Experiences from building a EUD business portal

Nikola Tanković*, Tihana Galinac Grbac**, and Mario Žagar***
* Superius d.o.o., Pula, Croatia
** Faculty of Engineering, University of Rijeka, Croatia
*** Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
nikola.tankovic@superius.hr, tgalinac@riteh.hr, mario.zagar@fer.hr

Abstract - End user development (EUD) is the idea of providing end users, professionals outside computer science community, means to develop their own software. This idea has received special attention by domain experts who would like to easily modify software applications to their needs, without intervention of technical professionals and without learning conventional programming languages. The end user development is in fact a system property, closely related to system adaptability that should be built into a system to enable number of modifications at different system levels securing powerful expressions that would satisfy the end user needs. This is a challenging requirement for every software provider. There are a number of identified system perspectives, critical to EUD, and general guiding principles that should be addressed. However, in reality, building these properties into the concrete system, a number of complex and interrelated issues have to be solved. In this paper we report challenges that we addressed while implementing end user development capability into a software system within business domain.

I. INTRODUCTION

End User Development is focused around concept of providing means and tools for non-professional developers to participate in software development. Lieberman at el. give the following definition [1]: "End-User Development can be defined as a set of methods, techniques, and tools that allow user of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact."

Spreadsheet is the most popular and well-known EUD tool, but there is an increase in the number of solutions such as scripting programming languages or graphical mashup tools that combine existing functionality into composite applications. The main reason for the popularity of EUD is the cost and time expense in traditional on-demand software development that many organizations face. The downside is the lack of quality in terms of reliability and security, especially in mission critical computing.

Our motivation for building a EUD business portal is to enable our customers with a tool for creating applications that implement simple business processes yet uncovered by their existing IS solutions. Such implementations are commonly non-critical with lower return of investment. This decision was made after a decade of software development experience in business domain designing and delivering various customer requests.

Building an end-user development system from technical perspective means combining and applying numerous existing software engineering concepts that are offered to improve the software lifecycle process. Usually the ideas behind these concepts are clear however their implementation is full of traps and obstacles.

Here we will elaborate on one particular case study how some of the most famous software engineering concepts are implemented and we elaborate the problems that have to be solved. The following concepts are addressed: automated customization of products to customers, automated production process, automatized delivery process and web oriented end user development process.

II. CONTEXT OF THE STUDY

We illustrate the concepts of this paper based on an example of software development in the supply chain management (SCM) domain. More precisely, the focus of this paper will be the software for supporting the continuous control of trading contracts. Trading companies are businesses usually selling different kinds of products to consumers. The products are ordered from different producers, offered in a retail stores and resold to buyers. One typical example is food shop. The relation between trading company and producer is defined by trading agreement. This contract includes metrics such as minimal number of products to be exposed on shelves, product positioning inside shop or minimum stock per product to be ensured. Contracts are subjects of control and future management that is performed by field agents dispatched by producers themselves. They ensure that for each of their products being sold, metrics agreed in contracts are being fulfilled across retail stores. In sequel, the software for supporting agents at managing trading contracts will be referred as Manage Trade software.

The company that is developing the Manage Trade software has long tradition and numerous customers. Our customers are before mentioned producers and distributors of goods. The core software system has been evolving for more than 10 years. A numerous improvements have been introduced in the software system. Here we will mention just few that have implications for our future elaborations.

1. When contract is signed between a producer and trading company, a number of agents are engaged to control and manage the contract execution. For that purpose, a number of software applications - one per agent - are developed for the purpose of one specific contract. Each software application has implemented one smaller derivation from the contract. For that purpose we introduced a feature of automated customization in our Manage Trade software system. That feature enables to automatically derive a number of customized software applications for every different contract.

2. Since we have many customers we have identified similarities in the agreements and contract management processes. A logical step was to automate software production with maximal reuse of existing software. Therefore, we developed our Domain Specific Language (DSL), a script language that describes the differences introduced with each contract by inheriting from common predefined constructs. That way all the specifics in applications for different customers are expressed easily.

3. We developed a Web based system coordinating and supplying data to applications from all agents by using a single server database instance for multiple contracts and producers.

4. Further step in improving our business was to automate the software delivery process. In that aim we implemented a Web based system that enabled Web based delivery of software applications to the agents of specific contract.

5. Our next step in improving our business is empowering our customers to develop their contract support system and thus reuse the concepts of the End user development (EUD) paradigm. A specific feature would be introduced in our Web based system to enable end user development. This feature we will refer as EUD portal.

III. EUD PORTAL – CONCEPT AND DEFINITIONS

EUD portal is a Web based system for definition, automatic construction and delivery of software applications in a certain domain of business. We observe and describe EUD portal use cases from different perspectives:

1) From *application development perspective* it is a web platform for modeling software applications using a provided DSL language. Resulting software applications

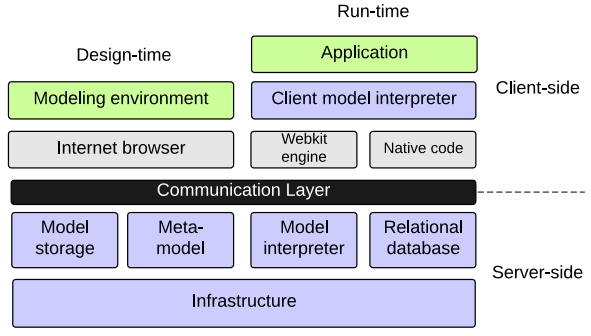

Figure 1. EUD portal by layers

are limited to elements and expressive level of DSL provided. In our case, we provide adequate DSL for Manage Trade domain where producers create applications to support their agents at work. The targeted end-user for this use case is a business expert who usually has some experience and expertise in simple programming such as creating Excel documents or scripting simple programs. Description of end-user programming interface used is out of the scope for this study and is still in development stage. We address only technical underlying challenges of building a portal with these properties.

2) From *runtime perspective*, the execution of Manage Trade applications, EUD portal leverages Software as a Service (SaaS) delivery model [2]. Each agent uses an application delivered and managed by EUD portal.

A. Application development perspective

Contracts between producers and retail shops vary in content. Agreements in these contracts evolve over time meaning that agent applications change. In addition, some producers also required additional functionality such as sales order creation or collecting additional data for their business intelligence systems. Given all that, our portal needs a lot of expressive power. To satisfy such a large variety of requirements, we pursued a model-driven approach, and defined a common AGM Meta-model typical for our tenants requirements presented at [3]. AGM meta-model possesses expressive power to model applications that can process a variety of structured data, business processes and validation rules to minimize data collection errors. AGM was defined from years of experience in SME business domains such as Manage Trade. The guidelines for defining a solid meta-model are provided at [4].

Figure 1 illustrates that design-time, or sometimes referred to as modeling-time, together with run-time environments reside on the same web infrastructure. Model storage is a specialized graph database Neo4J[1] containing end-user defined models. It is used by EUD portal as a central model repository of every modeled application, and for storing meta-model elements and definitions. Models are constructed using before mentioned DSL language for Manage Trade domain inside modeling environment provided by portal. This web-based environment is at a very basic and early stage, providing only a simple DSL editor to input or modify application definition. This area is a subject to future improvement with introduction of graphical modeling environment for a more user friendlier approach [5].

[1] Neo4J – open-source graph database, available at http://www.neo4j.org

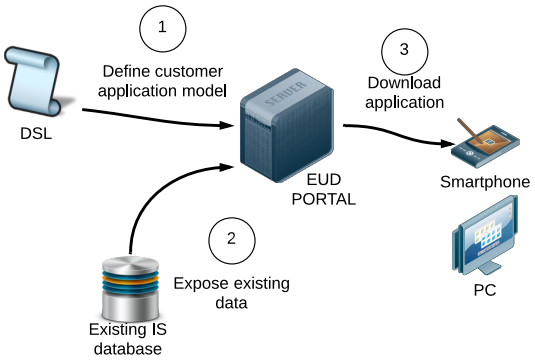

Figure 2. Basic steps involved in creating application

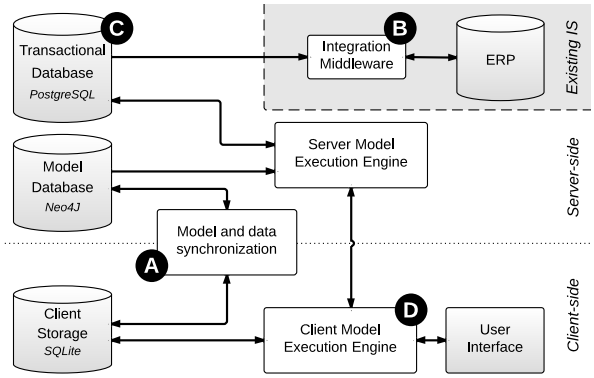

Figure 3. EUD portal architecture

Overview of complete application creation scenario is shown in Figure 2. Steps are simplified omitting ones like tenant registration or management of users. It gives the high-level picture of three-step application development:

1) In first step business expert in the role of end-user programmer models the application using the interface provided from EUD portal. This modeling step includes definition of data structure, data transactions that will be carried by agents, and a model of user interface. Transactions are modeled as a set of data collection forms. In Manage Trade domain agents use these forms to collect information that is analyzed to see if contracts are being met. Therefore, these forms are modeled according to contract contents.

2) Each organization exposes data from their existing IS through an ODBC [2] database connector or uses connectors to SAP ERP[3] or Microsoft Dynamics NAV[4] systems. The tools are provided by portal. This step is often completed with some help of professional developer or system integrator who has a deeper knowledge of SQL[5] language and database procedural languages. The structure of exported data in this step must conform to data structure defined in previous step.

3) Final application is automatically assembled and downloaded to each of agent devices. Manual download is required, but future updates are automatic and seamless.

EUD portal does not assemble each application as different executable specific to each agent, but rather provides a common executable for every agent, which is capable of interpreting provided model at runtime. We refer to this executable as a *client engine*. Model is synchronized to each client engine upon first usage and updated as necessary. Model interpretation in favor of generating end applications makes this an interpretative MDD approach. Differences between MDD approaches and benefits of each, together with rationale for choosing interpretative approach are described in [6].

B. *Runtime perspective*

1) From runtime perspective, EUD portal is a distributed software system composed of server and client components. Server-side components provide data storage services for assembled applications. Each agent continually collects information at retail stores that is stored in portal. This is achieved using a relational database. Single database instance is shared between agents and data is isolated accross companies.

To support data storage for multiple agents and organizations in a single database, we used a multi-tenancy design of the portal and underlying relational database. This means that each customer shares hardware and software resources leading to better resource utilization. A *shared-table design* [7], [8] is used to isolate data inside relational database instance. In our implementation, a single tenant correlates to single customer – producer or distribution organization.

EUD portal also possesses Web services and tools that provide integration with existing IS. In our business case, agents require immediate access to different information contained within existing Customer Relationship Management (CRM) and Material Management (MM) systems. Such information includes details about retail shops and products. Each agent also receives daily work schedules provided from these existing systems. They define a list of retail stores to visit and ensure contracts are being followed correctly. This work schedules are synchronized from outer systems to EUD portals central database and then further passed to each agent application.

Client side components are packaged as a hybrid mobile and desktop application [9]. Hybrid application composing means that applications are built using web technologies: JavaScript and HTML, combined with native modules developed separately for each platform we provide: PC, Android and iPhone. These native modules

[2] ODBC – stardadized middleware for accessing database management systems, originally developed by Microsoft

[3] SAP – SAP AG's Enterprise Resource Planning software, available at http://www.sap.com

[4] Dynamics NAV – ERP software product from Microsoft, available at http://www.microsoft.com/en-us/dynamics

[5] SQL – special domain specific language for managing data contoained in database management systems

are referred as *native wrappers*. They use Webkit[6] open-source engine to display HTML and JavaScript contents. That way we obtain functionality specific to native applications and still benefit from web technologies that facilitate software delivery and multiplatform execution [10]. Native functionality includes: access to file-system, photo camera, gyroscope, GPS module, device information and phone address book, depending on platform.

IV. CHALLENGES AND FUTURE DIRECTIONS

Many technical challenges arose during implementation phase. Those will be described in following subsections, each section covering different technical aspect of EUD portal. To position each problem according to EUD portal layer or component, we relate to Figure 1, which provides an overview on portal run-time and design-time layers, and Figure 3, which shows the main components used.

A. Communication between server and clients

As shown in Figure 3, labeled (A), data synchronization between portal and end applications is achieved with a distributed synchronization module residing both on server and client side. This empowers each client application to conduct computation over data independently, which unburdens server infrastructure resources. This is also very useful for supporting *offline* data processing, meaning that agent applications can process transactions on smartphones without data network, and postpone synchronization of completed transaction back to server after connection is reestablished. Modeling language provides expressions to filter data that is to be synchronized on each agent application.

It was quite difficult to achieve a stable synchronization mechanism given that resulting applications were designed by end-users so the synchronized data volume was often unpredictably large. This problem is even more present when data is sent over old generation mobile networks.

To resolve this issue, we are researching a mechanism that will predict the volume of data at run-time and warn the developers when such volume hits a certain threshold. That way, end-users can decide to lower the volume of data by filtering some less important data.

B. Integration

A key part of any business system is interoperability - efficient cooperation with existing systems. This is in our experience one of the greatest barriers for using EUD in business organizations. Many existing, on-premises IT systems are inaccessible or difficult to integrate with. A most common method of integration in our experience is by using intermediate tools to connect to their database systems. In Figure 3 this tools are noted (B). It can be observed that relational transactional database is connected with *integration middleware* for connecting to existing IS databases. These tools are configurable applications provided to be installed inside existing IS environments. The reason for on-premises installation is to evade strict firewall policies many of our customer posses and their inability to provide web services. Although SOA architectures are getting more common, our customers do not posses such solutions, so data is synchronized with their databases through ODBC drivers and a configurable set of SQL commands.

We already stated that for a stable integration, assistance from professional developer is required and this is quite a drawback for end-user development due to unpredictable costs and time factors when engaging professional help. After the transition to service-oriented architecture this problem will be easier to tackle, as there is many research on integrating web services with EUD principles. Dörner et al. [11] are researching end-user development integration with Service-Oriented Architectures (SOA) by making web services more understandable to end-users. Similar pattern can be applied here. However, our customers are from small and medium business (SMB) sector, and are still just considering migrating to SOA. Direct integration with their databases is still the most flexible and common solution.

C. Persistence Layer

Data persistence, marked (C) in Figure 3 is the most important part of any business transaction processing system and this was where we faced many challenges. When we did our research for choosing the right database to store EUD portal customers data, we had several main requirements:

- ACID properties,
- An open-source solution is preferred,
- Schema-less database is preferred; as it makes storing end-user designed data objects easier to implement.

We have chosen a relational open-source PostgreSQL[7] database. PostgreSQL, like any ordinary relational database, requires schema to be defined prior to usage. We gave up on this aspect in favor of ACID properties and reliability, as schema-less NoSQL[8] databases are still mainly not mature enough for business production usage, like storing important and sensitive data. Main deciding factors for PostgreSQL were durability, robustness and open-source code.

To achieve best hardware resources utilization, we implemented a row level multi-tenancy meaning that

[6] Webkit – open source layout component designed to render web pages, available at http://www.webkit.org/

[7] PostgreSQL – open-source object-relational database management system released under PostgreSQL license, available at http://www.postgresql.org

[8] NoSQL – common names for databases that store data in forms other than tabular relation used in relational databases, many of which do not require a schema to be defined

tenants share same tables with each row having tenant identification.

Since tenants had different schema structures, we had to share tables and columns across tenants. For that reason each column had the same universal variable length string type. This table and column abstraction on database side brought a small performance drawback because the benefits of relational schema were not fully utilized. To satisfy data isolation between tenants, we implemented a layer between the portal application code and database that does the mapping between user-defined schema and real database schema. This layer also brought a small degradation of data access performance. Similar approach has been implemented by the Force.com Salesforce1 solution [12] on top of Oracle relational database. We have not applied enough attention to properly address this issue and further research is required in this area. More research towards implementation of this feature natively inside databases is required, such as [13].

Another big issue was database scaling, ability operate effectively regarding to database size. We did not automatically track each tenant metrics on table sizes nor implement a mechanism for automatic index creation to rectify slow queries. Instead, we performed database optimization and administration manually. Taught by this production experience, we defined a set of characteristics that a fully automated EUD business system persistence layer should have:

1) Mechanism for keeping track of per-tenant meta-data and metrics: slow queries, overall data size, data fragmentation, and index efficiency. Basic solution is provided at Figure 4 where execution time is monitored together with requested query. Collected data is analyzed for decision such as automatic creation of additional indexes, data sharding and partitioning. Similar approach is applied at [14] on cost-driven basis.

2) Data sharding should take care not to divide data inside common tenant and to divide shards evenly. Such mechanism should track fastest growing tenants and reserve enough storage for further growth. If a single tenant gets larger than a single shard (database) it should be further split by relocating less accessed data.

Modern Platform-as-a-Service (PaaS) solutions offer APIs for managing hardware infrastructure. This could also be used for automatic new creation of additional database instances when total capacity is close to being exceeded with automatic data reallocation. An example of such solution is Amazon Auto Scaling[9] but it is limited in terms that it allows only predefined set of rules like defining peek times, or CPU usage thresholds when a scaling should occur. Scaling API should be combined with analysis of existing tenant meta-data and metrics for optimal resource allocation. Combined with a database that easily scales should provide a solid ground point towards building a database layer capable of handling large quantities of different tenant schemes. A good

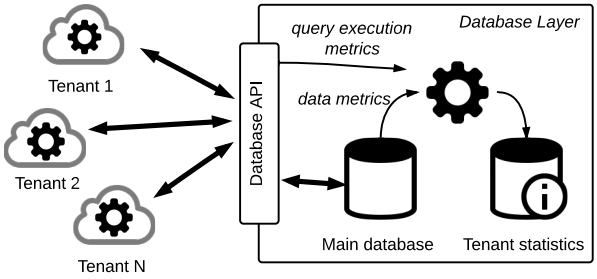

Figure 4. Mechanism for collection tenant data size and query performance metrics

database engines categorization with a special review to scaling is given in [15]. This work shows a promising future for NoSQL schema-less database solutions once they become more reliable.

D. Extending modeling capabilities

Certain transactions conducted by agents' required additional functionality that could not be expressed with our DSL language. Therefore, we provided means for writing simple JavaScript[10] functions extending provided modeling capabilities e.g. specific tax calculations, custom advanced validation controls or other types of more complex algorithms. They were executed on agent applications and can be modeled to execute on certain application events like startup of the application, beginning or end of transaction, periodically, and on different user interface actions.

Using these extension scripts in practice revealed a few additional problems:

1) *Security concerns* – client side JavaScript code is not safe from manipulation, so an additional JavaScript interpreter located at portal should revalidate those client-side calculations upon successful synchronization of data back into the portal database.

2) *Inefficiency* – certain scripts can perform much better on server side, e.g. manipulating a larger set of data. This requires more resources than client side computing can offer.

3) *Poor usability* – to write an extension, one needs some sort of assistance from professional developer who knows JavaScript. This was not suited for an end-user.

To resolve performance issues, an algorithm based on heuristics could predict functions to be executed on server side e.g. function with frequent access to large amount of data.

V. CONCLUSIONS

Implementing EUD portal in the web environment where technology is still not specialized enough for such dynamic use cases offers several technical challenges to surpass. Main concerns are infrastructure barriers, integration issues and scalability.

[9] AAS – Amazon Auto Scaling solution, available at http:// http://aws.amazon.com/autoscaling/

[10] JavaScript

Computational and network infrastructure is limited, but end-users do not possess understanding of those limitations which results in unstable and underperforming applications. Integrating EUD systems with existing solutions is another key barrier for end-users because it is often very complicated even for professional developers to understand and find a way to integrate with closed systems.

Regarding data persistence layer, greatest challenge is achieving multiple schema multi-tenant data storage on classical database management solutions. Having a user-defined data structure imposes a need for higher flexibility to change data structures in runtime. We proposed using a schema-less database or an abstraction layer over relational databases. A mechanism that would gather meta-data and usage statistics for each tenant is also needed to tackle self-optimization. Such mechanism should take decisions and actions over data partitioning, sharding and indexing.

To enable automatic scaling, platforms could be built on IaaS solutions that enable API's for dynamically creating new worker and storage nodes to take on extra traffic. Figure 5 shows an overview on key portal architecture characteristics: 1) multiplatform execution, 2) web-based WYSIWYG modeling environment, and 3) execution engine with ability to use infrastructure APIs for automatic scaling. Self-adaptivity in forms of carefully configured algorithms should take decision to scale by managing storage and computation resources from IaaS provider during runtime. This is crucial for EUD solutions where application resource footprints are highly dynamic and unpredictable.

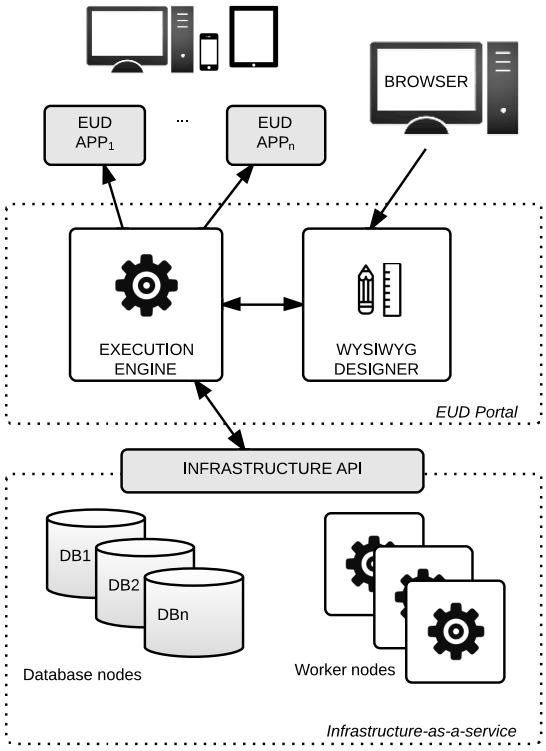

Figure 5. Proposed EUD portal architecture

VI. ACKNOWLEDGEMENTS

This work is supported in part by the Croatian Ministry of Science, Education and Sport, under the research project "Software Engineering in Ubiquitous Computing".

[1] H. Lieberman, F. Paternó, M. Klann, F. Paternò, and V. Wulf, "End-user development: An emerging paradigm," *End User Dev.*, vol. 9, pp. 1–8, 2006.

[2] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, p. 50, Apr. 2010.

[3] N. Tankovic, D. Vukotic, and M. Zagar, *Executable Graph Model for building data-centric applications*. IEEE, 2011, pp. 577–582.

[4] R. Lagerström, J. Saat, and U. Franke, *Enterprise meta modeling methods–combining a stakeholder-oriented and a causality-based approach*, vol. 29. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 381–393.

[5] V. Deufemia, C. D'Souza, and A. Ginige, "Visually modelling data intensive web applications to assist end-user development," in *Proceedings of the 6th International Symposium on Visual Information Communication and Interaction - VINCI '13*, 2013, p. 17.

[6] N. Tankovic, D. Vukotic, and M. Zagar, "Rethinking Model Driven Development: analysis and opportunities." pp. 505–510, 2012.

[7] Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang, and W. H. An, "A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing," in *2008 IEEE International Conference on e-Business Engineering*, 2008, pp. 94–101.

[8] J. Espadas, A. Molina, G. Jiménez, M. Molina, R. Ramírez, and D. Concha, "A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures," *Futur. Gener. Comput. Syst.*, vol. 29, no. 1, pp. 273–286, Jan. 2013.

[9] A. Charland, B. Leroux, and B. A. Charland, "Mobile Application Development : Web vs . Native," *Commun. ACM*, vol. 54, no. 5, p. 49, May 2011.

[10] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proceedings of the 6th Balkan Conference in Informatics on - BCI '13*, 2013, p. 213.

[11] C. Doerner, F. Yetim, V. Pipek, V. Wulf, and C. Dörner, "Supporting business process experts in tailoring business processes," *Interact. Comput.*, vol. 23, no. 3, pp. 226–238, 2011.

[12] C. D. Weissman and S. Bobrowski, "The design of the force. com multitenant internet application development platform.," 2009.

[13] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang, "Native support of multi-tenancy in RDBMS for software as a service," in *Proceedings of the 14th International Conference on Extending Database Technology - EDBT/ICDT '11*, 2011, p. 117.

[14] S. Schulte, D. Schuller, P. Hoenisch, U. Lampe, R. Steinmetz, and S. Dustdar, "Cost-driven Optimization of Cloud Resource Allocation for Elastic Processes," *Int. J. Cloud Comput.*, vol. 1, no. 2, pp. 1–14, 2013.

[15] J. Pokorny, "NoSQL databases: a step to database scalability in web environment," *Int. J. Web Inf. Syst.*, vol. 9, no. 1, pp. 69–82, 2013.