HAAGA-HELIA
University of Applied Sciences

Integration of Maventa electronic invoicing web service into PlanMill business application suite.

Gavin Van Dok

25.11.2011

HAAGA-HELIA
University of Applied Sciences

Abstract

Date of presentation 6.10.2011

Business Information Technology

Author or authors Gavin Van Dok	**Group or year of entry** 2008
Title of report Integration of Maventa electronic invoicing web service to PlanMill business application suite	**Number of pages and appendices**
Teacher/s or supervisor/s Zahid Anwar	

PlanMill Cloud is an online, subscription-based B2B product which supports service-based companies' business processes such as project management, employee time reporting and customer billing. PlanMill recognized a need for more efficient generic automation within its infrastructure. With increasing demands, especially in northern Europe for businesses to use electronic invoicing there was urgency for an automated setup and configuration of PlanMill's online services with a reliable electronic invoicing operator.

As a part of the 14.1 Version release on 22.9.2011 PlanMill's goal was to completely automate the registration, settings, verification and enabling of electronic invoice sending (which includes printing and other services offered by the operators). The chosen operator was 'Maventa' – a locally based and emerging player in the e-Invoice market. The integration would take advantage of Maventa's comprehensive open API web service (Bravo version).

This report describes the architecture and explains the methods to be used. It then follows the requirements gathering, creation of user stories and corresponding Behaviour Driven Development models (BDD's), selection and requirements testing of the API methods and Maventa's class libraries with JSP tools, data mapping between to the two systems, and finally the refinement of the development BDD's into useful cross-browser test cases. Due to some resource restrictions and the version release deadline the project made some compromises for a less functional first-release of the feature.

An appropriate new UI would need to be designed to implement the BDD's within PlanMill's Java-based factory model architecture. The Java implementation was to be primarily undertaken by a senior developer utilizing the results of the JSP tests and in accordance to the BDD's. Any UI layer applied business rules would be handled by JavaScript and be the responsibility of the author.

Keywords
web service integration, e-invoicing, API, PlanMill, Maventa

Table of contents

1 Introduction 1

1.1 Background 1

1.2 PlanMill's service segment customer base 1

1.3 Current methods, processes and resources 2

2 SaaS model launch goals 3

2.1 Limits of current sales model 3

2.2 Preparing for larger customer base with increased automation 3

2.3 Customization versus self-service features 4

2.4 Feasibility studies and profitability 4

2.5 Implication of research results - reorganization of resources 6

3 Working Methods for project 7

3.1 Scrum backlog, sprint of shippable product 7

3.2 Behaviour Development Design BDD artefacts 9

3.3 Kanban production process system 9

4 Architecture of PlanMill 10

4.1 Factory method design model 10

5 Existing functionality and the need for further development 12

5.1 Invoicing and limited e-invoice functions 12

5.2 Need for generic functionality for a multi-branched wizard 13

6 User stories and Behavioural Development cases 13

6.1 User stories to describe the desired functionality 13

6.2 Selection of functionality to be built in first Sprint 14

6.3 Acceptance criteria for user stories 15

6.4 First draft BDD definitions 16

7 Requirement gathering process 17

7.1 Research of API methods to match the user stories 17

7.2 Collection of important concepts, method responses and errors 18

7.3 Research of the Maventa UI and infrastructure 19

7.4 Clarification of registration process into state diagram 20

8 Database requirements 21

8.1 New code fields for recording states 22
8.2 New columns required in the existing Account object table 22
8.3 Design of a new Keystore table 22
9 UI design 23
9.1 BDD cases to wireframes 23
9.2 Wizard design considerations 24
10 Rationalization phase 25
10.1 Reduction of scope for first sprint 25
10.2 Assumptions made for unnecessarily complicated preferences 26
11 Finalization of Design specifications 27
11.1 Refinement of BDDs 27
11.2 Creation of clear additional specifications linked to BDD's Business rules 27
11.3 Data mapping 28
12 Implementation 29
12.1 System specification 29
12.2 Distribution of implementation across PlanMill architecture 29
12.2.1 Server side implementation 29
12.2.2 Client side implementation 30
12.2.3 Production-use JSP tool for adding keys directly to the Keystore 30
12.2.4 Script-handled behaviour design requirements 33
12.3 Communication 33
13 Acceptance testing and documentation 34
13.1 Evolution of BDD cases into Acceptance test cases 34
13.2 Documentation of completed functionality 36
14 Learning outcomes 38
14.1 Checklist for integration of an application with a third-party web service 38
Bibliography 40
Attachments 1
Appendix 1 - User stories 1
Appendix 2 – CONFIDENTIAL ..... **Error! Bookmark not defined.**
Appendix 3 – CONFIDENTIAL ..... **Error! Bookmark not defined.**
Appendix 4 - Acceptance BDD test case for Submit Registration form 2

1 Introduction

1.1 Background

PlanMill has been operating in its current form since 2001 though it has a longer legacy as a Project management tool. The business suite now supports Project management, Customer relations management, and some ERP processes such as invoicing. Its strength is its focus on service-based industries that invoice mainly on a time-reporting basis, and often in a project driven environment. PlanMill appears to fill a demand that applications like Microsoft's Dynamics Navision (which support more product-based ERP processes) are not satisfying.

One important feature of PlanMill Cloud is the ability to directly create invoices from the project revenues and sales orders in support of the complete sales process. However research carried out by the author had confirmed that with their currently manual configuration PlanMill could not profitably support new customers with less than 5 users. Thus PlanMill recognized its need to continually develop towards a better 'SaaS' (Software as a Service) mass-configuration and automation leveraging model to have any growth in this potent 'micro' segment.

1.2 PlanMill's service segment customer base

PlanMill specializes in SME's within the broad category of service-based businesses. These vertical segments of these groups include IT services, IT software project companies, Media, and Accounting, and Financial Services.

PlanMill's current online infrastructure and segmentation is geared to mainly medium sized business with 20-50 users though they have several clients supporting over 250 users. PlanMill still services a small market share in larger on-premise solutions. These were out of scope and not directly considered in this project. However the current

function rich architecture and more complex configurations and setup processes are a result of this larger corporation heritage.

1.3 Current methods, processes and resources

Currently any customizations and connections to third-party services require manual configurations, custom stored procedures and business layer coding by consultants and developers. There is a waiting list for these customizations, they can be costly and there always the risk of mistakes in the requirements gathering and implementation of the code. The major problem is of course growing liability of managing and maintaining the many thousands of lines customer specific code, parameters and scripts throughout continual evolution of the business suite. In a multi-tenanted architecture such as PlanMill, direct customizations create risks of failures in the customer instances each time the regular version upgrades are launched. The thankfully smaller group of highly customised instances must be considered and tested thoroughly each upgrade.

Business process support model

PlanMill is about supporting the business processes of customers in its target segments. The most commonly used function is the time reporting function for employees as they report billable or non-billable hours to projects, tasks and requests. These activity entities may or may not be attached to revenues with billing rules. In the case that revenues are generated they will in turn require invoicing of their corresponding customers' accounts which are managed in the CRM functions.

Sales order based invoicing facilities were introduced to PlanMill in 2010 by popular demand, although invoicing in general had been available since 2006. These functions completed the support of the clients' business process in a holistic approach from prospect to quote to order and then implementation through to sending invoices to customer and to accounting software. By encouraging efficiency with data

centralization and best practice models PlanMill seeks to grow its customers in a mutually beneficial manner.

2 SaaS model launch goals

2.1 Limits of current sales model

PlanMill has a proven strength in the mid-sized company segments as it offers customizations and higher standards of service.
The PlanMill online application suite known as 'PlanMill Cloud' is a push towards the low capital outlay cloud architectures that more and more companies are seeking. Cloud application solutions offer quick start-up, subscription based fees and lower maintenance issues – leaving the backups and recovery planning to the vendors.

The time of this project the sales and setup processes at PlanMill actually only profitably supported accounts with at least 5 users. The pre-sales time with the processes was preclusive of anything smaller due in no small part to the function-richness of the product and the many questions that arise from prospects. The SaaS model strategy of providing a no-obligation, free trial-period access to the working product was impossible due to the manual setup and configurations required to get started.

2.2 Preparing for larger customer base with increased automation

This project was a part of a longer term parent project to redesign the business application software, as well as improving the support software such as helpdesks, to prepare better for capturing the 'long-tail' of the business spectrum. In other words, the longer term strategy is the to make these smaller customers a more viably profitable option for inclusion in the target market as well as develop better processes for handling the larger segments such as 10-20 users companies. With better nurturing of prospects and empowerment of customers to self-service, supported by well-designed

automation, both the conversion rate of prospects to subscription members and the growth of existing customer could be expected to increase.

The automation of the registration and configurations for electronic invoicing services including hands off activation of timed invoicing jobs (known as chrons) which was undertaken by this project was a flagship implementation. As it was so different to any pre-existing automation functionality it was hoped to be a model for automated support of all necessary configurations, settings, instance creations, sample data and essential data objects to get a new user working with PlanMill with as little manual 'hands-on' dependency as possible.

2.3 Customization versus self-service features

One of the proposed benefits of cloud solutions with mass configurable shared architecture is that the evolution of the product benefits all the users (York 2009). This is the complete reverse of the situation with the pure customization approach. Customizations can result in alienation of the customer from subsequent versions upgrades and functions enhancement as more and more compatibility issues inevitably surface. There were at the time of writing clear evidence of this in several cases with clients using old functionality due to their customizations and the sensitivity or risk to those customizations of failure if subjected to new code paradigms. PlanMill was actually designed for customization optimization, as we shall see in the architecture overview section. However, as Web 2.0 shifts demands a new level of user experience this originally innovative architecture approach simply cannot be relied upon for an optimal SaaS application

2.4 Feasibility studies and profitability

PlanMill has made a decision to progressively enhance its web-based business application suite 'PlanMill Cloud' to allow more self-service facilities for their customers.

Profitability studies carried out by the author strengthened suspicions of the inefficiency of the manual set up and configuration process. As an illustration a spread sheet table was created to link the dependencies of number of subscriptions, non-billable pre-sales time (e.g. setting up a free-trial instance) and conversion rates of lead-to-subscribing-customer with average cost figures. We considered the SaaS 'long tail' example cases of 1, 5, and 10 user subscription customers. For a given average subscription fee of 50 euros per user, a consultant labour cost of 100 euros per hour and a lead to sales order conversion rate of 10 the 'zero point' for profit was determined for the pre-sales time wastage. The results were very interesting for the consultants involved in these processes.

Taking modest non-billable post sale maintenance post sales For the 10 subscription user customer group if any more than 3 hours was spent per customer on presales and just hour per month non-billable on post sales then profits would turn to into negative liabilities.
For the 5 customer group the figure was correspondingly 1,5 hours and the single subscription customer would only have to use 20 minutes of a consultant's time with free-trial handling and questions before he would risk eating the company's hard-won profits.

Even more interesting was the consideration that as much as the long tail of micro customers offers a promise of a mass customer base – it also carries with it a high exposure to liability. If these large numbers of small subscriptions and profit margins per account are not managed correctly they risk dragging a company down by a difference of just minutes spent per customer pre-sales. With just 500 customers won against a conversion rate of 10. If presales time exceeds 25 minutes then single subscription customers would have, after one year of paying their subscription, generated 'profits' of minus 200 000 euros - a very substantial loss despite all that hard work by the consultants.

2.5 Implication of research results - reorganization of resources

PlanMill experienced a growth burst of 100 per cent in the two years preceding this project with much of that growth over a single financial year. In the aftermath of this growth burst PlanMill could affirm that they were in possession of a product-service combination that was in high demand in the SME market while at the same time there was a realization that their own business processes with the current infrastructure are limiting any further sustainable growth. They simply could not service the customer base's needs with their setup and continued customization demands all along with any possible challenges, bugs etc. that could arise in each case.

The customization demands are both a reflection of the niche market that PlanMill finds itself in along with a symptom of the lack of inherent self-configuration functionality in the application. PlanMill's current customers are small enough to be interested in a non-enterprise ERP system, being precluded from highly priced start-up costs and unnecessarily complex systems such as SAP and still possibly Microsoft Dynamics Nav. However many of the customers are mature enough to have complex business processes that require assistance to set up in PlanMill, and not able to be modelled without some customizations. It was the situation at the time of writing that a small group of even more established clients consumed the most resources, all-be-it billable work, with little or none of that development benefiting any the other customers directly.

Thus the growing realization across the company from developers through to management has been that despite the short term profitability which is proffered by satisfying the current customers with billable customizations - these activities actually block the finite development resources from:

1) Creating more generic functions that could be reused for all customers
2) Redesigning the architecture to facilitate the support of a multiplied customer base

3) Developing systems that actually leverage the power of automation and allow PlanMill to reach more of its potential market.

The author had performed his internship at PlanMill being involved with an API integration of complex customized instance with a then necessarily complex front-end web application. In addition he also experienced first-hand the day to day demand on resources which result from the continual flow of smaller customization tasks. The resource drain affect the process chain in all the areas of request handling, estimates, requirements, implementation, fixes, billing, and maintenance.
With multiplication of customers you have directly proportional multiplication of load across the complete spectrum of activities – the exact situation which the SaaS seeks to avoid.

This project was thus a positive shift from these existing billable and customer specific activities to non-billable endeavours. The primary goal was to empower PlanMill's application environment with a seamless setup of e-invoicing. Mutual benefits as vendor or reseller of the service provided some profit generating motivations but primarily it was cost-saving driven with a need to reduce the customer care and setup load generated by interest and implementation, billing and maintenance issues related to e-invoicing.

3 Working Methods for project

PlanMill draws its working processes from the Agile development model. It takes the major elements from the Scrum subset of Agile though without strict adherence to the full practices. Pair programming, user stories, minimal documentation and test driven development are some of the important aspects that define PlanMill's approach.

3.1 Scrum backlog, sprint of shippable product

As a part of the requirements gathering phase discussions with the customer are focussed on the eliciting a set of clear user stories which describe the expected

functionality from the system or feature. These are without exception inclusive of the following points:

Who wants to do it?
What is wanted to be done?
Why I want to do it?
What are the minimum acceptable behaviours?

User stories are readable by all the stakeholders as well as the developer team. They are clear and include small enough functionality and complexity to be completed in a single 'sprint' period such as 30 days.

A user story could read like:

> "As a manager user of the system I want to get an activity report from a given company so I can make decisions of which campaigns they would be suitable for.
>
> Acceptance criteria: I can choose the period and it is phased into months."

The requirements gatherer, if they are not the actual the coder cannot be certain of the full consequences of a given desired functionality at a code level. This is why the implementing developer should make an estimate based on the current functionality and assets as to the time demand of each story. From this estimate the decision of where to draw the line of a 'shippable product' for that sprint can be made; whether another story could be included or should the primary one be broken into smaller user stories.

For the larger projects PlanMill as per the Scrum model maintains a product backlog comprising of all the stories on the customer's 'wish list' that must wait until subsequent sprints and for resources, prioritised in order of importance.

The essence of agile development is frequent communication and daily incrementing through development iterations of committed code. This strict methodology was still

not yet fully implemented at the time of writing however good version management practices had been established.

3.2 Behaviour Development Design BDD artefacts

The User story model from agile development is enhanced by breaking the general story into elemental steps. Taking the acceptance criteria into account these steps are then translated into workable Behavioural Development Design (BDD) cases. BDDs are in fact testable cases and should be completed with all the preceding stories, given conditions, actions, results, error and useable example data value

The Model for the BDD case is as follows:

Given story and scenario,

Given, when, then. Examples.

3.3 Kanban production process system

The context of the project was the introduction of a production management environment based on the Kanban system. This is a method developed in Japan in factory production companies to avoid bottle necks and encourage the smooth prioritised accomplishment of tasks. It has also been proven to be effective in intelligent material production like software.

Kanban in a similar spirit to Scrum (which avoids telling the developers HOW to do the implementation and rather WHAT the feature should do), avoids the traditional paradigm of assigning of tasks to people. Instead Kanban model puts the onus on the performer pick the task up from a prioritised waiting list. This allows some level of choice as the performer matches themselves to the task and also encourages more ownership in the process. The reason that Kanban has worked for many companies is that it is simply an excellent system for telling people what to do. (Baudin 2011). It could also be speculated that workers perform generally better and with more

motivation when they only have one or two tasks to consider at a given point in time - rather than a mounting list of assigned work.

All PlanMill requests for new functions and customizations pass through the following stages of the Kanban process: Requirements, Coding, Testing, Documentation, Production release. The principle is that a limited number of tasks should ever be underway in all of these phases at any one point in time. Jobs are given clarity to be completed more quickly and no stages of the process are then able to be omitted intentionally nor forgotten

4 Architecture of PlanMill

The PlanMill online business application suite uses a combination of open source and propriety software. The programming language used for the primary system is Java, running on Apache Tomcat servers. The database used is Microsoft SQL Server 2008 (while also supporting SQL Server 2005).

The application is by definition a multi-tenant paradigm with shared server environment and a default shared client. However each customer instance has its own partitioned Database instance and own customer specific UI instance which inherits all the common instance parameters and configurations from the shared client unless customer parameters exist.

4.1 Factory method design model

PlanMill has been designed according to the Factory method model to allow for more generic object oriented structure and allow for easier customizations with only a small increase in overhead of the complexity (Niinioja 2010).
The implementation of this model is through the partitioning into the broad layers of UI, Interface redirector, Business modules and Database - all of which have relevance in this project's design definition and specifications.

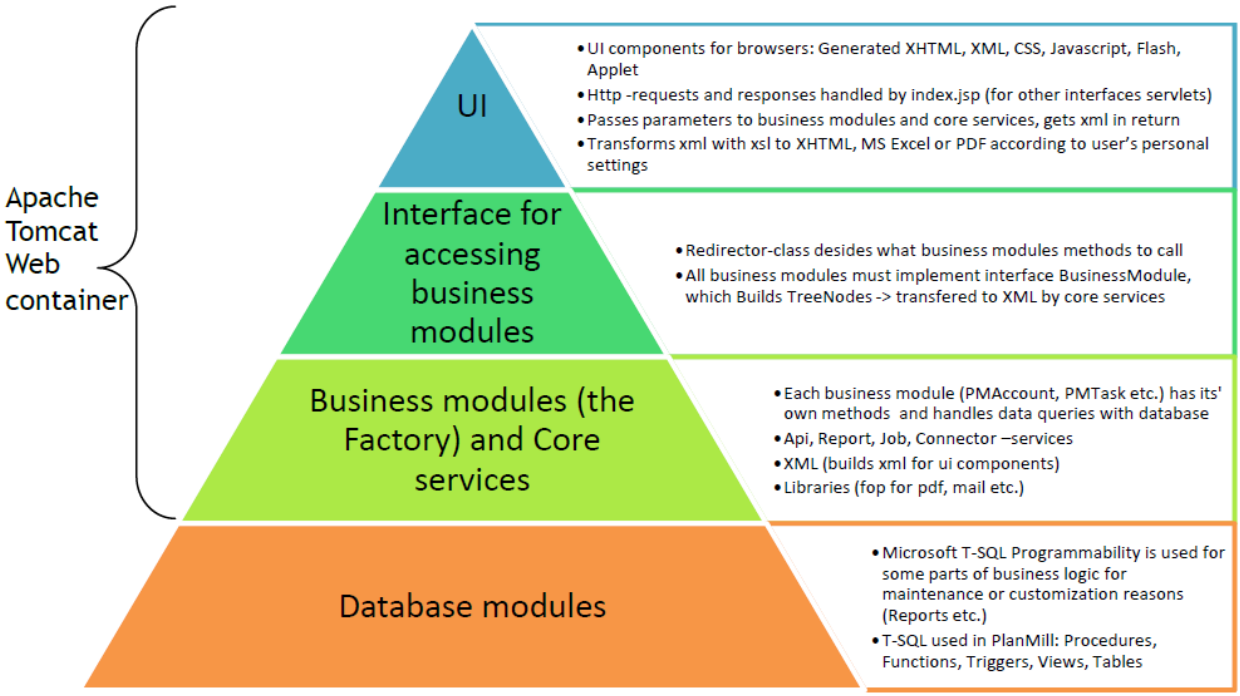

Figure 1.0 Architectural layers of PlanMill according to the factory model (Niinioja 2010)

Rather than attempting to explain the full architectural intricacies of the system at this point - from an unapplied theoretical perspective - we will cover them further in the context of the project implementation distribution in section.

The multi-tenanted functional design is partitioned by applying access rights to each subscription and user. Access rights implement access to functionality based on the subscribed-for services of PlanMill's client and the designated role of the user. Each user is checked for access rights to tools and categories and actions each page load. Optimization is important to keep the system useable despite its rich functionality.

5 Existing functionality and the need for further development

5.1 Invoicing and limited e-invoice functions

The pre-existing functionality of PlanMill included creation of invoices from Sales order or project generated invoices. For all customers who subscribed to this level of product group the invoice could simply be outputted in PDF form and handled manually. A select group of customers, with manual set up and configuration had the ability to transform the invoice files into the Finvoice-standard xml files for transfer via an electronic invoicing operator to their customers. This functionality had been setup originally for an intermediary other than Maventa however a small pilot group of customers were sending invoices electronically with Maventa by using these Finvoice files. There existed some technical issues with the current situation as Maventa has a different model for handling the sending process compared to the original operator. The desired channel was not always carried out for a given e-invoice address as Maventa required explicit declarations in the SOAP envelope used for the Finvoice. to ensure the correct delivery. This issue would need to be solved as a part of this project.

In any case PlanMill had observed there to be a distinct market gap between any existing software or SaaS offerings that could satisfactorily provide this standardized service and the SME companies demanding it. Other ERP solutions that can handle invoicing such as SAP are costly, overly complex for SME's and require substantial investment in understanding their application to the company's processes. The pilot projects were attempting to stop-gap meet the need and test for further development like this project.

Better handling of e-invoicing for a large group of customer demanded some level of automation of the registration and setting preference process. Maventa had a UI for its customers in place however PlanMill's goal was to avoid the need for the user to enter the Maventa environment and be baffled by the setting choices, Maventa's own wizard as well as the management of access keys

5.2 Need for generic functionality for a multi-branched wizard

Automation of the registration process pointed towards a multi-branched wizard type registration. Initial feasibility for this highlighted the need for several generic type handlings and functions to be added to the architecture. These included: radio buttons, long text in forms, popups in forms, links and XHTML parsing in form field caption. None of these had been necessary before and they are a good example of how a product is forced to evolve as more powerful and informative interfaces are created.

The form of the primary UI had been suggested to be a wizard however the question arose as to whether the concept of 'wizard' was really matched correctly to the need. If the user simply needed to fill out some form details and let the automation handle the rest then it could simply be a simple form. The existence of a wizard at Maventa possibly pointed to the need to break down the decision making process for the customer into phases. However with the integration project a major advantage would be that much information may already pre-exist in the customer's PlanMill database. The complications for the integration project arose due to the existence of the alternative direct registration channel. This fact generated the possible case combination of pre-existing company registrations and pre-existing user accounts eventually described as four distinct scenarios in section 13.1. Also account activation requirement enforced by Maventa created a user-dependent break in the registration and setup process while trying to abstract it away from the third party.

6 User stories and Behavioural Development cases

6.1 User stories to describe the desired functionality

From initial discussion about the functionality needs and consideration of the pre-existing resources the project embarks formerly with the creation of the first draft user stories. Around ten user stories were created and considered for breaking down into clarified BDD cases. These were formulated from a brainstormed list of possible

functionality desires imagined by the stakeholders. The brain-storm wish list included the following for various actors in a customer company:

> 1) I want to register my company for e-invoicing so I can send invoices electronically to my customers
> 2) I want to add my existing Maventa account to PlanMill for sending e-invoices direct from PlanMill
> 3) I want to choose the preferred invoice sending channels for my customers and let PlanMill set the channel for each customer based on the available information
> 4) I want to see a list view of all my customer companies which the invoice channel for each based on the preferences set and flags possible missing information.
> 5) I want to browse and match companies that are listed for e-invoicing to my own customers in PlanMill
> 6) I want to add other PlanMill users to be able to send e-invoices via Maventa

The user stories were formalized according to the model discussed previously in the Working Methods section and managed from a collaborative wiki space. The primary user story was selected to be related to the registration process and read as follows:

> "As a finance user I want to register my company with Maventa so I can bill customers using efficient e-invoicing"

The full list of formalised user stories is attached at Appendix 1

6.2 Selection of functionality to be built in first Sprint

From the user stories the following functionality was selected to be delivered as a shippable product at the end of the sprint, coinciding with the version release:

1) Automated registration and setup of a new Maventa company account from a PlanMill company account's summary view (including generation and storage of new Maventa access information)

2) Setting up invoicing in PlanMill with a pre-existing Maventa company account (mapping of existing account to the corresponding PlanMill account and storage of existing Maventa access information)

3) Automation of invoicing channel based on preferences.

The first two functionalities would be implemented by use of a multi-branched wizard. While the third would be the responsibility of some automated decision logic initiated by a relevant trigger and based on settings applied in the registration process.

The second sprint would add the functionality to manage subsequent users (add, update, delete) with the 'wish-list' suggestion of automatic user creation based on PlanMill user access rights. The architectural design for the first sprint would still need to prepare for multiple users despite the delay for that story.

Discussions with CEO and chief project manager defined the acceptance criteria for the now agreed shippable product of automatic registration. All the other remaining user stories would comprise the product backlog in appropriate priority.

6.3 Acceptance criteria for user stories

Acceptance criteria encapsulate the extra business rules that are not a simple to define as a measurable condition. Rather than simply listing all the acceptance criteria we will discuss some of the important ones that affect the project. Here is an example of an acceptance criterion needing to be transformed into BDDs:

"When Customer clicks register PlanMill collects all relevant fields for invoicing in addition to the required fields for registration and fills them with any existing data in the database."

With the form being generated within the PlanMill system and inheriting any known relevant information the user would not be required to fill out a completely empty form. So in contrast to the common case of registering to a non-integrated service, there was the chance to include more fields without creating a so called 'churn point'.

Purpose of this criterion - it both eases the process of registration by avoiding any unnecessary double-handling of any existing data and at the same time leverages the opportunity to improve the quality of existing information in PlanMill and prevent errors later when user tries to create invoices

Another important acceptance criterion was that the keys should be all stored in a new DB object table called a Keystore in the database for each instance. This would allow implementation of multiple user key support, as opposed to the current three pilot customers who were all using an instance bound set of shared and unencrypted parameters. The keys would need to be encrypted before storage at the database as a security acceptance criterion. Each key should be protected from misuse because each user would be accountable for invoices sent with their key and as they had hierarchical roles that affect the creation and management of more keys and users each user key and company UUID should be handled carefully.

6.4 First draft BDD definitions

So from the above acceptance criteria the first draft BDD cases were able to be defined. They would be refined much more clearly later however it was important to start partitioning the behaviour of the functionality for the requirements research. The first BDD's included the following

1) Register new company account new user
2) Register new company account with existing user.
3) Set up e-invoicing with existing Maventa Account

4) Apply company settings for invoicing

For example BDD 1) could read as:

For a given <User>,
And <User.role> ('finance'),
And <Account.Id>,
And <Account.Type> ('My Company')
When clicking the 'Register' button
Then Update form fields to the Database and make API call with registration parameters to Maventa.
Result: If succeful save returned keys to Keystore and notify user of activation email.

The practice is to use the form element id, which for the most part corresponds to DB column, as much as possible to aid mapping and later possible automation of such behavioural test cases.

7 Requirement gathering process

7.1 Research of API methods to match the user stories

The documentation for the Maventa Open API lists all the available methods. Each method includes some example code in Ruby and Python. Neither of these resemble closely Java however the code was useful to at least determine some required fields for the methods. Inspection of the user stories and BDDs pointed to a list of possible useful and essential methods for the first and second sprints. They have been listed below:

register_company – in case of creating a new company account (BDD case 1 and 2)

company_show, user_show - in case of an existing Maventa account needing be appropriately mapped to PlanMill (BDD case 3)

company_settings_update - for setting the preferred or recommended settings for other invoice channels such as printing services and email (BDD 4)

set_invoice_receiving - (BDD case 4)

user_create –for adding subsequent users (additional user story)

company_lookup – used for actively determining if a customer is able to accept electronic invoicing using Maventa as the intermediary (additional user story)

7.2 Collection of important concepts, method responses and errors

As a result of the minimal documentation the JSP test tools were also very useful for teasing out and recording both the successful response strings and any possible error messages from each method. These would be essential for string matching at the implementation stage for the various predictable conditions and were added to the requirements specifications which supplement each BDD case.

Some important concepts that were established from the API research included:

1) The user API keys and Company UUID needed as access credentials with any API calls,

2) The use of the email address as Username identifier, and the actual e-invoice identifier which could be the EDI (OVT in Finland) or a Maventa Id (referred to more generically as the invoice net address).

3) The vendor key and server address for the API were two additional important concepts but these were simply common to all users and set in parameters. PlanMill as a vendor had to manage both a testing and production version of

these so it was important to build some functionality that ensured the two were not mixed between the testing and production release phase.

Company identifier and all the API keys are in actuality each a Universally Unique Identifier (UUID). The UUID is a 128 bit hexadecimal number that uses among other things a clock-time based algorithm to generate a globally unique number without the necessity for any standards control authority. (Leach, Mealling and Salz, 2005)

Continuing with our discussion of important concepts and the user activation and its impact on the process to achieve a registered-and-verified state was recognized to be an important issue. The verification emails were noted to possibly distract the user away from the integrated registration process by offering too much unnecessary information such as displaying the keys plus an invitation to log into the Maventa UI. This could confuse the user and raise questions which it was the goal of the integration project to avoid.

7.3 Research of the Maventa UI and infrastructure

In parallel to the method selection, implementation and testing process the Maventa UI and direct registration process was inspected for important points. Both Maventa's own registration form and the company settings viewable when logged-in were considered and tested for behaviours in comparison to the API method results.

One important point which arose from this was the business rule associated with handling of VAT numbers in EU excluding Finland (where the VAT number is also the business identity number where applicable). This one small issue had consequences for the project UI behaviour and mapping issues - PlanMill reserves a place in the account for both BID and VAT while Maventa needs only to capture whichever one is relevant. Some client based script would definitely be required to

handle this.

The impact of the invoice channels preferences offered by Maventa were attempted to be clarified further by navigating through Maventa's UI and by using the system. As no invoices were actually sent on the test server it was difficult to conduct testing of the delivery method.

Printing services are an important channel in the digital handling of invoicing to reach the customers that do not yet accept electronic invoices while still increasing the efficiency of the invoicing company processes. Any invoices marked for printing service are forwarded by Maventa onto Itella's 'Ipost' service where they print the attached PDF and deliver them via the local post. The cost for this service at the time of writing was between 1-2 euros per page.

As a third alternative Maventa also provides online invoice previewing and downloading of PDFs with notification links to the customers email address. When not explicitly indicated the invoice method or preferences Maventa tries to deliver the invoice first as an e-invoice, then email, then finally as printing service. This rose questions as to how to explicitly predict the sending channel - due to the fact that PlanMill was using the simpler finvoice_put method rather than the more explicit invoice_put method available.

7.4 Clarification of registration process into state diagram

To help describe the concept of company registration status from the view-point of PlanMill a diagram was created encompassing the following states:

Unregistered, Registered – unverified, Registered –no keys, Registered -verified

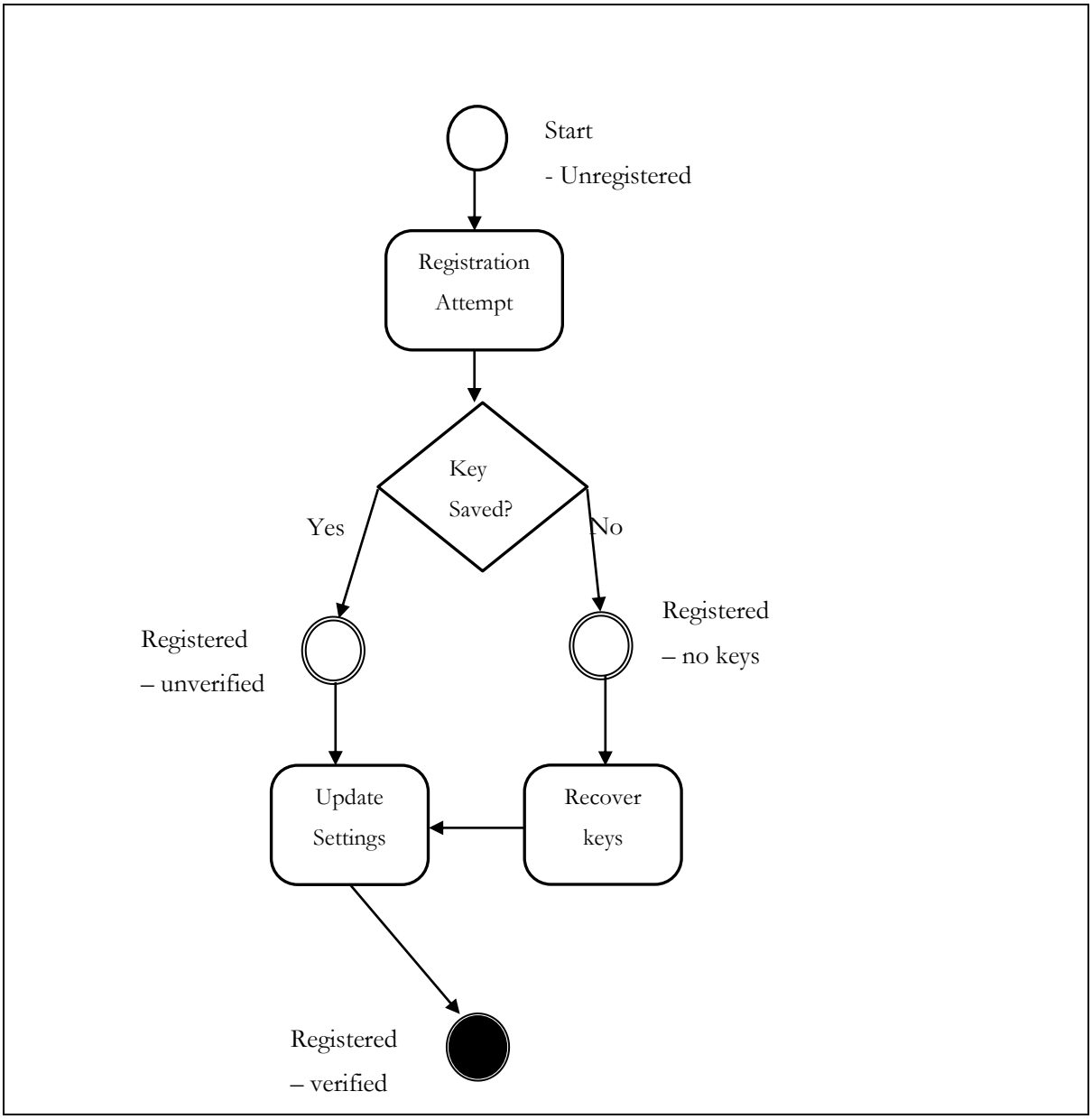

Figure 3 – State diagram for Registration concept

8 Database requirements

To support the new concepts introduced by the Maventa system requirements some adjustments would be necessary to the database. These would take the form of changes at 3 levels of persistency in the structure: code field in a generic table, column additions to the existing account table and a completely new table.

8.1 New code fields for recording states

As mentioned the activation requirement for a new account/user created a need for a state indicator in the system. Settings could not be updated and invoices sent until the user had activated the account via the email link.

Rather than impacting the Account table for this status record an integer code would be saved to the generic code table and thus no unnecessary space would need to be reserved in the DB. Other concepts would however affect the Accounts table.

8.2 New columns required in the existing Account object table

The currently unsupported concepts still needing to be bound to the company account were Printing services (or Ipost) region, Ipost type, Sending operator, and finally the Invoice receiving option. It was decided for management purposes to add these to the account table as null-allowed columns to be used when applicable.

8.3 Design of a new Keystore table

The access data being bound to both accounts and users and the multi-user acceptance criterion determine a need for Keystore. This was a service that would have use for all third-party applications in PlanMill and thus demanded the most persistent handling in the form of new table.

Concepts that needed reflection in the Keystore included:

1) User bound API key / company bound UUID,
2) User / Company association has many to many multiplicity,
3) initial administrator user immutability (company cannot exist without a user and user key),

4) subsequent users,
5) User roles (ADMIN or USER).
6) Active or Inactive status of users
7) Email used as identifier at Maventa as this was bound to the user API key at the time of creation.

These concepts were encapsulated in a simple excel table with some example data rows which established unique key sets for each company /user combination. This table would be used as a model for an iterative implementation of the Keystore by an SQL creation script. The primary design of the table was as follows:

Key store table design

Id	PersonId	AccountId	AppName	KeyName	KeyData
2776100	2689817	2776096	Maventa	user_api_key	############
2776102	2689817	2776096	Maventa	company_uuid	############
2776116	2689817	2776096	Maventa	user_role	ADMIN
2776117	2689817	2776096	Maventa	email_address	test@test.com

9 UI design

9.1 BDD cases to wireframes

By combining the draft BDD cases, PlanMill acceptance requirements and Maventa requirements with the research knowledge gathered the UI could begin to be described in 'wireframe' form. This was actually a process that occurred in parallel to the research of the methods to help visualize the user experience and clarify complexities. A multi-branch wizard that started with a choice of new registration or adding existing account keys would now need to be implemented.

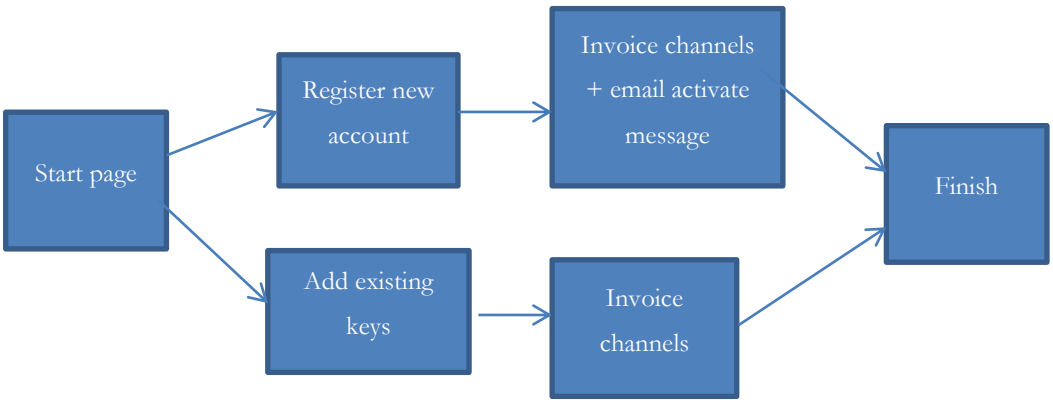

Figure 4. Wizard navigation first draft version

The most important page for the user would be the registration form. It would have the responsibility of collecting any information already existing for the company in PlanMill and demand any missing information to be completed before proceeding to perform the API call. The wireframe image for this page has been included at the end of this section at Figure 5.

The wireframes were also excellent tools for clarification discussions with the stakeholders in regards to the acceptance criteria such as messages to clients. As they are only a starting point for design the UI there is no need to retain them as artefacts at the end of the project -they may not resemble anything like the final UI. Thus Figure 5 the only wireframe included in this document.

9.2 Wizard design considerations

Research on UI and wizard design highlighted several important points described here as a checklist (Constantine and Lockwood 1999):

1) Question and choices should be worded as the intentions of user to make the consequences as clear as possible. (For example a 'Register me' button rather than generic 'Next' or 'Submit'.)
2) Upon launch inform the scope of the wizard to the user
3) Maintain state where possible between steps if wizard is aborted before completion.

4) Keep the user informed, and use simple language

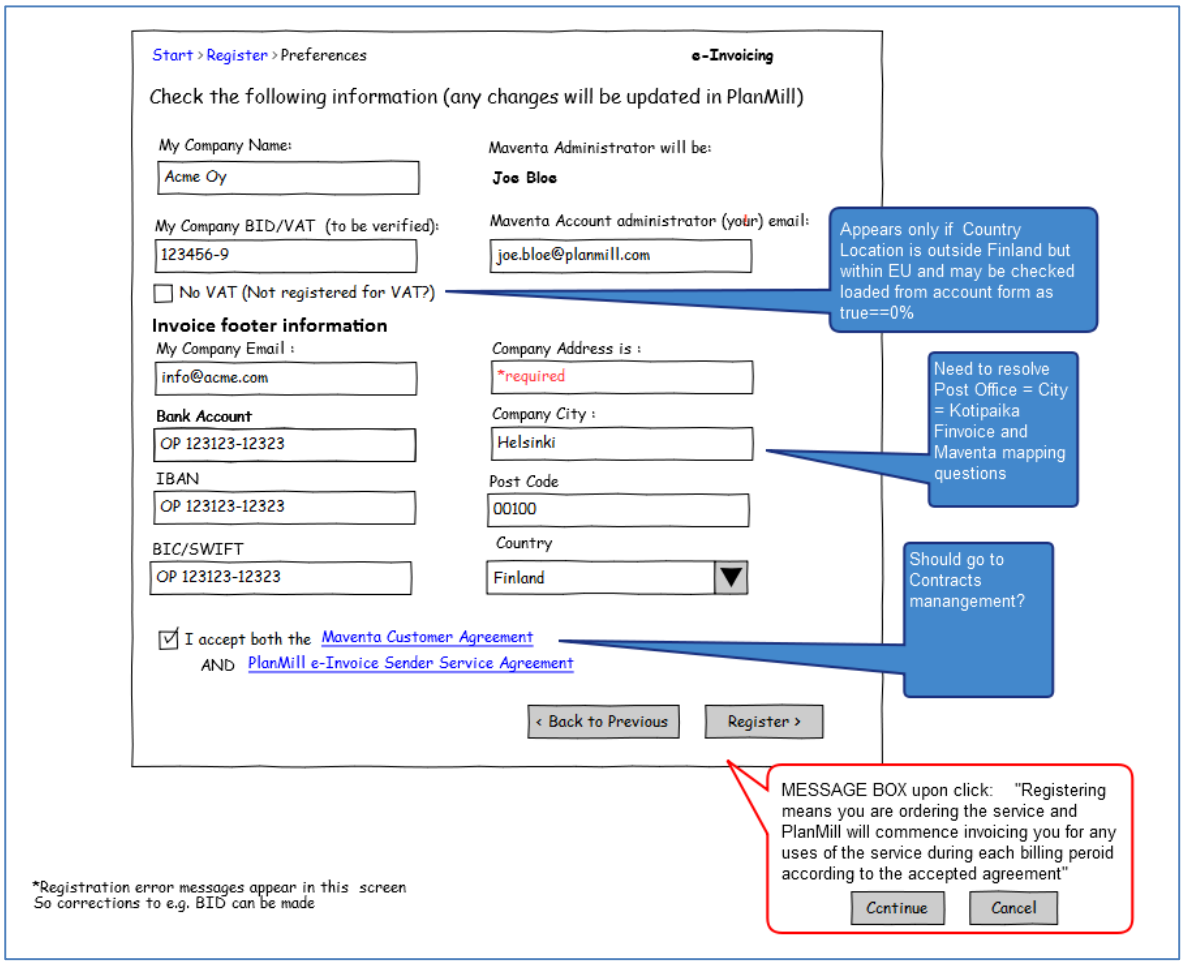

Figure 5 – Wireframe draft example of Registration form in PlanMill wizard

Best attempts were made to satisfy the discussed design recommendation as shown in the preliminary wireframe design example at Figure 5. Notice the inclusion of navigation breadcrumbs, checkboxes using intent wording-structure captions, and the contextually named buttons - all specific design optimization choices.

10 Rationalization phase

10.1 Reduction of scope for first sprint

The requirements phase took more time than expected partly due to lack of documentation and also raised some complications for modelling the multi branching

choices. The time restrictions and the requirement to have a shippable product at the end of the sprint forced a re-evaluation of the scope of user stories. The project was consolidated to now only implement the '*Registration of a new company*' story in the 14.1 version wizard. This was of course the primary story. The second story would still need to be covered but instead of within a multi-branched wizard it would now be implemented in a JSP tool as stop-gap solution. This persistent need for the second story was driven by three known and pressing cases of existing Maventa accounts owners – namely the pilot customers which had been setup manually using customer instance residing parameters (as semi-generic customizations).

Clarification was made that the multiuser supported design would need to bypass handling of the user binding and point to a single administrator keys until the implementation of the creating and manage user story in the next sprint.

10.2 Assumptions made for unnecessarily complicated preferences

The invoice channels settings offered by Maventa risked overly complex logic in the product backlog automations of customer account billing settings. After producing extensive BPMN decision modelling it was decided to be in the interests of stake holders to make some useful assumptions of settings:

The default values for Printing-services were for enablement to the world and economy post. These were now promoted to assumptions removing any user choice concerning these. Assumed preferences were to now be presented to the user as simplified channel choices available at the individual customer account level. The choices were reduced to only the e-invoicing or printing services for this sprint. The third option at the customer account would be to override the Maventa channels by selecting the currently available channel option 'email'. This rationalization explicitly ignored the Maventa available channel of emailing the invoices to the customer.

The basis for these decisions was experience with the current pilot group's members. Consultants had generally found that clients preferred PlanMill to choose a recommended configuration rather than be presented with a multi-faceted set of preference choices.

11 Finalization of Design specifications

11.1 Refinement of BDDs

Communication of the requirements research results amount to refined, normalized inheritance explicit, and ordered BDD cases and scenarios. This means that in theory every action should be represented by a BDD or scenario. An example of this refinement was the addition of a 'Load registration form' BDD case just for the correct behaviour of the system when the user clicked 'Set up e-invoicing'. This BDD would be responsible for describing the access rights and the relevant form fields to be loaded. The subsequent 'Registration of new account' BDD case would inherit the access rights and correct fields and then only need to test the registration dependent data such as Business Id and country.

Example data was supplied for unit testing purposes in the implementation iterations.

11.2 Creation of clear additional specifications linked to BDD's Business rules

Any complex business rules that could not be encapsulated in scenarios were added to supplementary specification tables. Examples of these business rules included the country dependent submission of the VAT number or Business id field. A list of the current EU countries was needed to be included as key value pairs with the short country code for handling in an appropriate script. Error responses and success responses were included in the scenarios.

11.3 Data mapping

To accompany the BDD cases the implementing developer required data mapping tables between the PlanMill objects and their database columns, and the corresponding Maventa objects and API parameter fields. Figure 6 is an example of the data mapping between the systems relevant for the Save, (Show) and Update settings BDD cases.

Maventa service	Planmill DB mapping	Parameter -default value	Required use	Notes (e.g form alias)	Wizard form / constant
Method: company_settings_update					
status string	N/A			Used only for company_show_settings	""
ipost_own_pdf boolean	N/A		Update - PlanMill Assumes value	should be always true for sprint 1	"true"
ipost int	N/A			Updated by Maventa: used only for company_show_settings.	""
ipost_request int	Account.PrintingService	printing_service	Update	gets checked by maventa and is set to ipost_type value if approved) -> 0 = disabled, 1 = Finland, 2 = EU, 3 = World	
acc_man_email string	N/A		Deprecated - do not map		""
website string	Account.Website				
email_remind_freq int	N/A		Update - PlanMill assumes for simplicity	Set to 7 days? 0 = off (Default is 2 if not changed) Is only the reminder repeat frequency if an invoice is sent by email (not used in sprint 1)	"7"
ipost_verified boolean	N/A			Updated by Maventa: used only for company_show_settings.	""
ipost_type int	N/A		Update - PlanMill assumes	applicable only in Finland, 0 = Economy, 1 = Priority (Default is 0)	"0"
Method: company_invoice_receiving					
invoice_receiving boolean	Account.InvoiceReceive	receive_invoices			Invoice receiving check box

Figure 6 - Data mapping table between Maventa and PlanMill for Company Settings concepts

12 Implementation

12.1 System specification

The system specifications for using the Agile minimal documentation methodology comprised of the clarified BDDs, the finalized UI screen shots especially the forms, the supplementary business rules, data mapping tables, Keystore table design, and a state diagram for the process of becoming a registered and verified user

12.2 Distribution of implementation across PlanMill architecture

The implementation phase, in the spirit of Scrum, does not tell the developer how exactly to code the feature. However it was useful to allocate the design model to the appropriate architectural layers.

12.2.1 Server side implementation

At the server a main Java class module for the business layer was created. All the logic implementing the state machine model for the invoice set up process would need to reside in this server side class. Actions linked to corresponding Java functions were accessed via arguments passed into the class by the web requests.

Main functions implemented in the server were:
Load registration – a database read query and form build
Save registration – a database update query and API call
Load invoicing channels - a database read query and form build
Save invoicing channels - a database update query
Verification of account – API call updating settings, database query changing the verified status code field, and finally an update that enables the running the invoice sending jobs.

The other new server-side class to be built was the Keystore service class. This class was effectively the interface that handled the encryption and all access methods for the

cached data. The Maventa business module would call upon the partitioned services of the Keystore which was to be built for the use of other services also.

The main business module class for the project also depended on services from the pre-existing 'MaventaClient' class. This class handled all the web service connections and served the important 'MaventaApiPort' function object for accessing all the API methods. The binding of Maventa access credentials to both user and account required the enhancement of this existing Maventa client class. Previously a single set of shared parameters and single company limitations were now to be replaced with user specific keys and a support for multiple companies. This amounted to the addition of the account id to the required arguments passed into the Maventa client class and alteration of the key getting methods to now point to the Keystore interface.

12.2.2 Client side implementation

The mapping between the systems is the responsibility of concise forms residing at the client side. The forms exist as field, enumeration and language string parameters, which are mapped to columns in the database tables. The language strings enable dynamic multi-language support -PlanMill at the time of writing was available in Finnish, English and German. Simple concatenations and operations, and JavaScript insertions are performed via in this layer as field parameter attributes along with element types and style class declarations. The attributes are parsed through a Form-builder service class into XML and then transformed by relevant XSL sheets into XHTML. New forms to be implemented at the shared client were the Registration form to support the first page of the wizard and the Invoice channels form for the second.

12.2.3 Production-use JSP tool for adding keys directly to the Keystore

As the Keystore was encrypted it necessarily required an internal use interface for any manual row entries. Also as discussed in section 10 Rationalization phase, some stop-gap handling of the user story '*I want to set up my company at PlanMill with an existing Maventa account*' was still needed despite its omission from the 14.1 release wizard.

Until the implementation into the wizard this user story would be dependent on internal use only of a JSP tool by a PlanMill consultant. These tools were in frequent use already for various extra functionalities that were available to super users only.

The tool would need to consider carefully the registration-verification state of the company that it would produce. As the final goal was to set up the invoicing company to be ready to send invoices it was soon recognized that the JSP tool should implement all the functionality handled by the registration of a new company – not just add the keys.

To clarify this it was found useful to create a combined BPMN process model diagram and review the complete process from registration, configuration through to activation verification and sending invoices. The diagram as seen below, also considers the important company account's verification status which is held by the integer code discussed in section 8.1.

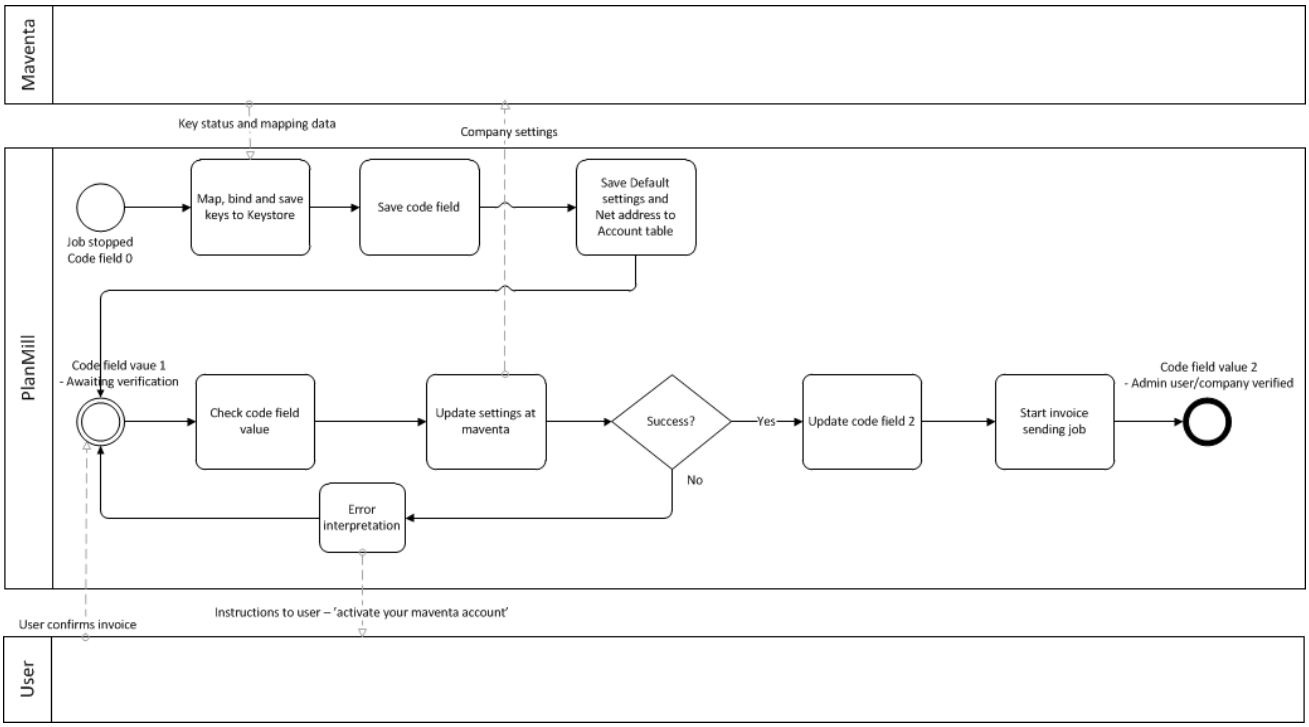

Figure 7. Business Process Notation Model of Setup e-invoicing with JSP tool

From this model an appropriate BDD case could be defined and applied for testing. It is worth noting that the tool did not need to 'reinvent the wheel'. Once the correct consistent state had been modelled any functions developed in the Maventa class could simply be called as needed. The JSP tool would in fact be the first production use of the new services long before any new user would access the wizard as it would be applied first thing on release night for the pilot customers. Later the logical decisions for the tool should in theory be supported by some level of automation in a further enhanced version of the wizard

After implementation of correct the information handling the acceptance criteria for the BDD still needed to be consider. This included the slightly complicated issue of mapping the user bound key to the correct user and the company bound key to the correct company.

It was interestingly discovered that Maventa's open API lacked a direct 'show key own user' based on the API key. The user_show method only accepted the apparent foreign key for the user details objects and as consequence it did not implement any direct link back to the user API key. This was either a relational model or security abstraction however it created the problem that if there is more than one user linked to a given company the API methods can do nothing more than list the all of users attached to it. This would leave you to guess who actually owns the key.

As a consequence the email for a given key used at the time of registration would need to be supplied by the client to correctly map any set of verified user key and company UUID. In section 13.2 as an example of sprint delivery documentation, a screenshot has been included which demonstrates the resulting tool.

12.2.4 Script-handled behaviour design requirements

The remaining business rules for the BDD's which were not able to be handled on the class level would need to be implemented also client side in JavaScript. These included the following:

1) Visibility of the 'Set up e-invoicing' wizard initiation button link in the account summary page for the company.
2) Opening of the UI overlay container for the wizard and the initial call to the business class to load the form
3) The country dependent VAT number relevance rule and corresponding enabling and disabling of fields
4) The consequent 'No VAT' choice in the case of VAT country relevance
5) The closing of the wizard when 'done'

12.3 Communication

Some working methods utilized in this project were newly introduced to PlanMill and experiencing the normal resistance stage from employees. Some of the artefacts provide to the senior developer were found to be useful while others were not or the model was not yet understood. Despite the availability of the collaborative wiki resource the feature request maintained in the PlanMill operations instance was the only reference point used by the senior developer. The requirements documents then had to be carefully linked to the relevant pages in PlanMill's collaborative wiki space containing the project artefacts such as BDD's and other specifications or the developer would not find them.

13 Acceptance testing and documentation

13.1 Evolution of BDD cases into Acceptance test cases

Agile and Scrum methodology encourages the relinquishment of redundant artefacts as the development processes proceeds. As a part of this the BDD's used for design should make way for persistent test cases with useful data examples.

The BDD for acceptance testing for the 'Submit registration form' considered the following four clear scenarios:

1) Unregistered BID/VAT for the given country, unregistered user email
2) Previously registered BID/VAT for the given country
3) Unregistered BID/VAT for the given country, previously registered user email with the PlanMill wizard from given PlanMill instance.
4) Unregistered BID/VAT for the given country, previously registered user email at Maventa but not yet stored in PlanMill (for that given instance).

As the system uses a web-based UI, cross browser issues need constantly to be considered. PlanMill at the time of writing was supporting IE 7+, Chrome, Firefox, and Safari. Test cases had to be created with examples for each of the different browsers. These would be run as a part of the acceptance testing for the 14.1 release each on a browser allocated virtual machines).

The test examples needed to be carefully selected to not conflict at the third-party application with either other testers or developers using the Maventa test server. For example Business ids for the test company along with user emails needed to be explicitly allocated so each browser tester would get an opportunity to test with unique and previously unregistered data. This could be done by using the 'Company lookup' method implemented in one of the method testing JSP tools.

The part of the resulting BDD test case pertaining to scenario 1) appeared as follows:

Given story: BDD Load Register Maventa form

Finance user checks/fills missing fields before proceeding.

Scenario 1:

Business ID / VAT has NOT been registered before and Administrator email address has NOT been registered before.

Given <Account.Id.>

And <User.Id>

And <User Agreement> = True

And <Location1.CountryId> (enumeration list)

And the submitted (<Account.BusinessId> OR (<Account.VatId> + <Country>)

And remaining <e-invoicing registration relevant fields>

When User clicks register (plus confirm order popup)

Then a succesful Maventa API call registration forwards user to the Invoice channels screen with Acount.Name and Person.Email

Result:

Keystore has 4 rows for the user/account combination key/uuid , email and ADMIN role

And <Account.InvoiceNetAddress> updated with Maventa_Id returned from api call.

Examples:

Test Name	<Account.Name>	<Account.BusinessId>	<Account.VATId> (if appears) (if enabled)	No VAT (if appears)	Licence Agree	Maventa Admin Email <Person.Email>
Scenario 1 Happy Day	wega Informatik AG	FI11223355 + 10(IE7) 20(IE8) 30(FF5) 40(FF?) 50(CHR) 60(SAF)	N/A	N/A	Checked	maventa_(browser)@planmill.com maventa_ie7@planmill.com maventa_ie8@planmill.com maventa_safari@planmill.com maventa_chrome@planmill.com maventa_firefox@planmill.com (override the suggested current user email)

Figure 2. A segment of the cross-browser acceptance test data

The complete test case is attached as Appendix 4

13.2 Documentation of completed functionality

The final phase of the sprint includes the documentation of the shippable products in release for release notes and user guide suitable for both PlanMill consultants and end user clients. Also importantly documentation was required for any new related JSP tools such as our user key /company UUID insert tool. JSPs due to their rudimentary interfaces often need more clarified documentation as they are quickly designed for internal use and without much if any consideration for the user experience or intuitiveness.

The author is of the opinion that relying on a tool being available only for internal use can be a big risk for costly errors in the future - especially in the case of this project in which the tool binds users and companies. Due to the lack of automation inherent in the JSP tool the guide leads the user step by step through the necessary decision points. Some effort was made to protect the user from entering just any keys to the Keystore by adding a verification layer however the mapping of the account and person could prove to be very confusing.

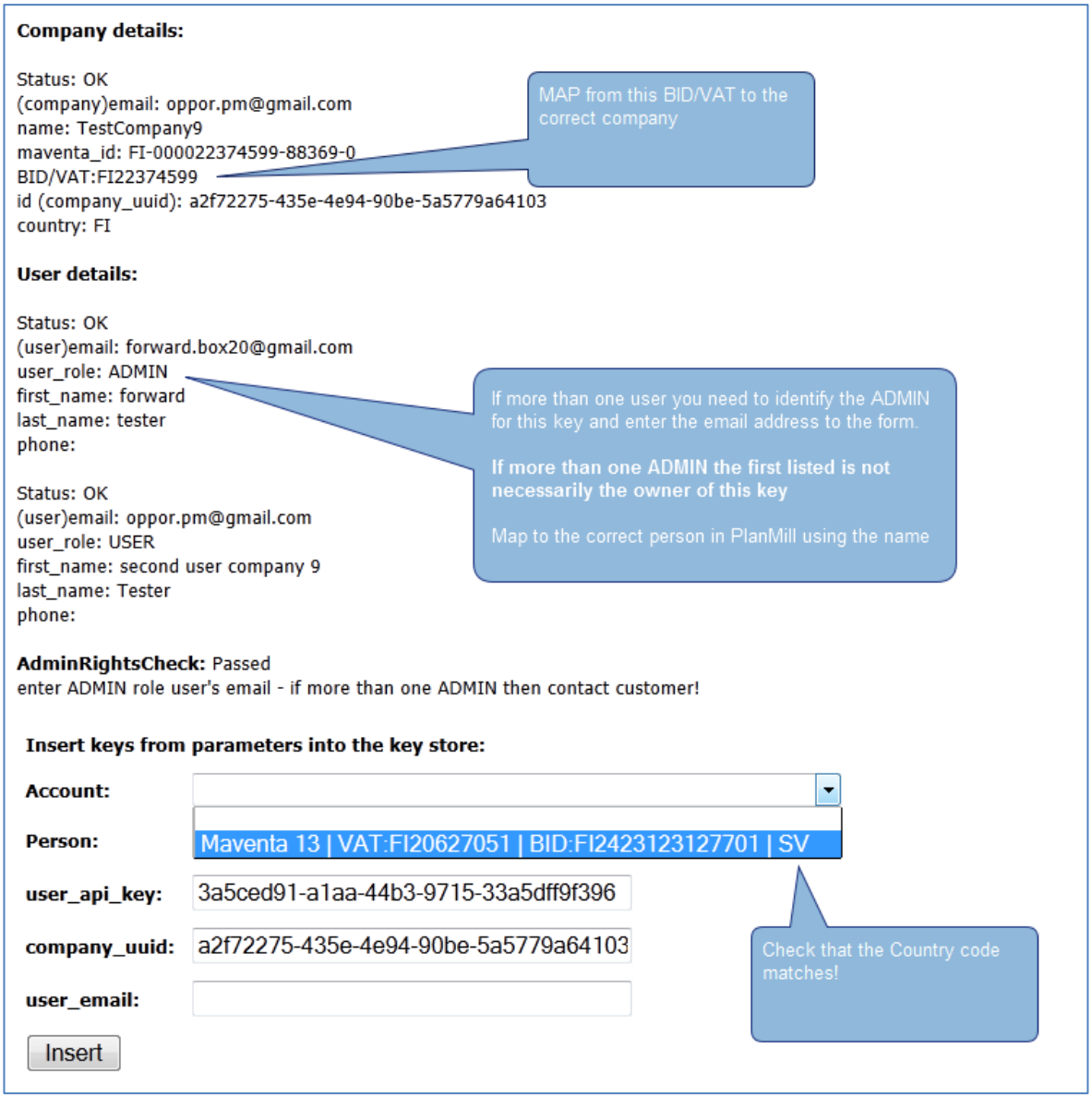

Figure 8. Screen shot of JSP tool as provided in the user guide for internal consultants

14 Learning outcomes

During the implementation phase some integration dependency issues arose which highlight the vulnerability such projects have with constantly developing API's and web services.

The author discovered the need for clearer BDD cases and example test data to aid the implementation process. Along with this the people using the model should agree on what works for them – what is useful and what is not.

Conveying implementation specifications important for further development but not relevant to the current BDD's was found to be a challenge e.g. the multi-user design and the user binding of keys despite the stop-gap solution of a single set of administrator access keys for a given company. Some solution that indicates these needs contextually within the BDD is required.

14.1 Checklist for integration of an application with a third-party web service

A good way to describe the learning outcomes appears to be a checklist of questions to ask when embarking on an integration project.

Are there SLAs covering changes to the API or web service - will they be notified in advance with useful adaptation suggestions? Any change in an expected result can have large effects on a process flow.

Are there any the third party direct communications to the user that bypass your own system? For example the email activation and welcome email in this project: does the third party take into account that the alternative registration process for the user via the API creates many redundancies for instructions and may require very different information?

Will there be good open communication lines between you and the service provider? What is the level of documentation for the service and available methods? The former consideration may make up for the lack of documentation and vice versa.

What is the stability level of the version - is there a new improved service nearing release for that rectifies some limitations?

What can and can't be expected to be done – Are most important user stories even covered by a method?

What are the differences between the testing server and the operational server behaviours? These should be clearly defined to prepare the correct actions for return responses when the system goes live. For example in our tests the VAT validation check was deactivated for the test server and consequently omitted from the test cases.

Bibliography

Joel York 2009, Top 10 Dos and Don'ts of SaaS, http://saas-top-ten-10.chaotic-flow.com/saas-top-ten-do-Enable-Mass-Customization.php Quoted: 18.9.2011

Michael Baudin 2011, What's Unique about the Kanban system http://www.sme.org/cgi-bin/get-newsletter.pl?LEAN&20010209&2

Marjukka Niinioja, PlanMill Architecture Overview, Technical guide, Release 12.2, 4.2.2010

Leach, Mealling and Salz 2005. RFC 4122 -A UUID URN Namespace http://www.ietf.org/rfc/rfc4122.txt Quoted: 18.9.2011

Constantine and Lockwood 1999, Software for Use – A practical guide to User-Centered Design, Edition 6 2004,

Attachments

Appendix 1 - User stories

As a...	I would like to...	so that I...	Acceptance criteria
Finance person	Check all or some or one of my non e-invoicing customer accounts against the register from the accounts list view	i want to keep increase the number of ie-nvoice recipients.	BDD View invoicing flagged columns in list view
	Check a customer account against the e-invoicing register from the account form	i when creating new or editing a customer account i can ready them for e-invoicing.	BDD Company Lookup
	Setup Mycompany is when I have already registered with Maventa	can perform e-invoicing from PlanMill with correct settings	BDD verify keys BDD Match existing user to API key owner and My Company to Maventa account BDD save keys to keystore and update relevate status
	Set up My company for e-invoicing (sending only) with Maventa as a new user of maventa	can perform e-invoicing with any of my customers that are available for receiving e-invoices	BDD Load Register Maventa form BDD Submit Register My Company for e-invoicing Maventa BDD Set Settings for My Company Assumed sending channels: Just invoice sending option needs to be in this form. 1. Setup is always visible but only enabled if PlanMill PSA, ERP or Finance add-on to PROJECT (checked from enabled access groups) 2. PlanMill price list for e-invoice transactions is clearly available even before setup is started 3. Setup can't be completed without accepting terms and conditions for Maventa and for PlanMill 4. Maventa account username and password are sent to user in separate emails (just like when opening account in Maventa website). 5. Maventa account is created with same company and address information as PlanMill My company from where it is created
	Edit the Maventa settings for My company	can update the details for my company like email	BDD load Invoice Channel from BDD submit invoice channels form
	Add an additional My company for e-invoicing with Maventa	can manage multiple companies in planmill as the same user of Maventa	This is handled as a scenario in the register my company user store >
	Add a new user to a given MyCompany Account		BDD Add user to Maventa Registered MyCompany Based on Access rights

Appendix 4 - Acceptance BDD test case for Submit Registration form

Given story:BDD Load Register Maventa form

Finance user checks/fills missing fields before proceeding.

Scenario 1: Business ID / VAT has NOTbeen registered before and Administrator email address has NOTbeen registered before.

Given <Account.Id.>

And <User.Id>

And <User Agreement> = True**And** Maventa Admin Email <Person.Email> is not registered at Maventa

And <Location1.CountryId> (enumeration)

And the submitted (<Account.BusinessId> OR (<Account.VatId> + <Country>) is Not Registered at Maventa yet

And remaining <e-invoicing registration relevant fields>

When User clicks register (plus confirm order popup)

Then a succesful Maventa API call registration forwards user to the Invoice channels screen with Acount.Name and Person.Email (at moment will be different if overriden)

Result:

Keystore has 3 rows for the user/account combination and <Account.InvoiceNetAddress> updated with Maventa_Id returned from api call.

Scenario 2: Business ID / VAT HAS been registered before

Given <Account.Id>

And <User.Id>

And <User Agreement> = True**And** Maventa Admin Email <Person.Email> is not registered at Maventa

And <Location1.CountryId> (enumeration)

And the submitted (<Account.BusinessId> OR (<Account.VatId> + <Country>) is Not Registered at Maventa yet

And remaining <e-invoicing registration relevant fields>

When User clicks register (plus confirm order popup)

Then Form is returned with Error (examples table) and also AlertBox directing user to provide user/company keys for Submitted(<Account.BusinessId> OR <Account.VatId>)

(Appendix 4 continued)

Result:
Keystore has NO rows for the user/account combination

Scenario 3: Business ID / VAT has NOT been registered before, Administrator email address HAS been registered before VIA PlanMill.
Given <Account.Id>
And <User.Id>
And <User Agreement> = True**And** Maventa Admin Email <Person.Email> is registered at Maventa and in keystore
And <Location1.CountryId> (enumeration)
And the submitted (<Account.BusinessId> OR (<Account.VatId> + <Country>) is Not Registered at Maventa yet
And remaining <e-invoicing registration relevant fields>
When User clicks register (plus confirm order popup)
Then a succesful Maventa API call registration forwards user to the Invoice channels screen (with <Person.Email> and <Account.Name>)
Result
Keystore has 3 rows for the user/account combination
AND <Account.InvoiceNetAddress> updated with Maventa_Id

Scenario 4: Business ID / VAT has NOT been registered before, but Administrator email address HAS been registered before at Maventa (not via PlanMill)
Given <Account.Id>
And <User.Id>
And <User Agreement> = True**And** Maventa Admin Email <Person.Email> is not registered at Maventa
And <Location1.CountryId> (enumeration)
And the submitted (<Account.BusinessId> OR (<Account.VatId> + <Country>) is Not Registered at Maventa yet
And remaining <e-invoicing registration relevant fields>
When User clicks register (plus confirm order popup)
Then No keys page is returned directing user to provide keys for user and company with given submitted(<Person.Email>)

Result: Keystore has NO rows for the user/account combination

(Appendix 4 continued)

Examples:

Test.Name	<Account.Name>	<Account.BusinessId>	<Account.VATId> (if appears) (if enabled)	No VAT (if appears)	Licence Agree	Maventa Admin Email <Person.Email>
Scenario 1 Happy Day	wega Informatik AG	FI11223355 + 10(IE7) 20(IE8) 30(FF5) 40(FF?) 50(CHR) 60(SAF)	N/A	N/A	Checked	maventa_(browser)@planmill.com maventa_ie7@planmill.com maventa_ie8@planmill.com maventa_safari@planmill.com maventa_chrome@planmill.com maventa_firefox@planmill.com
Scenrio 2: BID already taken	Maventa BID_Taken	FI123456999 (all testers use this one)	N/A	N/A	Checked	<CurrentUser.Email> (any valid email is ok for this test)
Scenario 3: Email already taken (from previous registration of another company)	Maventa Email_in_Keystore	FI11223355 + 11(IE7) 21(IE8) 31(FF5) 41(FF?) 51(CHR) 61(SAF)	N/A	N/A	Checked	maventa_(browser)@planmill.com (use the exact email in scenario 1 as it should have been stored to keystore)
Scenario 4 Email Taken but not found in keystore (e.g used in registration of another company outside	Maventa Email_Not_Found	FI11223355 + 12(IE7) 22(IE8) 32(FF5) 42(FF?) 52(CHR) 62(SAF)	N/A	N/A	Checked	forward.box20@gmail.com (this should not have been used in test instance previously)

(Appendix 4 continued)

<Location1.CountryId> enumeration	Company Email	Bank Account	IBAN	BIC	Error?
FINLAND	oppor.pm@gmail.com	123456	123456	123456	No Error
FINLAND					ERROR: Registration Failed, BID Alread Taken + AlertBox
FINLAND					No Error
FINLAND					Blank Screen + AlertBox