Cardoso, J. and Miller, J. A Internet-Based Self-Services: from Analysis and Design to Deployment. IEEE International Conference on Services Economics (SE 2012), IEEE Computer Society, Hawaii, USA, 2012

Internet-Based Self-Services: from Analysis and Design to Deployment

Jorge Cardoso
Dept. Engenharia Informatica/CISUC
University of Coimbra
Coimbra, Portugal
Email: jcardoso@dei.uc.pt

John A. Miller
Computer Science Department
University of Georgia
Athens, GA, USA
Email: jam@cs.uga.edu

***Abstract*—Many worldwide economies have moved away from manufacturing and became service-oriented. As a consequence, research on Internet-based Self-Services (ISS) will foster the uptake of service exports and trading since they can replace many face-to-face interactions and make service transactions more accurate, convenient and faster. Currently, ISS are poorly supported since they are created using generic software techniques and development methods. Systematic methods are needed and must go beyond service design, a well established research area, to account also for service analysis, implementation, and deployment. In this paper, we explore a systematic development approach, called SEASIDE, based on four scientific advances: enterprise architectures (EA), model-driven developments (MDD), model-view-controller pattern (MVC), and platform as a service (PaaS). By selecting the Incident Management service from the Information Technology Infrastructure Library, we have evaluated our approach. Our research indicates that the proposed method enables to focus on the business and design side of services, and reduce development time and costs.**

***Keywords*-service engineering; service design; self-services; service architecture; EA; MDD; MVC; PaaS;**

I. Introduction

The research done so far in the field of services has taken three main and distinct paths. On the one hand, and from the computer science community, services have been seen from a *software and IT perspective*. This has originated extensive research around the concepts of Web services and software-oriented architecture (SOA) [1], [2]. On the other hand, a stream originating from business and marketing has targeted the development of service management approaches and the economical analysis of services from a *sales, communications and business models perspective* [3], [4]. Models for understanding and modeling service processes, models for understanding customer satisfaction and tools for measuring service quality have been some of the results. And a stream originating from service design has taken the *design perspective* [5], [6]. It has targeted the 'deep dive' into the world of actors within service systems, aiming at the development and prototyping of scenarios for service improvements and innovations.

Our claim is that research on services should now focus on bridging the results already obtained, i.e. from software-based services, business models, and service design, to provide systematic approaches to develop Internet-based Self-Services (ISS) [7] which enable a faster time to market and involve lower costs. ISS are a subtype of services driven by *self-service technologies* [8] which provide technological interfaces allowing customers to use services independently of the involvement of direct service employee. Self-ticket purchasing and self-check-in for a flight using the Internet are examples of Internet-based self-services.

Indeed, the Internet and W3C/OASIS standards have a significant role for software-driven services but they are not sufficient to support self-services. The challenge here is how to "industrialize the manufacturing" of services for international trading. The formalization, conceptualization and integration of this service triangle (technology, business, and design) will enable, among other things, to increase the exports of services, reduce time-to-market from service analysis to service deployment, and provision services to consumers with lower operation costs [9]. ISS can replace many face-to-face service interactions to make service transactions more accurate, convenient and faster by using the Internet as a delivery medium and a Web browser as an interaction tool.

The purpose and objective of this paper is to understand how the systematic analysis, design, implementation and deployment of Internet-based self-services can benefit and be achieved from the intersection of four main research streams: a) enterprise architectures (EA), b) model-driven developments (MDD), c) model-view-controller architecture (MVC), and d) platforms as a service (PaaS). The combination of these four approaches resulted in a development methodology called SEASIDE[1].

Our approach was applied to the Incident Management Service description from the Information Technology Infrastructure Library (ITIL [3]) which was used as a use case. We followed an end-to-end systematic engineering which was finalized with its deployment using a platform as a service and explored the difficulties, advantages and challenges of the approach. We hypothesize that the combination of these four research streams can support a systematic engineering methodology which enables to reduce the complexity, reuse common patterns and structures, automatize implementation, and simplify the deployment of ISS. Enterprise architectures enable to reduce the complexity of designing services by

[1]SEASIDE = SystEmAtic ServIce DEvelopment

using abstraction and divide-and-conquer mechanisms. Model-driven development initiatives foster the automatized generation of code and reduce ISS' implementation costs. Model-view-controller patterns reduce code organization complexity. And, platform as a service cloud-based infrastructures enable to simplify the deployment of services and reduce operation costs.

The findings of our research indicate that the proposed SEASIDE methodology enabled to reduce service analysis complexity by using weak semantics and taxonomies. The use of automatic code generation techniques enabled the consistency and traceability between ISS models and service implementation. It has also enabled to support a more efficient implementation phase. The use of an adaptation of the original MVC pattern enabled a higher and consistent quality of the generated code and instructions with respect to errors, maintainability and readability. By using a PaaS for ISS operation, the time and complexity of deployment was reduced.

This paper is structured as follows. Section I, the introduction, provides the background to our work, identifies limitations of current and requirements for future research, and gives an overview of our approach. Section II describes the main related research in this area. In section III we describe our systematic methodology and depict a use case from ITIL to illustrate its applicability in practice. Sections IV, V, VI, and VII describe the contribution of EA, MDD, MVC, and PaaS, respectively, to our approach. Section VIII presents our achievements and the lessons we have learned. Finally, section IX presents our conclusions.

II. Related work

We have previously developed the ISE (Inter-enterprise Service Engineering) methodology and workbench [10], [11], one of the first attempts to devise a Service Engineering procedure for designing business services [12]. ISE has some similarities with the method presented in this paper since it was also inspired by the Zachman framework (see [12]) and relies on model-driven development concepts. Nonetheless, the differences are important and reflect the experience we have gained with the development of ISE. On the one hand, ISE was conceived to analyze and model services and did not foresee the implementation and deployment phases of the service engineering lifecycle. The exploration of these two additional phases makes the proposal described in this paper a complete, end-to-end solution. ISE adopted the first five perspectives (i.e. abstractions) of the Zachman framework. We have learned that the granularity of these perspectives was too fine to provide a useful modeling solution. As a result, in our new proposal for ISS engineering, we do not consider the conceptual and physical perspectives. This decision made the approach simpler to use and more appealing to stakeholders. On the other hand, the focus of ISE was on the horizontal/vertical integration, synchronization, and transformation of models using model transformation languages and toolkits such as MOF QVT or ATL (see [10], [12], [11]). This approach revealed to be too complex since management tools for model transformation were not mature for multi-model scenarios. In this paper, model-driven developments are used in a much simpler way to generate software code and instructions.

Several researchers have investigated how models, such as UML diagrams, could be used to automatically generate semantic Web services descriptions [13], [14], [15], [16]. In [13], the authors proposed an approach to produce MVC-based skeletons for Web applications from UML and OWL-S descriptions. In [14] and [15] the authors use UML class diagrams to represent data models and UML activity diagrams to represent business processes. The approach is used with semantic Web services specified with OWL-S by applying XSLT transformations to UML diagrams. Timm and Gannod [16] also present an automated software tool that uses model-driven developments to generate OWL-S descriptions from UML diagrams. Compared to our work, we seek to understand how various models (e.g. data models, service blueprinting, network graphs, user interfaces, USDL, etc.) can co-exist and be transformed into software components that implement an ISS. Handling several models provide a more real service development environment. We rely on model-driven developments to also generate software code. But in our work, code is organized according to a modified version of the MVC pattern structured to implement and deploy ISS services. We believe that the MVC pattern needs to be specialized to reflect domain knowledge of services.

The telecommunication industry is realizing the value of exposing WSDL/SOAP Web services to consumers using governance systems and best practices. For example, SAPO, a subsidiary of Portugal Telecom (the largest telecommunications service provider in Portugal), developed a Service Delivery Broker [17] to design and deploy Web services. The system enables the creation of Web services UI front-ends to enable consumers to easily access and buy functionalities (e.g. SMS packages and GPS points of interests). Our work differs in two fundamental aspects. On the one hand, we target the study of Internet-based self-services and do not restrict our approach to WSDL/SOAP services. On the other hand, we seek to demonstrate that it is possible to develop automated end-to-end approaches to implement and deploy services in the 'cloud'. Overlapping objectives include understanding how a PaaS can reduce the cost of service management and delivery.

III. Systematic Service Development

One of the cornerstones of our SEASIDE methodology is the adoption and adaptation of an enterprise architecture framework to provide a service architecture to organize services' developments and verify their completeness. The term "completeness" refers to the verification that a service has all the models, interfaces, linkings and flows needed so it can be analyzed, modeled, implemented and deployed.

A. Service architecture

The enterprise architecture adopted to provide a service architecture was the Zachman framework for four main reasons: (1) our previous research on using this framework for the

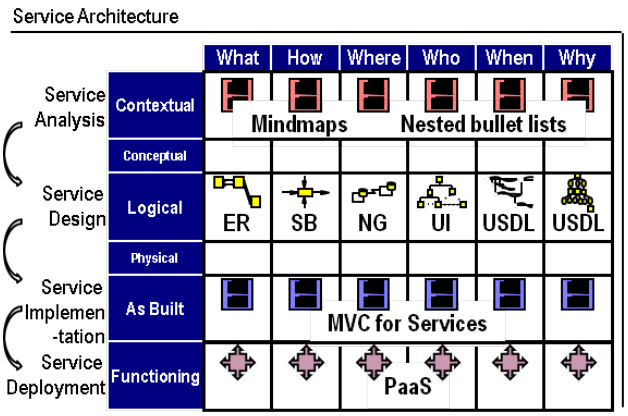

Fig. 1. Service architecture for the systematic development of ISS

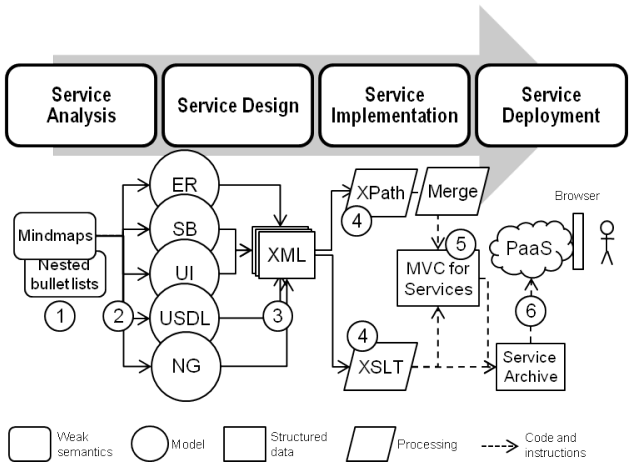

Fig. 2. The process of developing ISS using our systematic approach

engineering of services was positive (see [10]); (2) there is a clear mapping between the framework and the components of the approach proposed in this paper. For instance, MDD maps to the logical perspective of the Zachman framework and MVC maps to the 'as build' perspective; (3) it provides simple cognitive models, such as mindmaps and nested bullet lists, which help stakeholders dealing with abstraction and supports the complexity of services using a divide-and-conquer approach; and (4) it addresses the separation of concerns paradigm raised by [18] by dividing and structuring services' important aspects into six dimensions relevant for ISS: data, processes, networks, people, time, and motivation.

Based on our previous research, we have realized that the original framework structure, while adequate to model enterprise architectures, had too many abstractions which made it difficult to use. As a result, we decided to "discard" two abstractions since their original contribution did not fit exactly our services' needs. The conceptual and physical abstractions were left out in our service architecture. On the other hand, one abstraction was added when looking at our previous work on the ISE framework. The functioning abstraction was added to capture the commercial PaaS which can be used to operate ISS. The resulting service architecture is illustrated in Figure 1 and is composed of the following abstractions:

- *Service analysis* corresponds to the contextual abstraction and contains mindmaps and nested bullet lists.
- *Service design* corresponds to the logical abstraction and contains models, such as processes and user abstract interfaces, managed using MDD.
- *Service implementation* corresponds to the 'as build' abstraction and contains a specialization of the MVC pattern.
- *Service deployment* corresponds to the functioning abstraction and contains Internet-based self-services running on a PaaS.

The service architecture is a blueprint which drives the systematic development process of ISS from service analysis, service design, service implementation to service deployment.

B. Service development method

The following paragraphs describe the workflow which generates MVC-based ISS applications from a service architecture (see Figure 2). In the first and second step, stakeholders construct service models based on the weak semantics captured in the analysis phase. These steps are manual and involve an intensive discussion of the objectives of an ISS to be developed. In a third and fourth step, models are serialized to XML and code and instructions are automatically generated from models by applying XSLT transformations and customized transformation engines using XPath. E-R models generate SQL instructions, service blueprints generate controllers (which are coded using, for example, Ruby, Java or PHP) and inject control-flow instructions into views (e.g. HTML instructions), the low fidelity prototypes representing the UI of services' tasks front-ends are used to generate Web pages (see Figure 4), and USDL generates information instructions to the views (e.g. quality of service information) and controllers (e.g. if-then-else conditions that specify the operating schedule for our ITIL IMS use case). The injection of code was made using specific `<TOKEN>` tags placed in the receiving code which was replaced by the code being injected. In the fifth step, the code and instructions generated are integrated and organized according to an MVC-based structure customized to services and named MVC for Services. Information on the deployment is extracted from the network graph and packed into a service archive along with the MVC for Service structure. The networked graph contained information on the parameterization of the commercial PaaS and the instructions needed for deployment. In the last step, the service archive is automatically deployed into a commercial PaaS platform. After this last step, the ISS is available for use by end users which can interact with it using a Web browser.

C. ITIL service use case

As a use case, we decided to apply our SEASIDE end-to-end development approach to analyze, design, implement and deploy the Incident Management Service (IMS) specified

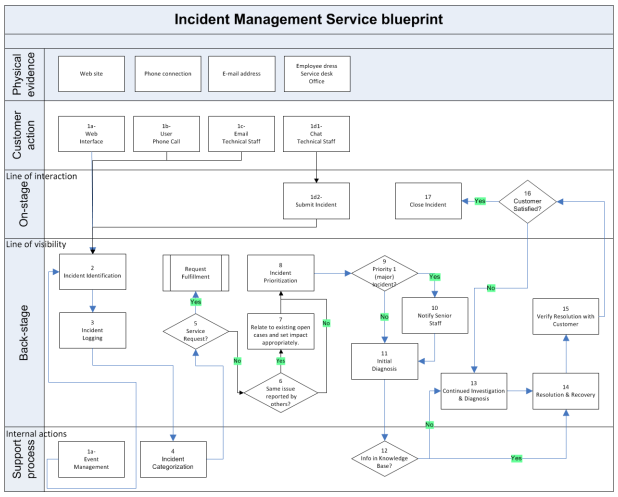

Fig. 3. ITIL Incident Management Service blueprint

in the Information Technology Infrastructure Library (ITIL). The primary objective of the IMS is to resolve incidents (e.g. application bugs, disks-usage thresholds exceeded, or printers not working) causing an interruption of processing in the quickest and most effective possible way. Figure 3 provides a simple representation of the process model behind the IMS. The decision to use this service in our research was made since ITIL provides detailed descriptions of well known services enabling other researchers to replicate our experiments and findings.

While nowadays software solutions which provide ITIL services already exist (e.g. ServiceDesk Plus and Aegis Help Desk), our objective is not to develop similar solutions or provide a more or less functional alternative. Our goal is to demonstrate how architectures, model-driven developments, service patterns and cloud-based infrastructures are important building blocks for the systematic engineering of ISS.

IV. Service analysis with weak semantics

The service analysis phase is prescribed according to the Zachman framework's contextual abstraction. In this phase, a description of the most important elements associated with the service and its parameterization is created using mindmaps and nested bullet lists to identify the main concepts as a taxonomy (this approach was also used by the Untangle project [19]).

A taxonomy provides *weak semantics* for the domain of ISS. The term weak semantics [20] (or *lite* semantics) refers to semantics that can be identified based on simple structural and syntactic formalisms and contain simple information in small "chunks" (*deep semantics* deals with the issues of human cognition, perception, or interpretation). The taxonomy adopted classifies service information entities in the form of a hierarchy, according to relationships of the real world entities which they represent. In our work, the meaning (or semantics) of the relationships varies and depends on the service architecture dimension under study. For example, for the 'what' dimension, relations represent 'is-part-of', and for the 'how' dimension, a relationship indicates an activity or a sub-process. Therefore, a long and profound study with service stakeholders is required to model its most important aspects. As a brief example, we used nested bullet lists to represent a taxonomy for the ITIL IMS service of our case study. The taxonomy for the 'what' dimension was composed by the concepts: Incident, Solution, Customer, etc.

1) *Incident*. Unplanned interruption of an IT service.
   a) *Priority*. How quickly the service desk should address the incident.
      i) *Impact*. The effect on the business.
      ii) *Urgency*. The need for a fast resolution.
   b) *Supervisor*. The responsible actor.
2) *Solution*. Steps to solve the incident.
3) *Customer*. The actor which submitted the incident.
4) ....

For example, the Incident concept has nested concepts and contains all the information needed to manage incidents such as incident Priority composed of the concepts Impact and Urgency.

For the 'how' dimension, several concept functions, on the provider and customer side, were identified. On the provider side, the following nested concepts exist: Categorization, Prioritization, Investigation, and Closure. The concepts also form a taxonomy of the most important functions of the ITIL IMS:

1) *Categorization*. Classify incidents according to a category and subcategory.
2) *Prioritization*. Use of metrics to determine priority.
3) ....
4) *Investigation*. A largely human process to identify the sources of the incident.
5) *Closure*. Closed incidents remain in the system for reference purposes.
   a) *Survey*. Carry out a user satisfaction survey.
   b) *Documentation*. Ensure that the incident record is fully documented.
   c) *Recurrence*. Determine whether it is likely that the incident could recur.
   d) *Formal closure*. Formally close the incident.

The creation of mindmaps and nested bullet lists is carried out for all the dimensions of our service architecture, i.e. for the 'what', 'how', 'where', etc. dimensions.

The reader is referred to our previous work on service analysis with the ISE methodology (see [10], [21], [12]).

V. Service design using MDD

The service design abstraction aggregates several models designed according to our service architecture framework (see Figure 1). While the selection of models may vary, we have conducted our experiment to model the ITIL IMS using an E-R model (ER) for modeling data ('what'), service blueprinting (SB) [5] to model functions ('how')(see Figure 3), a network graph (NG) to model locations, a low fidelity prototype to model user interfaces (UI) with people ('who')(see Figure 4),

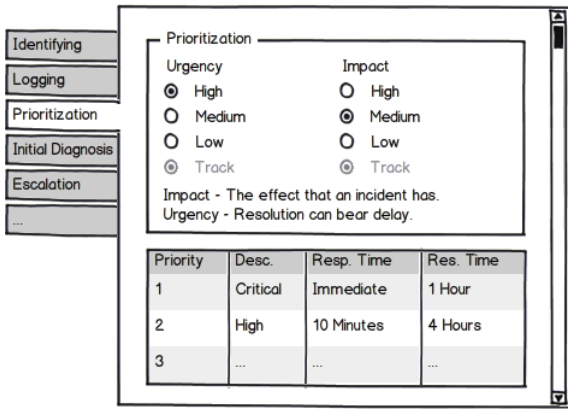

Fig. 4. Low fidelity prototype to design the UI for the ITIL IMS categorization and prioritization activities

and USDL [22] to model timings and motivations ('when' and 'why').

The software implementation of the various ISS models is not trivial since developers need to posses skills to understand various languages. Given the substantial learning curve of these languages, researchers have proposed that software artifacts (WSDL, EJB, CORBA Objects, etc.) can be generated automatically from models (such as UML diagrams and ER models). If it was also possible to generate ISS software implementations directly from ISS models using MDD then it would be possible to generate complete ISS applications directly from widely understood representations. This would reduce implementation costs, increase homogeneity, and reduce programming errors.

For Internet-based self-services to realize the full benefits of MDD, it is important to find ways of formalizing and organizing architectural artifacts according to a service architecture to afterwards allow an automatic code generation [23]. As such, we rely on MDD to abstract models of ISS and systematically transform these models to concrete implementations [24] organized according to an adapted version of the MVC pattern (called MVC for Services) to be later deployed in a PaaS. The transformation of ISS' models builds upon our previous work [25] in which we use of XSLT and XPath-based instructions to transform the models exchanged in business-to-business transactions. The system developed made use of XSLT instructions to transform transactions for global trading (represented with ebXML, xCBL, etc.) into a company's private and internal domain data model (represented with an ontology expressed with OWL serialized with XML).

MDD's defining characteristic is that self-service developments primary focus on services models rather than computer programs. This is because the complexity of service systems that software typically deals with is often very far from statements of data, functions, interfaces, and rules that constitute programming languages. Since ISS are a very specific type of applications, we rely on domain-specific modeling languages, such as service blueprinting and USDL, to formalize with a high precision the structure, behavior, and requirements of self-services. The use of specific languages addresses the inability of generic languages to alleviate the complexity of services and express domain concepts effectively. Since most of the models we have used are well-known, we will only describe two of them which required extensions and/or adaptations: (1) service blueprinting with BPMN and (2) business description with USDL.

A. Service blueprinting

Service blueprinting [5] provides a visual solution to express the intentions and goals of ISS while linking them to customer's needs as the service back-end process progresses. Typically, service blueprintings are created using "paper and pencil" or office tools (e.g. Visio).

Since service blueprinting tools with an XML serialization are not common, we have used the BPMN notation to design the crucial aspects of services' processes involving actors and customers. The mapping between the two representation languages was done in the following way. Service blueprinting segment processes were mapped into BPMN's swimlanes. Four swimlanes were created to capture customer actions, on-stage employees' actions, back-stage employees' actions, and support processes. The separations of swimlanes corresponds to the line of interaction, line of visibility and line of internal actions. Physical evidence was specified with a special swimlane. Once this mapping was done, we have modeled the ITIL IMS process illustrated in Figure 3.

Afterwards, we have experimented two distinct tools to transform the service blueprint modeled with BPMN, and serialized with XML, into an MVC-based pattern: (1) Sparx Systems' Enterprise Architect and (2) Bonita BPMN workflow editor.

Using Enterprise Architect, the XML serialization of BPMN was highly complex since there were a considerable number of references to components' IDs, such as 'EAID_7A41FEBE_-D787_4514_AE57_386CECB19C89'. The parsing of XML was not sequential. For example, to retrieve tasks flow there was the need to navigate by "SequenceFlows" and retrieve input and output IDs, then obtain source and destination IDs, store them into a temporary variable, retrieve the respective node and retrieve the corresponding name. This process was extremely complex using XSLT. Mostly because XSLT proceeds only forward. When there is the need to move from one node to another, it is difficult to develop a clean code. As a result of this complexity, we decided to create our own XML transformation engine based on XPath instead of using XSLT. The custom made transformation engine is very flexible since it is implemented with a general-purpose language (e.g. Ruby) and relies on an XPath library to query and select nodes from XML documents. For applications which require only a small set of transformations which do not change frequently over time or are not reused, it is an appropriate solution. Therefore, we decided to make a specific parser to extract the various BPMN elements. As a second experiment, we carried out the same actions but this time using Bonita BPMN workflow editor. The results were significantly different. The

XML specification generated to model the ITIL IMS was simple and easy to understand. As a result, a transformation engine based on XSLT was rapidly developed. This was the most efficient and effective approach.

Both approaches extracted BPMN elements from the XML serialization, i.e. roles, transitions, tasks, etc. which drove the generation of code and instructions that were stored into our MVC for Services structure. For example, roles were important to create automatically an access control list (ACL) and login views for the various actors involved during the provisioning of the IMS. Transitions were used to set the control-flow of the IMS application into the MVC controllers. Control-flow information and event-based information was generated and took the form of control constructs such as If-Then-Else and Repeat-While of the BPMN process model. Each task part of the blueprint is associated with a view designed with a low fidelity prototype. For example, the task 'Incident Prioritization' from Figure 3 is associated with the UI sketched in Figure 4. Low fidelity prototype are created manually (we have relied on the application Balsamiq Mockups). The association is made by matching the task name with the filename of the XML file describing the UI.

Independently of the approach followed (i.e. XPath or XSLT), the generated code was never final. We estimate that approximately 25% of the skeleton and basic structure of an ISS was automatically generated (and 75% must be manually coded). There are always adjustments that need to be made in order to make the code runnable. Those adjustments included refining software programs, HTML user interfaces, database connections, etc.

Nonetheless, to increase the degree of automation, several extensions can be made to our approach. One the one hand, if an organization uses a consistent UI layout for all their Web applications they can create a UI template and add it the view structure of MVC. This will enable to produce, with a high degree of automation, UI which are close to their final visual aspect. The UI code will be injected directly into those templates. On the other hand, the process notation provided by BPMN to model blueprints can be explored to a greater extend. For example, events can be used to represent messages (e.g. e-mails) send by the user and notifications received the application that must be shown to the user. Once events are used in blueprints, they can also trigger the generation of code. Furthermore, BPMN2 provides data objects of type input and output which can be used to automatically build database queries to retrieve data from the MVC's model. As a last improvement, the BPMN annotation element can be use to encode domain specific information which can be used when generating the MVC component.

B. Service description using USDL

The Unified Service Description Language (USDL[2]) [22] was developed to describe various types of services ranging from professional to electronic services. The specification is being driven by important players in the field of business and IT services such as SAP AG and Siemens. We have used it to enhance the description of the ITIL IMS with a special emphasis on business and operational aspects such as quality of service, pricing and legal aspects among others. To transform the USDL specification, we developed a set of XSLT instructions responsible for its transformation into Web page code (i.e. HTML) which was afterwards injected into the views of the MVC for Services structure. USDL was used to provide additional information on the IMS to end users, such as the legal rights and obligations, and the quality of service rendered to end users. Operation schedules were extracted and transformed into software code in the form of business rules. These rules were injected into MVC controllers to specify the periods (i.e. days/hours) when the IMS was available to end users.

Instead of using QVT- or ATL-based approaches, and to simplify the transformation process, MDD relied on the use of XSLT and XPath to transform models into code. The preferred way to generate code was to use XSLT. Nonetheless, the XML representation of some models was extremely complex. In such a case, XSLT did not enabled to construct simple and straightforward instructions and we relied on a more expensive solution with XPath to manually identify XML tags, understand their semantics, and manually encode transformations using a generic programming language.

VI. Service implementation using MVC for Services

The model-view-controller architectural pattern, originally deployed in Smalltalk, is widely used in the construction of Web applications. As such, we believed it would also be suitable for ISS. We propose an adapted version of MVC to create MVC-based skeletons for Internet-based self-services. The generated software code, database instructions, user interfaces, etc. were structured according to an adapted and compliant version of the MVC pattern which we call MVC for Services. This pattern was tailored to account for the specificities of ISS. The adapted MVC followed the following principals:

- The *model* stores the code of the data models of the service architecture (i.e. 'what' dimension).
- The *view* stores the instructions of the UI models of the architecture (i.e. 'who' dimension).
- The *controller* stores the code of the blueprint and USDL models of the service architecture (i.e. 'how', 'why' and 'when' dimensions).

The MVC was structured according to the service blueprint's segments. In our running example, the MVC for Service has been structured with customer actions, on-stage employees' actions, back-stage employees' actions, and support processes. This MVC for Services skeleton construct enabled to streamline the automated process of code generation since many PaaS providers accept to upload MVC-based applications to their platforms.

Internet-based self-service implementations, organized according to MVC, are automatically generated using

[2]http://www.w3.org/2005/Incubator/usdl/

MDD and stored in a service archive which contain PaaS dependent deployment information. The MVC's model is obtained from the ER model developed during the service design phase. In this phase, a SQL physical model is created. We have explored the applicability of Enterprise Architect 9.1 and MySQL Workbench 5.2 to design and export the ER model to XML. Afterwards, XSLT was used to extract tables and attributes names from the XML document. Names where passed to Heroku and using the scaffold command, the tables with attributes were created automatically in the database. For example, `rails generate scaffold Incident name:string priority:string description:text` was used to create a table named 'Incident' with three attributes: 'name', 'priority' and 'description'. Other existing Heroku commands include 'belongs_to' and 'has_many' to model relationships and association between tables.

The views (coded in HTML) are generated from the information obtained from the service blueprint (each human task originates a view and roles to determine access privileges to views), from low fidelity prototypes of user interface (for each view a UI model is created), and from the USDL specification (information on QoS and legal constraints is obtained). The ISS developer can afterwards modify the generated views or create other views using a 'green field' approach. In the last step, our approach generates software code (e.g. Ruby or PHP) which constitute the MVC's controller. The controller is the middle component between the models and the views. As a simple example, the following method encodes the decision point identified with label 9 in the blueprint from Figure 3:

```
def create
 @incident = Incident.new(@params['priority'])
 @incident.date = Date.today
 if @incident.priority == ''1''
   redirect_to :action => 'Notify_Senior_Staff'
 else
   redirect_to :action => 'Initial_Diagnosis'
 end
end
```

This method first creates a new Incident object and initializes it from the parameters posted by the activity 'Incident_Prioritization'. If the priority of the incident is 1, it redirects the control-flow to the next activity specified in the IMS blueprint, i.e. 'Notify_Senior_Staff', otherwise, it redirects the flow to the activity 'Initial_Diagnosis'.

VII. Service deployment using PaaS

To deploy the service archive created, our approach relied on the use of a platform as a service. A PaaS provides an Internet-based software delivery platform for multi-tenant, web-based applications that will be hosted on the provider's infrastructure thus reducing costs and increasing scalability.

The service archive, containing our MVC for Services and deployment information extracted from the network graph of the service architecture, was deployed to the PaaS. PaaS provides access to an abstract middleware infrastructure where the generated ISS code is uploaded by the PaaS provider and presented on the Web. With PaaS, an ISS can be build without installing any tools and deployed without requiring any specialized system administration skills. Heroku, Google AppEngine, Force.com, Bungee Connect, LongJump, WaveMaker are all instances of PaaS.

In our experiment to deploy our ITIL ISS we selected the Heroku platform[3] – a provider that emphasizes ease of use, automation, and reliability for Web applications – since it supported Ruby on Rails, a popular programming framework with a focus on simplicity and productivity. This language has shown to greatly reduce both the number of lines of code and the development time of our ISS when compared with other languages such as Java. Therefore, it is well suited for our initial goal of reducing development costs. The following commands were retrieved from the network graph and executed automatically to deploy the ISS:

1) $ heroku create <imservice> –stack cedar
2) $ git init # Initialize the code repository
3) $ git commit ... # Several commits were made
4) $ git push heroku master

Ruby software applications using Rails follow the model-view-controller design pattern. This means that the selected PaaS knows exactly where all the necessary directories, models, views and controllers of our ISS are located and how they are structured.

VIII. Achievements and lesson learned

Our initial goal was to analyze, design, convert into code and deploy the ITIL IMS from our use case to a PaaS. All models' transformations and deployment needed to be achieved with a minimal effort and as close as possible to a "double click" paradigm. This goal was accomplished since only manual "cosmetic instructions" needed to be done to MVC's views and some "hard-wiring" was needed to the controllers and models.

The use case followed gave us several insights on how the future development of Internet-based self-services will look like. The main lesson learned is that by using the proposed SEASIDE systematic methodology we can observe a faster, simpler and more structured approach to services' developments. While many other lessons were learned, we decided to only refer the six more significant:

1) Using only one modeling dimension was not sufficient to develop an ISS. Therefore, an EA was used to provide five dimensions (i.e. data, function, locations, time, and motivation) which were important to consider.
2) The Zachman framework enabled to specify services' models with domain specific languages thereby enabling the use of very specific, small languages which were easy to manage.
3) Since ISS have a strong business orientation, MDD provided formal, high-level models which were readable by service domain experts and business owners which did not needed to face IT and technology.

[3]http://www.heroku.com

4) The externalization of service blueprinting to model the behavior of ISS, instead of hard-wiring processes into code, increased stakeholders' understandability.
5) MDD and MVC enabled to check and enforce compliance since ISS applications had to comply with a service architecture.
6) While the methodology was used to create an Internet-based self-service of the IMS, we believe that the approach can be applied to other types of services as well (e.g.).

Based on our current achievements, future directions of research include the development of an integrated workbench for ISS development and the execution of comparative benchmarks to test the developments costs using various strategies (e.g. using different PaaS, different models, and different programming languages).

IX. Conclusion

This research resulted in a systematic service development methodology, called SEASIDE, to create Internet-based self-services. Results indicate that the approach is suitable for the 'massification' of services' since it reduces development complexity and costs, and time to market. The insights gained demonstrated the applicability of integrating EA, MDD, MVC, and PaaS to support a systematic and step-by-step guidance for ISS development. The use of an enterprise architecture required its adaptation to a simpler model which revealed to be intuitive and powerful for ISS' stakeholders. The separation of ISS models from the code using MDD enabled stakeholders with no programming skills to participate in the ISS development process. Nonetheless, manual adjustments to the code were always necessary but had a small impact to development time. The creation of the MVC for Service pattern resulted in a lower cognitive load for developers when adjusting code since it was organized according to a structure that closely resembled the elements of a service blueprint. The use of a PaaS approach to deploy ISS was surprisingly fast and simple since our MVC for Services was deployed in the 'cloud' using a small set of instructions and enabled a transparent scalability.

References

[1] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86–93, Mar/Apr 2002.

[2] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall, 2005.

[3] A. Hochstein, R. Zarnekow, and W. Brenner, "ITIL as common practice reference model for IT service management: formal assessment and implications for practice," in *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*, 2005, pp. 704–710.

[4] V. A. Zeithaml, M. J. Bitner, and D. D. Gremler, *Services Marketing: Integrating Customer Focus Across the Firm*. McGraw-Hill, 2008.

[5] S. Fließ and M. Kleinaltenkamp, "Blueprinting the service company: Managing service processes efficiently," *Journal of Business Research*, vol. 57, no. 4, pp. 392 – 404, 2004.

[6] S. Nenonen, H. Rasila, J. Matti, and S. Karna, "Customer journey: a method to investigate user experience," in *Proceedings of the Euro FM Conference Manchester*, 2008, pp. 54–63.

[7] J. O. Thomas, Y. A. Rankin, and N. Boyette, "Self service technologies: eliminating pain points of traditional call centers," in *Proceedings of the Symposium on Computer Human Interaction for the Management of Information Technology*. New York, NY, USA: ACM, 2009, pp. 9:60–9:63.

[8] M. L. Meuter, A. L. Ostrom, R. I. Roundtree, and M. J. Bitner, "Self-Service Technologies: Understanding Customer Satisfaction with Technology-Based Service Encounters." *Journal of Marketing*, vol. 64, no. 3, pp. 50–64, Jul. 2000.

[9] J. Cardoso, M. Winkler, and K. Voigt, "A Service Description Language for the Internet of Services," in *First International Symposium on Services Science (ISSS'09)*, Leipzig, Germany, 2009.

[10] V. Bicer, S. Borgert, M. Winkler, G. Scheithauer, K. Voigt, and J. Cardoso, "Modeling services using ise framework: Foundations and extensions," in *Modern Software Engineering Concepts and Practices: Advanced Approaches*, A. H. Dogru and V. Bicer, Eds. Information Science Pub, 2011, pp. 126–150.

[11] H. Kett, K. Voigt, G. Scheithauer, and J. Cardoso, "Service engineering in business ecosystems," in *Proceedings of the XVIII International RESER Conference*. Stuttgart, Germany: Fraunhofer IRB, 2008, pp. 1–22.

[12] J. Cardoso, K. Voigt, and M. Winkler, "Service Engineering for The Internet of Services," in *Enterprise Information Systems X*, vol. 19. Springer, 2008, pp. 17–25.

[13] C. V. S. Prazeres, C. A. C. Teixeira, E. V. Munson, and M. da Graça C. Pimentel, "Semantic Web Services: from OWL-S via UML to MVC applications," in *Proceedings of the 2009 ACM symposium on Applied Computing*. New York, USA: ACM, 2009, pp. 675–680.

[14] I.-W. Kim and K.-H. Lee, "Describing Semantic Web Services: from UML to OWL-S," in *Proceedings of the 2007 IEEE International Conference on Web Service*, july 2007, pp. 529 –536.

[15] J. H. Yang and I. J. Chung, "Automatic generation of service ontology from UML diagrams for semantic web services." in *Proceedings of the 1st Asian Semantic Web Conference*, 2006, pp. 523–529.

[16] J. Timm and G. Gannod, "A model-driven approach for specifying semantic web services," in *Proceedings of the 2005 IEEE International Conference on Web Service*, vol. 1, July 2005, pp. 313–320.

[17] Microsoft, "SAPO Portugal Telecom subsidiary helps ensure revenue opportunities in the cloud," 2011.

[18] D. L. Parnas, "On the criteria to be used in decomposing systems into modules." NY, USA: Springer, 2002, pp. 411–427.

[19] I. Horrocks, D. L. McGuinness, and C. A. Welty, "The description logic handbook," F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds. New York, NY, USA: Cambridge University Press, 2003, pp. 427–449.

[20] A. P. Sheth, "Panel: Data semantics: what, where and how?" in *Proceedings of the Sixth IFIP TC-2 Working Conference on Data Semantics: Database Applications Semantics*. London, UK: Chapman & Hall, Ltd., 1996, pp. 601–610.

[21] J. Cardoso, M. Winkler, K. Voigt, and H. Berthold, *IoS-Based Services, Platform Services, SLA and Models for the Internet of Services*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2011, vol. 50, pp. 3–17.

[22] J. Cardoso, A. Barros, N. May, and U. Kylau, "Towards a unified service description language for the Internet of Services: Requirements and first developments," in *IEEE International Conference on Services Computing*. Florida, USA: IEEE Computer Society Press, 2010.

[23] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, "Developing applications using model-driven design environments," *Computer*, vol. 39, no. 2, pp. 33 – 40, Feb. 2006.

[24] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Proceedings of the 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37–54.

[25] J. Cardoso and C. Bussler, "Mapping between heterogeneous XML and OWL transaction representations in B2B integration," *Data & Knowledge Engineering*, vol. 70, no. 12, pp. 1046–1069, 2011.