

Howto Start with UiaML

Contributed by Alex
 Monday, 08 October 2007
 Last Updated Monday, 07 January 2008

This howto is more a little tutorial that will help you to get started with UiaML. The tutorial will give you a method to get started, but because UiaML is a tool and not a method, it doesn't mean that that method given in this howto is the right- or the only one.

Preliminary phase Before you start using UiaML you should have an answer to the questions:

- what is the project all about and what is/are the core message(s)?
- which functionality should be provided (you can use UML - Use Cases to determine an answer to this question)?

Let's do this with an example: "De Drop Shop" (or "The Liquorice Shop" in English)

Drop is a famous typical dutch sweet that comes in many flavors and forms. "De Drop Shop" is a small fictive shop that deals in "Drop". "De Drop Shop" has no web presents yet and the owner thinks it's time to get a website. The website should provide information about the location and the opening hours of the shop. The shop also wants to express their expertise in the field of liquorice by providing information about the liquorice plant itself and the liquorice production process. Finally the shop wants to show the visitors of the site a small piece of the available assortment and allow the customer to buy them online.

The last one is actually a use case, so we specify a simple use case template for it.

Use Case

Order liquorice

Actor

Customer

Pre condition

none

Main scenario

1. The customer requests a list of the available products
 - 2.1 The system shows the customer a list of products and field to specify the order quantity of each product.
 - 2.2. The system will also ask the customer about the delivery address, his/her e-mail address and will offer the customer to add some comments to the order.
3. The customer submits his/her order
 - 4.1 The system will check whether or not there has been ordered at least one product and whether the delivery address and e-mail address has been provided.
 - 4.2 The system will show the delivery address, e-mail address, comment, a list of the ordered products and the total price of the order.
5. The customer confirms the order
6. The system sends an e-mail with the order toward the shop owner with a cc toward the customer.

Post condition

The customer has submitted an order

Alternative scenario

At 4.1) The check results in an invalid order. The system goes back to step 2.1 and will inform the customer why the

order didn't check out.

At 3, 5) To abort the order, the customer can navigate toward another page (like the homepage).

{mospagebreak title=The UiaML sitemap view}The UiaML sitemap view Done with the preliminary phase, we can continue by modeling the UiaML sitemap view (SMV). This phase requires the following steps:

- Draw a sitemap (representation of a single application window) and name it
- Define the contentarea's that will cover you goals as specified on the semantic level
- Define the containers (pages) that will hold the contentarea's
- Define flow between your pages and additional contentarea's in as required to support you goals.

We'll exercise those steps by continuing our Drop Shop example. We start by drawing a sitemap symbol, that represents the application window that holds our order site and will name it: "De Drop Shop".

Next we determine the contentareas that will hold the information we want to provide. In our example case we will need at least contentareas for:

- the location and opening hours of the shop
- liquorice background information
- and a menu that will enable us to navigate the site.

We further add the required documentation belonging to the image above.

Contentarea ID

Liquorice information

Description

Information about the liquorice plant and the production process of the liquorice sweets.

Contentarea ID

Menu

Description

Navigational menu

Contentarea ID

Welcome

Description

Opening hours and location information of the shop.

Next we will place the contentarea's into pages. Since the menu contentarea is required by each page, we duplicate it. For each page we need to define whether it is a homepage (there can only be one, indicated by a thick frame) and/or a landing page.

Page ID

Information page

Startpage

No

Landingpage

No

Page ID

Homepage

Startpage

Yes

Landingpage

Yes You might have noticed the menu consistently drawn on the left side. This is no problem, because the UiaML core is about the semantic design. The UiaML graphical design plugin will solve this issue. Next we add the navigational links to the model. Each link starts within a contentarea and point toward a page. Of each different contentarea there is exactly one contentarea the defines the links. That contentarea has its contentarea identifier underlined. As you can see from the image below; the liquorice information contentarea has no links and the defining contentarea of the menu contentarea is the one within the homepage. The menu contentarea of the information page inherits the links from the menu contentarea of the homepage.

By inheriting the links the menu from the information page inherits a link from the menu contentarea toward the homepage. So instead of drawing the defining contentarea of menu within the homepage (see image above) we could have also chosen to draw it within the information page (see image below).

Adding the use cases As you might have noticed: we skipped the contentareas / pages that will show the assortment and handle the order respectively the ones that express the "Order licquorice" use case. For documentation purposes we want to model those contentareas / pages separately in order get the focus right and to avoid a crowded sitemap that might hold several use cases. We start with an empty sitemap that has the same identifier than the sitemap we used above. Next we add a contentarea that will show the licorice assortment and will allow us to order them by entering a quantity per product and providing the delivery address, email-address and comment field as stated in step 2 of the main scenario of the use case. We further a contentarea that will give us an overview of the order (see step 4.2 of the use case). And finally one contentarea that will hold our e-mail message (see step 6 of the use case).

Contentarea ID

Assortment

Description

List of products that can be ordered online, as wel as fields to provide the delivery address, the email address and a comment.

Contentarea ID

Order

Description

Overview of the ordered products and entered field values. The order will be definitive as soon as the customer hits the submit button.

Contentarea ID

Ordermail

Description

Same information as provided by the order, but this time there will be no submit button and the information will be e-mailed toward the shop owner and the customer. In order to support the alternative scenario of step 3 and 5 we'll add the menu contentarea to our model. And place the contentareas within page containers .

Page ID

Productpage

Startpage

No

Landingpage

No

Page ID

Order page

Startpage

No

Landingpage

No

Page ID

Ordermail

Startpage

No

Landingpage

No The last step to finish our sitemap is adding the links. First we will add a link from the assortment toward the OrderPage. But as step 4.1 of the use case states, the system will have to check whether the provided information is valid and in case it's not will navigate back toward the Productpage (see alternative scenario at 4.1 of the use case). To express this within the UiaML sitemap view we use the Choice element .

Choice ID

Validate

Decision description

Check if an email address and a delivery address has been provided and whether or not there has been ordered any product.

Decision base

We need to model the contentarea view first to determine the decision base. Next we need to think about the integration of our sitemap with the rest of the site. Let's say that we will be able to reach our Productpage from the contentareas Welcome and Menu. To express the link from the menu/welcome contentarea toward the Productpage without drawing the Productpage itself within our first sitemap we use a reference page symbol . In the sitemap that holds our Productpage we'll use reference contentarea symbols to express the origin of the links toward the Productpage. We will use the same trick to express a link from the order contentarea toward the Homepage.

We also adapt the SMV that shows the Homepage to get a consistent navigation model. The last thing we'll need to take care of is to express that our email get send as soon as the link from the order contentarea toward the Homepage is fired. We'll do this by using an associative link from the submit link toward the OrderMail page. UiaML currently can not express that our OrderMail page is actually an e-mail. It might also be a file that will be written to our harddrive or a page that will be printed. That it will be an e-mail has to be expressed within your documentation.

{mospagebreak title=The UiaML contentarea view}

The UiaML contentarea view This view zooms in on the content of the contentareas defined within the sitemap. For each contentarea defined within the sitemap you'll have to determine and specify all the possible contentarea elements (CAE).

We'll exercise those steps by continuing our Drop Shop example. Our “Drop Shop” example contains the following contentarea's:

- Welcome
- Liquorice information
- Menu
- Assortment
- Order
- Ordermail

Let's specify them one at a time. {mospagebreak title= - The Welcome contentarea}

The Welcome contentarea In the sitemap view (SMV) we already defined some properties of the welcome contentarea.

Contentarea ID

Welcome

Description

Opening hours and location information of the shop. We also added a link from the welcome contentarea toward the Information page. In the Contentarea View (CAV) we have to honor the description given in the SMV and have to be consistent with the links defined within the SMV. The CAV of the welcome content area starts with a contentarea symbol . The contentarea ID has to match with the name of the contentarea used within the SMV.

Next will add a text label that will hold information about the opening hours and another text label that will hold address information on where to find the shop.

TextLabelID

Opening hours

Intended content

A table that shows the opening hours for each day

TextLabelID

Address

Intended content

The exact address of the shop

With those two text labels we fulfill the description of the welcome contentarea as specified within the SMV, but it wouldn't be a very friendly site. So lets add some additional text labels to welcome the costumer and give him some idea what this site is about, supported by some images . We also will add a street map to make it easier for the customer to find the address.

TextLabelID

Welcome header

Intended content

A nice greeting

TextLabelID

Welcome text

Intended content

Text that will inform the customer that he is on the website of the Drop Shop. A shop that is specialized in liquorice sweets.

ImageID

Shop-Outside

Intended content

A photo of the shop taken from outside, showing the front of the shop, so that visiting customers will be able to recognize the shop from the website.

ImageID

Shop-Inside

Intended content

A photo of the shop taken from inside, showing the broad choice of liquorice offered.

ImageID

Streetmap

Intended content

An image of a streetsmap with the shop at the center, showing the streets within a circle of 500 meters. You might notice that we nest some CAE in each other to express that those CAE belong together. But it doesn't say anything about the graphical design since that is handled by another UiaML plugin. Finally we have to add the link toward the information page. We define that the link will find its origin at the CAE: Welcome text. Since we already have defined within the SMV that the link will point to the information page, it's enough that within the CAV the link points somewhere outside the contentarea symbol. The link identifier must however match with the link identifier used within the SMV. {mospagebreak title= - The Liquorice information contentarea} The Liquorice information contentarea As specified within the SMV the Liquorice information contentarea doesn't hold any links and will provide information about the liquorice plant and the production process of the liquorice sweets. This means the page will hold a couple of text labels and images to meet the expectations.

TextLabelID

Liquorice information header

Intended content

A header that indicates that this contentarea covers information about the liquorice plant and sweet production.

TextLabelID

Plant information header

Intended content

A header that indicates that the related text is about the liquorice plant

TextLabelID

Plant information

Intended content

A text that provides a lot of information about the liquorice plant itself

ImageID

Liquorice plant

Intended content

An image of the liquorice plant

TextLabelID

Production info header

Intended content

A header that indicates that the related text is about the liquorice sweet production.

TextLabelID

Sweet production information

Intended content

A text that provides information about the liquorice sweet production process itself.

ImageID

Sweet production

Intended content

A photo showing the craftsmanship of traditional "drop" production. {mospagebreak title= - The Menu contentarea}The Menu contentarea The menu will holds three links: Home, Information and Products. So it's a very small menu. Each link will be triggered by a text label CAE. Notice again that the the UiaML core isn't about the look and feel, so we don't specify the direction of text, the lettertype or size etc.

TextLabelID

Liquorice information

Intended content

A label that indicates that this link will show the user information about the liquorice plant and production process.

TextLabelID

Home

Intended content

A label that indicates that this link will lead the user back to the homepage.

TextLabelID

Assortment

Intended content

A label that indicates that this link will lead to page that shows a part of the available assortment and will allow the user to place an order.

```
{mospagebreak title= - The Assortment contentarea}
```

The Assortment contentarea Within the SMV we already defined the assortment contentarea as follows:

Contentarea ID

Assortment

Description

List of products that can be ordered online, as well as fields to provide the delivery address, the email address and a comment.

Let's model the corresponding CAV by adding one item at a time, starting with the header and the list of products. Until now we haven't defined what aspects of a product should be shown, so let's say that each product of our product list consists of a name, an image, a weight and a price indication as well as a short description. Since this site is a very low-budget site, we will only list 10 products. We could model this by placing 10 elements for each product aspect in the Assortment CAV, but we can also use a list element instead.

The AssortmentHeader is a simple TextLabel element. So the documentation of it is straight forward:

TextLabelID

AssortmentHeader

Intended content

A label that indicates that this contentarea shows a part of the available assortment and will allow the user to place an order.

Documenting the elements within the list elements would require that we add 10 tables for each element within our list element. To be able to express that a certain tables belong to the same product instance we use add a referential path to the element identifier, like Product[5].Name or Product[7].Image (Note that there is a dot between the element identifiers; The dot is chosen to ease the communication with developers who often program using an object oriented programming language). But since the description for each element within our list remains the same we use a generic indicator (n) to express that the element properties remains the same for each instance.

TextLabelID

Product[n].Name

Intended content

Name of the product

TextLabelID

Product[n].weight

Intended content

Weigth of the product

TextLabelID

Product[n].price

Intended content

Price of the product

TextLabelID

Product[n].description

Intended content

Description of the product

ImageID

Product[n].Product image

Intended content

Image of the product

Finally we have to add the properties of our list itself to our documentation.

ListID

Product

Nr of items

10

Sort element

Name

Sort order

Alphabetically in ascending order

Recursion

No

We now have a list but we want to be able to select items from the list. By adding a min and max selection our list becomes a SelectList element . Since our customers are not required to order anything we can set the min selection to 0. But our customers may order all of our products, so the max selection will become 10.

Because our List element has become a SelectList element we have to adjust our property table.

SelectListID

Product

Nr of items

10

Sort element

Name

Sort order

Alphabetically in ascending order

Recursion

No

Min selection

0

Max selection

10

Mandatory

No

Intended purpose

Selected products are ordered by the customer.

Now our customers may only order 1 item of each of our products. So we have to add something to our SelectionList to be able to express an order quantity. We do so by adding a TextInput element together with a corresponding TextLabel element to tell the customer about the purpose of the TextInput element.

TextLabelID

Product[n].quantity

Intended content

Label before the input field that tells the customer that the input field is meant for the quantity.

TextInputID

Product[n].quantity.QTY

Intended content

The initial value of the input field should be zero.

Intended purpose

Tells the shop owner how many items of a product the customer orderd.

Mandatory

No

Next we will add the remaining elements mentioned within the description of the contentarea.

You might have noticed that the InputText elements of delivery address and e-mail address have a shadow. This is a visual representation of the mandatory property.

TextLabelID

txtDeliveryAddress

Intended content

Label before the input field that tells the customer that the input field is meant for the delivery address.

TextInputID

txtDeliveryAddress . DeliveryAddress

Intended content

none

Intended purpose

Delivery address of the ordered items

Mandatory

Yes

TextLabelID

txtEmailAddress

Intended content

Label before the input field that tells the customer that the input field is meant for the e-mail address.

TextInputID

txtEmailAddress . EMailAddress

Intended content

none

Intended purpose

E-Mail address of the customer. The shop owner will contact the customer using this address to handle the payment and other issues regarding the order.

Mandatory

Yes

TextLabelID

txtComment

Intended content

Label before the input field that tells the customer that the input field is meant for the delivery address.

TextInputID

txtComment . Comment

Intended content

none

Intended purpose

Here the customer can add a comments that will be send along with the order.

Mandatory

No

To submit the order we need an order button. We represent the button by a simple text label that can trigger a link.

TextLabelID

OrderButton

Intended content

Label that indicates that the customer should click on it to proceed with the order.

If we take a look at our sitemap than we see that the order link will result in a validation. In the specification of the Choice Element “validate” we already stated that we need information from the contentarea first in order to determine the decision base property. Based on the decision description, we see that the decision is based on the contentarea elements txtDeliveryAddress . DeliveryAddress and txtEMailAddress . EmailAddress. To determine whether or not a product has been ordered, we can either state that any item of the CAE Product has been selected. But determine whether or not there is any QTY item within the SelectList Product that has a value greater than zero might be more specific. Our new specification of the SMV Choice Element “Valide” therefore becomes:

Choice ID

Validate

Decision description

Check if an email address and a delivery address has been provided and whether or not there has been ordered any product (there must be a QTY field with a value greater than zero).

Decision base txtDeliveryAddress . DeliveryAddress txtEMailAddress . EmailAddress Product[n].quantity.QTY

Finally the validation might generate in an error event resulting in a navigation back to the Productpage. Displaying the page as specified we get the problem that the customer doesn't get any feedback and therefore doesn't know that he didn't pass the validation check. To solve this problem we could add a popup to the link, we however choose to display the error messages within the assortment contentarea.

The error messages are just normal text labels containing a text that informs the customer about the mistake he made. Including the error message within this contentarea doesn't mean that they are always visible (that is handled by another plugin), but that they semantically belong to the content of the assortment contentarea. Getting back to this contentarea not only means that an error message should be shown, but also that the values entered previously should be visible again. So we not only have to add the property tables of the error messages, but also have to change the intended content of the text input elements.

TextLabelID

errDeliveryAddress

Intended content

Error message that informs the customer that the provided delivery address isn't valid.

TextLabelID

errEMailAddress

Intended content

Error message that informs the customer that the provided email address isn't valid.

TextLabelID

errNoProductsSelected

Intended content

Error message that informs the customer that no products has been selected and the order therefore isn't valid.

TextInputID

txtDeliveryAddress . DeliveryAddress

Intended content

- the previously entered delivery address, in case the contentarea is shown due to an error generated by the validation check.
- otherwise there is no content to be shown.

Intended purpose

Delivery address of the ordered items

Mandatory

Yes

TextInputID

txtEMailAddress . EMailAddress

Intended content

- the previously entered e-mail address, in case the contentarea is shown due to an error generated by the validation check.
- otherwise there is no content to be shown.

Intended purpose

E-Mail address of the customer. The shop owner will contact the customer using this address to handle the payment and other issues regarding the order.

Mandatory

Yes

TextInputID

Product[n].quantity.QTY

Intended content

- the previously entered quantity, in case the contentarea is shown due to an error generated by the validation check.
- otherwise the initial value of the input field should be zero.

Intended purpose

Tells the shop owner how many items of a product the customer orderd.

Mandatory

No

{mospagebreak title= - The Order contentarea} The Order contentarea In the SMV we specified that the Order contentarea gives an overview of the order and is the last step to finalize the order.

Contentarea ID

Order

Description

Overview of the ordered products and entered field values. The order will be definitive as soon as the customer hits the submit button.

So all we have to do is show the entered values as a text labels and offer the customer a change to submit the order.

TextLabelID

OrderHeader

Intended content

A label that indicates that this contentarea shows an order and is the last step to finalize the order.

TextLabelID

txtDeliveryAddress

Intended content

Label that indicates that the value that follows is the provided delivery address

TextLabelID

DeliveryAddress

Intended content

The delivery address as entered in CAV:Assortment . CAE:DeliveryAddress.

TextLabelID

txtEmailAddress

Intended content

Label that indicates that the value that follows is the provided e-mail address

TextLabelID

EmailAddress

Intended content

The E-Mail address as entered in CAV:Assortment . CAE:EmailAddress

TextLabelID

txtComment

Intended content

Label that indicates that the value that follows is the provided comment.

TextLabelID

Comment

Intended content

Comment as entered in CAV:Assortment . CAE:Comment

Our customers are allowed to order less products than we have provided in the assortment contentarea view. So all we know about the length of our list is that it will be between 1 and 10. The actually length of the list however remains a secret. We therefore use the x in our model to indicated this phenomena.

TextLabelID

Product[x].Name

Intended content

Name of the ordered product.

TextLabelID

Product[x].price

Intended content

Price per product of the ordered product

TextLabelID

Product[x].quantity

Intended content

Label that indicates that the value that follows is the provided quantity

TextLabelID

Product[x].quantity.QTY

Intended content

Quantity as entered in CAV:Assortment . CAE:Product[n].quantity.QTY

ListID

Product

Nr of items

x

Sort element

Name

Sort order

Alphabetically in ascending order

Recursion

No

TextLabelID

Totalprice

Intended content

Total price of the ordered products.

TextLabelID

SubmitButton

Intended content

Label that indicates that the customer should click on it to finalize the order.

{mospagebreak title= - The Ordermail contentarea}

The Ordermail contentarea As soon as the customer submits the order a mail with the order will be send toward the customer as well as toward the shop owner. We included the mail because it also contains content and is part of our system. The used protocol to transmit the content (HTTP, E-Mail, FTP, ...), can't be modeled with UiaML. The contentarea of the Ordermail can be modeled like this:

TextLabelID

OrdermailSubject

Intended content

A label that indicates that this contentarea shows an order submitted by a customer.

TextLabelID

ShopEMailAddress

Intended content

E-mail address of the shop owner who should receive the e-mail

TextLabelID

txtDeliveryAddress

Intended content

Label that indicates that the value that follows is the provided delivery address

TextLabelID

DeliveryAddress

Intended content

The delivery address as entered in CAV:Assortment . CAE:DeliveryAddress.

TextLabelID

txtEmailAddress

Intended content

Label that indicates that the value that follows is the provided e-mail address

TextLabelID

EMailAddress

Intended content

The E-Mail address as entered in CAV:Assortment . CAE:EMailAddress

TextLabelID

txtComment

Intended content

Label that indicates that the value that follows is the provided comment.

TextLabelID

Comment

Intended content

Comment as entered in CAV:Assortment . CAE:Comment

TextLabelID

Product[x].Name

Intended content

Name of the ordered product.

TextLabelID

Product[x].price

Intended content

Price per product of the ordered product

TextLabelID

Product[x].quantity

Intended content

Label that indicates that the value that follows is the provided quantity

TextLabelID

Product[x].quantity.QTY

Intended content

Quantity as entered in CAV:Assortment . CAE:Product[n].quantity.QTY

ListID

Product

Nr of items

x

Sort element

Name

Sort order

Alphabetically in ascending order

Recursion

No

TextLabelID

Totalprice

Intended content

Total price of the ordered products.

{mospagebreak title=Documenting it all} Documenting it all There is no standard on how to document the model in its totality. The only documentation requirement is the property table of each symbol used. There are however some experiences of best know practices on how to document an UiaML modeled site.

- Start with a model that shows the homepage in it's center. From there you can have links toward reference pages.
- Use a chapter for each use case. Starting with the use case itself, followed by the corresponding sitemap view (SMV) and the corresponding contentareas (CAV).
- You might give a total sitemap at the appendix.

Example: The Drop shop UiaML specification (PDF) .