Towards Model-Driven Development of Staged Participatory Multimedia Events

Jan Van den Bergh, Steven Huypens, and Karin Coninx

Hasselt University – transnationale Universiteit Limburg
Expertise Centre for Digital Media – Institute for BroadBand Technology
Wetenschapspark 2
3590 Diepenbeek
Belgium
{jan.vandenbergh,steven.huypens,karin.coninx}@uhasselt.be

Abstract. The industry nowadays is showing an increasing interest towards an extended interactive television experience, called participation television. This increasing interactivity brings the creation of such television events closer to the creation of regular software as we know it for personal computers and mobile devices. In this paper we report on our work in model-driven development of one kind of such interactive television shows, staged participatory multimedia events. More specifically, this work reports on the domain-specific language we created to model these events and the generation of abstract prototypes. These interactive prototypes are built using web-languages and can be used to perform early evaluation.

1 Introduction

In recent years the entertainment industry has known an increasing focus on interactive and social programmings. The traditional passive, lay back television medium is getting more interactive to compete with other popular interactive entertainment devices. Every day more television shows allow user interaction in one way or another, e.g. well-known TV-games are being extended to get a new social dimension[1].

Participation television is the kind of interactive television with the highest degree of public involvement. The watchers are no longer constrained to merely passive viewing or anonymous participation, but they can become part of the show if they want to. This can be accomplished by not only using a remote control and keyboard for interaction, but by adding devices such as microphones and webcams to create input for the show and by using cross-medial aspects; e.g. users can view or interact using their remote control and television set, their PC or even their mobile phone.

The creation of participation television shows is complex, not only regarding the social, creative and managerial aspects, but the software needed for those shows is becoming very complex and more similar to traditional software engineering projects compared to the production of traditional broadcast television shows. Within the IWT project Participate, we extend existing software engineering techniques for the creation of interactive desktop applications so they can be applied to participation television.

[1] http://research.edm.uhasselt.be/kris/research/projects/telebuddies/

In this paper, we report on our results obtained by combining a model-based development approach in combination with prototypes in the earliest phases of development. We show that it is possible to combine high-level models to generate interactive abstract prototypes of participation television formats.

We will start the remainder of this paper by shortly discussing some related work within the model-based community and how participation television formats are designed and realised, followed by the introduction of our approach and modeling notation for high-level participation television specification. This is followed by a discussion of how we use current standards from the World Wide Web Consortium (W3C) to construct interactive abstract prototypes. Finally, we explain how we generate the interactive abstract prototypes from the high-level specification and draw some conclusions.

2 Related Work

At the moment several tools (from companies such as Aircode, Alticast, Sofia, iTVBox and Cardinal) are commercially available for the development of interactive television applications. These tools provide a graphical environment enabling a non-technical user to easily create simple iDTV software or websites. They require no technical knowledge like Java, MHP or DVB-HTML [6] from the designer and thus ease the creation of iDTV. Most of them also offer an emulator which enables the author to preview the result of his work on his own PC, rather than deploying his output to a set-top box.

These environments are however too limited for the development of participation television. They are mostly centered to designing the graphical layout of various pages and the addition of some common iDTV modules and components. There is no support to add the interaction components needed for the participation of the viewer in a television show.

A number of tools have been created that allow early prototyping based on models. Canonsketch [2] is a tool that provides synchronized models at different levels; at the highest level, a UML class diagram is used to represent the structure of a single dialogue using the notation proposed in [10]. The Canonical Abstract Prototypes notation (CAP) [4] allows to describe the functionality offered by a user interface at an abstract level using a limited set of icons that are linked to specific areas of the screen. At the most concrete level, a preview using HTML is provided.

Elkoutbi et al. [7] use UML diagrams to define user interfaces at an abstract level using annotated collaboration diagrams and class diagrams from which statechart diagrams can be generated. Based on these statecharts, complete functional prototypes are generated. The approach is concentrating on form-based user interfaces for a single user. The specifications that are used as input, however, have to be rigorously defined in comparison to what we want to accomplish.

Pleuss [12] proposes a methodology that uses UML to model multimedia applications. A code generator for Flash supporting his methodology is under development.

Several model-based approaches for the development of user interfaces are taking into account some form of context-sensitiveness. The TERESA-tool [11,9] allows the semi-automatic creation of multi-platform user interfaces (mostly targeted to the web) starting from a task model. Another task-based method is described by Clerckx et al. [3], who provide tool support that enables derivation of a context-sensitive dialog model from a context-aware task-model. In combination with a presentation model, the tool is able to generate context-aware prototypes.

Other approaches, such as Damask [8] for low-fidelity prototyping use a purely informal sketch-based interface. Damask uses models, originating from the model-based design community, in the backend but does not expose the designer to these models to enable consistent prototyping of multi-device user interfaces. Damask allows interactive testing of the sketched interfaces, but models cannot be reused.

3 Staged Participatory Multimedia Events

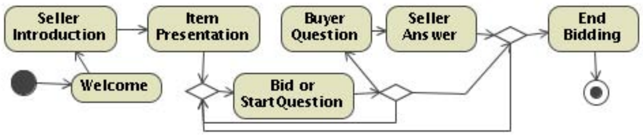
Broadband end users currently witness an evolution towards ubiquitous entertainment being delivered over their High-Speed Internet (HSI) line. As such the broadband pipe is used to deliver complete experiences to the home. Unfortunately, this enables the home-based users only to consume more professional content (Hollywood movies on demand) and to get some low level interaction with the broadcast television: time shifted viewing, voting, etc.

The goal of Staged Participatory Multimedia Events (SPME) is therefore to actively engage end-users and turn them into true participators, thus providing a stage for users to participate in challenging interactive television applications that do not exist today. Several devices like microphones and webcams will be used in these formats to enable the different participators with true interactivity. In the future, this will lead to television shows with thousands of active participants, where complex software is needed to cope with these new formats.

In the remainder of this paper we will use an auction as an example of a SPME. The auction SPME could start when a regular auction is being broadcast and at least one of the viewers has offered an item for sale. When enough interested viewers are registered, an auctioneer can decide to start the auction. After a short welcome message, he introduces the seller to the interested viewers. The seller then has the opportunity to promote the item, after which the auction starts. Any of the registered viewers can then make a bid on the item or ask questions to its seller. When a satisfactory price is reached, the auction is concluded. The control flow specification and a prototype implementation of this scenario are shown in Fig. 1.

It is clear that creating such an interactive show requires establishing and creating a complicated software infrastructure. It is our intent to facilitate the creation of such shows by offering a set of models from which the necessary code, targeting a set of pre-build networked components, can be generated. These models should abstract away a lot of the details and complexity of such infrastructure. In our approach this abstraction is reached by using layered models. The models at the highest level of abstraction do not have a direct relationship with the software infrastructure but relate to the structure of a SPME and the structure of the user interface that the participants of the SPME interact with on their television set. These high-level models can be used without the lower level details to enable early design evaluation using generated prototypes. These models and prototypes are discussed into more detail in the following sections.

(a)



(b)

Fig. 1. The auctionTV scenario: (a) a high-fidelity prototype implementation ©Alcatel (with permission), (b) control flow specification

4 Specification Language

We created a domain-specific modeling language that uses abstractions familiar to those involved in the creation of participation television. We chose to define a domain-specific language instead of traditional models used in model-based user interface design or software engineering because this allows us to create a language that uses concepts familiar to the target audience, creators of a SPME, and still can be translated into the required software concepts. The attentive reader will notice that the content of some models used in model-based design of user interfaces, such as dialog model, user model and presentation model, is combined in new models in the domain-specific language.

In this paper we will limit ourselves to those language parts relevant for the creation of the first high-level models. The first part is the general flow of the show. It is specified in the scenario model, which is built around the concept of *scenes* or *template instances* and is discussed in section 4.1. The second, and last, relevant part describes the screen composition for the different roles that are involved within a scene or template as is explained in section 4.2. Before discussing these two models, we start with the necessary definitions for the used terminology.

Definition 1. *A* template *is a reusable behavioral artifact containing a flow of actions in which a specified number of participants having different roles can be involved. A template can receive data as input and output parameters.*

Definition 2. *A* scene *is an instance of a template. The value of all input parameters of a scene needs to be specified either explicitly in the model or implicitly through derivation from output parameters of another scene in the scenario diagram.*

4.1 Scenario Model

The scenario model describes the structure of a SPME using scenes (see definition 2). The overall layout of the diagram is based upon the graphical language used to program LEGO mindstorms[2] . The language also features a horizontal bar, the *heading*, at the top that serves as a starting point for one or more parallel flows[3]. The flow of a program

[2] http://mindstorms.lego.com/

[3] Our current tool support does not take parallel flows into account.

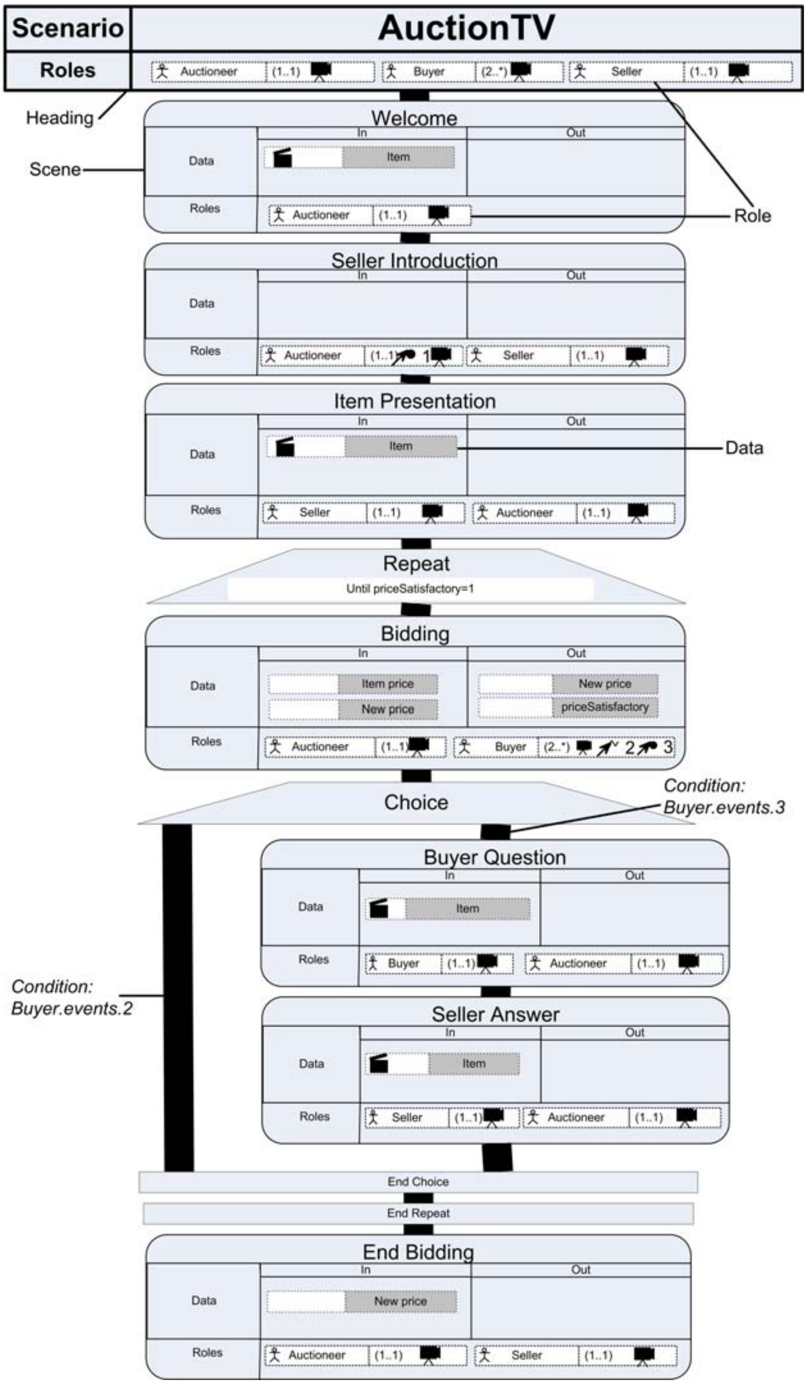


Fig. 2. An annotated example of a scenario diagram

is specified from top to bottom. Each block in the diagram corresponds to one scene of the scenario whereas in LEGO mindstorms it corresponds to one action. Loops and conditional behavior are specified using two separate blocks as can be seen in Figure 2, which shows a scenario model for the auction SPME discussed section 3. The different parts of the scenario-model are marked in the Figure and shortly discussed in the remainder of this section.

Heading. The heading gives some generic information about the scenario, specified in that diagram. It contains the name of the scenario as well as all roles that are required to start the execution of the scenario.

Scenes. A scene is depicted as shown in Fig. 2. The scene representation consists of three main areas. The upper part specifies the name of the template, while the middle part specifies the data flowing into the templates (center) and out of the template (right). The lower part shows the roles that are actively involved in the template. A role is actively involved when users having that role can cause an event that triggers or ends part of the behavior specified in a template or can give streamed input to the modeled system, such as live-video through a webcam.

Roles. The graphical representation of each role in the diagram shows role-specific information in addition to the role name. The minimum and maximum number of participants having that role are shown between parentheses while the events that can be triggered as well as the required input devices are represented using icons. The icons for the events correspond to the tools of the Canonical Abstract Prototype notation [4], while the icons for the input devices are stylized representations of a camera and a microphone .

Two predefined role types can be used in scenario models: *all*, representing all participants of an SPME, *other*, all participants except those having roles explicitly specified in the template.

Data. The only information visible in the diagram about the parameters and the results of a template, are its name and an indication of the data type. A stick-Figure is shown for role-related data (e.g. when the role of a participant changes within a template), a stylized camera for live streaming video, a stylized microphone for live streaming sound, and a stylized clapper board for media recorded in advance. For all other types of parameters no type indication is provided.

4.2 Screen Model

For the screen model, we adopted the Canonical Abstract Prototype notation [4]. This notation uses rectangular regions with icons that identify the type of abstract component that spans this region. Three major types of abstract components exist with different subtypes: *generic abstract tools* ↗ (actions, operators, mechanisms or controls that can be used to operate on materials), *generic abstract material* □ (containers, content, information, data that is operated upon) and *generic abstract active material* ↗□ (a combination of both other types).

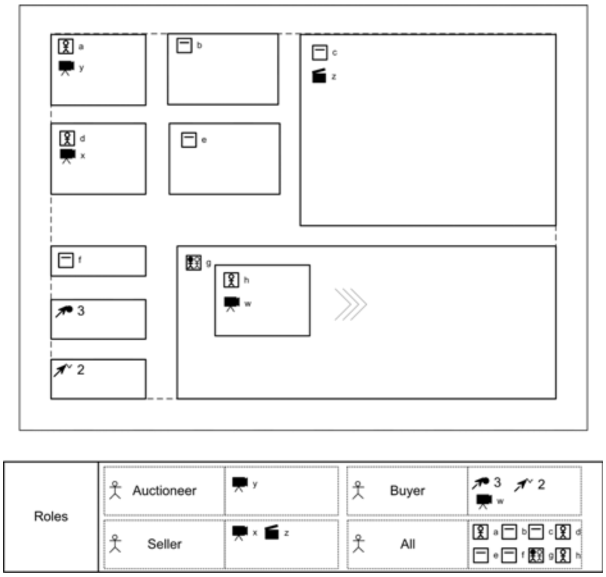


Fig. 3. An example of a screen model

Our notation is shown in Fig. 3. The differences with the Canonical Abstract Prototype notation are driven by the difference in targets. The standard Canonical Abstract Prototype notation is used to give designers an overview of the functionality that is to be shown to a user on a screen or in a window on a screen. Depending on the designer's creativity, the final screen layout can be entirely different.

The screen model, however, is intended to design user interfaces that target entire television screens. A single model can describe the different screen compositions that are used within one or more templates. Therefore each icon is combined with a character that identifies the component. Each character-icon combination appears within both the screen layout and the role section establishing the connection between the participants and the abstract components he can observe. Multiple character-icon combinations can be placed within one region with each combination visible to participants with a different role. We also introduced two new icons to identify user interface components that show a participant (participant element ▣) or a collection of participants of which zero or more can be emphasized (active participant collection ▦). A concrete example of the latter type of user interface component can be seen in the lower right corner of Fig. 3.

Because involvement of the participants is an important aspect of a SPME, icons referring to the origin of multimedia materials are added to the icons in CAP-notation [4]. The sources can be either live multimedia streams 🎥 or recorded multimedia 🎬 . The sources are linked to users with a certain role with means identical to the ones used to link user interface components with participants with a certain role.

5 Interactive Abstract Prototypes Using XML

The dynamic abstract prototypes are expressed using a combination of XHTML, XForms [5] and CSS. XHTML merely serves as a container for embedding the dynamic abstract prototype. XForms and CSS are respectively used for expressing the structure of the dynamic abstract prototype and the styling and positioning of the abstract components. This combination was chosen because the tools to display these specifications are freely available and the specifications are relatively easy to read. Furthermore, the style and layout, the structure of the show, the user interface and the runtime data are all cleanly separated. The choice for XForms is also motivated by the fact that it is designed to be embedded in another XML-based language and is completely declarative (including event handling). This enables reuse for more concrete prototypes, for which the XForms-structure could be largely reused in for example a SMIL [1] document. XS-miles[4] is being ported to the MHP platform and will be able to show content expressed using XForms and SMIL .

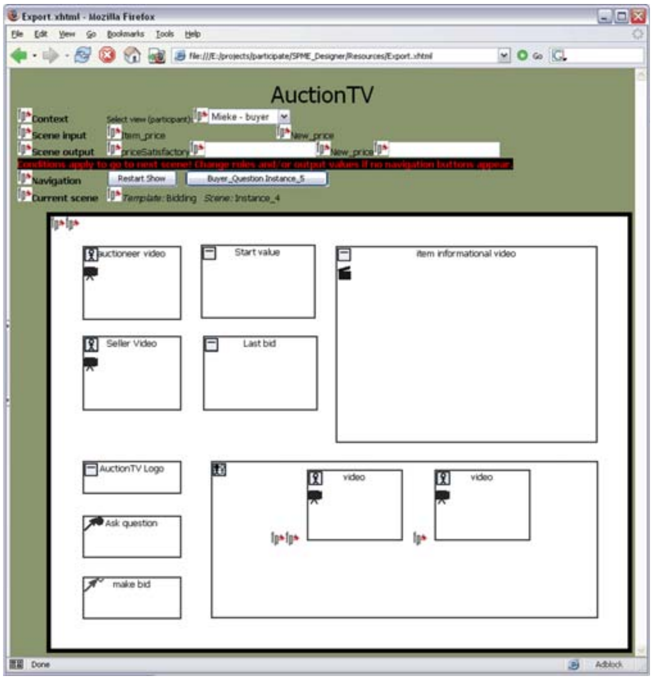

Export.xhtml - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///E:/projects/participate/SPME_Designer/Resources/Export..xhtml
AuctionTV
Context
Select view (participant):
Mieke - buyer
Scene input
Item_price
New_price
Scene output
priceSatisfactory
New_price
Conditions apply to go to next scene! Change roles and/or output values if no navigation buttons appear
Navigation
Restart Show
Buyer_Question Instance_5
Current scene
Template: Bidding Scene: Instance_4
auctioneer video
Start value
item informational video
Seller Video
Last bid
AuctionTV Logo
Ask question
make bid
video
video
Done
Adblock

Fig. 4. An example prototype

An example of a prototype corresponding with the *Bidding* template in the scenario in Fig. 2 and the screen layout in Fig. 3 is shown in Fig. 4. It shows a typical screen during *prototype simulation*. The upper part contains controls for navigating through the abstract prototype. At the top left one can select a participant with a certain role. In

[4] http://www.xsmiles.org

this case the participant is Mieke and has the role “bidder”. At the next row, the values of all parameters of the current template are shown. The next line similarly shows input fields for the corresponding output, followed by triggers for navigating through the abstract prototype, including a trigger for restarting a simulation and triggers for all transitions to other scenes that are possible in the active context (selected participant and parameter values). The last line of the top part displays the current template and scene. The rest of the screen shows the actual prototype using the CAP notation. User interface components in the abstract prototype can show a tooltip when hovering over them giving concrete information about its function. The tooltip for the abstract component *Ask Question* in Fig. 4 shows a button, *Question*, that triggers a transition to the corresponding scene.

The remainder of this section provides more detail about how XForms and CSS are combined to create the dynamic abstract prototypes. The overall structure of a document describing a prototype is shown in Fig. 5. The document consists of three major parts: (1) simulation related data, including the participants of the simulated show, scenario structure and the applicable constraints, (2) the prototype manipulation controls and (3) the description of the user interfaces associated with the templates.

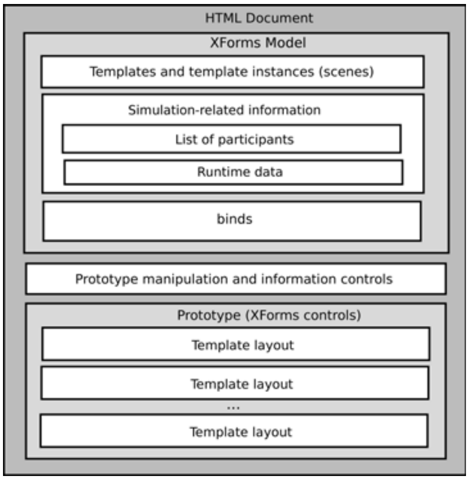

HTML Document
XForms Model
Templates and template instances (scenes)
Simulation-related information
List of participants
Runtime data
binds
Prototype manipulation and information controls
Prototype (XForms controls)
Template layout
Template layout
...
Template layout

Fig. 5. The document structure of a prototype document

Templates and scenes. The template structure is coded into XForms instances and is shown in Listing 1.1. It lists all templates in the same order as they appear in the scenario model described in section 4. Each template has a name and all corresponding scenes appearing in the scenario are described in nested tags. Each scene also has a name and a next scene specification. Notice that the elements *instances* and *next* are used because the XForms processor requires an element to only contain one type of sub-element in order to iterate over them.

Listing 1.1. DTD for template and scenes structure

```
<!ELEMENT scenario (template+)>
  <!ATTLIST scenario name CDATA #REQUIRED>
<!ELEMENT template (instances)>
  <!ATTLIST template name CDATA #REQUIRED>
<!ELEMENT instances (inst+)>
<!ELEMENT inst (next?,params?)>
  <!ATTLIST inst name CDATA #REQUIRED>
<!ELEMENT next (option+)> <!ELEMENT option EMPTY>
  <!ATTLIST option templ CDATA #REQUIRED>
  <!ATTLIST option inst CDATA #REQUIRED>
  <!ATTLIST option conditional CDATA #REQUIRED>
<!ELEMENT params (param+)>
<!ELEMENT param (#PCDATA)>
  <!ATTLIST param name CDATA #REQUIRED>
  <!ATTLIST param input (true|false) #REQUIRED>
```

Choices and iterations are not directly coded into the templates although each instance can have multiple following templates, instead all possible next templates following a specific scene are mentioned. The template *Bidding* in Fig. 2, for example, can have both *Buyer Question* and *End Bidding* as next scenes. The Figure however only shows *Question* as next scene, because no satisfactory bidding is reached yet. Note that also *Bidding* is not listed as an option because a navigation element to the currently active page can be confusing. Furthermore, this is a prototype that has no link to program logic. When the current user in the simulation is no buyer, none of the next options would be shown, because only buyers can trigger a transition to another scene. A warning (on black background) is displayed whenever some transitions could be hidden due to unsatisfied constraints (see Fig. 4).

Bind expressions. This section of the document contains mainly bind-tags that indicate relevancy of navigation controls (the “next triggers” for navigating to other scenes), or user interface components that are only relevant for a certain role. The generation of bind-expressions for the “next-triggers” results in a number of bind tags for unconditional transitions that equals the maximum number of unconditional transitions for a single scene and in one bind tag for each conditional transition. Additional bind expressions are provided to ensure that input and output values for scenes are always displayed correctly.

Simulation related information. All simulation-related data, is also encoded using an XForms instance. The show related information contains all runtime information about the show and a list of participants (with name, role and other related data such as media streams). The runtime information includes the information about the currently active participant, the currently active scene and tags that can be referenced to by controls that are shown conditionally (such as the possible transitions to another template). Among these tags there is one tag for each bind related to a “next trigger”. The relevance, combined with a CSS rule indicating that disabled XML-elements should not be displayed, allows hiding of all irrelevant items.

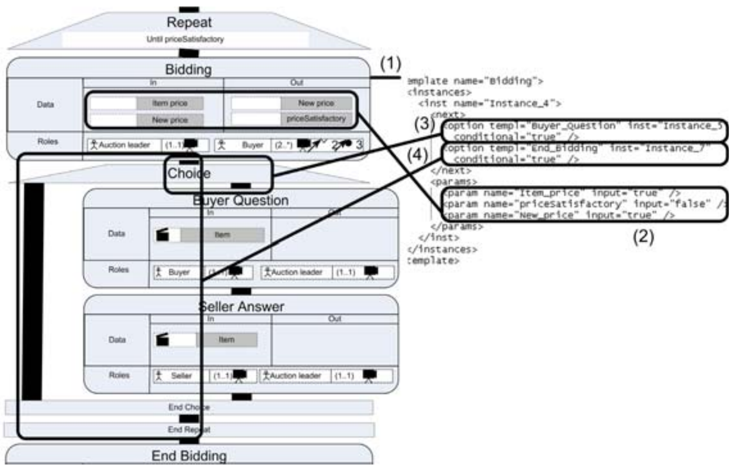
User interface specification. The user interface specification is entirely contained within one XForms group (from now on referred to as *group*), representing the screen. This group contains a set of groups, one for each template. These groups are shown one at a time, depending on the currently active scene as described in the XForms model.

Each group contains XForms controls for each user interface component in the screen model. When controls are only visible to a certain role, they are only made relevant to this role, and consequently only shown to the relevant users. Additional information is shown when hoovering over the controls in the CAP-notation (using an ephemeral XForms message). The FormsPlayer plugin for Internet Explorer allows embedding XForms controls in hints (displayed in most browsers as tooltips) Fig. 4. This enables using XForms outputs, combined with appropriate CSS-rules to show the CAP-notation of the components in the html-page and to show low-level user inteface components in tooltips for establishing real user interaction. In this way, the abstraction can be used to spark creativity, while keeping the interactivity of low-level controls.

6 Generating Dynamic Abstract Prototypes

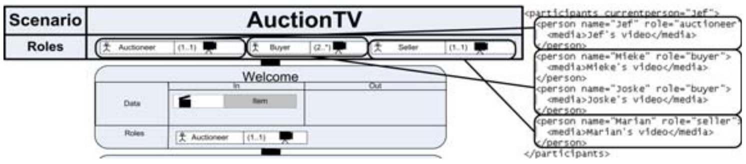
The prototypes can be automatically generated from the models specified in section 4. We will shortly discuss the main aspects of this algorithm in this section: the generation of the templates and template instances section in the XForms model, the generation of a list of participants, and the generation of the prototype's user interface.

Templates and template instances. For each scene in the scenario, the template is added. When a scene is followed by a choice-construct, all possible next scenes are added as options to the list of scenes that can follow the current scene except when the next scene is the current scene. The generation of the scenes is illustrated in Fig. 6



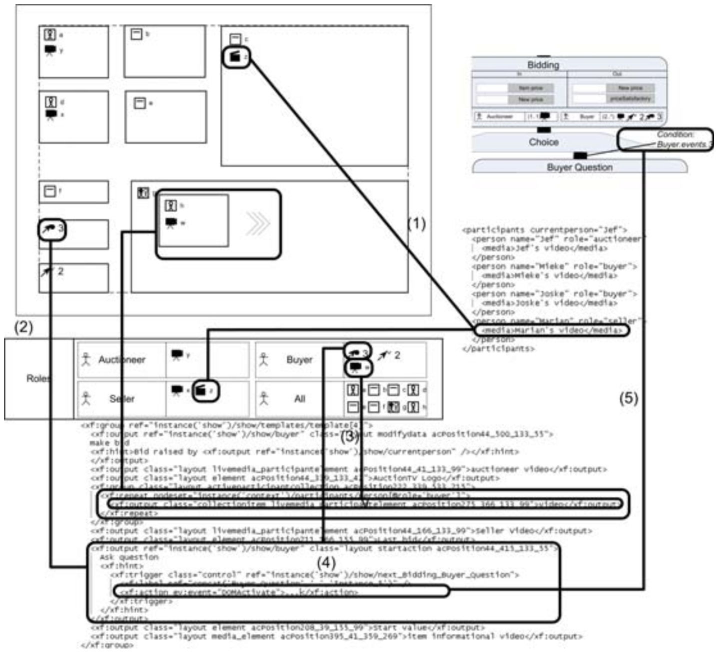
```
emplate name="Bidding">
<instances>
  <inst name="Instance_4">
    <next>
      <option templ="Buyer_Question" inst="Instance_5
        conditional="true" />
      <option templ="End_Bidding" inst="Instance_7"
        conditional="true" />
    </next>
    <params>
      <param name="Item_price" input="true" />
      <param name="priceSatisfactory" input="false" /
      <param name="New_price" input="true" />
    </params>
  </inst>
</instances>
template>
```

Fig. 6. Generation of the templates and scenes: (1) template generation, (2) parameter generation, (3) and (4) generation of next options



```
<participants currentperson="Jef">
  <person name="Jef" role="auctioneer">
    <media>Jef's video</media>
  </person>
  <person name="Mieke" role="buyer">
    <media>Mieke's video</media>
  </person>
  <person name="Joske" role="buyer">
    <media>Joske's video</media>
  </person>
  <person name="Marian" role="seller">
    <media>Marian's video</media>
  </person>
</participants>
```

Fig. 7. Generation of the participants



```
<participants currentperson="Jef">
  <person name="Jef" role="auctioneer">
  | <media>Jef's video</media>
  </person>
  <person name="Mieke" role="buyer">
  | <media>Mieke's video</media>
  </person>
  <person name="Joske" role="buyer">
  | <media>Joske's video</media>
  </person>
  <person name="Marian" role="seller">
    <media>Marian's video</media>
  </person>
</participants>
```

```
<xf:group ref="instance('show')/show/templates/template[4]">
  <xf:output ref="instance('show')/show/buyer" class="layout modifydata acPosition44_500_133_55">
  make bid
  <xf:hint>Bid raised by <xf:output ref="instance('show')/show/currentperson" /></xf:hint>
  </xf:output>
  <xf:output class="layout livemedia_participantelement acPosition44_41_133_99">auctioneer video</xf:output>
  <xf:output class="layout element acPosition44_339_133_42">AuctionTV Logo</xf:output>
  <xf:group class="layout activeparticipantcollection acPosition222_339_533_215">
  | <xf:repeat nodeset="instance('context')/participants/person[@role='buyer']">
  |   <xf:output class="collectionitem livemedia participantelement acPosition275_366_133_99">video</xf:output>
  | </xf:repeat>
  </xf:group>
  <xf:output class="layout livemedia_participantelement acPosition44_166_133_99">Seller Video</xf:output>
  <xf:output class="layout element acPosition211_166_133_99">Last bid</xf:output>
  <xf:output ref="instance('show')/show/buyer" class="layout startaction acPosition44_415_133_55">
    Ask question
    <xf:hint>
      <xf:trigger class="control" ref="instance('show')/show/next_Bidding_Buyer_Question">
      | <xf:label ref="concat('Buyer Question', ' ', 'instance 1')" />
      | <xf:action ev:event="DOMActivate">...</xf:action>
      </xf:trigger>
    </xf:hint>
  </xf:output>
  <xf:output class="layout element acPosition208_39_133_99">Start value</xf:output>
  <xf:output class="layout media_element acPosition395_41_359_269">item informational video</xf:output>
</xf:group>
```

Fig. 8. Generation of the participants and user interface controls: (1) generation of media links for participants, (2) generation of repeated elements which contain media from participants with role *buyer* (3), and (4) trigger that is only visible to buyers and causes a transition (5)

for the template *bidding* of the scenario in Fig. 2. When other media are connected to participants with a certain role, such as the item that the *seller* offers to the auction, can be derived from the screen model. Fig. 8 (1) shows an example.

Participants. The initial list of participants is generated based on the roles present in the scenario heading, while the remaining participants are generated based upon the roles present in the screen models that are linked to the templates that are used in the scenario model. In this case, only roles that can add participants are considered. This means that the following role-types cannot cause the creation of new participants: roles that are already represented in the list, the meta-role *all*, and roles that are created during the scenario (i.e. they are mentioned as output of one or more scenes.) The generation of the participants can be done completely using the heading (see Fig. 7, because all roles are actively participating).

User interface. Most of the user interface generation process is straight-forward; for each abstract user interface component in a screen model, an output-control is generated with the appropriate style and position in CSS. Some triggers can cause a transition between scenes. Fig. 2 shows how this is marked in the scenario. The event-labels and the roles that can cause these events are used as a constraint on the transitions leaving the choice statement. In this case, a XForms trigger has to be generated that causes the transition (see Fig. 8 (4) and (5) for an example.)

7 Discussion and Conclusions

We presented an approach to make early evaluation of a special kind of participation television, staged participatory multimedia events (SPME), possible through the automated creation of abstract prototypes from a limited set of models, defined in a domain-specific modeling language. The modeling language has been succesfully used to express a limited set of participation television scenarios. We believe that despite the abstractness of both the models and the prototype, they are able to express the participation aspect using the icons that were added to the Canonical Abstract Prototype (CAP) [4] notation. Further work will show whether all icons in the original CAP notation are relevant to participation television and whether additional icons are required.

It is our firm believe that using this kind of high-level models through tools allows an effective approach to quickly get a good grip on the required functionality of such an application. We have created proof-of-concept tool support, based on MS Visio and the corresponding SDK, for the notation and prototype generation. The models shown in Fig. 2 and Fig. 3 were created using this tool support in less than half a day including filling in a limited amount of previously unspecified details. These models included enough information to derive a running abstract prototype from those specifications. Despite the fact that our experience is still very limited, we believe that our approach can be used to verify whether all required functionality is being thought of and to create some preliminary results, which do not have to be thrown away, very quickly.

The use of abstraction has the advantage that one is not tempted to spend a lot of time in perfecting details in the early stages of design. The disadvantage is that the resulting abstract prototypes do not have the feel of a more concrete prototype and thus one cannot get feedback about the feel from users. Replacing the CSS-background images with concrete images provides more concrete information when necessary.

The abstract prototypes can now be completely generated from the presented models. We are planning to do some more tests with the defined model types and will extend our model generation approach to incorporate additional models that can provide us with more details about the timing and the user interactions. We will also investigate the generation of more concrete and higher-fidelity prototypes from the presented models complemented with lower-level details and a model providing more information about the interactions from participants with the SPME infrastructure within a scene.

Acknowledgements. This research was performed in the context of the IWT project Participate of Alcatel Bell. Part of the research at the Expertise Centre for Digital Media is funded by the ERDF (European Regional Development Fund), the Flemish Government and the Flemish Interdisciplinary institute for Broadband Technology (IBBT).

References

1. Dick Bulterman, Guido Grassel, Jack Jansen, Antti Koivisto, Nabil Layaïda, Thierry Michel, Sjoerd Mullender, and Daniel Zucker. Synchronized multimedia integration language (smil 2.1). http://www.w3.org/TR/2005/REC-SMIL2-20051213/, December 13 2005.
2. Pedro F. Campos and Nuno J. Nunes. Canonsketch: a user-centered tool for canonical abstract prototyping. In *Proceedings of EHCI-DSVIS 2004*, volume 3425 of *LNCS*, pages 146–163. Springer, 2005.
3. Tim Clerckx, Frederik Winters, and Karin Coninx. Tool support for designing context-sensitive user interfaces using a model-based approach. In *Proceedings of TaMoDia 2005*, pages 11–18, Gdansk, Poland, September 26–27 2005.
4. Larry L. Constantine. Canonical abstract prototypes for abstract visual and interaction design. In *Proceedings of DSV-IS 2003*, number 2844 in LNCS, pages 1 – 15, Funchal, Madeira Island, Portugal, June 11-13 2003. Springer.
5. Micah Dubinko, Leigh L. Klotz, Roland Merrick, and T. V. Raman. Xforms 1.0. W3C, World Wide Web, http://www.w3.org/TR/2003/REC-xforms-20031014/, 2003.
6. DVB. Multimedia home platform. http://www.mhp.org/, 2006.
7. Mohammed Elkoutbi, Ismaïl Khriss, and Rudolf Keller. Automated prototyping of user interfaces based on uml scenarios. *Automated Software Engineering*, 13(1):5–40, January 2006.
8. James Lin and James A. Landay. Damask: A tool for early-stage design and prototyping of multi-device user interfaces. In *8th Internation Conference on Distributed Multimedia Systems (International Workshop on Visual Computing 2002)*, pages 573–580, San Francisco, CA, USA, September 26–28 2002.
9. Giulio Mori, Fabio Paternò, and Carmen Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Sofware Engineering*, 30(8):507–520, August 2004.
10. Nuno Jardim Nunes and João Falcão e Cunha. Towards a uml profile for interaction design: the wisdom approach. In *Proceedings of UML 2000*, volume 1939 of *LNCS*, pages 101–116. Springer, October 2000.
11. Fabio Paternò. Towards a uml for interactive systems. In *Proceedings of EHCI 2001*, pages 7–18. Springer-Verlag, May11–13 2001.
12. Andreas Pleuss. Mml: A language for modeling interactive multimedia applications. In *Proceedings of Symposium on Multimedia*, pages 465–473, December12–14 2005.