaneously consume a vast amount of resources.
- **Business layer (horizontal):** The business layer contains the business logic, varying between different industries. Ideally, industry specific processes are only present within the business layer and can be extended if needed.
- **Integration layer:** The integration of middleware allows continuous data processing even if some components fail. The middleware is an independent system component providing the data. One of the greatest advantages is the integration of mobile devices. The middleware abstracts the communication between components using the Publisher Subscriber pattern. Hence the message-exchanging components do not need to "know" each other. The Subscriber-Publisher pattern works as follows: The services cannot communicate directly (and they are technically independent of each other), thereby they communicate through a message broker (intermediary). A service can publish events - for example "received new order" or the event "new goods arrived". Services can subscribe to events (e.g. events of order) in order to perform appropriate actions or trigger consequential events ("Process Order" in this example).

- The **message broker** notifies those services at each occasion that the corresponding event subscribed. Service-orientation is a typical middleware architecture characteristics with advantages such as "communication after contract", autonomy and standardized orchestration of services. The "contract" is given by a WSDL definition of operations, endpoints and XML schema of the messages. The WSDL plays an important role in the overall Web services architecture since it describes the complete contract for application communication.
- **Client layer (horizontal):** The architecture of the client should support mobile clients if data shall be provided as time and location independent. Apps (Mobile applications) should be developed client side following the Mobil-Crossplattform-principle. This will result in apps that are web based and run in a local web container of the mobile system. Offline access is guaranteed if the client sided architecture contains caching mechanisms.
- **Web layer (horizontal):** The portal architecture must provide functionality through the browser, for example, searches, personalization or categorization. Browser independence is a property that ensures that all functions and features are available throughout the different browsers. This is especially important when mobile devices are thought to be integrated.
- **BI / Analytics layer (vertical):** This layer assists in turning data into wisdom in order to support decision making for management. Data is processed to support the analysis required to achieve knowledge and insight (i.e. statistical analysis, forecasting and segmentation). For example, an analytics package could take historical sales transactions and forecast the expected revenue over the next six months. Also, intelligent pattern recognition or clustering algorithms are used in this analytics layer.

At this point, some components within architectural layers should be explained:

- **Clients, Middleware und Services (vertical):** External applications access the services (REST-Services) for states and data. Devices and mobile devices send events and data (e.g. during manufacturing or other processes) to the message broker. Vast amounts of data can be streamed via WebSockets. Vice-versa mobile applications and the portal can receive events from the MessageBroker and acquire (Event-Subscription pattern) or request stateless data through the services. To increase performance, services can be circumvented and data accessed directly from the database. The services which perform business logic interactions reflect the business processes. Therefore, they form the business layer.
- **Web layer (vertical):** The application server can be found here, receiving events from the MessageBroker as well as corporate data from the services and analytical data from the in-memory server.

CONCLUSION AND FURTHER RESEARCH

The findings derived from the Delphi study lead towards a novel definition of the ES architecture in order to meet the requirements of a hyperconnected world. Specifically, we present a new architecture of Enterprise Systems that incorporates the anywhere accessibility of data, it's anytime option supported by the integration layer and data layer, and any type of data as seen in the client, web, and BI/analytical layers. One of the constituting elements of an ES was the existence of a single database. In the advent of an increasing amount of data and requirements, respective of the business analysis, various sources of data including real-time streams needs to be considered. New technologies such as NoSQL databases or in-memory databases and middleware, which are streaming-capable, could be used to fulfil these requirements.

The novelty of the architectural proposal (Figure 5) lies within the data, integration and BI/Analytical layer. The data layer now comprehends data and its transformation from/into internal and external sources. Data can represent real-time information and long term business information through the integration of middleware. Trends like Big Data and the integration of external systems are now included in the BI/Analytical layer.

Future research shall be focused on how ES providers can create and capture value using the proposed architecture, hence the impact on the business model. Further on, special fields of application (e.g., ES for e-commerce) could be a fructiferous basis for further research.

APPENDIX

The Delphi study consists of interviews that were conducted with CIOs. The interviews addressed three main areas of investigation: the current ES architectures, their limitations (pain points) and future improvements.

Block	**Topic to explore**	**Debate into which CIOs were dragged**
1	The current ES architecture	How would you describe the architecture of your Enterprise System? o Monolithic o Multi-Layered o Strictly oriented to the functions o Web-based o High focus on extensibility (Module-based) o Big Data-oriented o Alternatives: …
2	Architectural Imperfections	What are the current architectural difficulties? Do you see a trend towards a new architecture?
3	Future requirements for architectures	Which requirements a new ES architecture must comply in the future?

REFERENCES

1. Andresen, K. *Design and Use Patterns of Adaptability in Enterprise Systems* Gito Verlag, Berlin, 2006, p. 147.

2. Armour, F.J., Kaisler, S.H., and Liu, S.Y. "A Big-Picture Look at Enterprise Architectures," *IT Professional* (1:1) 1999, pp 35 - 42.

3. Bass, L., Clements, P., and Kazman, R. *Software architecture in practice* Addison-Wesley, Boston, 2003.

4. Bengel, G. *Grundkurs verteilte Systeme : Grundlagen und Praxis des Client-Server-Computing - inklusive aktueller Technologien wie Web-Services u. a. ; für Studenten und Praktiker* Vieweg, Wiesbaden, 2004.

5. Bradford, M. *Modern ERP : select, implement & use today's advanced business systems* North Carolina State University - College of Management, Raleigh, 2010.

6. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. *Pattern-oriented software architecture : a system of patterns* Wiley, Chichester, 1999.

7. Gerald, B.L., and Cluskey, M. "Research in Foodservice Management," in: *Research : successful approaches*, E.R. Monsen and L. Van Horn (eds.), American Dietetic Association, Chicago, 2008.

8. Ghani, I., Rimal, B.P., and Jeong, S.R. "Tiny-Notational Approach for Software Architecture Visualization," *International Journal of Computer Applications* (43:4) 2012, pp 38 - 42.

9. Giachetti, R.E. *Design of enterprise systems : theory, architecture, and methods* CRC Press, Boca Raton, 2010.

10. Gorton, I. *Essential software architecture* Springer, Berlin; Heidelberg; New York, 2006.

11. Gronau, N. *Nachhaltige Architekturen industrieller Informationssysteme bei organisatorischem Wandel* Habilitationsschrift im Jahr 2000, Oldenburg, 2000, p. 320.

12. Gronau, N. *Enterprise resource planning Architektur, Funktionen und Management von ERP-Systemen* Oldenbourg, München, 2010.

13. Gull, C. *Web-Applikationen entwickeln mit NoSQL* Franzis, Poing, 2011.

14. Hopkins, B., Leganza, G., Cullen, A., and Cahill, M. "The Top 10 Business Technology Trends To Watch: 2012 To 2014," Forrester Research.

15. Jablonski, S. *Guide to Web application and platform architectures* Springer, Berlin [u.a.], 2004.

16. Jamshidi, P., Ghafari, M., Ahmad, A., and Pahl, C. "A Framework for Classifying and Comparing Architecture-Centric Software Evolution Research," in: *17th European Conference on Software Maintenance and Reengineering*, Genova, Italy, 2013.

17. Jost, M., Nilakanta, R., and Willis, J. *Qualitative research methods for education and instructional technology* Information Age, Greenwich, Conn., 2002.

18. Kaisler, S.H., Armour, F., and Valivullah, M. "Enterprise architecting: Critical problems," 38th Annual Hawaii International Conference on System Sciences, IEEE, 2005, pp. 224b-224b.

19. Kogent Learning Solutions, I. *SAP ABAP questions and answers* Jones and Bartlett Publishers, Sudbury, Mass., 2010.

20. Mak, G. *Spring recipes : a problem-solution approach* Apress, Berkeley, Calif., 2008.

21. Pessi, K., Magoulas, T., and Hugoson, M.-Å. "Enterprise Architecture Principles and their impact on the Management of IT Investments," *The Electronic Journal Information Systems Evaluation* (14:1) 2011, pp 53-62.

22. Rico, D.F., Sayani, H.H., and Field, R.F. "History of Computers, Electronic Commerce and Agile Methods," in: *Advances in Computers,* V.Z. Marvin (ed.), Elsevier, 2008, pp. 1-55.

23. Scott, A. "Object-relational mapping," *Softw. Dev.* (4:10) 1996, pp 47-50.

24. Senthilvel, G. "DotNetWorkflow Concepts," 2008.

25. Stephane, G., Nabelsi, V., Passerini, K., and Cakici, K. "The Next Web Apps Architecture: Challenges for SaaS Vendors," *IT-Professional* (13:5) 2011, pp 44 - 50.

26. Sultanow, E. *Zusammenarbeit in verteilten Projekten: Dekomposition, Barrieren und Lösungen im Kontext der Webentwicklung* Gito, Berlin, 2010.

27. Sultanow, E., Brockmann, C., and Gronau, N. "Enterprise Systems Ecosystem: A case study based comparison of software companies," in: *16th American Conference on Information Systems (AMCIS)*, Lima, Peru, 2010.

28. TheOpenGroup "The Open Group Architectural Framework (TOGAF)."

29. Vasconcelos, A., Sousa, P., and Tribolet, J. "Information System Architecture Metrics: An Enterprise Engineering Evaluation Approach," *The Electronic Journal Information Systems Evaluation* (10:1), // 2007, pp 91-122.