A content management tool for the QES system

Developing of a web-based content management tool for touch printers

Master of Science Thesis of in the programme Software Engineering & Technology

DEJAN MILOJEVIC

Department of Computer Science and Engineering
CHALMERS UNIVERITY OF TECHNOLOGY
Gothenburg, Sweden 2010

A content management tool for the QES system
Developing of a web-based content management tool for touch printers

DEJAN MILOJEVIC

Examiner: JOACHIM VON HACHT

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden

Department of Computer Science and Engineering
Gothenburg, Sweden, June 2010

Abstract

Queuing systems are an important part of stores, hospital, banks and other departments. QMatic AB is a world-leading company in the area of queuing systems and effectiveness of customer flows. The core products when handling customer flows are the touch printers, containing a screen where a visitor at i.e. a bank office can choose the type of service demand. The visitor gets a ticket printed and is then put into a virtual queue; there is a computer system is taking care to call the waiting clients to different workstations to be served.

The problem QMatic has is that the customers buying the touch printers are not able to manage the content in a feasible way. To solve the problem an investigation has been done by developing a proof of concept content management tool for editing content that can be presented on the touch printers. This report will show how the analysis and modeling is done to finally in an iterative way implement defined use cases, to finally produce a prototype of an editor tool integrated into QES; a web-based enterprise system to handle customer flows. With the tool an administrator is able to in a straightforward way produce content that could be presented and used on the touch printers.

The result is a web-based editor where end-users can create content by using drag-and-drop and finally publish the content on the touch printers. In the appendix of this report there is screenshots of the developed application.

Acknowledgments

This Master Thesis report has been written as the final part of the Master programme Software Engineering & Technology at Chalmers University of Technology, Sweden. The subject was chosen in collaboration with QMatic in Gothenburg, where the thesis also has been performed.
I would like to thank the supervisors for their help and the support that they have given throughout my work:

QMatic
Jonny Dahlberg
Niclas Bentley

Also thanks to rest of the RnD-team, including external consultants.

Chalmers University of Technology
Joachim von Hacht

Dejan Milojevic
Gothenburg, Sweden
2010

Table of Contents

1 Introduction 1
1.1 Queuing and Queuing systems 1
1.1.1 Types of queuing 1
1.2 QMatic AB 2
1.2.1 About QMatic 2
1.2.2 Customer Flow Management 2
1.2.3 QMatic Enterprise Suite 4
1.2.4 Touch printers 4
2 Background 5
3 Problem 5
4 Purpose 6
5 Scope 6
6 Method 6
6.1 Development process 6
7 Technical background 7
7.1 Java Enterprise Edition 7
7.1.1 EJB3 7
7.1.2 Java Server Faces 7
7.1.3 RichFaces 7
7.2 QMatic Enterprise Suite 7
8 Vocabulary 8
9 Pre-study 8
9.1 Market analysis 8
9.2 Domain description 9
10 Requirement elicitation 10
10.1 Use cases 10
10.2 Functional requirements 11
10.3 Non-functional requirements 11
10.4 Usability requirements 11
10.5 Excluded from requirements 11
11 Analysis 12
11.1 Usage – Workflow 12
11.1.1 Activity diagram 13

11.1.2 Preliminary GUI 14
11.2 Domain model 16
12 System architecture/High-level design 17
12.1 Server-Client architecture 17
12.2 Modularized architecture 17
12.3 Design 18
12.3.1 Holding the state 18
12.3.2 Client-server interaction 19
12.3.3 Content relation between editor and display mode 20
12.3.4 Rendering 20
13 Development 21
13.1 Iteration 1 21
13.1.1 Use case #13 Choose a branch type 21
13.1.2 Use case #3 Select page template 22
13.2 Iteration 2 23
13.2.1 Use case #1 Create a page flow 23
13.2.2 Use case #2 Add pages to page flow 24
13.3 Iteration 3 25
13.3.1 Use case #4 Add components 25
13.3.2 Use case #5: Remove component 28
13.4 Iteration 4 29
13.4.1 Use case #6 Modify page properties 29
13.4.2 Use case #7 Modify component properties 30
13.5 Iteration 5 32
13.5.1 Use case #8 Link components to next page 32
13.5.2 Sub-task: Switching among pages 33
13.5.3 Use case #9 Connect components to service 34
13.6 Iteration 6 36
13.6.1 Use case #12 Add overlays 36
13.7 Iteration 7 37
13.7.1 Use case #10 Preview a page flow 37
13.8 Iteration 8 38
13.8.1 Use case #11 Publish page flow 38
13.9 Summary 39

14 Result 40
15 Discussion 40
15.1 Does the tool cover all needs? 40
15.2 Is the tool usable enough? 40
16 Conclusion 40
17 Future work 40
17.1 Input Components 40
17.2 Conditional overlays 41
18 References 42
19 Appendixes 44

1 Introduction

1.1 Queuing and Queuing systems

"Amazingly 70% of people will walk out of a shop if the queue is too long and with so much pressure on our time we are always looking for ways to avoid wasting it in a queue." [1]

"On average we spend 273 days of our lives in queues so it is not surprising that impatience gets the better of us and we start to get seriously annoyed after 13 minutes of queuing. What's more, nearly 10% of adults become seriously annoyed the moment they are in a queue." [1]

Queuing systems are a core part of the infrastructure in banks, stores, hospitals, and other departments. Because of grown complexity and amount of visitors the focus of using computer systems to handle different queuing systems is important.

In this report the keyword *queue* references to a line of people waiting in a row to get some service.

1.1.1 Types of queuing

The difference between following two queuing type should be understood before continuing.

Physical queuing, also called *linear queuing*, means that people are standing in a sequence row where the first one is served first and so on (see Figure 1.1). New people are staying behind the last person in the row and the procedure continues in that way. The advantage with physical queuing is that it is simple to maintain but the disadvantages is that people has to physically stay in row which can be boring and annoying.

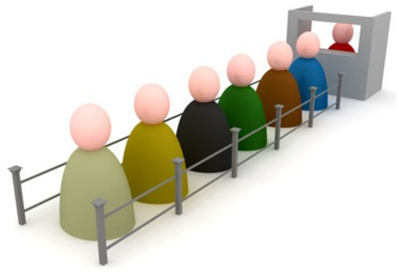


Figure 1.1 People standing in a physical queue waiting to be served.

Virtual queuing means that people are not ordering of in a physical queue but instead there is some system that holds the state of the queue. I.e. each person can have a paper ticket (see Figure 1.2) with a sequence number and the system controls that will be served next. The disadvantage with this types of queues it that they require systems to control the queuing; however there are many advantages person doesn't have to stand and wait in a row but can sit and rest or do other activities while waiting. The queuing waiting time is a psychological experience that can be managed and this is what virtual queuing does; it makes the waiting people gives the impression of feel like the waiting time is shorter than it really is. [2]

Figure 1.2 A ticket with a number representing the place in a virtual queue.

1.2 QMatic AB

1.2.1 About QMatic

QMatic AB is a world-leading company in the area of queuing systems and effectiveness of customer flows. The company was founded in 1981 by a Swedish entrepreneur and an engineer who had a restaurant business in Gothenburg, Sweden. They devised a system whereby customers could not only choose their meal but, by adding a display panel to their numerical system, see when it was their turn to collect it. QMatic's focus is to take care of the whole customer flows with help of their own developed Customer Flow Management process, CFM (see below).

The company operates in more than 110 countries with 40 000 installations of their systems over the world. The main office is located in Mölndal (Sweden) but there are subsidiary companies and distributors over the whole world. The systems are used in retail, retail finance, healthcare, education, and the public sector. Each year 1.7 billion users pass through a QMatic system. [3]

1.2.2 Customer Flow Management

Customer Flow Management (CFM) is a process, developed by QMatic, for managing customer flows - from arrival to the service. But it is also about getting information in form of throwback and feedback from the clients (i.e. clients of a bank) after they have been served. During the whole process different information is collected. The goal of the CFM process is to help their customers (i.e. a bank or store) to increase sales and productivity and to reduce costs by ensuring that a customer is at the right place, at the right time and is serviced by the most appropriate staff member. It also increases client and staff satisfaction by reducing the actual waiting time as well as the perceived waiting time. It also aims to create a relaxed environment and importantly for the clients, it establishes a controlled and fair waiting process. CFM can also generate data and insights about how customers' clients behave and how the staff serves them. [4]

The CFM process in divided into six steps:

1. *Pre-arrival*

   Before clients physically visits the shop, bank, hospital or public service centre the can book appointments via phone or the Internet before arrival. This reduces the time spent waiting can deliver the pre-visit data required for the service provider to staff more appropriately and deliver better customer service. [4]

2. *Arrival*

   On arrival, the visiting clients need to be placed in an appropriate queue. CFM stresses the possibility of segmenting the visiting clients in different queues if appropriate, rather than entering all customers in the same queue. The most common segmentation is based on clients needs, e.g. separate queues for separate services. It also allows the service provider to match customers with the staff who have the most suitable competence to respond to their needs. Another possibility for segmentation is to base it on clients' attractiveness. VIP clients that are considered important for the service provider could for example be positioned at the front of the queue. For the service provider the arrival and queue entry is the first opportunity to start tracking the client on site. This requires client identification and the creation of an initial data point. Usually touch printers are used when clients gives different information about their appointment and finally receives a ticket representing their queuing place. [4]

3. *Queuing/waiting*
   From a customer perspective an important criteria to ensure a high service experience is to secure an "in-process feeling", e.g. the customers need to know and feel that they have entered the queue correctly and will be served in the appropriate time by the appropriate service person. [2]

   The main purpose of this step is to minimize the perceived waiting time. This can be made by engaging the clients in active waiting, e.g. fill the waiting time with activities that reduce the perceived waiting time and hence enhance the waiting experience [2]. By identifying and keeping track of who is currently visiting targeted media can be broadcasted on television screens to keep clients informed and/or entertained while waiting. Other opportunities that both reduce the perceived waiting time and create additional value for the service provider are to engage the client in activities to prepare for the service and reduce workload for staff, e.g. by filling out forms etc.

   Also this can create opportunities for further shopping while waiting, either through strategically placed goods where the clients are waiting or by allowing the clients to move freely around the service providers premises while waiting to be served use media to stimulate further sales. If the service provider knows which individual clients are waiting, the media content can be adapted to target the specific needs of the clients waiting. [4]

4. *Serving*
   Because of that the visiting clients are identify and start tracked as soon as they enter the queue, the staff providing the services can start preparations before the customer actually arrives at the service point. For example, staff could call up the client's history on their screen. They can see every visit the client has made before, who they saw and what the enquiry was about. When the client is being served, data on their visit can be captured and made available for real-time insight through management dashboards. It can also be stored for later use. For instance, management could use the information to view customer wait times or find out how long different transactions actually take to complete. [4]

5. *Post-serving*
   After a client has been served, staff closes the transaction and relevant data – like wait-time and transaction time – are recorded. It is also possible to engage the client in other value-adding activities like answering client surveys. The information gained could be highly valuable for the service provider as it could reveal the client's perception of the service they have just received and highlight areas with improvement potential. It could also improve the client's service experience if they feel that they have the opportunity to make their voice heard. [4]

6. *Managing*
   This part means all the managing, i.e. queuing statistics etc.

If data from the CFM process is gathered and stored then managers can, at any point in time, evaluate the current processes. Reports can be generated on service times and client wait times. Operational inefficiencies can be identified and adder through process changes or training. Trend analysis and statistical reports are also useful tools for achieving and reporting on a variety of organization. [4]

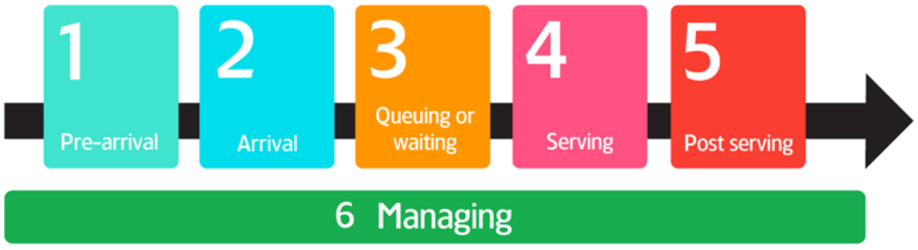


Figure 1.3 An overview of the CFM process

1.2.3 QMatic Enterprise Suite

QMatic Enterprise Suite (QES) is a web-based software platform that is the central point for handling the customer flow. The platform may handle the whole CFM process with functions as registering queuing events, calculating statistics, supporting dashboard management, and much more. Although there are a few standalone products (especially at smaller premises) QES is the centralized point for distributed products. A more technical description of QES will be done in Technical background section.

1.2.4 Touch printers

The touch printers are (hardware) devices consisting of a touch screen part and a printer part (see Figure 1.4). The touch screen is used to present a view for arriving clients and give the clients to interact by using the touch feature. Often the screen displays a number of buttons representing some services that a client is supposed to choose. It can either be that a client should register its arrival or just get a ticket for a queue to some workstation to be served. The printer part of a touch printer is used to print a piece of paper representing a queuing ticket, mainly with a number on it (placed in a virtual queue).

Figure 1.4 A touch printer

There are two models of touch printers; TP3xxx (Figure 1.4) and Vision (Figure 1.5). The mainly different is the design and hardware specifications.

The touch printers are the core part of the arrival step in the CFM process. They give the opportunity to direct clients to workstation that best matches their needs. This is done when clients are interacting with the touch screen. In the most simple way a number of services offered by current premises are listed there each service if linked to a workstation. But in more complex way there could be a number of choices to make on the screen, i.e. adding personal data like birthday number as input. Then software if used to calculate which workstation that would serve that client best. Furthermore the system can also send information to the serving workstation before the client arrives to it. This means that when that client gets in turn to be served the staff working at the workstation can get personal data presented to decreasing the number of questions needed to the client. Fewer questions to ask means that the serving time is decreased and overall queuing will be more effective. Some touch printers may have cameras taking a photo of the client that staff on the workstation uses for faster identification, not forcing the client to pick up any id cards.

The information can be stored for statistics used later by management parties. The main office can get an overview when there should be more workstation open and when there doesn't have to be so many. Also what sort of staff should be working when can be scheduled from the statistic data.

Finally a very powerful function is to show directed advertisement. I.e. when a client has gone through a flow of pages and waits for the ticket to be printed the touch screen can display some advertisement that is connected to the client's issue.

Figure 1.5 Vision

2 Background

QMatic has indications that there is a demand among their customers to being able to manage the content displayed on the touch printers by themselves. Therefore a study has been demanded, by QMatic, to investigate whether or not a tool can be built that satisfies the customer needs; that almost everyone, even non–experienced computer users, should be able to manage content displayed on the touch printers.

3 Problem

The content displayed on the touch printers is created programmatically, by QMatic engineers. The problem for QMatic is that updating has to be done in the same manner, makes it impossible for non-technicians to handle.

Although there are some possibilities to modify the content by using some sort of scripting it often introduces fails and bugs when customers are trying to modify the content. A very common "quick fix" in the real world is that customers are putting scotch over part of the screens to avoid that clients are pressing some screen areas not in use anymore.

The summarization of the problem is that staff, especially non-experience computer users, of customers buying the touch printers cannot change the content displayed.

4 Purpose

The purpose of this thesis is to investigate whether or not it is possible to build and integrate, into QES, an editor tool to be used by non-technical experienced end-users and that can handle all the functionalities demanded. If the problem could be solved it should be presented how to accomplish that. If it would be shown that the problem cannot be solved a result of the bottlenecks or other problems should be presented and discussed.

5 Scope

Within the scope of this thesis an investigation should show how the problem could be handled. The investigation should take in concern a suggestion of some prototype of a final tool. Following paragraphs should limit the scope:

- The usability of a possible tool should be good enough in measurement of some usability best practices together with feedback from the project customer, QMatic.
- Although prototyping would be a tool for the investigation to solving the problem, the output should not be a complete product but more an illustration of a suggestion.
- There should not be any complete market studies.
- There should not be any interaction studies.

6 Method

The method used to try to solve the problem will be to implement a prototype of a content management tool. The prototype should be a suggestion for a final product.

6.1 Development process

The development process will start by a pre-study part where a more detailed description of the domain will be analyzed of what a user should be able to perform by the tool.

The domain description will be an input to the second part which will be the requirement elicitation. In this part all use cases and requirements will be derived from the domain description.

After it has been analyzed what the user should be able to perform with the tool an analysis will be done to design how the tool should be used and how it should look like. Usage diagrams and GUI sketches will be done here matching the uses cases and the requirements from previous part. Also a domain model will be constructed.

After the domain has been analyzed and modeled, an initial system architecture and design will be done before starting any implementation. The step should how the tool should be technically built.

Finally the use cases will be detailed designed and implemented. The implementation will be divided into a number of iterations where each of the iterations will contain one or more use cases.

7 Technical background

All techniques important for the development process will here be briefly explained. For details please see references.

7.1 Java Enterprise Edition

Java Enterprise Edition (JEE) is a platform-independent, Java-centric environment for developing, building and deploying Web-based enterprise applications..The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multi-tiered, Web-based applications. JEE includes many API's from the Java Standard Edition (JSE). [5][6]

7.1.1 EJB3

Enterprise JavaBeans (EJB) technology is the server-side component architecture for Java EE. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology. [7]

7.1.2 Java Server Faces

Java Server Faces (JSF) technology establishes the standard for building server-side user interfaces. JSF is a request-driven MVC web framework based on component driven UI design model. [8]

JSF has managed beans, which provide the logic for initializing and controlling components for managing data across page requests (a single round trip between the client and server), user sessions, or the application as a whole. [9]

7.1.3 RichFaces

RichFaces is a component library for JSF and an advanced framework for easily integrating AJAX capabilities into business applications. [10]

7.2 QMatic Enterprise Suite

The QMatic Enterprise Suite (QES) is built on J2EE and EJB3. The system is divided into a number of modules. The system works as a centralized node for a lot of QMatic products; from it the most products can be administrated, i.e. generating statistical report, handling media content, and so on. I.e. all queuing processes pass this central system (i.e. clicking a button on a printer machine sends events to the system that registers and delegates commands further). The system is web-based taking us of JSF/RichFaces as front-end web-tire.

8 Vocabulary

Page flow
A page flow is a sequence of pages.

Page
A page represents what is displayed. A page will have a number of components placed in it.

Page cell
A page cell is an area of a page where the components are placed. It is the cell that holds the components and a page that holds a number of page cells.

Page template
A page template describes the look of a page, i.e. how many page cells it has and how they are divided into a number of rows and or columns.

Component
A component represents a piece of content. It is be a button, an image, or a piece of text. This is something an end-user can see on the screens.

Property
A property describes the look of a component. It could be i.e. color, size, or margin.

Service
A service is something that a branch offers (i.e. House loan, in a bank). These are defined in the system configurations of QES. Already exists and used in QES.

Branch
A branch represents premises, i.e. a bank office in Gothenburg, or an IKEA store in Stockholm. Already exists and used in QES.

Branch type
A branch type is a collection of branches divided into some logical way. Already exists and used in QES.

(Touch) Printer
A printer in QES represents a touch printer device. Already exists and used in QES.

9 Pre-study

9.1 Market analysis

Looking at the solution of the problem the QES has to be extended with software taking care of content management for the touch screens. This means that the software has to be web-based and well-integrated into QES. A quick look at the market hasn't shown that there exists a product that can cover this, instead an assumption has been done that the most effective way in form of time and money is to build the software.

The only question is whether existing techniques used in QES will be enough. However the goal is to take use of the existing techniques, but there can be decisions made to import other frameworks if it would be required.

9.2 Domain description

From previous section it has been determined that QES needs a sub system as complement that handles the content management on the touch printers. The tool should give users opportunity to create content in a simple way that can be added to the touch printers.

Below a more detailed domain description on how the tool should be will be presented. This can be seen as delivery demands from the project customer. The domain description will works as a direct input to retrieve use cases and requirements. Also concepts will be retrieved to later design the domain model. The concepts will be underlined and explained in next section.

A brief description of the domain

The tool should be an editor where users can create page flows. A page flow should consist of a number of pages that would be populated with different types* of components; i.e. images, texts, and buttons. Furthermore it should be able to customize the look of the components by changing different properties i.e. setting a color and size. The user should be able to link button components to navigate to another page in same page flow. Also the user should be able to link button components to services from the system configurations. It should be possible to be preview a created page flow and also publish to touch printers configured in the system configurations. The collection of services available for each page flow should depend on which branch the user chooses. The pages in a page flow should have a template consists of cells structured in some special way.

* (Motivation for these types: One could think to have cool flash animations and so on, but because of that we want a user of touch printers to not stay to long here).

10 Requirement elicitation

10.1 Use cases

These use cases are found out from the domain description in previous section. Furthermore each use case will be detail designed and implement inside iteration.

There are two main use cases; Create page flow and display (created) page flow. These use cases have a number of sub use cases, see Figure 10.1.

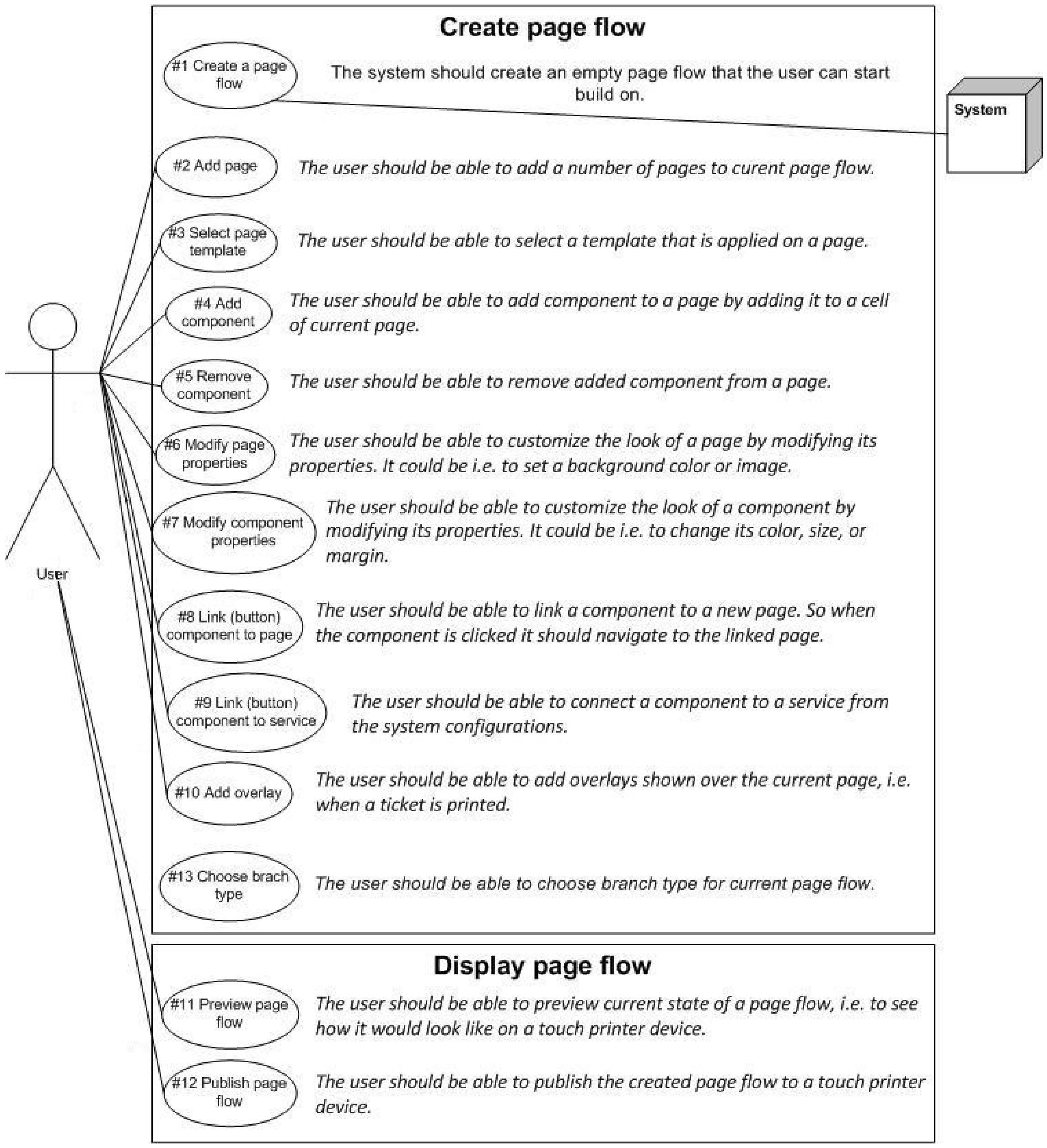

Create page flow
#1 Create a page flow
The system should create an empty page flow that the user can start build on.
System
#2 Add page
The user should be able to add a number of pages to curent page flow.
#3 Select page template
The user should be able to select a template that is applied on a page.
#4 Add component
The user should be able to add component to a page by adding it to a cell of current page.
#5 Remove component
The user should be able to remove added component from a page.
#6 Modify page properties
The user should be able to customize the look of a page by modifying its properties. It could be i.e. to set a background color or image.
#7 Modify component properties
The user should be able to customize the look of a component by modifying its properties. It could be i.e. to change its color, size, or margin.
User
#8 Link (button) component to page
The user should be able to link a component to a new page. So when the component is clicked it should navigate to the linked page.
#9 Link (button) component to service
The user should be able to connect a component to a service from the system configurations.
#10 Add overlay
The user should be able to add overlays shown over the current page, i.e. when a ticket is printed.
#13 Choose brach type
The user should be able to choose branch type for current page flow.
Display page flow
#11 Preview page flow
The user should be able to preview current state of a page flow, i.e. to see how it would look like on a touch printer device.
#12 Publish page flow
The user should be able to publish the created page flow to a touch printer device.

Figure 10.1 Overall use cases

10.2 Functional requirements

- User should be able to create a page flow
- User should be able to add pages to a page flow
- User should be able to add components to a page
- User should be able to modify a component
- User should be able to link a component to another page
- User should be able to add services from the system configurations
- User should be able to choose from with branch he wants the services
- User should be able to choose template for a page flow
- User should be able to preview created page flow
- User should be able to display created page flow on a touch printer

10.3 Non-functional requirements

- Because the tool should fill in the missing functionality in the existing platform it should also be integrated as a part of it, furthermore; the content management tool should follow the techniques and architecture from the enterprise platform, i.e. it should…
    - Be web-based
    - Integrated in QES, i.e.
        - Being able to communicate with other sub systems
        - Try to use as much as possible of current techniques and frameworks from QES
- The content management should not produce content with format only possible to present on specific displays.
- The tool should support different types of components

10.4 Usability requirements

- The tool should be able to be used be non-experienced computer users, so it should…
    - Be used in a clear and straightforward way
    - Be fast to learn to use

10.5 Excluded from requirements

- The way to present created content
- The way to push created content to printers
- System performance
- System security
- Support for different browsers

- The output will not be a complete product, i.e.;
    - Bugs are accepted
    - No exception handling included
    - No stable system expected
    - No expectation on clean code
- No expectation on testable code

11 Analysis

11.1 Usage – Workflow

“*People have a limited short-term memory – we can instantaneously remember about seven items of information. Therefore, if you present users with too much information at the same time, they may not be able to take it all in*”. [11]

The goal of the tool is to make it possible for the end-user to accomplish to create a page flow. With the usability requirements in mind the end-user should use simple and preferably an intermittently behavior when working with the tool. According to the 3-click rule [12] the number of clicks to accomplish each action will tried to be decreased as possible. Also, with directions from the citation above number of different selection will be decreased as much as possible.

To decrease number of information displayed at the same time the information should be divided at different pages; the end-user should perform a minor action at each step. Because some of the use cases above are dependent of other to be done before a wizard would be a good solution [13].

Looking at the use cases some of them will be performed once and some will be performed intermittently when creating a page flow (a publishing).

Done once	Done intermittently
Create a page flow	Add page
Publish page flow	Add component
Preview	Remove component
Select page template	Modify page properties
Select branch type	Modify component properties
	Link component to page
	Link component to service
	Add overlay

Table 11.1 Use cases grouped by how often they are used when end-user would create and publish a page flow.

All use cases in the right column in Table 11.1 can be done right after the "Create a page flow" use can has be done in the left column. Furthermore is dependent on other of the use case.

Use case dependencies					
Select page template Select branch type	Create a page flow	Add page	Modify page properties Add component	Modify component properties Link component to page Link component to service Remove component Add overlay	Preview Publish page flow

Table 11.2 Use cases dependencies. Use cases in each column are dependent of use cases in the nearest column to the left and so on.

11.1.1 Activity diagram

With Table 11.1 and 11.2 in mind the following activity diagram has been constructed describing in what order the end-user will perform each of the use case. There are one activity diagram describing the whole working flow and a separate to describe how to manage the content more detailed.

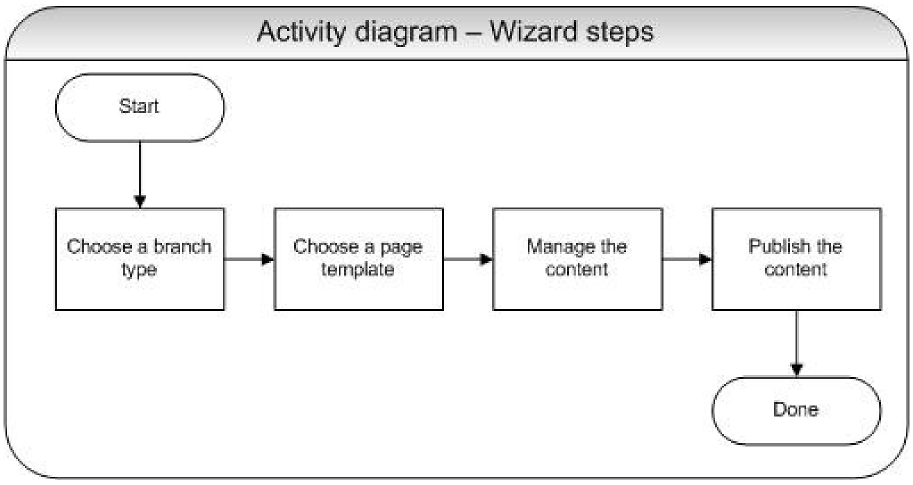


Figure 11.1 Activity diagram of the wizard steps the end-user should pass through.

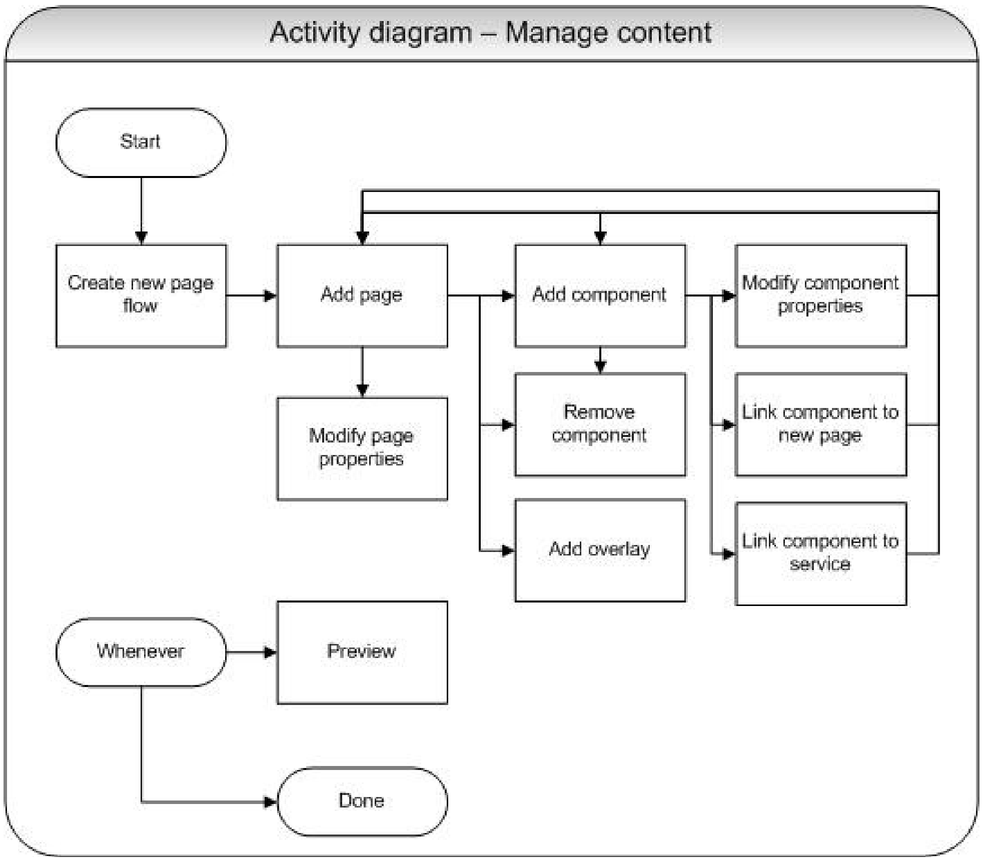


Figure 11.2Activity diagram of the content management part. Describes in what order the end-user should manage the content.

11.1.2 Preliminary GUI

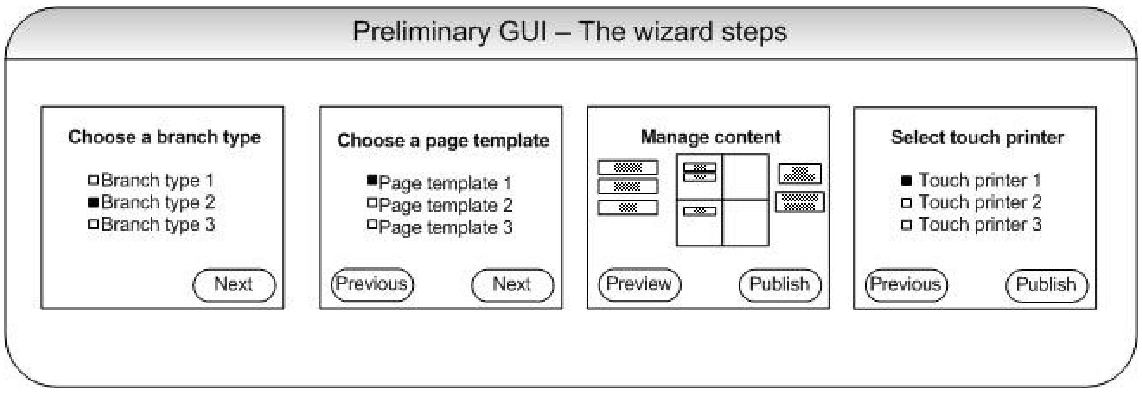


Figure 11.3 Preliminary GUI sketch of the wizard steps.

The wizard is divided into four steps which end-user should go through after done a choice at each of them.

Step 1: *Choose a branch type*
At this step the end-user gets all available branch types listed. The end-user selects one of the listed choices and continues to next step.

Step2: *Choose a page template*
The end-user gets all available page templates listed. The end-user selects one of the listed page templates and continues to next step.

Step3: *Manage content*
This step consists of an editor. At this step the end-user creates the page flow and populates it with pages and furthermore components. A more detailed sketch is presented in Figure 11.4.

Step4: *Select touch printer*
At this final step all available touch printers will be listed. The end-user should select one of them and choose to publish the created content at selected touch printer.

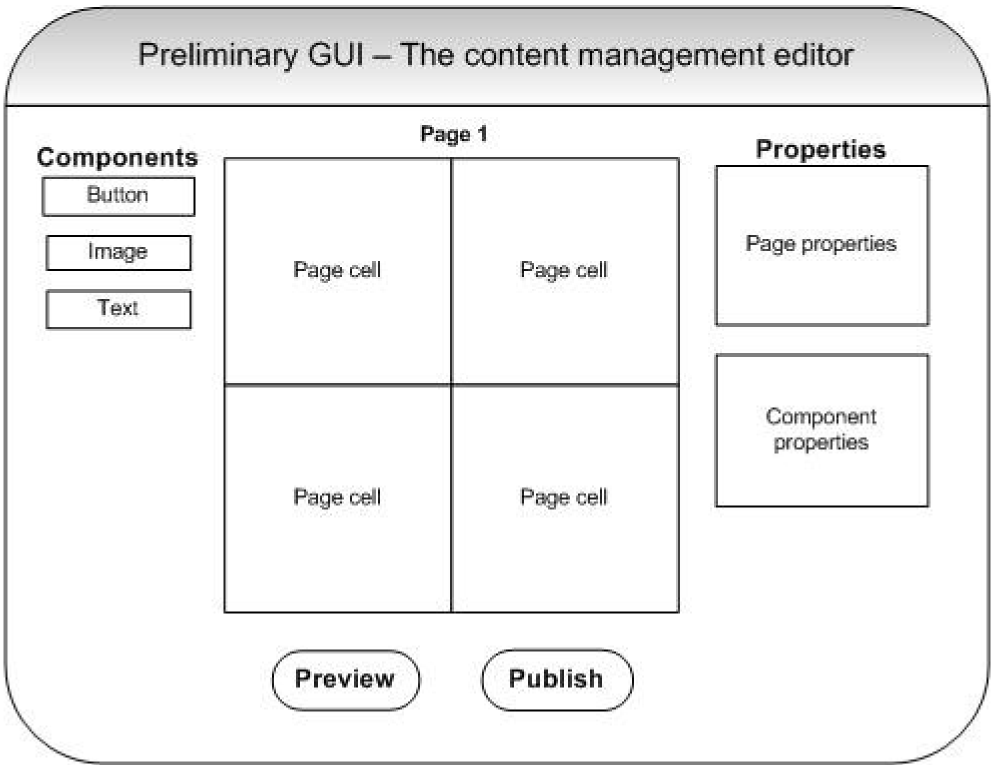


Figure 11.4 Preliminary GUI sketch of the content management editor.

Left column
Here a list of all component types should be listed. When the end-user wants to add one of them to the display area he drags a component from this list and drops in on the display area.

Center column
This is what is supposed to be displayed on a touch printer. The area here represents a page that would be divided into cells; depending of which template that the page (and other pages in same page flow) is using. In the cells the different types of component should be dropped.

Left column
The end-user should be able to modify each of the content added to the display area. When to so the end-user should get a couple of properties shown that he can modify and then save, then the content should be modified.

Mouse clicks
The end-user should be able to click on added content (right-, left-, or double click) to display the property parts. Double click: Go to next page, right click: show the menu, left click: show properties.

11.2 Domain model

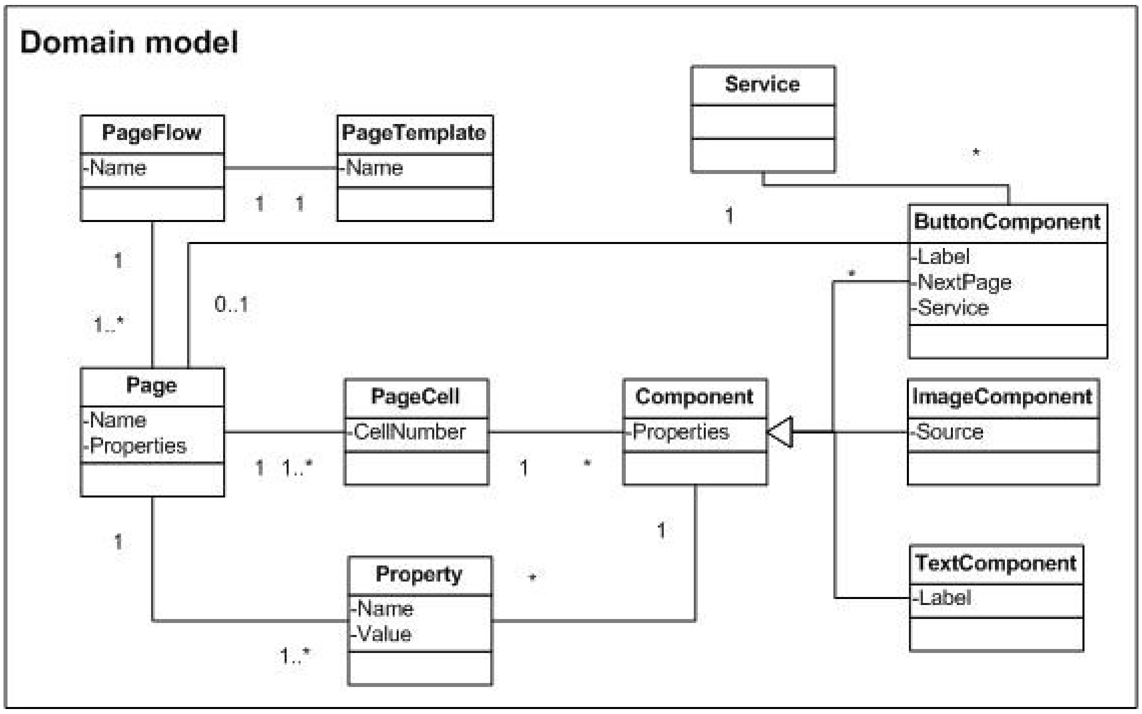


Figure 11.5 The domain model.

12 System architecture/High-level design

12.1 Server-Client architecture

As mentioned in introduction section QES is a central point communicating with other distributed products. For interaction with end-users QES uses a server-client architecture running the server part at an application server with web browser as client part. This will also be the case with the developed tool; the end-users will interact with a web browser when using the tool, and the web browser will in background interact with the server part.

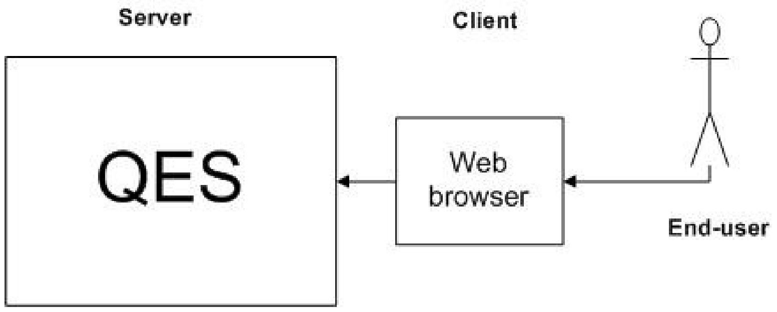


Figure 12.1 The client-server architecture.

12.2 Modularized architecture

QES consists of a number of modules, where each module consists of a Web module part and a corresponding EJB module part. To follow the architecture also the developed tool will be built in same manner. The tool will be a module called Touch Solution and divided at a Web module part and an EJB module part. The modules will have dependencies to the already existing core EJB module in QES to retrieve data from the system configurations (stored in database).

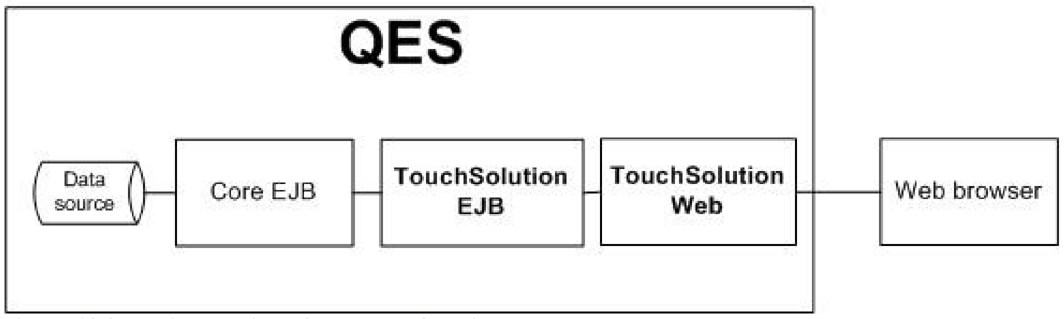


Figure 12.2 The integration of the TouchSolution module.

TouchSolution Web
This module will contain the web part of the touch solution part. I.e. all pages and managed beans (read about JSF for more understanding).

TouchSolution EJB
This module will contain the domain model. It is responsible to, in future, persist the objects.

Core EJB
This module is the core module in QES. Its responsibility is the communication with the database. To handle the persistence of the core domain objects.

12.3 Design

12.3.1 Holding the state

As shown in Figure 11.3 the end-user will go through a wizard making a choice at each step before continuing to the next. The first issue to consider is how the choices should be stored. The state of the whole wizard from previously steps has to be stored and available during the following steps.

A first design suggestion is to send request containing choices made in all previously steps. Although this is a straightforward way, there are unnecessary parameters to handle, at steps then are not related to. A more robust and maintenance-effective way would be to take use of some external framework for handle sequence of pages [14].

The solution made is instead to create a page sequence handler by adding a managed bean into the scope session. This bean, called PageSequenceBean, will hold the choices made at each step.

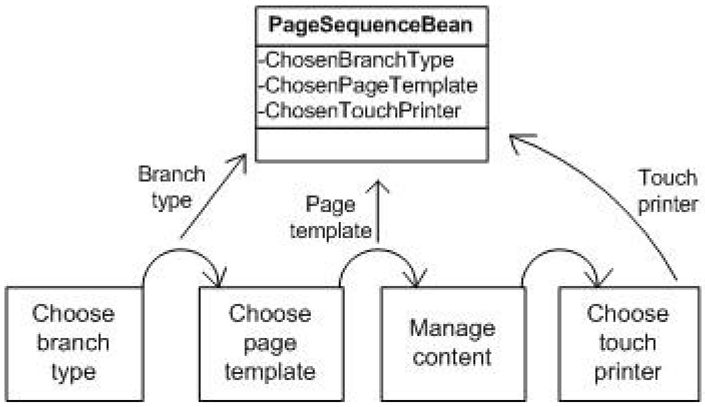


Figure 12.1Describing how the choices after each step in wizard will be stored in PageSequenceBean. Notice that the manage content step is more complex and will therefore analyzed more detailed later.

12.3.2 Client-server interaction

Second issue to solve is how the client and server would communicate with each other. When end-user are editing the content on the client-side, the browser, the changes needs to be sent to the server, so the server and the client have to be synchronized with each other.

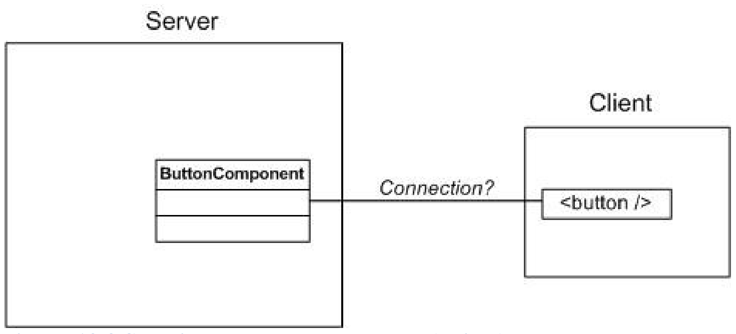


Figure 12.3 Showing a button component (object) on the server and a button-element on the client. How should they be synchronized?

The following is required:

- Changes on the client side have to be up-to-date on the server.
- Each Html-element on the client has to have a corresponding object on the server-side.

One solution could be to store all data in html-elements. The advantage is the performance but there is a disadvantage in growing complexity and robustness in holding the state, especially when switching between different pages with many components. A solution to decrease the complexity could be to just store selected page on the client and then use Ajax when switching selected page. Disadvantage is that a render engine has to be created on the client side.

A more robust solution is to take advantage of using RichFaces. Using managed beans the state can be held on the server side and then automatically update the client-side. This means that the server will hold the state of current page flow, including its pages and components.

How much data would it be in the memory?
Answer: An assumption of number of objects, in average:
1 page flow + 1 page template + 4 pages each with 3 page cell containing 1 component per page cell each having 4 properties.

Calculation: 1 + 1 + 4(3 + (3 * 1) + (3 * 4)) = 74 objects, which is no memory problem.

12.3.3 Content relation between editor and display mode

Another question is how to handle the content while it is in editing mode and when it is shown on the touch printers. In editing mode the editor-view has to be different in way so all components should be editable and so on, while when displaying the components on the touch printers they should be performing some action instead. The preliminary solution is to have different methods for generating code for current object for editing and presenting modes, see Figure 12.4.

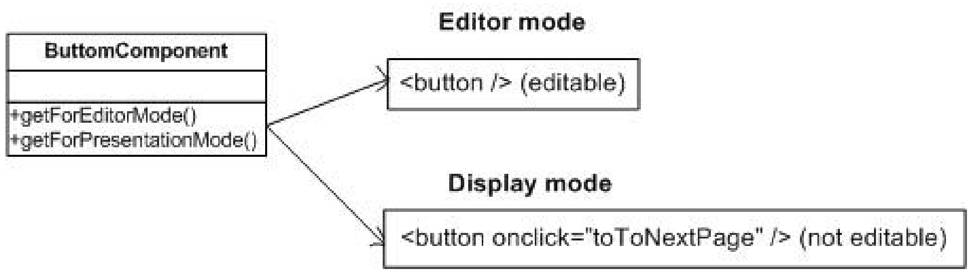


Figure 12.4 Showing how a button component should be represented differently depending on whether it is in the editor mode or the display mode.

The content created should be represented in an abstract way, object tree, so it later on, after easily persisted, can be presented in different ways. The presentation of the content on the touch printers will be discussed later.

12.3.4 Rendering

After the end-user has performed an action, i.e. drag-and-dropped a component to a page cell, and the information has been sent to the server, the editor-view has to be updated too. There are two suggested solution; either let the client-side, the web browser, take care about the updating of the editor-view by itself, or let the server send information to the client-side about what has to be updated, and how.

The first suggestion requires developing some developing engine on the client-side. A disadvantage is that when the client-side is updating itself it doesn't really know about what have happened with the information it has sent to the server.

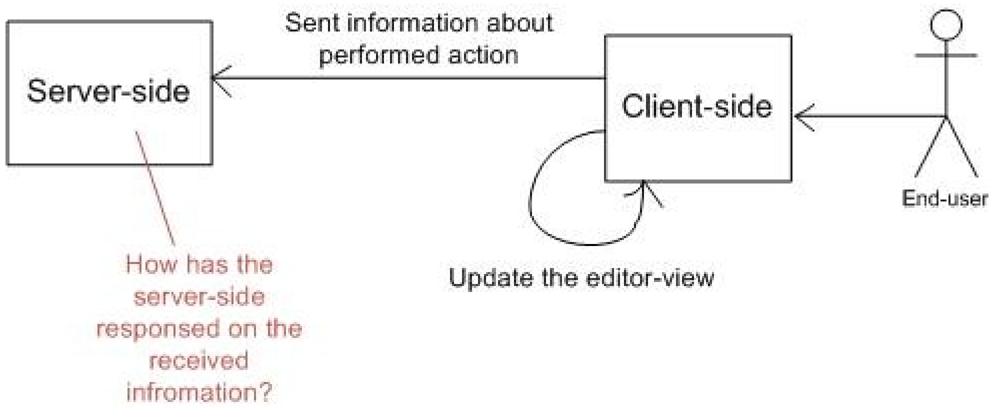

Server-side
Sent information about performed action
Client-side
End-user
Update the editor-view
How has the server-side responsed on the received information?

Figure 12.5 An overview of the information chain after the end-user has performed some action.

Instead the second suggestion will be realized; after the server-side has received the information from the client-side, and handled it, then it will send a response back and tell the client-side what to do; i.e. how to update its view.

The technique used to solve this will be to dynamically create JSF and RichFaces elements on the server-side, and finally the client-side will update pieces of its view depending on what action that has been performed.

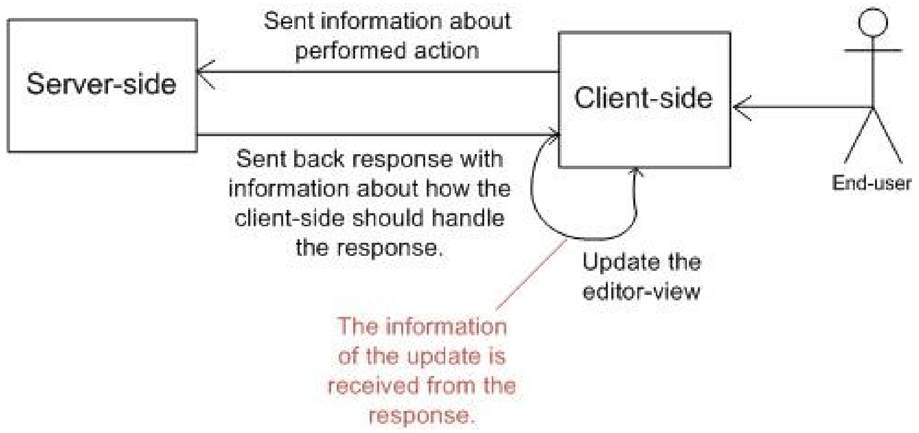

Server-side
Sent information about performed action
Client-side
End-user
Sent back response with information about how the client-side should handle the response.
Update the editor-view
The information of the update is received from the response.

Figure 12.6 Showing an overview of the solution when the end-user performs an action, how the information is exchanged between the server-side and the client-side.

13 Development

The development section will present the iterations of the development. During these iterations the use cases identified in the requirements elicitation section will be detailed designed and implemented. Each of the iterations will involve one or more of the use cases.

13.1 Iteration 1

This iteration will handle following use cases:

- Use case #3 Select page template
- Use case #13 Choose a branch type

13.1.1 Use case #13 Choose a branch type

The first choice the end-user has to do is to decide which branch type to create a page flow for. Branch types are retrieved from the system configurations in QES. Dependent of which branch type a branch belongs to tells which services the branch supports.

A managed bean, called BranchDataBean, will be used to retrieve the all available branch types and present them in the web browser. When arriving to this step the end-user will get a list of retrieved branch types where one of them has to be choose, in this case simply radio buttons.

After user has selected one of the listed branch types he navigates to the second step.

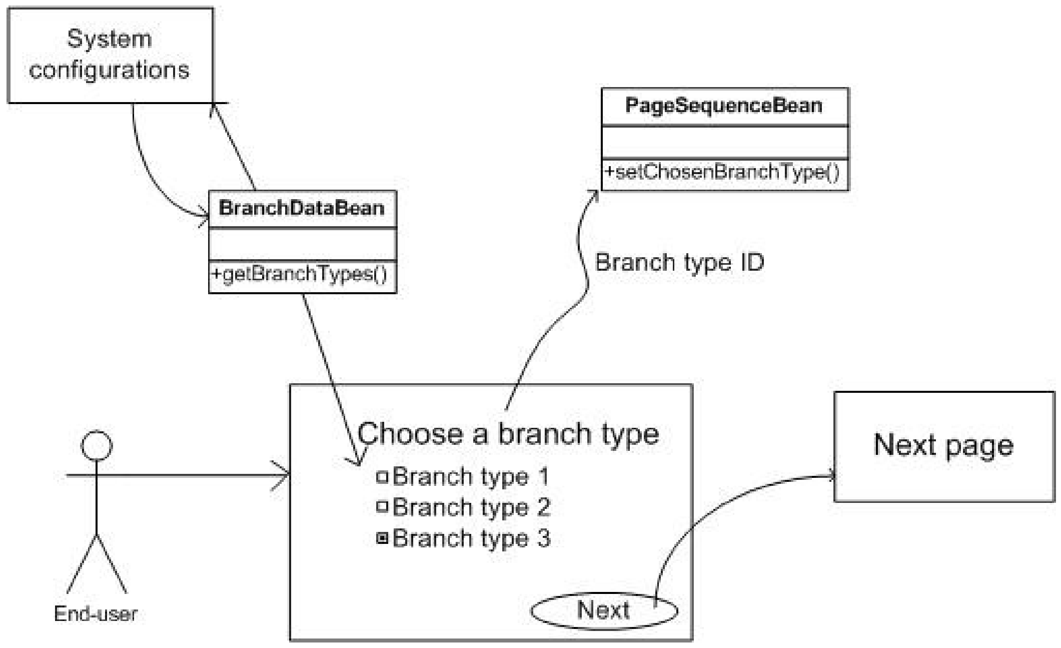


Figure 13.1 Overview of the process when end-user has arrived to the page to choose a branch types, and after a choice has been made and the next button pressed.

This use case has implemented the first step in the wizard, where the end-user chooses a branch type.

13.1.2 Use case #3 Select page template

The second step in the wizard is to let the user choose which page template that should be used for current page flow. A template is simple a page structure divided into rows and columns, which shows areas where components can be added. Because of that the content management tool should support different types of touch screens, there should be possible to have different types of templates. The decision for this thesis is to use same template on all pages in a page flow, this is not because of any usability but because of that this is an investigation and not a final product.

The creation of a template is made in a programmatically way (further template editors are to taken in concern and are out of the scope). The available templates will be listed so the user can select the one he wants and then continue to the next step in the wizard.

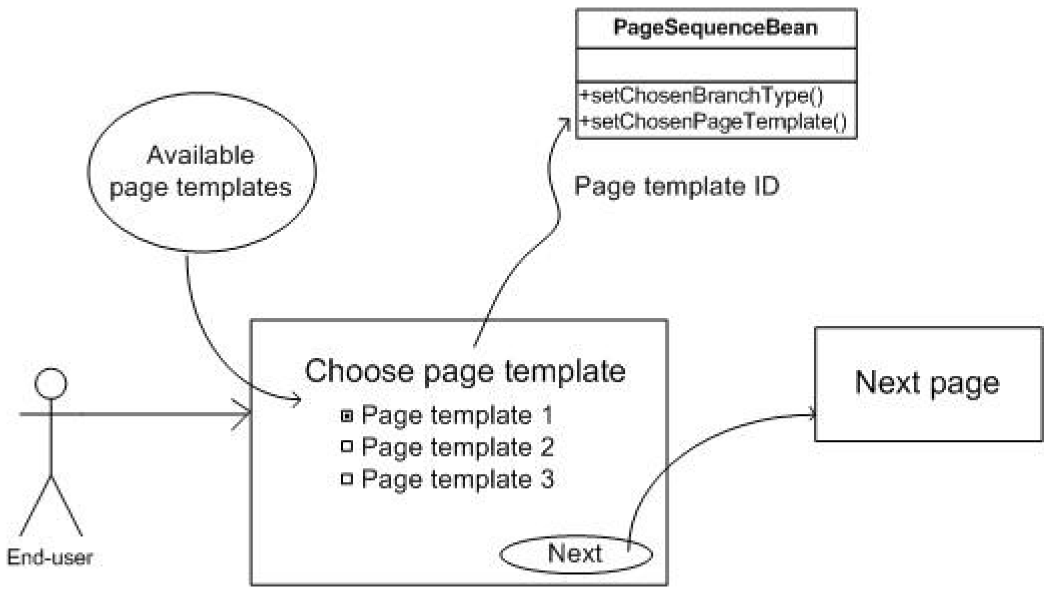


Figure 13.2 Overview of the process when end-user arrivies to the second step in the wizard, and chooses page template.

This use case has implemented the second step in the wizard, where end-user chooses one of the available page templates.

13.2 Iteration 2

This iteration will handle following use cases:

- Use case #1 Create a page flow
- Use case #2 Add pages to a page flow

13.2.1 Use case #1 Create a page flow

This use case is completely handled by the system. When end-user enters the editor the system does following:

- Creates a page flow
- Creates a page template (of type chosen previously during the wizard) and assigns it to the page flow
- Creates a page and assigns it as the start page of the page flow
- Populates the page with a number of page cells. The quantity depends of type of page template.
- Creates a managed bean, called PageFlowStateBean; Assigns it a reference to the page flow and add the bean to the session.

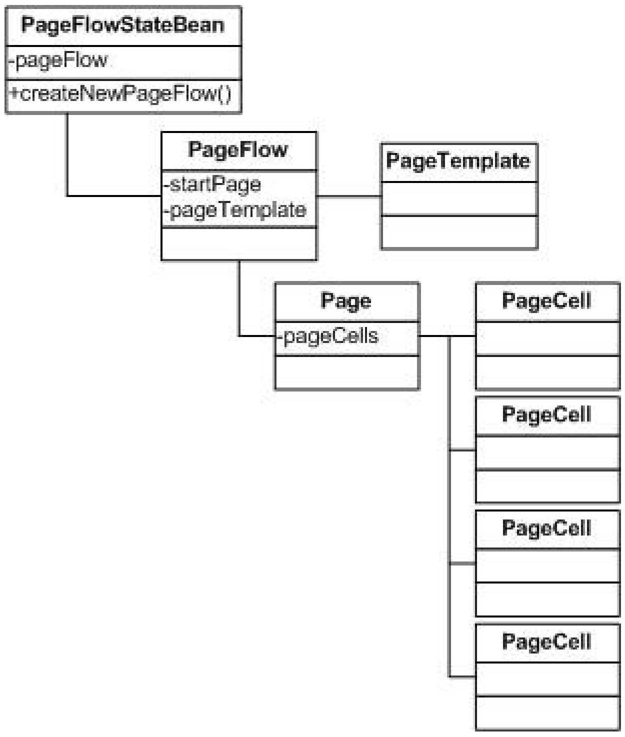


Figure 13.3 Overview of the created page flow populated with a generated start page and its page cells. The PageFlowStateBean is a managed bean placed in the session used to access the page flow tree.

On the client-side a web page will be presented for the user representing the current state of the page flow. The page flow itself is never represented in a view; instead the first page will be shown divided into its page cells.

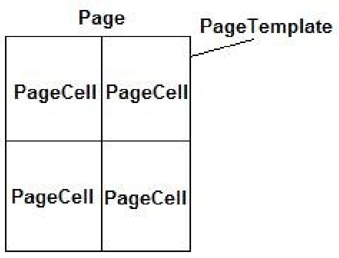


Figure 13.4 An overview of the GUI representing the page flow tree from Figure 13.3

This use case has implemented the generation of a page flow; prepared with a page and page cells that end-user can start working with.

13.2.2 Use case #2 Add pages to page flow

The page flow will have a list of all its pages. Each time a new page is created it should both have a reference to its page flow but also a reference of that page should be stored in the list of pages in the page flow. Also each time a new page is created it gets a couple of page cells generated (dependent of the page template). A page is never containing components but instead the components are stored in its page cells.

A page never refers to any other page. Instead there will always be a button component that refers to the following page, which it navigates to.

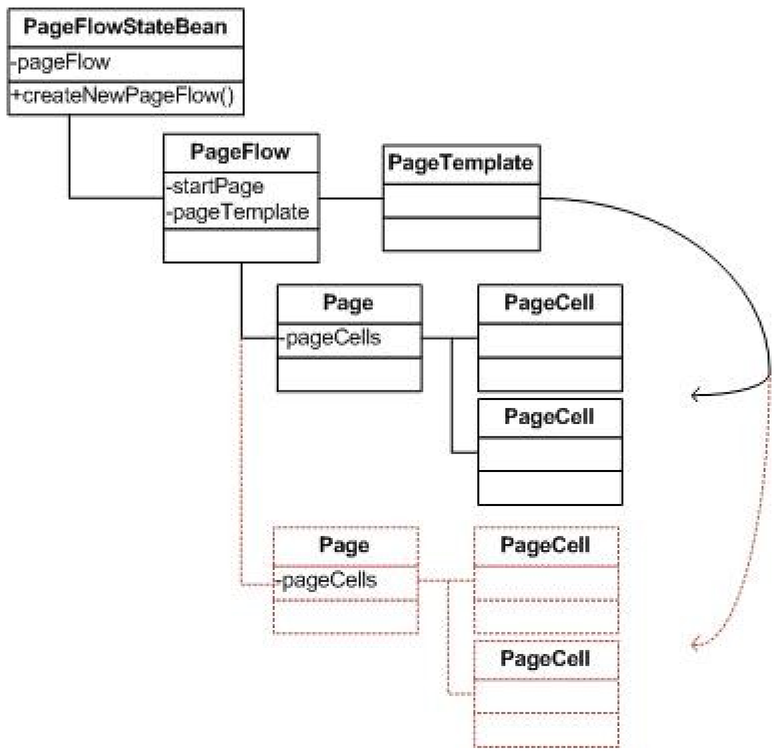


Figure 13.5 Showing how a new page is added the page flow. The tree is extended with a new page and two page cells; because all pages follow same page template, they will have same number of page cells.

This use case has implemented support for adding new pages to the page flow.

13.3 Iteration 3

This iteration will handle following use cases:

- Use case #4 Add components
- Use case #5 Delete components

13.3.1 Use case #4 Add components

As decided earlier components will be added by using drag-and-drop. Looking at RichFaces library it is possible to add drag-and-drop support on elements by adding a dragSupport or dropSupport element as child element. When changing the state of the view the page flow tree on the server should be updated.

When the user drops a component into a cell of the current page the following will happen:

- The server state has to be synchronized with the client:
    - The client has to send the type of component dropped
    - The client has to send the cell of current page the was dropped to
- The client-side view will be updated with the new state

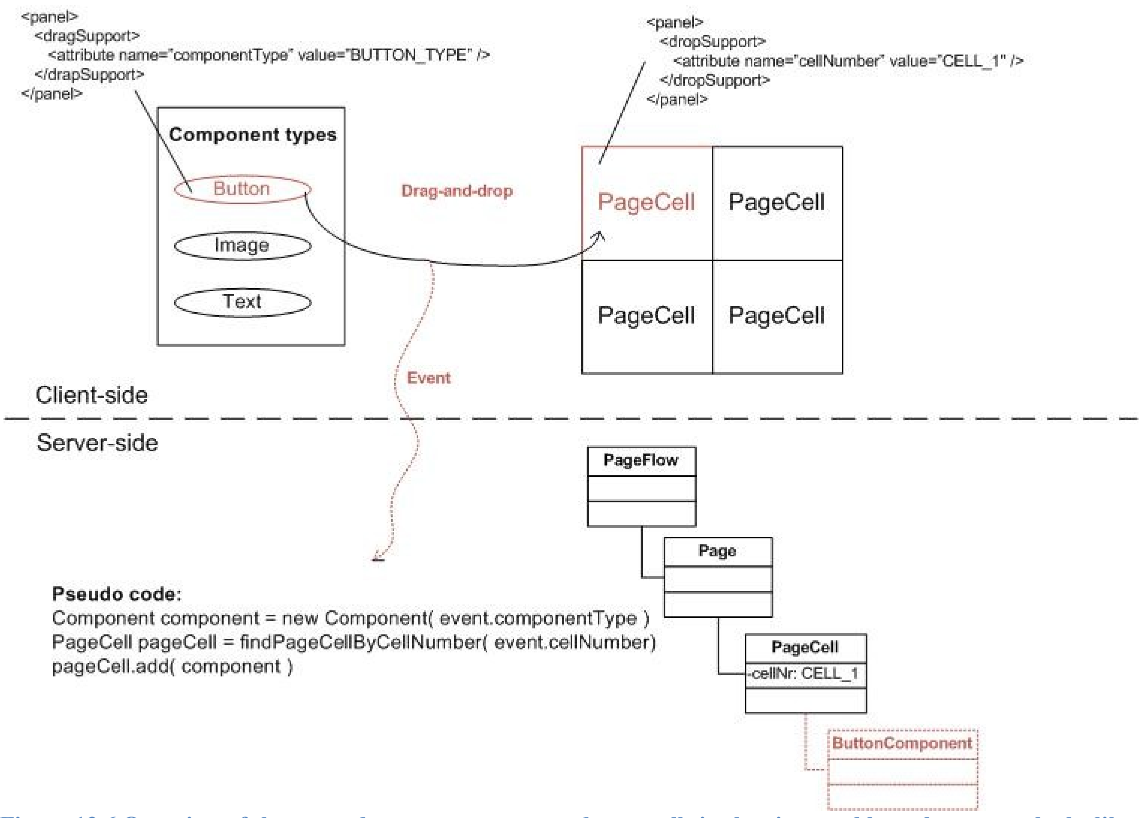
Before starting to develop the drag-and-drop support the view has to be created; Available component types have to be listed and the drag- and drop-support to be added.

The component types will be listed as panel elements adding dragSupport element as a child in each of them. Furthermore the component type has to be described. This will be done by adding an attribute with the component type to the dragSupport element; this will later make it possible to identify the dragged component type.

Also each page cell (panel element) has to be added a dropSupport element as child. Furthermore each page cell will be assigned an attribute with a number unique for each page cell of current page.

After the end-user has dropped a component type into a page cell, the server-side should be notified. This is done by adding an action listener to the drag- or drop-element (dependent of data needed, see RichFaces reference for more information). The invoked server part will be a managed bean, called RenderBean. When this bean is invoked it will retrieve data from the event and perform following actions:

- Create a new component with type retrieved from the event
- Find the cell with number retrieved from the event
- Add the component to the cell



```
<panel>
  <dragSupport>
    <attribute name="componentType" value="BUTTON_TYPE" />
  </drapSupport>
</panel>
```

```
<panel>
  <dropSupport>
    <attribute name="cellNumber" value="CELL_1" />
  </dropSupport>
</panel>
```

Pseudo code:

```
Component component = new Component( event.componentType )
PageCell pageCell = findPageCellByCellNumber( event.cellNumber)
pageCell.add( component )
```

Figure 13.6 Overview of the created component types and page cells in the view, and how the process looks like after the end-user has dropped a component type into a page cell.

After the server state has been updated the final step is to update the client view. A first possible solution is to send data in Json or xml format and serialize/deserialize the component on client-size, using JavaScript to append html code to cells. A disadvantage of this solution is that a rendering engine has to be created on the client-side. A better solution is to create the elements on the client-side dynamically. This means that java objects corresponding html elements are created on the server and then appended to the client view.

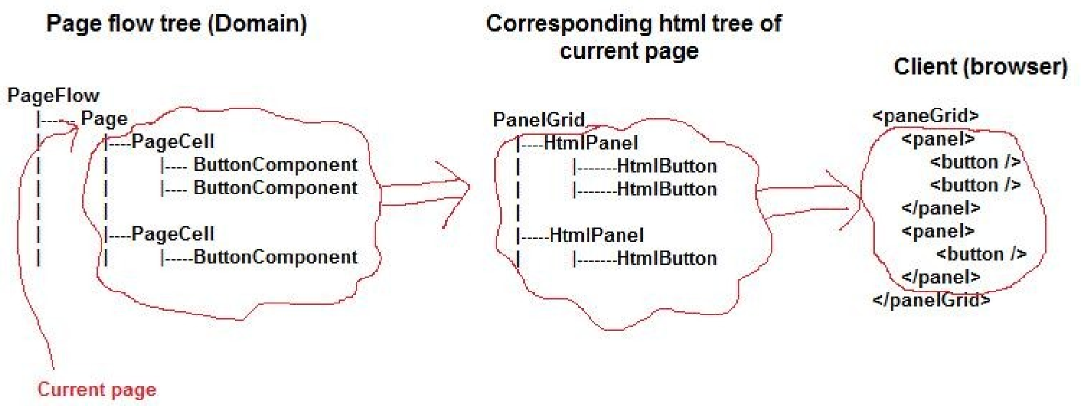


Figure 13.7 Showing how the page flow will be (manually) tranformed to an html object tree, that further automatically is transformed to html element on the client-side.

To accomplish the chain above the page cell panels in the view has to be bound to the RenderBean. This is done by adding bind property to the panel elements. Each page cell will then be bound to corresponding html object in the RenderBean (for detailed information about binding elements to managed bean please the RichFaces reference).

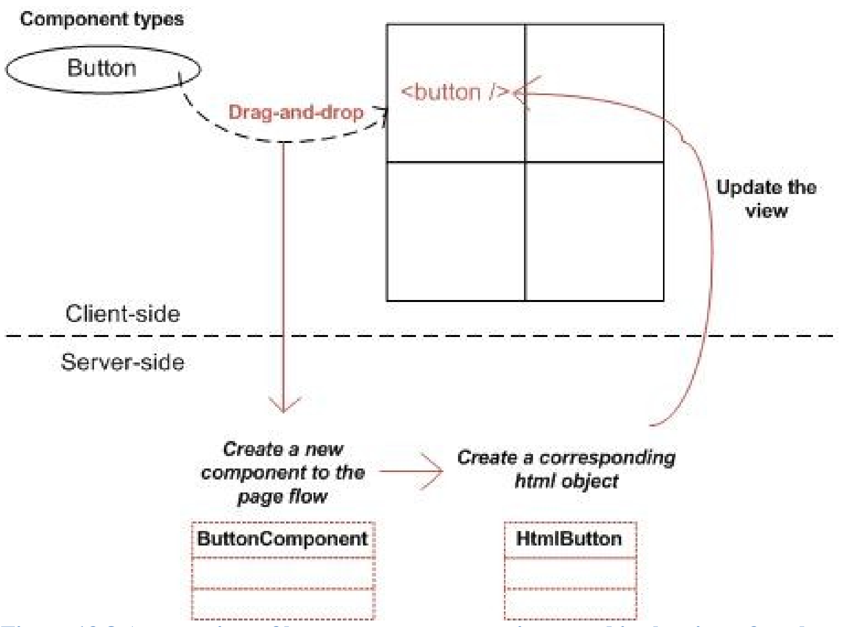


Figure 13.8 An overview of how a new component it created in the view after the server state has been updated.

This use case has implemented the support for end-user to add components by drag-and-drop arbitrary component type into a page cell.

13.3.2 Use case #5: Remove component

When adding a component in previous step an id unique (at least) within the component's page cell has to be assigned to the component. When user chooses to remove the component it has to be removed from the page flow tree and the html tree. After that the state is up-to-date and the view can be updated on the client-side.

To remove a component a RichFaces content menu is dynamically created for each component. In this menu a remove item is added invoking a method in RenderBean. For more detailed information about content menu in RichFaces please see RichFaces reference.

Figure 13.9 A context menu in RichFaces. See RichFaces documentation for details.

When removing an already added component from a page cell following scenario is performed:

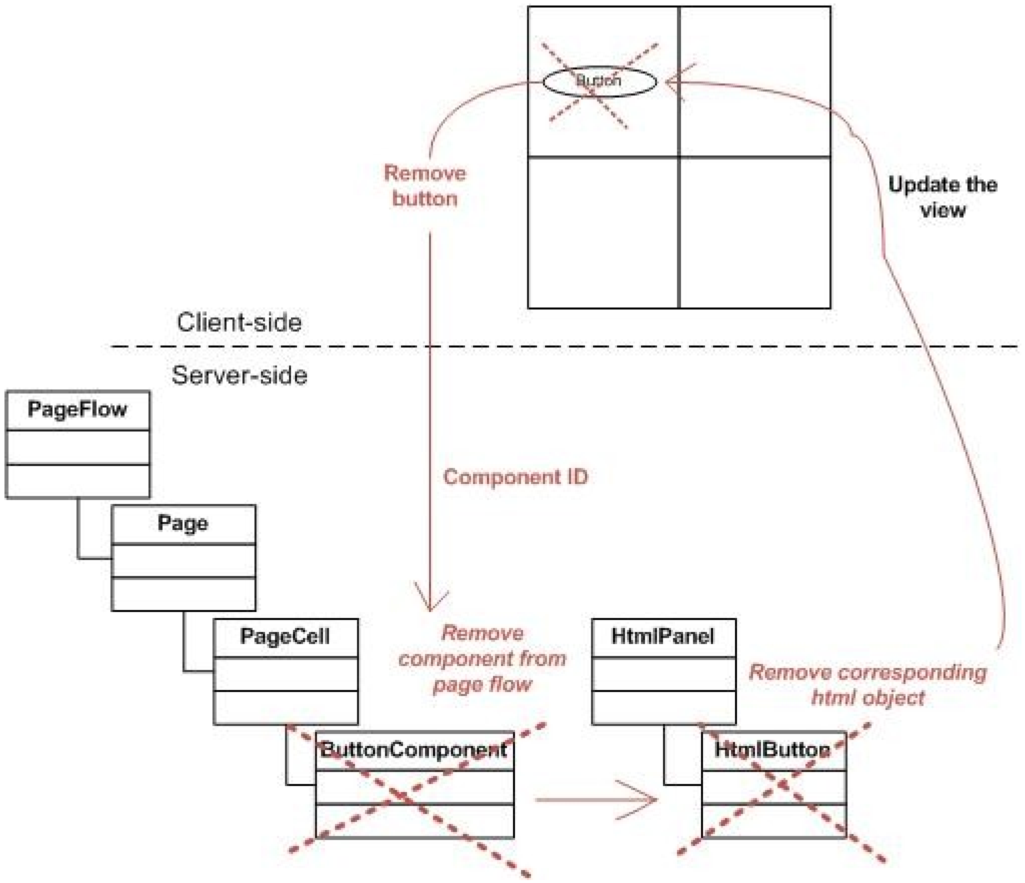


Figure 13.10 Overview of the process after the end-user has chosen to remove a component.

This use case has implemented functionality for end-user to remove added components from the page cells.

13.4 Iteration 4

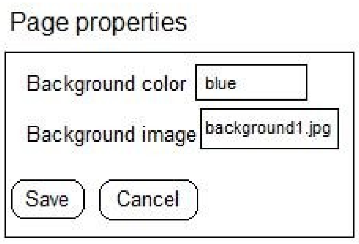
This iteration will handle following use cases:

- Use case #6 Modify page properties
- Use case #7 Modify component properties

This iteration implements the customization of the page and its components. The end-user should be able to change some properties and then the view should be updated with respect to the properties. The types of properties developed are just arbitrary to illustrate how the customization could be implemented.

13.4.1 Use case #6 Modify page properties

To customize a page the end-user will be able to change its background color or adding a background image. The end-user will be able to choose within a property box for current page. The property box should be updated when switching page.


Page properties

Background color: blue

Background image: background1.jpg

Figure 13.11 The picture shows the look of the box for the page properties. The user can change the background by choosing a color or image.

When the end-user has changed the properties and chosen to save the changes the view will be updated; i.e. the background of current page will immediately be updated in the view, but also stored in the page flow tree on the server.

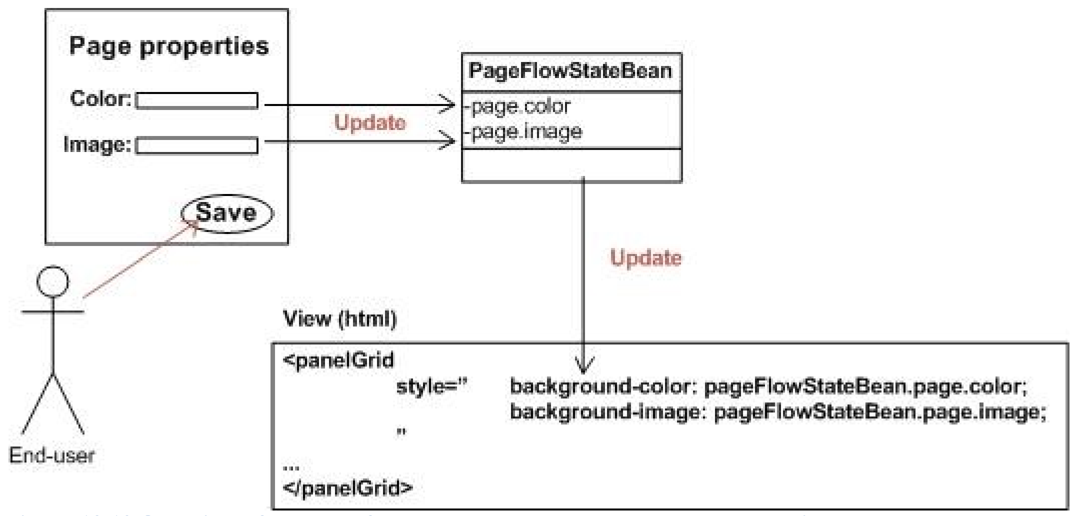
To accomplish that the following will be done:

- The end-user modifies the properties and chooses to save
- The property data is sent to the server that updates current page in the page flow
- The view is updated

The first thing to do is to add the properties to the page. In this case it will be simply done as variables, one for the background color and one for the background image.

In the (editor) view the panel grid element that is representing the page has to be added style attribute with the values of the page's background color and background image.

To set the properties to the page the property box in the view (see Figure 13.11) has to be linked to the page. This will be done in a simple way through the PageFlowStateBean which has a reference to the page. The page's setter and getter methods will be assigned to the properties, so when the view is updated then the page will get its own properties.



```
<panelGrid
          style="      background-color: pageFlowStateBean.page.color;
                       background-image: pageFlowStateBean.page.image;
          "
...
</panelGrid>
```

Figure 13.12 Overview of process after the end-user has changes page properties.

This use case has implemented functionality for the end-user to customize the look of a page.

13.4.2 Use case #7 Modify component properties

There are different types of component which can be modified in different ways.

Button component

The button component is supposed to represent a button on the screen and have the feature that a click on it navigates to another of the pages in the page flows. The modifications that should be supported is width and height of the button, background color, font color, and margin.

Image component

An image component represents an image on the screen; it is simple an image container. The modifications should be width and height of the image, margin, and a reference to its image file.

Text component

The text component type represents a text on the screen. The modification should be the font size, the font color, and margin.

The property entity

Because components could have a lot of different properties they will have a collection of property entities. This should also be the case for a page but because of a page had so few properties that solution was just enough for this thesis.

The property entity has a property name, a property value and an optional property suffix. On the client-side these are represented in form of cascading style sheet (CSS).

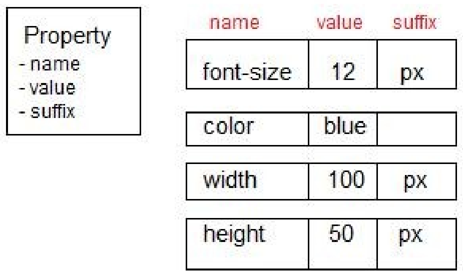


Figure 13.13 The structure of a property.

With difference from the page property box in the view the component properties should be shown when a component is selected. Because that different component types have different properties there should be a view for each component type. Initially each component property view should be hidden. When the user chooses to modify properties for a component the property box should appear in the view for that component type.

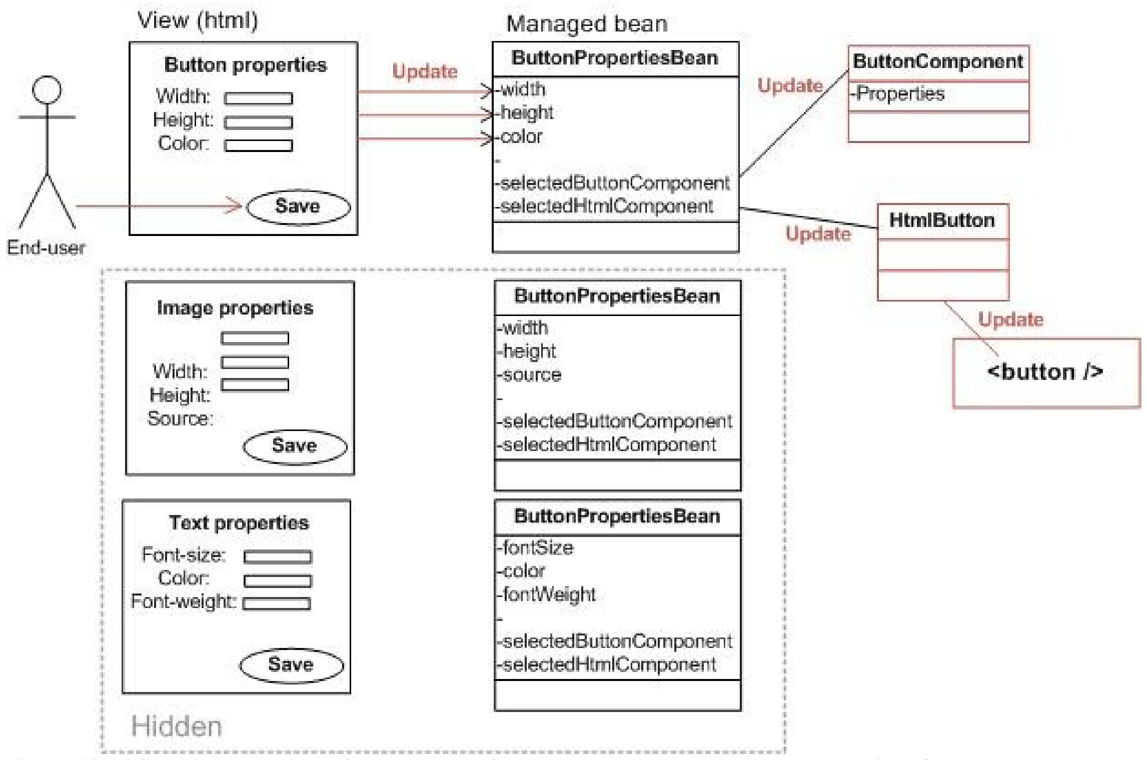


Figure 13.14 Showing overview of the process after the end-user has changed properties of a button component.

This use case has implemented the functionality for end-user to change properties for components.

13.5 Iteration 5

This iteration will handle following use cases:

- Use case #8 Link components to next page
- Use case #9 Connect components to services

13.5.1 Use case #8 Link components to next page

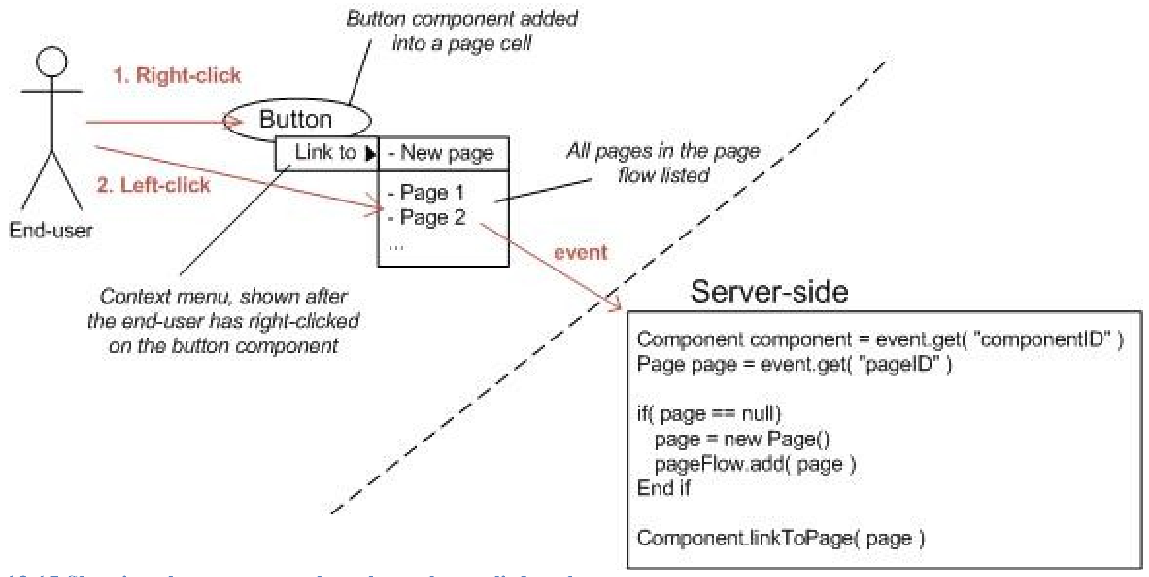
The button component is supposed to be able to navigate the client finally using it on the touch printers to another page in the page flow. Following has to be done to link a button component to a page:

- The end-user chooses in the editor-view to link a button component to one of the existing pages in the page flow, or to a new page
- The information is sent to the server-side that identifies the component and which page it should link to
- If it is an existing page the component gets that page assigned as the linked page, if it isn't an existing page then a new page is created and added to the page flow and the component is then linked to that new page
- The client-side gets updated

As for some previous use cases the end-user will use the context menu after right-clicked on a components to get a list of all pages in the page flow. The link to a page will be a menu item group.

The menu group will be created and append to each button component when at same time the component it created. There will be a condition for linking to a page; the component type has to be a button component. First a "new page" item will be created that is bound to a method on the server. Second a list of existing pages has to be added; also they will be bound to a method. The item for the "new page" has to contain an attribute with the component id. The items for existing pages have, besides to hold id to its component, also hold an id to the page. Because of that, as with components, each page has to have an id unique for a page in current page flow.

When the end-user clicks to link a button component to a page the server-side is invoked. The component is identified and linked to the page. The difference between a new page and an existing page is that a new is created and appended to the page flow while an existing will be found from the page flow and the linked to. The end-user can whenever change to link a component to another page.


Button component added into a page cell
1. Right-click
Button
Link to
- New page
- Page 1
- Page 2
...
End-user
2. Left-click
All pages in the page flow listed
event
Context menu, shown after the end-user has right-clicked on the button component
Server-side

```
Component component = event.get( "componentID" )
Page page = event.get( "pageID" )

if( page == null)
   page = new Page()
   pageFlow.add( page )
End if

Component.linkToPage( page )
```

13.15 Showing the processes when the end-user links a button component to a page.

This use case has implemented support for the end-user to link button components to a page in the page flow.

13.5.2 Sub-task: Switching among pages

At this point there are a couple of pages in the page flow. What has to be decided, and implemented, is how the end-user should switch between the pages in the editor.

First decision to make is whether the pages should be passed through a sequence as a wizard or if the user should be able to from each page navigate to all other pages. To make the content management more flexible and efficient the end-user will be able to switch between the pages in its own manner.

Adding to vocabulary

Current page
This means the page that the end-user sees, in the editor, or on the touch printer at the moment. When the end-user chooses to switch to another page in the page flow the current page is then switched.

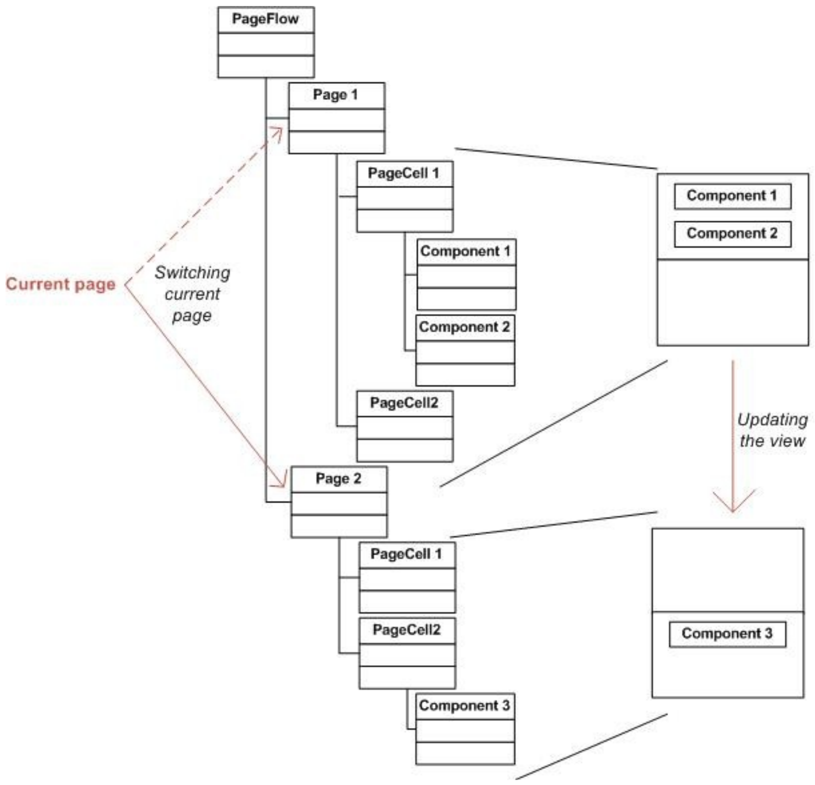
The following process will be performed:

- All available pages in the page flow should be listed and clickable for the end-user in the editor-view
- When the user clicks on a page in the list
    - Current view should be cleared
    - The new page should be pointed as current page
    - The view should be populated with the data of the new page

To accomplish that first the server-side has to be invoked. To update the view the following will be done:

- The html objects corresponding to the element on the client-side has to be updated for the new page
- The client-side has to be updated

To update with the new pages content the page has to be found. This is done by adding an id to each page that is unique within its page flow. So when the client-side invokes the server-side the id has to be sent to the server-side so correct page could be found. In the PageFlowStateBean that page will be referenced as the current page, so the server-side has the knowledge of which page new components should be added to.

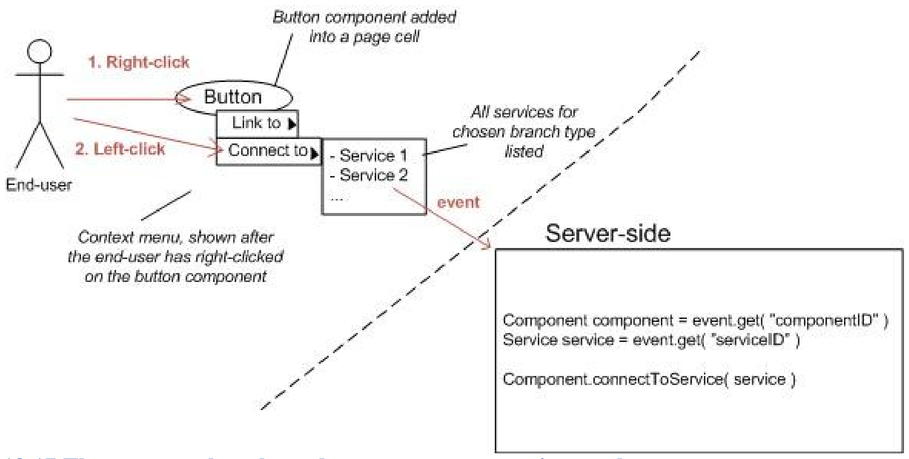


13.16 Showing the (behind) process when the end-user is switching between the pages in the page flow, in the editor.

At this subtask the implementation of switching between the pages in the editor-mode was implemented.

13.5.3 Use case #9 Connect components to service

To connect a component to a service data from the system configurations has to be retrieved. The first suggested solution of this issue is to append all services (that matches the chosen branch type) to the context menu of a button. The end-users would then, in same way as when linking to a page, right-click on a button component and choose which service it should be connected to. The process is then pretty same as when linking page to a button component.

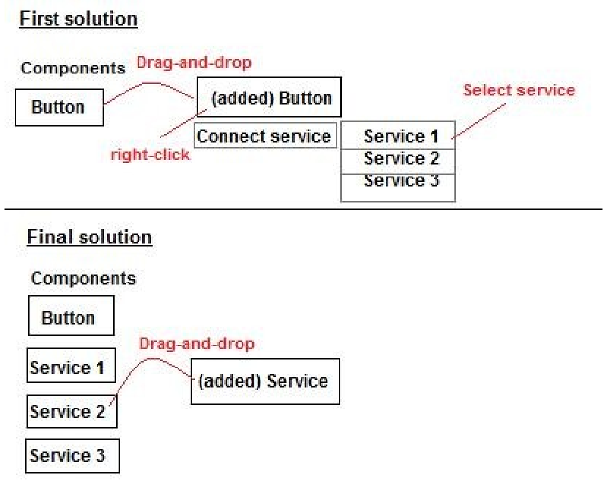

Button component added into a page cell
1. Right-click
Button
Link to
2. Left-click
Connect to
- Service 1
- Service 2
...
End-user
All services for chosen branch type listed
event
Context menu, shown after the end-user has right-clicked on the button component
Server-side
Component component = event.get("componentID")
Service service = event.get("serviceID")
Component.connectToService(service)

13.17 The process when the end-user connects a service to a button component.

With this solution the button component is actually of two different types; either it is a navigation button that navigates to another page, or it is a service button that performs some service.

To make the activity more user-friendly the button component will be split into two different components. This means that choosing to add a component of type button the user can only choose a page the button component would navigate to.

With the services there is instead a component of type service component. What will do with these is that they will all be listed in the view, each already connected to a service. So when user adds a service component no configuration has to be done; it is already connected to the service. With this solution instead both the design is better in form of that the components are split to perform a specific task. But also the solution makes it more user-friendly because of to accomplish the use case requires less clicks of the user, which is a 3-clicks rule best practice [12].



13.18 The difference between the two suggested solutions.

This use case has implemented the functionality for end-users to add service components connected to services from the system configurations.

13.6 Iteration 6

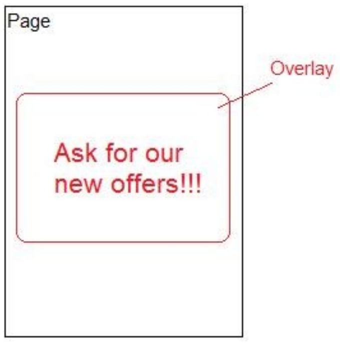
This iteration will handle following use cases:

- Use case #12 Add overlays

13.6.1 Use case #12 Add overlays

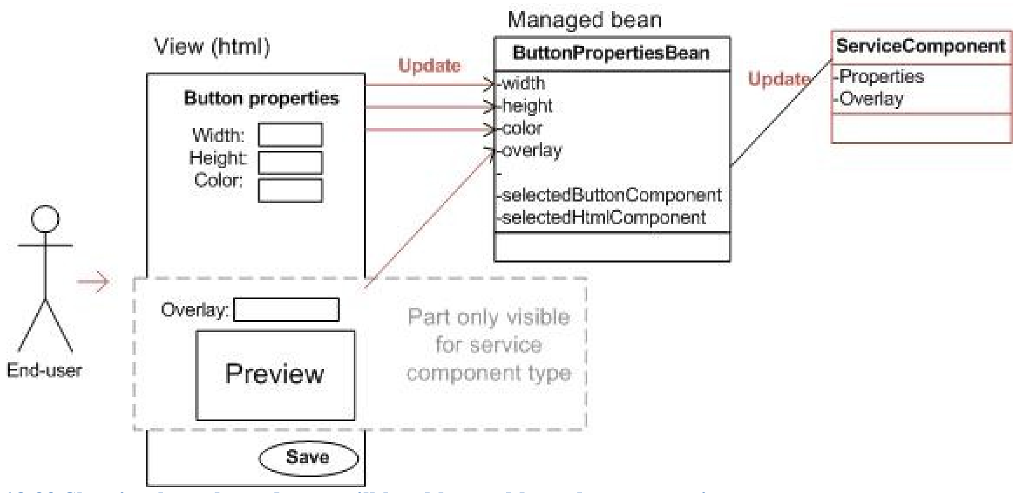
As described in the introduction sections overlays can be a powerful tool to showing targeted advertisement. The time that a client waits for a ticket to be printed some advertisement could be displayed on the screen. This editor will support end-users to add this ability. The overlays should be connected to service components.

The first question to answer is how the overlay should be done technically. Although the developed tool will have functions that cover editing of pages with components there will not be any editing mode for overlays. Instead an overlay will be a predefined image; this would also be most simple for a user; to have already predefined advertisement overlays when creating a page flow.



13.19 An overlay display over the page.

The overlays will be listed as images in view of the property box for button components, although it will only be visible when modifying properties for service components. The end-user will be able to choose an overlay from the list, preview it in a miniature box, and finally save the properties.



13.20 Showing how the end-user will be able to add overlay to a service component.

This use case has implements functionality for end-users to add overlays.

13.7 Iteration 7

This iteration will handle following use cases:

- Use case #13 Preview a page flow

13.7.1 Use case #10 Preview a page flow

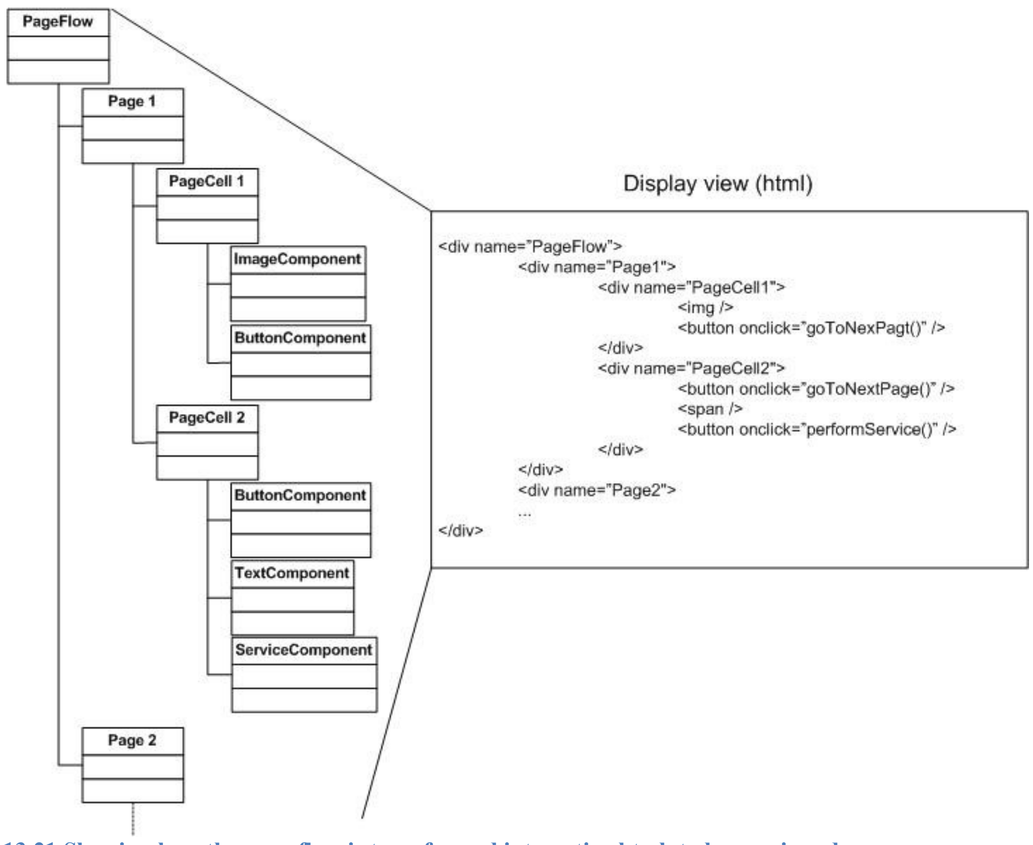
This use case gives the user opportunity to see how (the current state of) created page flow would look like in a presentation mode (on a touch printer). Because the created page flow is stored as an object graph in the memory (would be retrieved from a database in a final product) the page flow can be presented in many different ways. There are two possible choices:

- Create the presentation using JSF/RichFaces-tag library
- Create the presentation with native HTML and JavaScript

With the first choice there is a more powerful way to accomplish that. This was also done. But later on, because wanting to display the content on a touch printer, the second choice was chosen. This is because the first choice requires more from the hardware. For the small touch printers it is not possible to use that choice. So to give a hint of further solution (for a final product) the preview will be made using the second choice.

- The page flow object will be iterated and all pages will be created represented by native HTML
- Each column of each page will be iterated and components created represented by native HTML
- For button components JavaScript will be used to switch currently displayed page
- For service components later some Ajax-solution has to be made

The previewed page flow will be one simple generated HTML-page with all pages listed as parts under each other. All pages will be hidden but not the first one. Later on when navigating to a new page then that page will be shown and the previous one will be hidden. In this ways always the current page will be shown on top of the HTML-page.



```
<div name="PageFlow">
        <div name="Page1">
                <div name="PageCell1">
                        <img />
                        <button onclick="goToNexPagt()" />
                </div>
                <div name="PageCell2">
                        <button onclick="goToNextPage()" />
                        <span />
                        <button onclick="performService()" />
                </div>
        </div>
        <div name="Page2">
        ...
</div>
```

13.21 Showing how the page flow is transformed into native html, to be previewed.

This use case has implemented the functionality for end-users to preview created page flow.

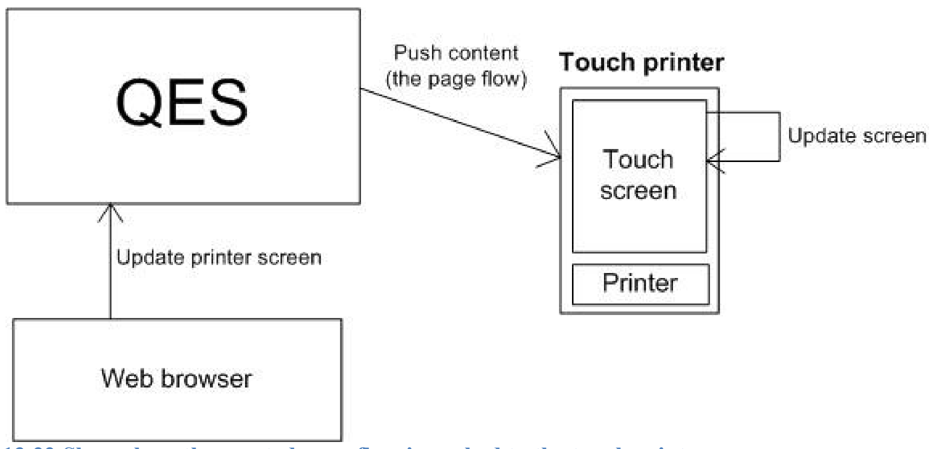
13.8 Iteration 8

This iteration will handle following use cases:

- Use case #11 Publish page flow

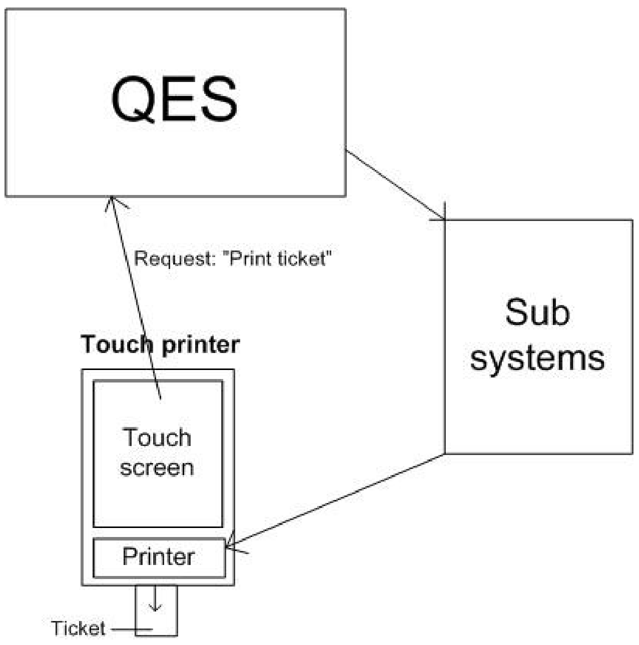
13.8.1 Use case #11 Publish page flow

This use case is more of a suggesting solution, to get hint of further work. To publish the page flow, the QES has to send data to the touch printer. This will be done using reverse-Ajax technique with help of DWR [15]. DWR gives opportunity to update part of a web page. But in this case we will just tell the browser to refresh and then the new content will be downloaded from the server.



13.22 Shows how the created page flow is pushed to the touch printer.

DWR will also be used to show that the service integration works. When user clicks on a service button a request has to be sent to the server. This will be done using DWR to send a simple Ajax-request with some data of the server. Furthermore the received request will be handled to send a ticket. Below an image shown how the data is sent between different devices to print a ticket:



13.23 Overview of printing a ticket. The collaboration between different system parts.

This use case has implemented functionality for end-users to publish created page flow on touch printers, and make it work with the ticket printing.

13.9 Summary

Now all use cases have been covered and a final result will be presented in the following section. There are functionalities sketched but not implemented; these will be discussed in the Future Work section.

14 Result

By implementation of the prototype it has been proved that it is possible to solve the problem. For a presentation of the prototype please see the appendix.

15 Discussion

15.1 Does the tool cover all needs?

The main point to discuss is how much of the needs the tool covers. First thing to look at is to compare the problem and requirement elicitations with the result. I think that this shows that the tool covers almost everything mentioned. There are a couple of functions brainstormed that will be presented in the future work section. Also talking to QMatic the response is that the functionalities supported by this tool cover a lot of marketing demands.

15.2 Is the tool usable enough?

Because of this thesis has excluded a usability research it cannot be any discussion referencing to any study material. Instead it can be said is that the developed tool makes it much easier and faster for an end-user to manage the content. It also involves more human because of that it doesn't require any programming skills at all.

16 Conclusion

The conclusion is that with help of this tool it is much easier and efficient for end-users to manage content on the touch printers. This has also been confirmed by QMatic.

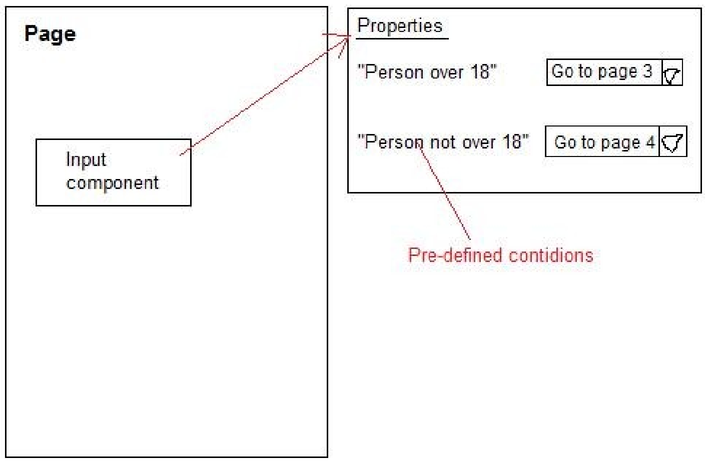
17 Future work

17.1 Input Components

One type of components not analyzed is their clients can get some data as input to the touch printers. This could i.e. be adding a personal number with a touch keyword or drawing its bank card. These types of components would require the user that managing the content to add conditions of the input data. For this type of components I would recommend to have another part split from the editor. At this part some administrator could set up a couple of conditions for the input component, i.e.:

- "If the age of the person is under 18"
- "If the drawn card belongs to a VIP customer"

Then in the editor the user should just add the component as it is done in the prototype and use the properties to add navigation to a page or service for each condition.



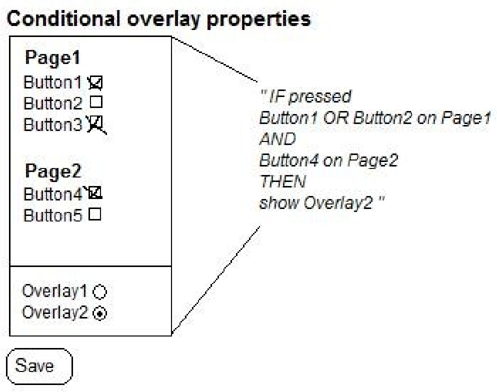
17.1 Input component.

17.2 Conditional overlays

In the prototype the user can add an overlay to each service. This means that the user can add redirect advertisement by adding overlays to related services.

An extension of the overlays would be to set a number of condition for the overlay to be shown; i.e. when client has pressed a service button on a touch printer and the ticket is getting printed, then there could be a condition "if the user chosen button 1 on first page and button 5 on second page then shown overlay 2".

To solve this in the editor the properties for service components could be extended with a matrix of all (navigation) buttons on each page. Then the user could mark a number of them that means that all these buttons have to be pressed by a client on the touch screen to make some overlay to be displayed.



17.2 Conditional overlay.

An example result could be "If the client have pressed that he is under 18 years old and pressed that he has no student account then show the overlay with advertisement about student account offer".

18 References

[1] Visa Promotion. *Intolerance to queues changing UK shopping habits*
Available: *https://www.visapromotions.net/pressandmedia/newsreleases/press233_pressreleases.jsp*
Accessed: 1 April, 2010

[2] David Maister. *The Psychology of Waiting Lines*
Available: *http://davidmaister.com/articles/5/52/*
Accessed: 1 April, 2010

[3] QMatic. *The QMatic homepage*
Available: *http://www.qmatic.com/*
Accessed: 26 April, 2010

[4] CFM. *Customer Flow Management white paper*
Available: *http://www.q-matic.com/PageFiles/238/CFM%20white%20paper_ver%20L1.5.pdf*
Accessed: 24 May, 2010

[5] Sun. *Java EE*
Available: *http://www.java.com/en/download/faq/j2ee.xml*
Accessed: 10 April, 2010

[6] Sun. *Java EE*
Available: *http://java.sun.com/javaee/index.jsp*
Accessed: 10 April, 2010

[7] Sun. *Enterprise Java Beans*
Available: *http://java.sun.com/products/ejb/*
Accessed: 12 April, 2010

[8] Sun. *Java Server Faces*
Available: *http://java.sun.com/javaee/javaserverfaces/*
Accessed: 12 April, 2010

[9] Sun. *Managed beans*
Available: *http://www.oracle.com/technology/tech/java/newsletter/articles/jsf_pojo/index.html*
Accessed: 12 April, 2010

[10] JBoss community. *RichFaces*
Available: *http://www.jboss.org/richfaces*
Accessed: 12 April, 2010

[11] *User interface design*
Software Engineering 8, Sommerville 2006

[12] *Testing the three-click rule*
Available: http://www.uie.com/articles/three_click_rule/
Accessed: 28 March

[13] *When to use a wizard*
Available: *http://www.uie.com/articles/wizard/*
Accessed: 16 April

[14] Spring Source. *Spring MVC*
Available: *http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html*
Accessed: 12 February

[15] *DWR*
Available: *http://directwebremoting.org/dwr/index.html*
Accessed: 26 April

Figure 1.1 *A physical queue*
Source: *http://www.supanet.com/where-does-our-time-go-queue-queuing-people-standing-in-line-5804633.jpg*

Figure 1.2 *A virtual queuing ticket*
Source: *http://www.faqs.org/photo-dict/photofiles/list/3185/4240queue_ticket.jpg*

Figure 1.3 *CFM Process*
Source: *http://www.q-matic.com/Global/International%20site/Images/CFM-Steps-2.gif*

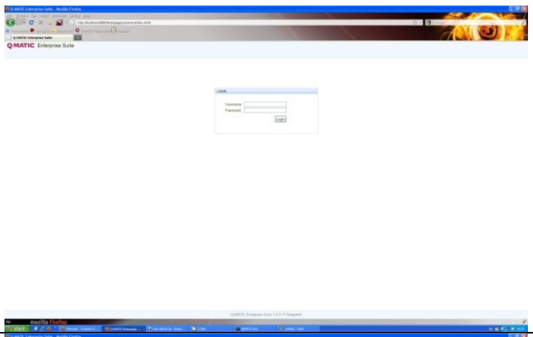
Figure1.4 *Touch printer*
Source: *http://www.qmatic-caribe.com/english/images/D.jpg*

Figure1.5 *Vision touch printer*
Source: *http://www.q-matic.it/prodotti/immagini/Vision.jpg*

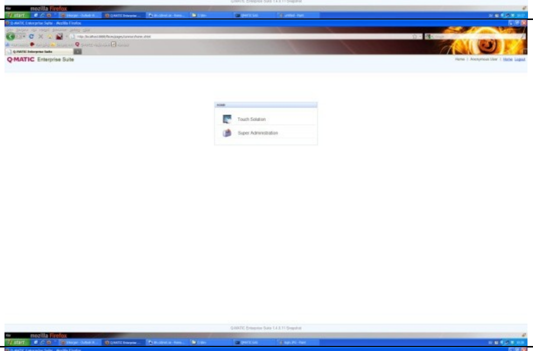
19 Appendixes

Appendix I – Prototype demo.

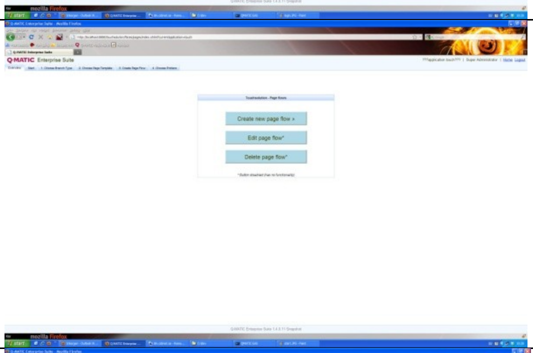
Appendix I – Prototype demo



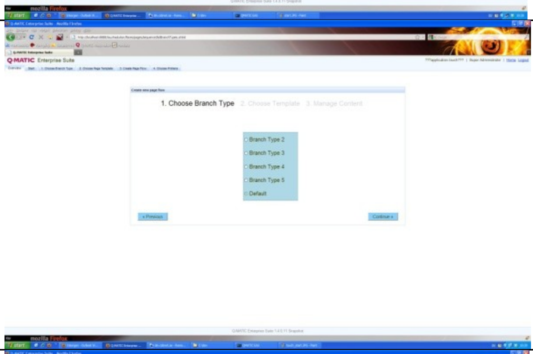
Login to QES.



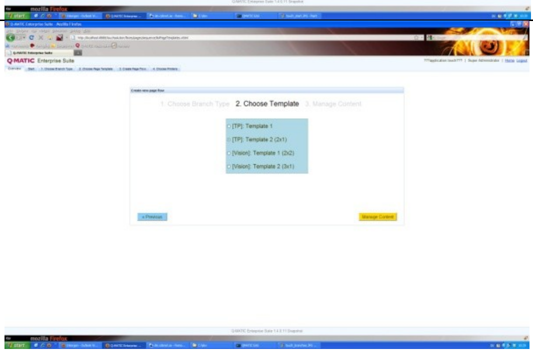
The touch solution module listed as available application.


Q-MATIC Enterprise Suite 1.4.0.11 Snapshot
Touchsolution - Page flows
Create new page flow »
Edit page flow*
Delete page flow*
*Button disabled (has no functionality)

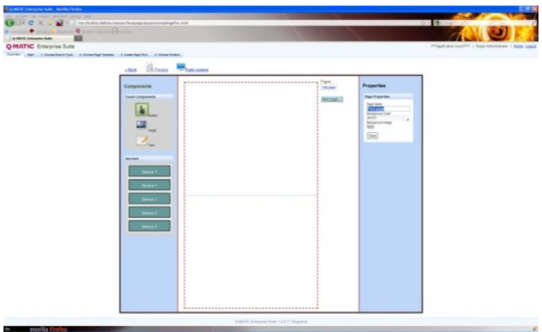
Main menu.



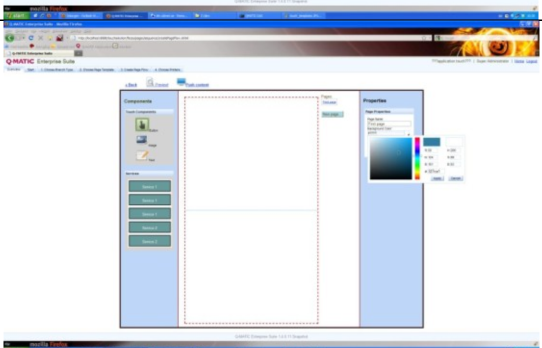
Choose branch type.



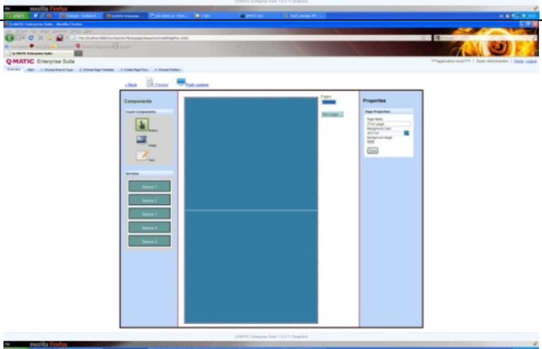
Choose page template.



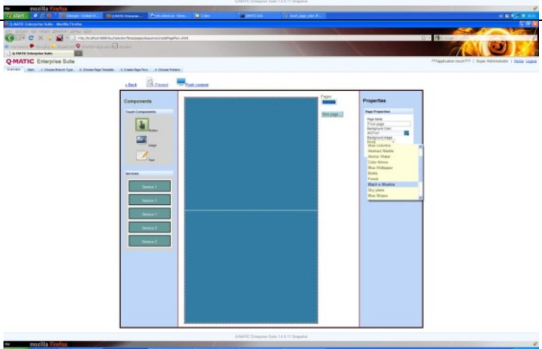
The editor.



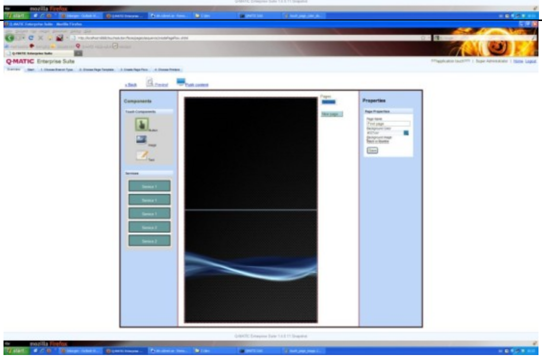
Choose page background color.



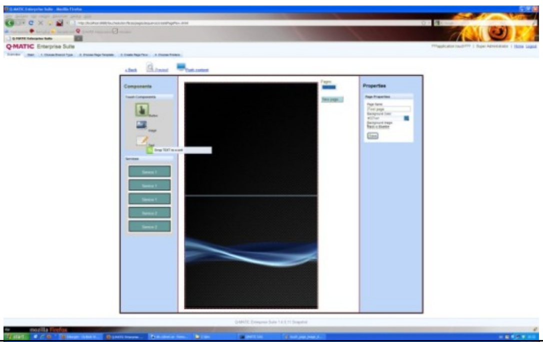
Background color changed.

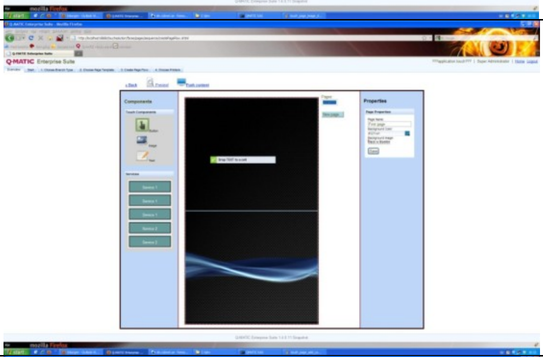


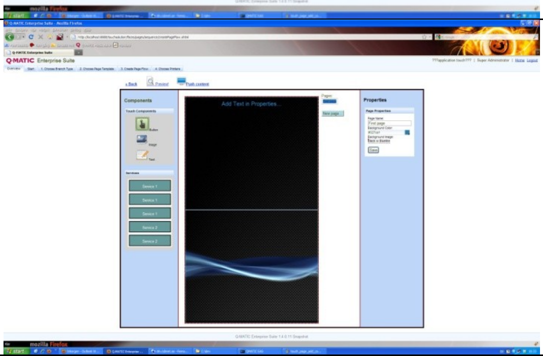
Choose page background image.

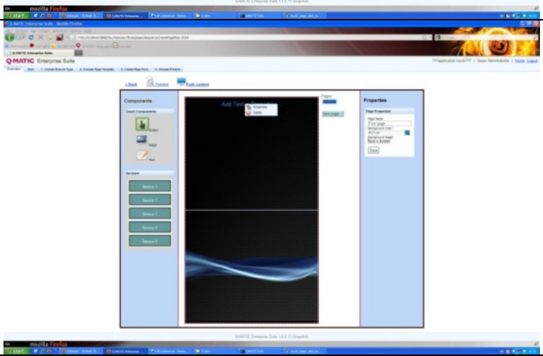


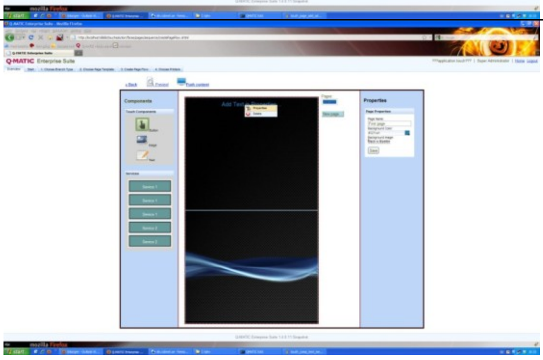
Background image changed.

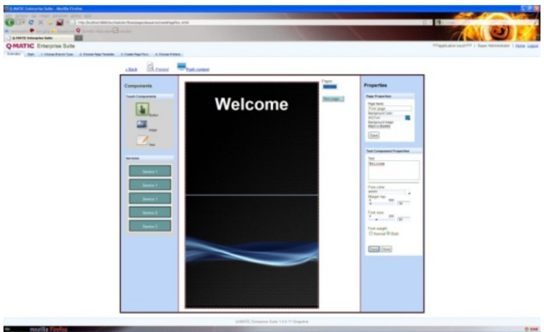

Drag a component type.


Drop to a page cell.

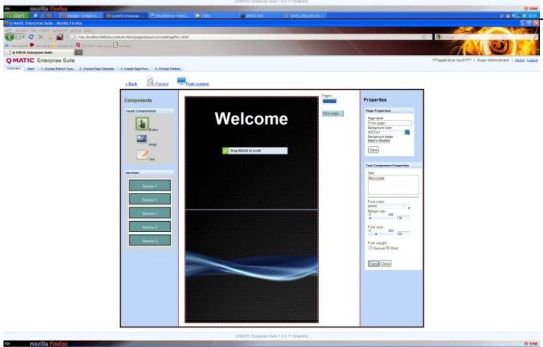

A text component dropped.


Right-click on the text component.

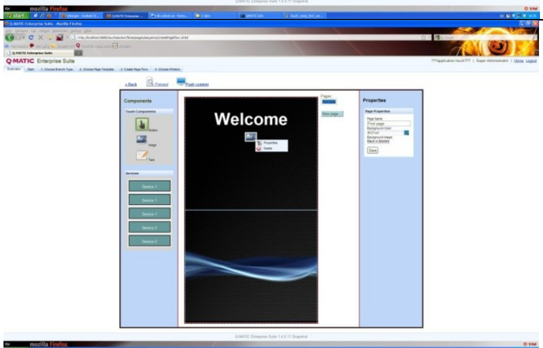

The context menu shown. Choose properties.



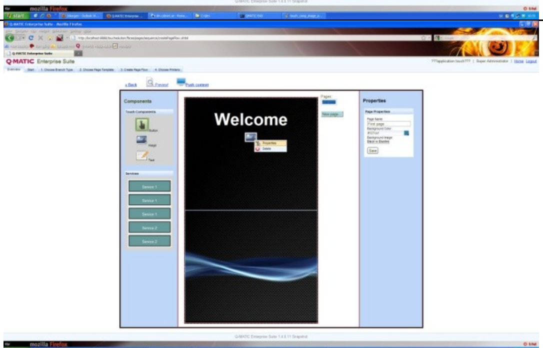
The component properties shown. Label changes to "Welcome" and font-size changed.



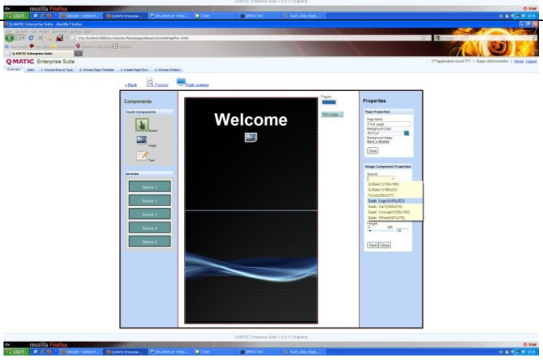
Drop a new component.



An image component dropped. Right-click.



Click on properties.



Properties for the image component shown. Choose image.

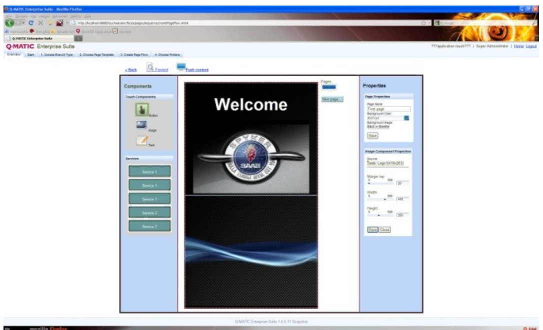
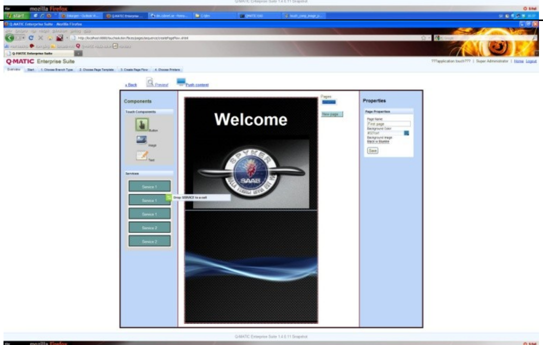
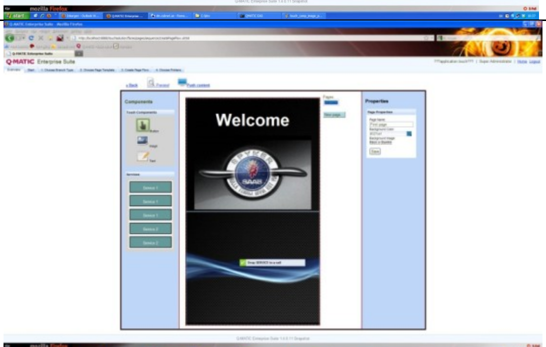


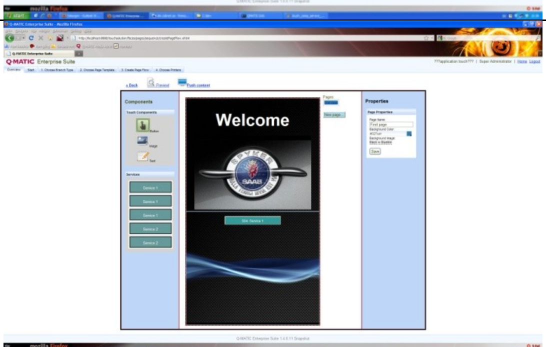
Image chosen and component updated.



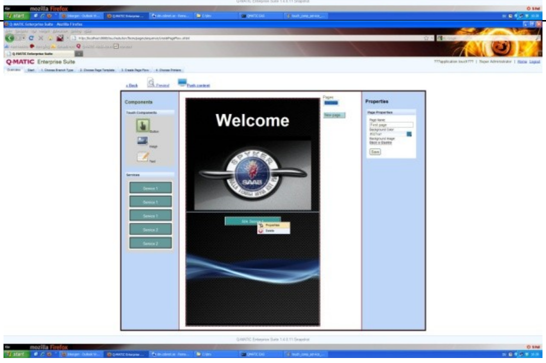
Drag a new component.



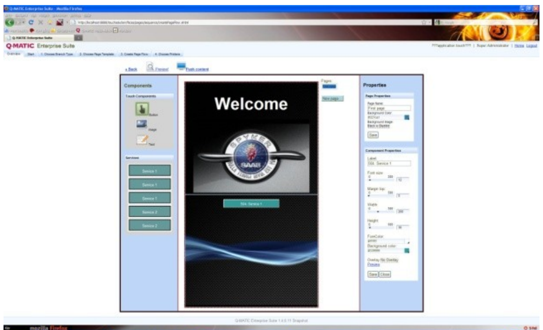
Drop.



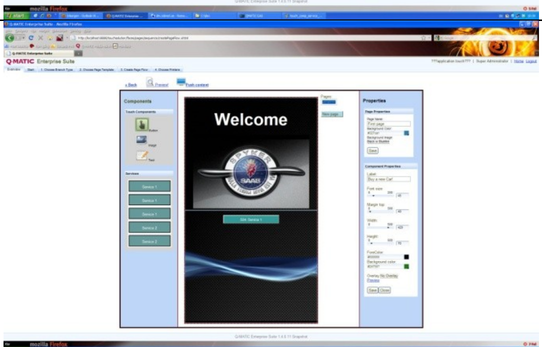
A service component dropped to second page cell.



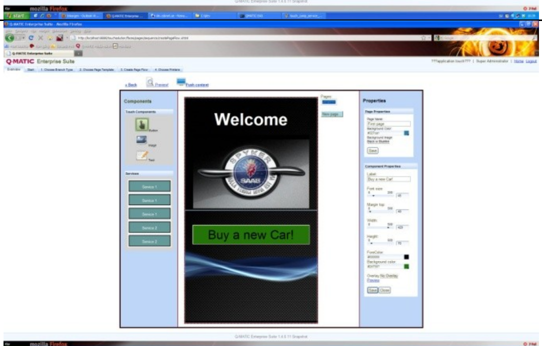
Right-click on the service component.



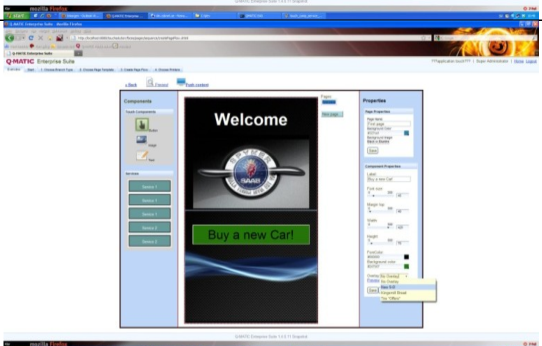
Properties for service component shown.



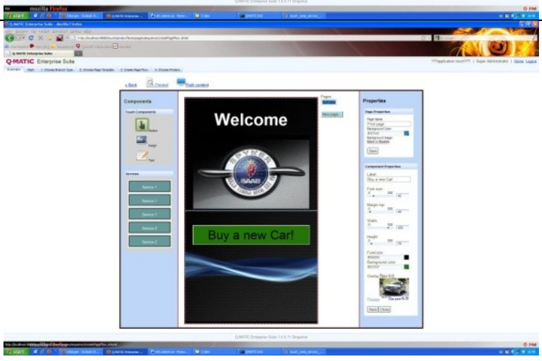
Change properties.



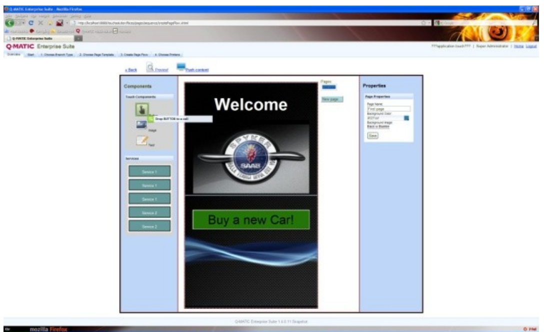
Service component updated.



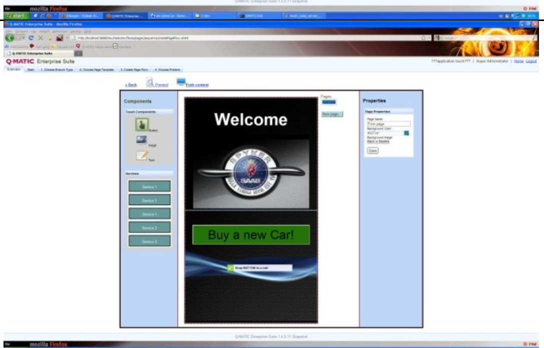
Choose an overlay for service component.



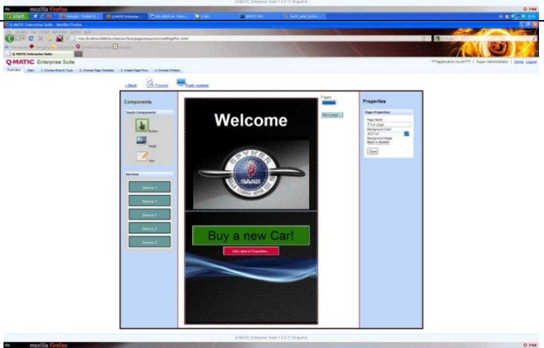
Preview chosen overlay.



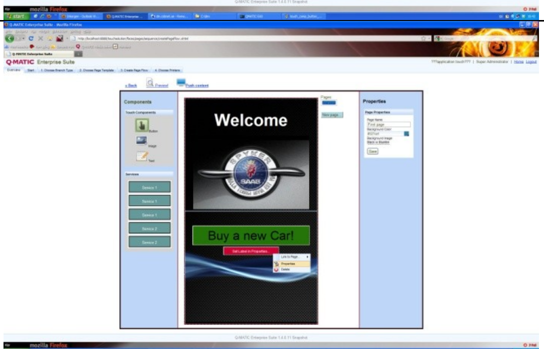
Drag a button component.



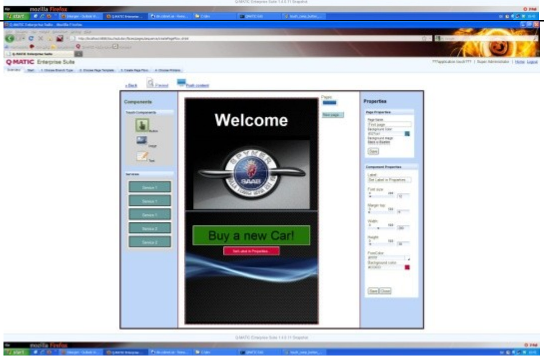
Drop.



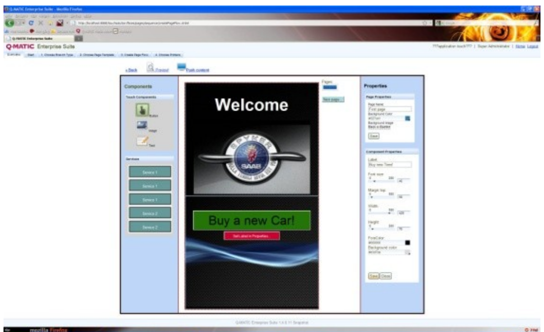
A button component dropped to second page cell.



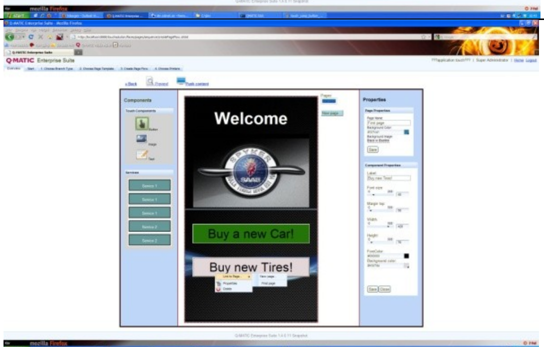
Right-click on the button component.



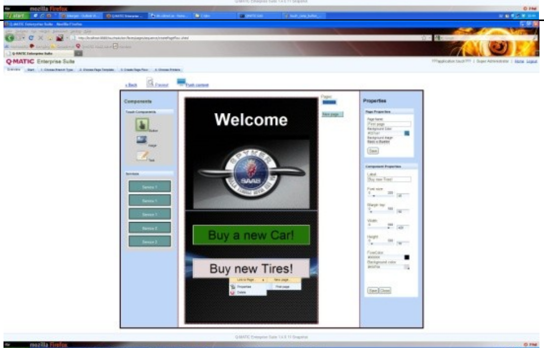
The properties for button component shown.



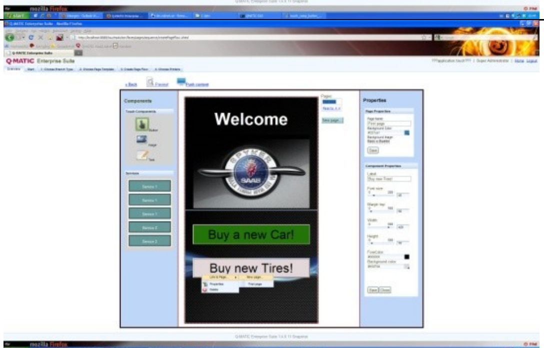
Change properties.



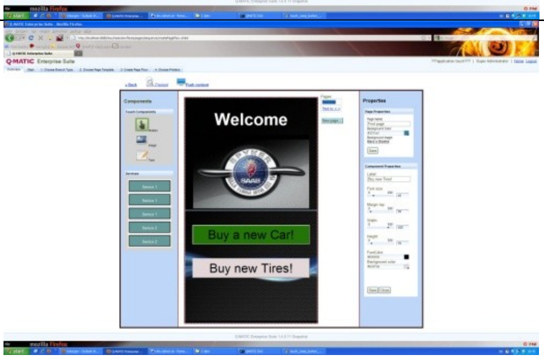
Button component updated. Right-click for linking to a new page.



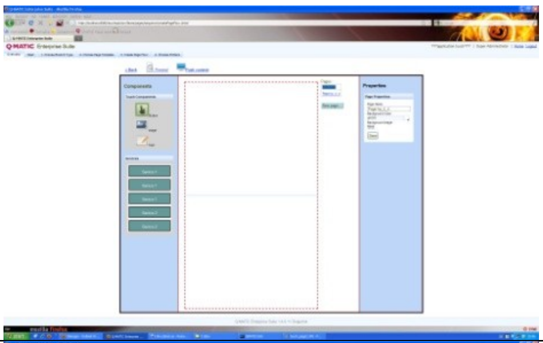
All pages in the page flow available, now just the start page.

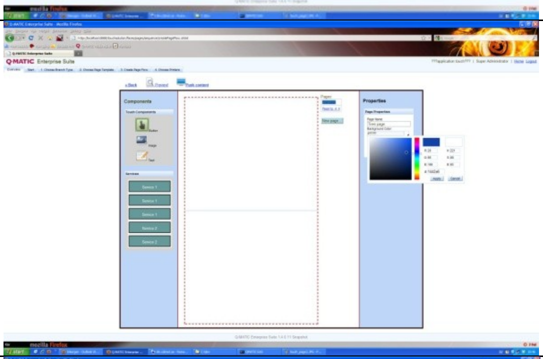


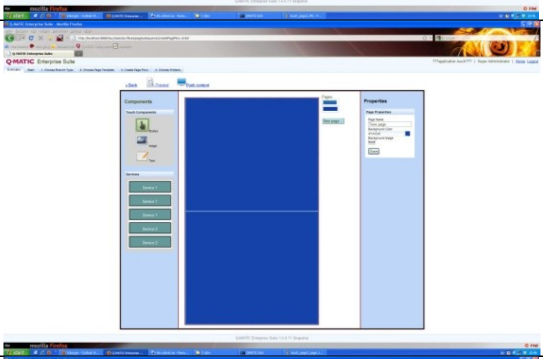
Choose "New page…".

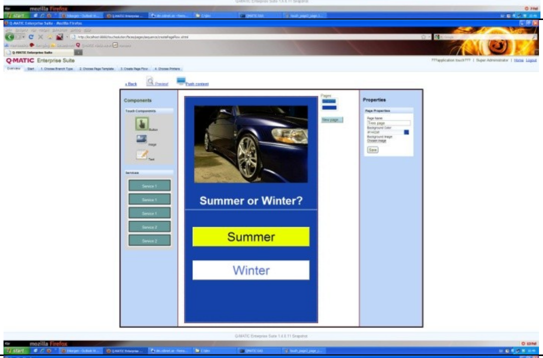


New page created and the button component linked to it.

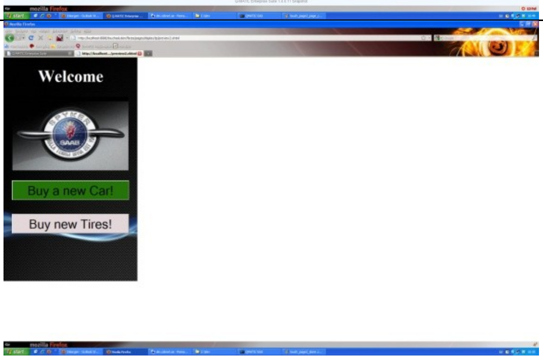

Switched to the new page.


Set a background color.

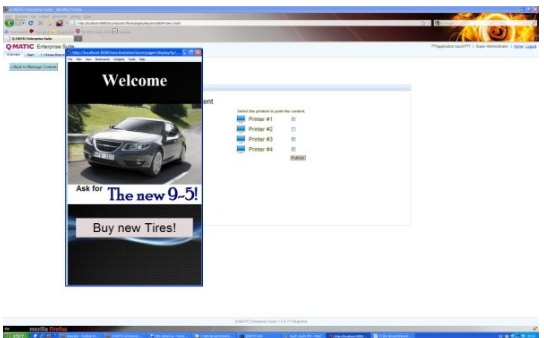

Background updated.



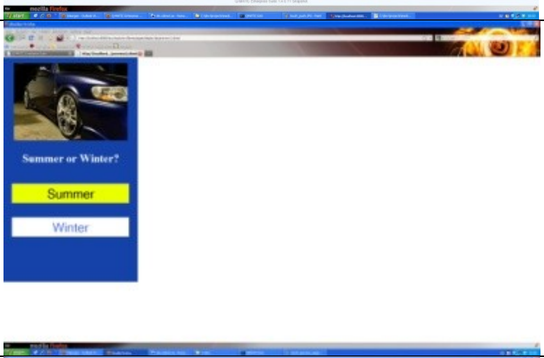
Adding new component and modifying their properties.



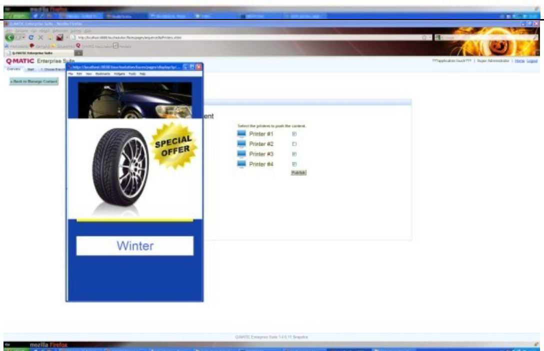
Preview.



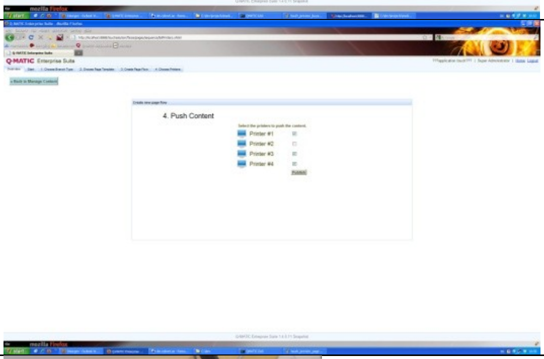
Preview of the overlay.



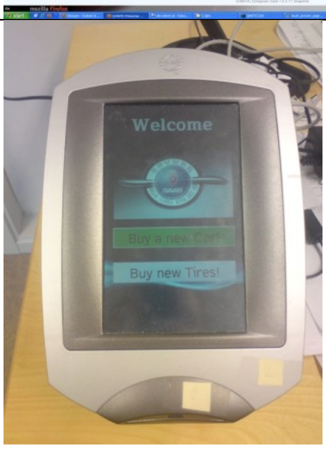
Preview of the second page.



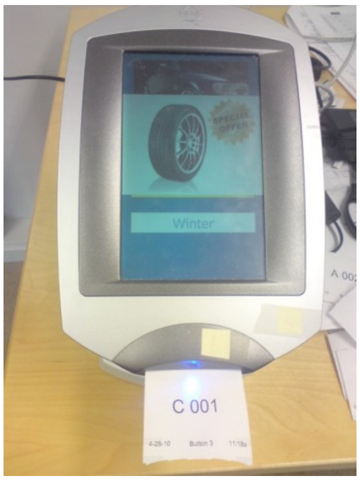
Preview of another overlay.



Touch printers listed. Choose which to publish the page flow to.



A touch printer device with the created page flow.



Showing an overlay while printing a ticket.