Multimedia Systems manuscript No.
(will be inserted by the editor)

Jan Van den Bergh · Bert Bruynooghe · Jan Moons · Steven Huypens · Bart Hemmeryckx-Deleersnijder · Karin Coninx

Using High-Level Models for the Creation of Staged Participatory Multimedia Events on TV

Abstract Broadcasted television shows are becoming more interactive. Some broadcast TV shows allow even home viewers without professional equipment to be part of them. In this paper we present an approach that takes this concept even further. In the proposed kind of participation television viewers will not only participate in the show through interaction or video streams, but also be able to create and host their own show. The core of the presented approach consists of the use of high-level models to describe the different aspects of the television show, and a generic runtime environment. This paper discusses this type of participation television, Staged Participatory Multimedia Events, and the supporting runtime environment in more detail. It also introduces the tool and the models that are used to support graphical creation of the structure and appearance of Staged Participatory Multimedia Events.[1]

1 Introduction

In recent years the degree of interactivity of television programs has steadily increased. When viewing a television show one can participate in it by voting or playing along. In some TV-shows (such as "De ThuisPloeg" in Flanders) this participation can mean that participating viewers are invited to the studio for the next episode or get a professional camera at their home. The growing availability of high-quality live streaming video enables integration of live video from viewers into TV shows. These live video streams are obtained via video phones (in the TV-show "Matina" in Italy) or camera-enabled mobile phones (in the TV-show "CultTV" in France). "Ze Live" is interactive program of one hour where youngsters can participate via audio calls, webcam calls or mobile video calls. It was broadcasted on Plug-TV, a youngster TV channel of RTL in Belgium.

At the same time there is a growing trend of end-users creating their own content and making it available to the general public. Users have been able to easily produce their own content and share it with others worldwide on blogs, sites such as MySpace and YouTube. Social networks as well as traditional media coverage can spread the users' creations to thousands of people around the world in a very short time.

These two trends are combined when end-users can create and participate in television shows that are broadcasted or multicasted on digital TV channels. Our research aims to establish this goal for a well defined subset of interactive television events with rich user participation. We call these events Staged Participatory Multimedia Events (SPMEs).

SPMEs are events whose director and participants can all be television viewers. In this paper, we only discuss the system intended for television sets, although the usage of other platforms is also supported. Table 1 gives an overview of all supported devices. One example SPME, AuctionTV, will be used throughout the paper. It is a live auction in which all participants, including the auctioneer and the seller view the event through their television set and will be discussed into more detail in section 3.

Internet service providers or broadcasters may use SPME as an added value to their existing services by including them into their interactive television offering (e.g. in a so called *walled garden*). This however requires that the technical knowledge to create and adapt these events is minimal and that the runtime environment can be integrated into the existing infrastructure.

After a discussion of related work, section 3 describes the structure and properties of such shows into more detail and introduces an example. Section 4 then discusses the ParticipationTV runtime infrastructure and the format description used to configure the flow of the SPME. This format description can be generated from graphical models. These

Jan Van den Bergh, Steven Huypens, Karin Coninx
Hasselt University – transnationale Universiteit Limburg
Expertise Centre for Digital Media – Institute for BroadBand Technology
Wetenschapspark 2, 3590 Diepenbeek, Belgium
E-mail: {jan.vandenbergh,steven.huypens,karin.coninx}@uhasselt.be

Bert Bruynooghe, Jan Moons, Bart Hemmeryckx-Deleersnijder
Alcatel-Lucent
Copernicuslaan 50, 2018 Antwerp, Belgium
E-mail: {bert.bruynooghe,jan.moons,bart.hemmeryckx-deleersnijder}@alcatel-lucent.be

[1] The original publication is available at www.springerlink.com

Platform	Input device	Streaming from C to S	Streaming from S to C	Application type
PC	Mouse *remote control* *game controller*	*RTP*	*RTSP*	Local application
	Browser controls remote control (mapped to keyboard)	RTMP	*RTSP*	Webpage using *Quicktime* Flash and *AJAX*
Set-top box	*remote control* *browser controls*	*SIP*	RTSP	Web page *proprietary webcam/microphone streaming* *proprietary RTSP playing* and AJAX
		Device Dependent	Device Dependent	Proprietary Solution
		Signal for IP-webcam / microphone	Broadcast signal	MHP + parallel return stream channel
Game console	Game Controller	Device dependent	Device dependent	Proprietary application
Mobile phone	MIDLET controls	SIP, ...	SIP, *MMS*, ...	Local application
	DTMF handled at server side	*SIP*	*SIP*	*SIP service attached to SIP number*

Table 1 Overview of all target platforms for SPME and the corresponding streaming technology from *C*lient to *S*erver and vice versa. Input devices, application types and streaming protocols that are already implemented and tested are mentioned in italics

models are discussed in section 5 followed by the discussion of the tool support. The paper ends with the presentation of conclusions and future work.

2 Related Work

There is a body of work regarding participation TV besides the effort discussed in this paper.

Inhabitated TV [2] bundled some of the experiments with a big participatory role of end-users in broadcast television (and cinema). Inhabitated TV featured a collaborative virtual environment (CVE) in which different layers of interaction and awareness were possible. All people involved could be divided into three groups: performers, inhabitants and viewers. The performers were immersed in the CVE through specialized equipment, while inhabitants could navigate, interact and communicate in the CVE using a standard internet connection and equipment. Finally, the viewers of broadcast TV had only limited interaction capabilities. Although the shows allowed participation in the show, the lack of rich expressions of the characters in the CVE made it difficult for viewers to associate themselves with the performers.

Such richer expressions were possible in some shows broadcasted later on, where live video streams from viewers were integrated in the show. Examples of such shows are CultTV in France and Mattina in Italy. The latter uses the Mycast system from Digital Magics [2] to integrate live feedback from viewers from videophones and webcams into the daily morning show by Rai Uno, a national TV station in Italy [9].

[2] http://www.digitalmagics.com

OpenTV offers a tool and framework that allows creation of Participation TV [3] without coding. The framework is focused on enhancing existing television shows and formats with interactivity and extended statistics and does not allow the advanced viewer-driven shows which integrate live video-feeds as we presented in this paper.

TVML [4] has a different focus and allows to easily create 3D non-interactive TV shows using a simple scripting language. SoapShow [5] allows users to create their own soap online using video, still images, text and sound clips. Currently it's limited to the web, but SoapShow is planning to show the best soaps on Dutch television.

Telebuddies [13] is a proof-of-concept framework that gives a social dimension to existing shows. It allows viewers of a television program, which are spread over different locations, to chat with one another and to play against each other in automatically composed groups based on similar interests. While both the telebuddies framework and SPME allow a social television experience between people at different locations and could both be used to augment *existing shows*, SPME are primarily television programs in their own right and can be used for both small and large audiences. In contrast to telebuddies, SPME can be described using a generic XML language (see section 4.2) and created using graphical models.

Finally, a different form of user participation is investigated within the project NM2 where television viewers can influence storylines. These storylines and the possible interactions are modeled using the Narrative Structuring Lan-

[3] `http://www.opentvparticipate.com/`
[4] `http://www.nhk.or.jp/strl/tvml/`
[5] http://www.soapshow.nl

guage [18]. This language deals only with the possible plots and interactions and the related media fragments. Layout of these elements cannot be specified. The way the interaction is performed is described in less detail in NSL than in SpIeLan as discussed in this paper. Their runtime system has however been used in a television show in Finland where viewers could influence the story by sending SMS messages.

3 Staged Participatory Multimedia Events

Staged Participatory Multimedia Events are broadcasted (for large audiences) or multicasted (for small communities) interactive television events that actively engage TV viewers and turn them into true participants. They provide a stage for viewers to participate in interactive television applications that are not necessarily publicly or commercially available yet. The basic requirement for viewing an SPME is having a television set.

In the classification of interactive television genres and formats composed by Jensen [12] and shown in table 2, SPME are targeted to support the "Games and betting" and "T-commerce" categories within a *walled garden* environment as shown in Table 2. The SPME technologies can however also be used in an enhanced TV setting. SPME modeling techniques and runtime technology might even be used to realize the interactive part of interactive advertising. This path however would need further investigation.

Format	SPME
Electronic Program Guides	-
Enhanced TV	x
Video-on-demand	-
Personalized TV	-
Internet TV	-
ITV advertising	-
T-commerce and home banking	X
Games and betting	X

Table 2 Interactive Television Formats and SPME

In order to participate communication from the TV (as a client) towards the server is needed. This up-link can be established through a set-top box with remote control as interaction device. Mobile phones can contact the server using text messaging or MMS. Also game consoles deliver this kind of connection capabilities to the Internet or any other broadband network today. Total participation can involve devices as webcams, microphones and other interaction devices, which also demand higher speed connections.

The format of an SPME can be defined using the graphical modeling language discussed in section 5.2 or directly using the XML-language that is used as a blue print of the show (see section 4). Show participants can have different interaction capabilities based on their *role*. Some roles may have hardware requirements such as a webcam and microphone. Each format allows all participants to chat with one another. Since the creation of an attractive format requires some technical skills, the creation of these formats will probably be accomplished by professionals or prosumers, while customizations are expected to be made by a broader audience.

An SPME is started when the first viewer activates a format. This first viewer does not only start the show but will also become its director, identified by the role *master*. All other viewers that join the active format initially get the role *participant*. The show is driven by viewer interaction or time-based events if desired. The format, however, determines the actions that can be performed by viewers based on their role and the actions that have already been performed. Examples of such formats can be a distributed auction or a quiz in which not only the public, but also the candidates and even the quiz master are TV viewers.

AuctionTV is an example of such a format and will be used for illustration throughout the paper. The format allows one of the television viewers to offer an item for sale through an auction. Media about that item is made available on the web before the start of the show. A (sub)set of the participants in the show are part of the show through live video streams. These live streams originate from webcams that belong to the television set. In the realized proof-of-concept, only the auctioneer and the seller are shown using live feeds. Another version of the AuctionTV format could also show all bidders.

The auctioneer, who can be any television viewer, can start a session on a specialized channel, offered by a media company such as a broadcaster or an internet service provider. Once the auctioneer has started the show, he gets the role *master* and other viewers of that channel can join the show. They will initially get the role *participant*. One of these participants can offer an item for sale and as such becomes the seller. The seller does no longer have the role *participant* but gets the role *seller* instead. Then the auctioneer initiates an interview with the seller, followed by the bidding process. Whenever a participant *p* bids, the auctioneer raises the price, confirming the bid. In doing so the role *winner* is added to the roles of *p* and removed from the previous bidder (if there was one). When an acceptable bid has been made and confirmed, the bidding process is ended by the auctioneer. The auctioneer finally does an interview with the winner of the highest bid. As long as no bid is made, the auction can be cancelled by the auctioneer, just as all other viewer actions, using a remote control (see Fig. 1).

The AuctionTV scenario demonstrates some of the technical properties that all SPME have in common:

1. Viewers are the centerpiece of SPME. They start and stop the show and control everything in between. The AuctionTV example is even completely driven by viewer actions. Although this need not be the case as timer events are also supported.
2. The person that starts the show is the show master and is in control of the show. All important decision regarding the flow of the show are made by the master. All other viewers of the show are initially participants.

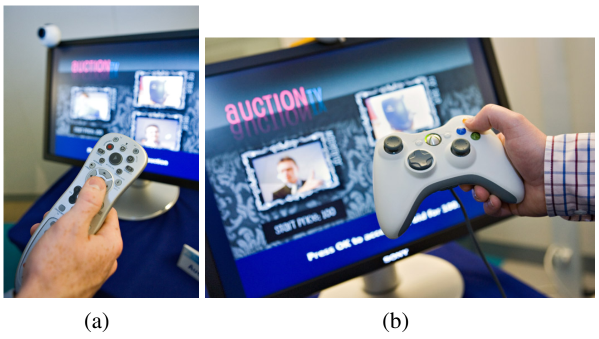


Fig. 1 AuctionTV application used with different interaction methods/devices (a) a TV remote control (b) a game controller

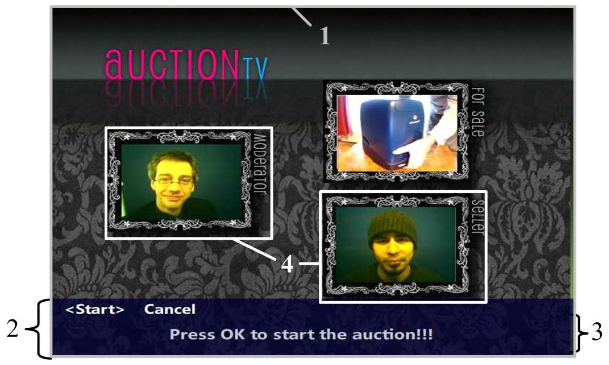


Fig. 2 Example structure of the screen contents of an SPME: (1) *background panel* featuring live streams of participants with different roles (4) and an *interaction bar* (2) containing the names of the available *interaction panels* as well as the content of one of these panels, in this case having label `Start` (3)

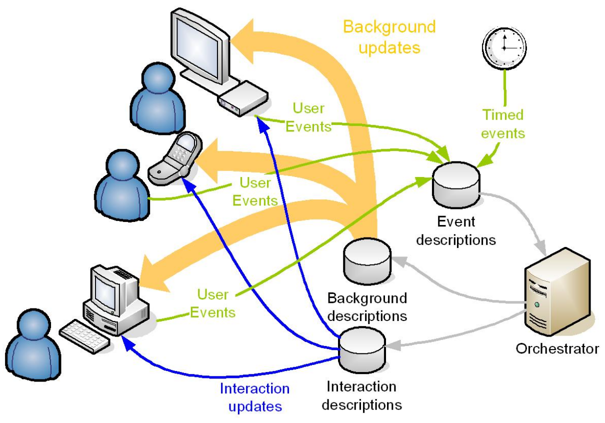


Fig. 3 Usage of SPME XML parts in the ParticipationTV runtime

3. Each person watching an SPME has one or more roles. The roles played by a person can change over time. E.g. one of the participants in the AuctionTV format gets the role seller and loses the role participant.
4. As viewer participation is key to SPME, visual and auditive integration of the key players in the broadcasted or multicasted media stream is important. Viewers can thus be part of the show through live video (and audio) streams provided by webcams as illustrated by Fig. 2.
5. SPMEs have a script; the consequences of viewer actions have predefined (predictable) results.
6. All SPME formats have a similar screen structure, which is illustrated in Fig. 2. The *background panel* consists of non-interactive content and is shared by all viewers whatever their role is.
7. On television sets, all interactions are presented in interaction panels or popups. Interaction panels are the preferred medium on television sets to present information and interaction capabilities, specific for one or more roles, to the appropriate viewers.
   Each interaction panel and popup allows the viewer to perform one task such as making a bid, setting the price of the item to be sold, or start the interview with the winner (a transition to a new screen layout). The interaction panels are layered on top of each other in one designated area of the screen, the *interaction bar* (see Fig. 2).
   A *popup* temporarily disables access to all interaction panels. A popup thus has behavior similar to that of a modal dialog box on a desktop computer. Furthermore, it allows the designer to escape the rigid form of the interaction panels for some important interactions or role-specific information.
8. Highly different interaction methods or devices can be used (see Fig. 1), even when only television sets are considered as a platform.

4 The SPME runtime support

The SPME runtime support consists of two parts: a description of the SPME using XML (from now on called SPME XML) and a generic runtime environment that interprets this description. The relation between SPME XML and its use in the Participation TV runtime is shown in Fig. 3, which shows only the key server-side component, the *Orchestrator*. It receives all events related to SPME. These events are generated by user interaction through a supported client system or by a timer associated to a SPME. It then ensures that all necessary actions are performed and that all user interfaces are updated. These updates can involve changing backgrounds or interaction panels. More information about the infrastructure is given in section 4.1, while section 4.2 discusses the different parts of SPME XML.

4.1 The runtime infrastructure

The ParticipationTV infrastructure consists of four major components: the Orchestrator, the VideoMixer, the ParticipationTV Service and an Instant Messaging server. Fig. 4 shows how these components interact. The figure clearly shows that the instant messaging (IM) server plays an important role in the infrastructure as it handles all communication between the participants and the Orchestrator, which handles the communication with the ParticipationTV Service.

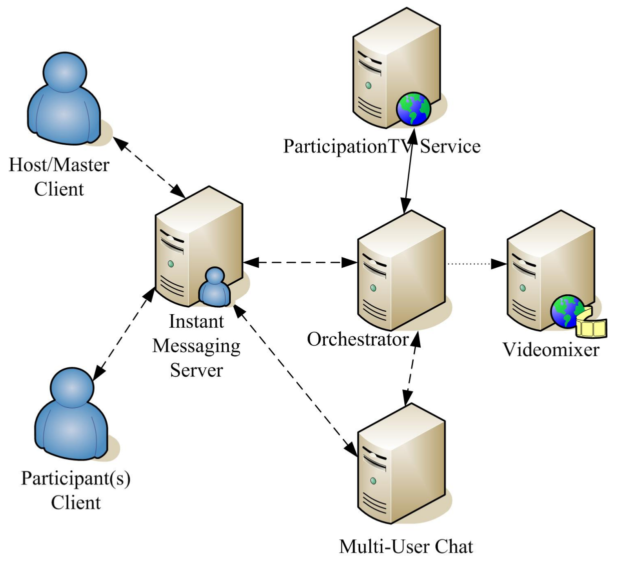


Fig. 4 The ParticipationTV Runtime infrastructure. Dashed connections use the XMPP protocol, the dotted connection uses proprietary XML-based communication and continuous line connection uses a webservice protocol.

The latter holds the list of all available SPME formats, while the Orchestrator deals with all communication related to the SPMEs while they are executing. The creation of the mediastreams and the transmission to the viewers of the SPME is handled by the last component, the Videomixer.

The choice for an IM server to handle all communication with the SPME viewers is based on the capabilities of the eXtensible Messaging and Presence Protocol, short XMPP [16, 17]. The protocol is based on XML, which means that it is easily extensible with ParticipationTV's own XML-based SPME language. ParticipationTV clients or server-side components that need to send ParticipationTV specific messages to each other can simply do so by embedding them in an XMPP message and the IM server will make sure that they will arrive at the corresponding entity. Furthermore the presence part makes it easy to see who of your contacts are online and you can invite them to play a TV program that you made or host, although people can also join a ParticipationTV program if they know the format and host.

The actual core of ParticipationTV is the Orchestrator, a server-side XMPP component that plugs into the IM server. The Orchestrator has a variety of functions: keeping track of ParticipationTV format instances and sessions, making sure participants have the correct roles, handling workflow, sending the right interactive components to all participants and steering the video mixer (see also Fig. 3).

Furthermore, the Orchestrator has a crucial role during the start of a SPME. Before a user can start a new SPME or join an already started SPME, the user has to log into ParticipationTV, which will send an XMPP message to the Orchestrator component to ask for the available formats. The Orchestrator will make use of the ParticipationTV web service to retrieve the list of formats. After the user has selected a format and states that he wants to be the host, the Orchestrator will exercise the following steps:

1. Load the corresponding SPME XML files into its workflow engine.
2. Create the new session and inform the web service of the newly created session.
3. Attach the role *master* to this user.
4. Start an instance of the video mixer.
5. Create a room in the Multi-User chat component [7] of the IM server.
6. Send a message back to the user that everything is set up correctly and that he can join the Multi-User chat component. All participants of the same format instance will end up in the same Multi-User chat room.

All other users that subsequently log in will be able to join the format instance/session created by this first user. The workflow engine that is part of the Orchestrator will now start to process the SPME XML files and will act accordingly:

1. Steer the video mixer, which composes the correct video image for all users.
2. If there is interactivity, make sure the user with the correct role will receive it and can act upon it.
3. Go through the flow until the format has finished or stopped by the user who has the role *master*.

4.2 The runtime SPME description

The runtime SPME description (SPME) consists of three parts (as shown in Fig. 3): the *flow description* (using events), the *background panels* and the *interactions*, shown as interaction panels and popups on television sets.

The flow description is a list of named *events* and the according event descriptions, which consist of a set of *commands*. The current set of commands relevant to each role is maintained in the state machine of the orchestrator to enable consistent view on the show for all participants, even when they join lately or are accidentally disconnected. There are three kinds of commands:

– The first category of commands applies to variables and roles. Both variables and roles are maintained in a Jscript [6] environment and can be resolved to strings to be used in the layout files. Variables will typically be used for the dynamic behavior of the show, and do support basic Jscript evaluation, while roles have to be considered as dynamic groups of users, which are typically used for role dependent content and interactions (popup dialogs and interaction panels). The first set of commands in Listing 1 fall into this category.
– The second category consists of a single timing-related command *setTimer*: upon execution, it starts a timer that spawns a new event when it times out.

[6] `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/jscript7/html/jsoriJScript.asp`

Listing 1 SPME XML - example commands

```
<AddToRole user="#INTERACTOR#"
    roleName="Seller" />
<RemoveFromRole user="#INTERACTOR#"
    roleName="PARTICIPANT" />
<SetVariable name="RequestedBid"
    value="100" />
<SetVariable name="RequestedBid"
    value="$RequestedBid$+100"/>

<SetTimer associatedEvent="NextScreen"
    expirationTime="10000" />

<SetBackground panelName="Intro" />
<AddInteraction user="#PARTICIPANT#"
    panelName="Sell"
    associatedEvent="BecomeSeller" />
<RemoveInteraction user="#MASTER#"
    panelName="Cancel" />
<ShowPopup user="#MASTER#"
    panelName="AckBid"
    associatedEvent="MasterAckBid"/>
```

– The last category of commands changes the layout of the screen and the interaction possibilities of the different users: a new *background panel* can be assigned, *interaction panels* can be added and removed, and *popups* can be displayed. Interaction panels and popups also have an associated event, which the user will spawn when he presses OK when the interaction panel or popup is active. These graphical items are designated by a *panelName*, which maps to a description in the corresponding layout files. Examples of this category of commands can be seen in the lower part of Listing 1.

The remaining parts of SPME XML concern the specification of the user interface, the *background panel* and the *interactions*. Originally, both were described in a ad hoc manner MyXaml [7], an open source language and library to describe GUIs of Windows applications, together with some terminal dependent custom controls. This offered the advantage that we could reuse the MyXaml libraries and quickly test the usage of the platform in the spirit of X'treme Prototyping [14, chapter 4]. Later on, we realized that many items in scenes were coming back several times and we needed dynamic scene changes, which were not obvious to describe in MyXaml.

In the current implementation, the description of the *background panel* is done based on SMIL [3], which allows dynamic positioning of text, video and images, while also allowing control of the audio. It also offers the possibility to separate the content description (body) from the actual layout and styling (regions and paramSets), which enhances the readability and reduces viscosity. The dynamic scene progression is described using the par and seq tags from SMIL, where the former are triggered by the events.

For the interactions, we plan to use a language on a higher level of abstraction than that of MyXaml, which we are still using today. This with the eye on supporting SPME on other platforms than television sets, such as mobile phones and PC's. It should thus be a language that is terminal independent, and describing the interaction itself rather than how it should be rendered. Based on our current tests, we already came up with some basic types of interaction: acknowledgment, multiple choice question, number input and plain text input. Listing 3 shows examples of such specifications using a preliminary XML syntax.

Listing 2 SPME XML - partial background specification using SMIL

```
<par trigger="StartBid">
   <text region="bidText"
      src="data:Bid: $RequestedBid$"
      trigger="StartBid, NextBid"/>
   <video src="webcam:#MASTER#"
      region="masterWebcam"/>
   <audio src="microphone:#MASTER#"
      region="masterWebcam"/>
   <img src="auctiontv/sellerinterview.png"
      region="sellerBg"/>
   <video src="ItemPresentation_SGI.avi"
      region="itemVideo"/>
</par>
```

Listing 3 SPME XML - envisioned interaction descriptions

```
<ack statement="Make a bid" name="AckBid"/>
<question name="Answer"
      desc="Which one is more fun?">
   <item value="Mont Ventoux"/>
   <item value="Kemmelberg"/>
</question>
<number name="NextBid" min="0" max="200"
    step="5" desc="What will be the next bid?"/>
<text name="NameInput"
    desc="Give your name:"/>
```

The applicable interactions should be communicated with the user in a form that is optimized for the type of terminal that is used. On PC, it may be rendered as a clickable interface allowing keyboard input, while on set-top box it needs an interface controllable with a remote control. On mobile devices, we can think of text to speech and vice versa in order not to overload the tiny screen.

5 Modeling SPME

5.1 Requirements

To support the creator of an SPME, who is not necessarily a professional with a deep understanding of the implementation issues but rather a knowledgeable viewer, a graphical modelling language with tool support has been constructed. The tool allows the creator to model an SPME using the graphical language and save it in the XML format that can be executed by the SPME runtime. This avoids the overhead of writing SPME XML by hand, which would be a challenge

[7] `http://www.myxaml.com`

for non-technical users not familiar with the implementation constructs of the SPME runtime.

To understand some of the choices that were made, we first present the requirements we set for the language:

R.1 : Supporting the creation of SPMEs; from concept development and concept prototyping over production and testing to prototype deployment [8];
R.2 : Independence of the final interaction devices, since these might be different for different users and not known in the early design phases;
R.3 : Emphasizing the user participation and their presence;

From these requirements we can derive a fourth requirement:

R.4 : Having a level of abstraction that allows the specification to be used during the concept development and prototyping stage, where attention to details about layout and graphic design should be avoided but not necessarily omitted.

Requirement *R.4* provides the required flexibility toward the deployment stage: the decision which platforms and interaction devices to target can be postponed until late in the process but does not hinder early testing [21] (*R.1*). Including other or new interaction devices does not require any changes apart from filling in device specific details at the final artefacts of the design process. As such, requirement *R.4* is directly related to requirement *R.2*.

With these requirements in mind, we created SplеLan (*SP*ME *i*nterface *lan*guage), a graphical modeling language for the design of SPME. It consists of three models: the *scenario*, the *scene stage* and the *scene script*. It is important to note that the names and the contents of these models have been slightly changed to better fit the updated requirements of the runtime infrastructure discussed in section 4 since earlier reports on SplеLan [21,22]. The visual syntax has also been updated, based upon an informal test with both programmers and non-programmers, people familiar with participationTV and people that were not familiar with it [19].

5.2 Scenario model: the overall scenario

The scenario model provides a high-level overview of the show format: it provides general information about the roles that viewers can have during the show and the scenario flow. A scenario is composed of a set of *scenes*. A scene is a contiguous part of the show during which the background panel (see section 3) does not change. The structure of the show is organized into a main flow, which can branch out if necessary.

Fig. 5 shows a scenario model for the format *AuctionTV*. At the top of the diagram the header contains general information about the format and the roles that viewers can play. In Fig. 5 there are four roles. Two of these roles (*master* and *participant*) are always present in an SPME specification since the master is the one that initiates the show and all other viewers that join initially get the role *participant*. Due to the definition of the role *master* there is only one user at any given time with this role. For each role, the tool also displays the media requirements. These requirements can be the availability of audio and video capturing and streaming equipment and possibly the availability of media fragments on the server (such as a video about the auctioned item for the seller). Note that each role is shown in a different color, which is also used for the visualization of the concepts associated with the role (such as events and media requirements).

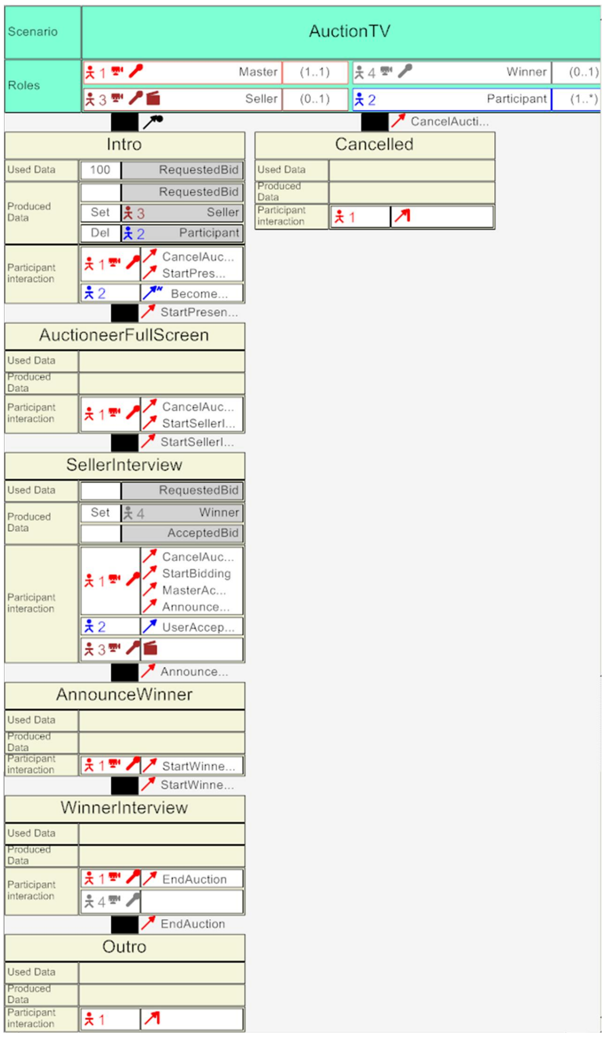


Fig. 5 Scenario of AuctionTV SPME

The first scene in the AuctionTV scenario in figure 5 is *Intro*. The model shows that at the start of this scene the variable *RequestedBid* with initial value *100* is used and set by a participant. This participant loses its initial role but gets

[8] More information about these stages of interactive TV shows can be found in [10].

Icon	Name	Semantics
↗	StartScenario	Starts the SPME
↗	EndScenario	Ends the SPME
↗	Event	Performs a simple action not defined by the other types
↗	SelectValue	Selects a value from a predefined set (currently only strings)
↗	EditValue	Edits a value (currently only numbers)

Table 3 Types of user-generated events in an SPME

the role *Seller* instead [9]. The master of the show, the auctioneer, can also perform an action (*StartPresenting*). This action triggers the following scene because it is also shown next to the connection to that scene.

The auctioneer (the *master*) gives an introduction to the auction in the second scene. The show then continues with an interview with the seller and the bidding process. In this process *participants* can accept a bidding price (event *UserAcceptBid*, variable *AcceptedBid*) set by the auctioneer. The auctioneer then announces the *winner*, a role that has been set during the bidding process, in scene *AuctioneerFullScreen*. When the *winner* is announced, the auctioneer starts an interview with the winner (scene *WinnerInterview*) and finally the show is ended.

There is however an alternative flow possible as long as no bid is made. This fact cannot be completely derived from the scenario model but is described in the script of the scene *SellerInterview* that will be discussed in section 5.4. The auctioneer starts the alternative flow when he decides to cancel the show (using the event *CancelAuction*). This event also appears at the start of the second flow, which consists of one scene, *Cancelled*, which means that this event starts the alternative flow.

The AuctionTV scenario contains different types of events. All events that are currently supported by SPME are shown in Table 3. The symbols for these events are taken from the Canonical Abstract Prototypes notation [6]. The semantics of these symbols is kept the same, although it is more restricted and more specific to SPME. The symbols for these events are abstract because highly different remote controls could be used to interact with the SPME, even when only the TV is considered as a visual medium, as can be seen in Fig. 1. A time event can also be used as a trigger for scene changes and is represented by a stylized sand glass.

As the AuctionTV scenario demonstrated, alternative scene flows are possible. Should this not be sufficient, control structures can be used. Discussion of the control structures is however out of the scope of this paper.

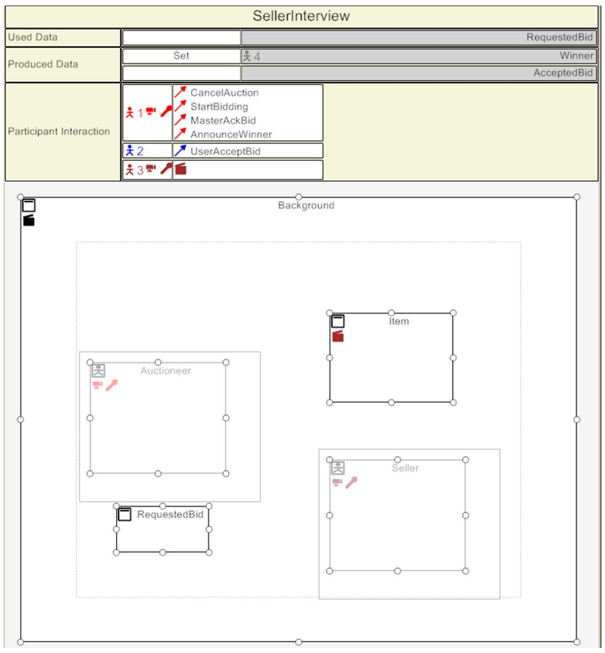


Fig. 6 Scene stage specification

5.3 Scene stage: the screen layout

The second model specifies the *scene stage*; the arrangement of the user interface controls on the television screen. The model consists of two required parts. The first required part is the scene header which contains the name of the scene, the roles actively involved in the scene, and the data used and produced during the scene.

The second part describes the screen layout. It completely specifies the *background* as specified in Fig. 2. In contrast to earlier versions of the model [20], the layout of the *interaction bar* and *interaction panels* is no longer explicitly specified in the scene stage. The representation of the user interface controls is based on the Canonical Abstract Prototypes notation [6], although some changes and additions have been made. Both parts are shown in Fig. 6.

The most important changes for SPME were the introduction of two additional types of *abstract components* (see Table 4): the *ParticipantElement* and the *ActiveParticipantCollection*. The first is an abstract representation of presence information of a single person participating in the show (such as a live video stream), while the second handles a group of people of which one or more can be *active*. Persons that are in the *active* state are highlighted. The *ActiveParticipantCollection* could, for example, be used in the AuctionTV scenario to show all participants that have made a bid. In which case, active could mean "having made the last bid".

Other changes give more details about more concrete things that are important for SPME: the type of media that is used is indicated using additional icons. A camera for live

[9] A person can only affect his own role(s).

Icon	Name	Semantics
⊟	Element	Displays media or text
☰	Collection	Groups a set of media or text elements
⊡	ParticipantElement	Displays media about a viewer
⊞	ActiveParticipant-Collection	Displays media about a group of viewers

Table 4 User interface controls for SPME

video streams, a microphone for live audio streams and a clapper board for pre-recorded media. All of these icons can be seen in Fig. 6. The ParticipantElement *Auctioneer* has both live video and audio, while the element *Item* shows a recorded media-stream. They are shown respectively in the middle-left and middle-right of Fig. 6. The safe zone (the zone that should be visible on all TV sets) is indicated using a dashed rectangle, while a thin-lined rectangle can be overlaid on ParticipantElements and elements to indicate the size and position of a mask image that is associated with these controls.

5.4 Scene script: the interaction

The third and last model specifies the scene script; it specifies which actions can be performed by the viewers in which order and what they result in. This section offers an overview of the model that should be sufficient to roughly understand the semantics. For additional details about the semantics of this model we refer the interested reader to [22], which discusses an earlier version of the scene script. The semantics between that version and the current version are, however, limited.

The scene script has a similar structure as the scene stage. The header is the same as that of the scene stage (see Fig. 6) and contains the same information as the scene in the scenario model. The remaining content of the model differs. An example of this contents is shown in Fig. 7. It shows that in the scene *Bidding*, the *master* can initially see two interaction panels (with associated events *StartBidding* and *CancelAuction*). The event *CancelAuction* ends the scene and hides the corresponding interaction panel, while *StartBidding* enables the *participants* to make a bid (event *UserAcceptBid*). When a bid is made, the auction can no longer be canceled because the event *UserAcceptBid* removes the interaction panel *Cancel* from the interaction bar of the *master*. The *participant* that makes the bid gets the additional role *winner*. In addition to this, the value for *AcceptedBid* is set. The master can then set a new bidding price (*AckBid*) or end the bidding process (*AnnounceWinner*).

The example shows that the basic building blocks in the scene script, *SystemActions*, correspond to interaction panels. The name of the interaction panel is shown at the top of the rectangular shape that represents the SystemAction.

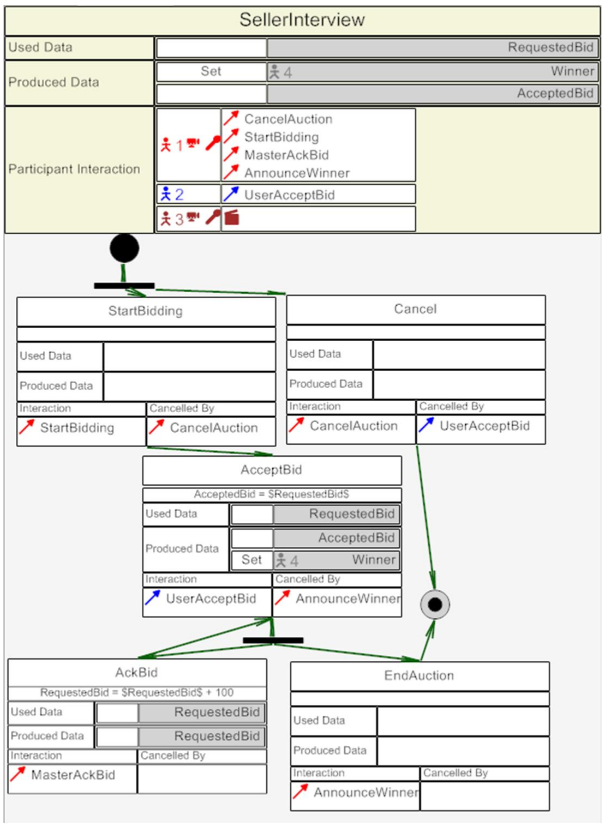


Fig. 7 Scene script corresponding to the header in Fig. 6

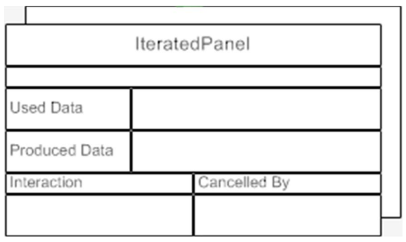


Fig. 8 IteratedSystemAction

When the system performs an action, such as modifying variables, this action is specified below the SystemAction's name. The data that is used and produced by this action is shown in the middle, as are changes in roles. At the bottom, the related events are displayed. At the left side the event is shown that can be triggered through the interaction panel, while the event at the right side will hide the interaction panel without triggering the related functionality.

There are two types of *SystemActions* that can be used in the scene script. The simple *SystemAction* used in Fig. 7 and the *IteratedSystemAction* (see Fig. 8). The latter differs from the first in that it allows multiple persons to trigger the same event (each triggered event causes the execution of the associated functionality). The designer can specify how many people have to trigger the event before the execution of the *IteratedSystemAction* is finished. This amount corresponds

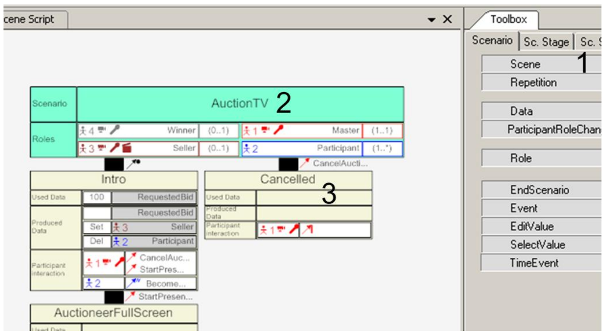
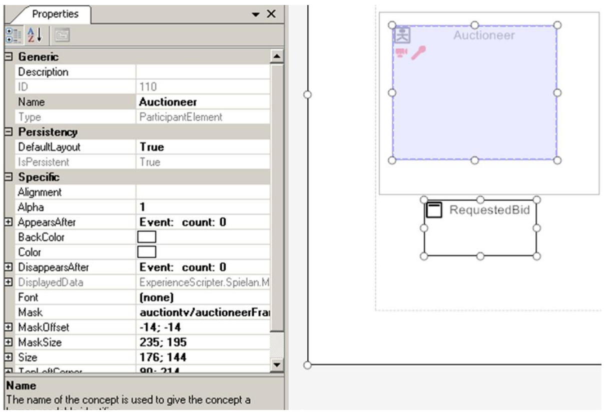


Fig. 9 Procedure to add a scene

by default to all people sharing the role associated with the trigger event, but an absolute amount can also be used.



Generic	
Description	
ID	110
Name	**Auctioneer**
Type	ParticipantElement
Persistency	
DefaultLayout	**True**
IsPersistent	True
Specific	
Alignment	
Alpha	**1**
AppearsAfter	**Event: count: 0**
BackColor	☐
Color	☐
DisappearsAfter	**Event: count: 0**
DisplayedData	ExperienceScripter.Spielan.M
Font	**(none)**
Mask	**auctiontv/auctioneerFra**
MaskOffset	**-14; -14**
MaskSize	**235; 195**
Size	**176; 144**
TopLeftCorner	**90; 214**

Name
The name of the concept is used to give the concept a

Fig. 11 A language construct has a graphical representation and associated properties with comments. The highlighted concept (named `Auctioneer`) is a *ParticipantElement* whose properties are shown in the Properties grid at the left side.

6 Tool Support

SpIeLan is supported by a custom-built tool, called *Experience Scripter* (Fig. 10). The tool has a layered design; the front-end, which was realized using the Piccolo-toolkit [1], contains all information regarding the concrete visual syntax, while the back-end consists of the abstract syntax and supports the model serialization to various formats. Both layers are loosely coupled as the back-end never directly invokes functionality of the front-end. Changes in the back-end are communicated to the front-end via an asynchronous publish-subscribe mechanism. The following two sections provide more details about the front-end and the back-end respectively.

6.1 Design of the front-end

Since not all envisioned users of the tool will be programmers nor have highly technical skills, we tried to follow design guidelines for end-user development (EUD) such as those that were discussed by Repenning and Ioannidou [15] as closely as possible during the design of the Proof-of-Concept tool. We will shortly discuss how these guidelines reflect into the tool we designed.

In our tool we tried to make most syntactic errors impossible. Scenes can only be connected to other scenes or to the header. For example, one can add a scene after an existing scene by selecting *scene* from the toolbox and then clicking on the existing scene. Adding a scene before the first scene, such as *Intro*, of a flow can be done by first clicking on the scenario header and then on *Intro* (see Fig. 9). Similarly only syntactically correct scene scripts can be constructed.

Since a graphical diagram cannot represent all information about a language construct, a property panel is provided that contains the additional information about the selected language construct. Certain properties are common to all diagram elements: name, type, id and description. The latter allows the tool user to give some additional detail about what is meant by a specific construct. It thus offers a similar function as comments in a programming language, but is limited to the scope of a single graphical element. Fig. 11 shows the property panel for the ParticipantElement named *Auctioneer*. In addition to the general properties, it also has some specific properties that complement the information in the diagram or provide a more convenient way to change this information. E.g. detailed positioning can be easier to accomplish using the properties grid, while initial arrangement can be accomplished in the diagram.

The current implementation supports incremental development and decomposable test units. The most logical test units smaller than the complete SPME are scenes. Though previews can be shown of incomplete specifications, interaction panels will only be shown when they are specified in the scene script; events that have no corresponding entity in the scene script are thus ignored for prototype generation. User interface elements will have to have some associated media files before a concrete prototype can be generated. High-level prototype generation such as described in earlier work[21]) can provide an alternative in earlier stages of development. These high-level prototypes do not require the specification of interaction panels nor the specification of media files for the user interface elements.

To get a better idea of the issues that should be tackled by the tool to optimally support the user in defining and understanding SpIeLan models, an informal user test of the graphical notation was performed [19, pp.154–160] as well as a broad brush evaluation using the cognitive dimensions framework [11]. These evaluations resulted in features of the tool that relate to the following cognitive dimensions:

viscosity The viscosity, resistance to change, was kept as low as possible. For example, adding a ParticipantElement in the scene stage automatically adds the necessary media requirements (such as support for streaming video and audio) to the visualization of the relevant role in all necessary places. Another example is that when a media element is present in several scenes, changes in its lo-

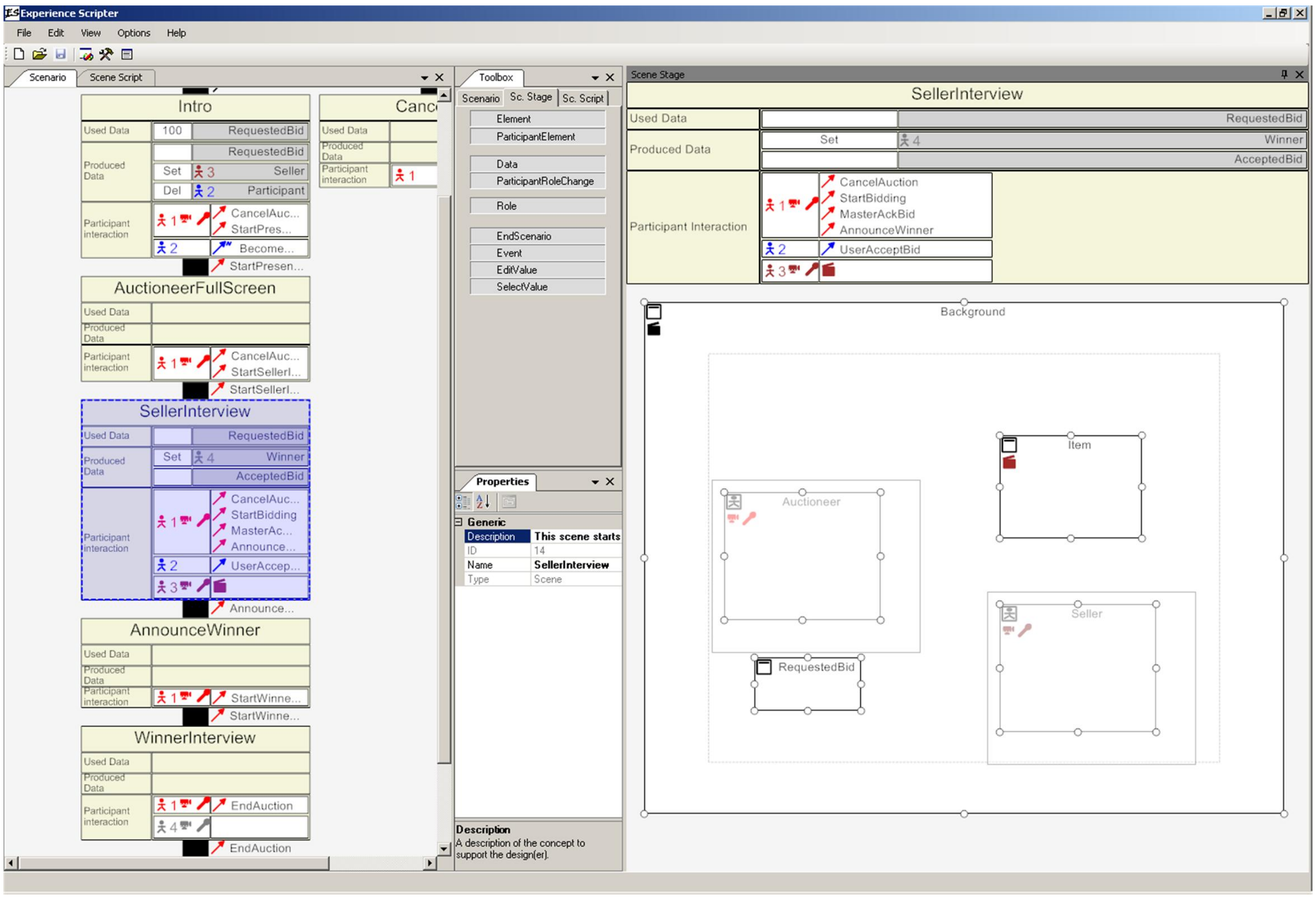


Fig. 10 Proof-of-Concept tool support (Experience Scripter) showing from left to right: a scenario model, the toolbox from which concepts can be dropped on the models and the properties grid, and the scene stage. All these parts can be positioned as dockable or floating windows by the tool user.

cation can be performed on all instances using a single action.

abstraction Some unnecessary abstractions, such as numbers associated to events and roles, were removed from the original notation as they proved to be confusing to some users. Some concepts, such as those related to the layout and composition of the interaction panels were removed altogether because they were implementation dependent and could be generated automatically.
Some additional abstractions were created to make editing the diagrams easier. For example, some user interface components seemed to appear in almost all scenes. This motivated the definition of an additional property for user interface components: *persistent* (see also Fig. 11). A user interface component that is marked as persistent appears, by default, in all scenes of the SPME. Once marked as persistent, changes to a persistent user interface component made in one scene are reflected in all scenes. When this behavior is not desired in a certain scene, one can set another property (*DefaultLayout*) to false to reflect these changes.

hidden dependencies Nearly all dependencies between the different models are indicated symbolically. For example, the information about the scenes that is given in the scenario model is duplicated in the scene stage and scene script.

visibility The tool supports juxtaposing all models related to a scene. Local visibility is optimized by showing the most relevant information in the models, sometimes requiring information duplication over the various models, and showing the other information about the selected concepts in a separate properties pane.

secondary notation The graphical representation of the models themselves has very little support for secondary notation. However, each concept can be annotated with free text. Although the graphical display of the variables that are used is an integral part of the graphical notation, they are (currently) not used to generate the SPME. In some sense, they can thus also be considered as a secondary notation that increases visibility. The availability of these notations allows the gradual specification of more details by people with the appropriate knowledge.

6.2 Back-end

The back-end of the *Experience Scripter* is built around an implementation of the meta-model or abstract syntax of SpIeLan. This meta-model consists of 38 classes, including the common base class *Concept*. Four of these are abstract classes and three others have no visual representation. About half of these classes are directly represented on the toolbox and thus are created directly by the users of the tool. All other concepts are created implicitly by the tool together with the concepts that are represented in the toolbox or during serialization.

The back-end of the tool is currently able to serialize the models to two different formats: a direct serialization to XML [5] that includes all model information and the SPME runtime format discussed in section 4.2. A scaled-down runtime system can be accessed directly from within the tool and allows to quickly check the effect any changes made to the models on a single machine. The generation of XForms [8]-based prototype descriptions as discussed in [21] is incomplete but is straight forward given the fact that it was completely implemented for an earlier version of the language and the changes in the meta-model since then are minimal.

6.3 Discussion

Having presented the current tool-support, we now revisit the requirements set in section 5.1. Requirement *R.1*, support for the creation process of SPMEs form concept development to prototype development, was addressed by offering the possibility to work at multiple levels of detail. One can start by creating the scenario flow and giving only textual descriptions for the scenes. These can be refined by a rough, high-level sketch of the interface in the *scene stage*, while the roles and the events they can trigger in the different scenes can be specified in the *scenario model*. The sequence of the actions and the reaction of the system can be specified at a later point in the *scene script*, while fine grained positioning of the media-elements and specification of the associated video clips, fonts and still images can be performed in the *scene stage*.

The tool and the models allow to specify the show at different levels of formality (rigid specification using predefined constructs or free form text) and detail. This does, however, not mean that the tool and models can be used throughout the complete development cycle, nor that it facilitates discussion between all people that are involved in the design process since these may have highly different backgrounds.

Facilitating such discussions might require incorporating some of the ideas presented in [4]. The tools presented in this paper, provide alternative views for one model at different levels of formality or abstraction. For our tool this could mean that one or more additional views for the scene stage and scene script could be offered that use the final presentation (available on a specific platform). The scenario model could also feature alternative visualizations where the scenes could be visualized using sketches of the screen or a textual description of the scene. A more polished user interface featuring extensive (semantic) drag-and-drop and syntax-highlighting for the scripts in the scene script might also be useful. More user tests involving the people that make the current interactive TV applications would be necessary to determine the necessity of these additions.

The complete specification is independent of the final interaction devices; only events and corresponding interactions are specified in the models. The concrete interaction visualization is determined by the runtime-environment. Nonetheless, a designer can preview the creation during the development. The fact that only events and associated SystemActions are used to specify user interaction satisfies requirement *R.2* (be independent of the final interaction devices). Furthermore, also the specification of the background panel can be done at a high level of abstraction in the *scene stage* without worrying about the graphic design, which means that requirement *R.4* is also satisfied.

Requirement *R.3* demands that user participation and presence is emphasized. This requirement is satisfied by the prominent specification of the involvement of users with a specific role for each scene in all models. The visual and audible presence is also emphasized by two dedicated kinds of user interface controls (see Table 4) and the role specification in the scenario header.

7 Conclusions and future work

In this paper we described a new type of participation TV: Staged Participatory Multimedia Events (SPME) on TV. It gives an unprecedented amount of control to television viewers: they are no longer just viewers but participants or even the host of a show. They can be part of the show through the use of currently available technology such as webcams, microphones and a remote control. Examples of SPME formats can include existing TV shows adapted for greater viewer participation, AuctionTV, adaptions of board games, shows for specific communities or even a headbangers competition.

A flexible proof-of-concept runtime infrastructure was created. This infrastructure uses open specifications and protocols such as SMIL, MyXaml and XMPP and has been demonstrated and tested at various events related to interactive television.

The creation of a format can be done using the SPME XML or through a graphical modeling language, SpIeLan, from which the SPME XML can be generated. Proof-of-concept tool support for SpIeLan has been realized. The design of this tool was based on general guidelines for the creation of end-user development environments [15], user feedback during informal user tests and a broadbrush evaluation using the cognitive dimensions framework [11].

Although further tests are still needed to be able to draw definitive conclusions about the possibility to let end-users create SPME, this paper demonstrates that model-driven creation of participation TV such as SPME is possible.

Revisiting the requirements set in the beginning of the paper, we can state that the realized proof-of-concepts indicate that it is feasible to create SPME using SpIeLan and the supporting tool with limited technical knowledge. Furthermore, the realized runtime environment is flexible and builds on existing standards increasing the chances that it the SPME can be integrated in the infrastructure of interested service providers or broadcasters.

Future work will be conducted along several paths. One of the major tasks will be to extend the platform support for SPME with, for example, a web-based PC-client and a mobile phone client as shown in Table 1. The interaction methods for the viewers will also be adapted to these target platforms. The introduction of a high-level language for interaction descriptions should minimize the effort for creators of SPME that target all these platforms.

More large scale user tests with the formats are needed to validate the viability of SPME as a commercial platform. Therefore such tests are planned with one or more of the then supported platforms.

Also the tool support and the modeling language, SpIeLan, will be the subject of further research. Improvement of the tools usability and its suitability for end-user development of SPME is an important point for further research.

While the focus for the presented infrastructure and modeling support was largely focused on the T-Commerce and Games categories of interactive television. Future research will reveal whether they can also be used for other kinds of interactive television or even outside the world of interactive television.

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).

We also thank the anonymous reviewers for their extensive feedback and suggestions.

References

1. Bederson, B.B., Grosjean, J., Meyer, J.: Toolkit design for interactive structured graphics. IEEE Trans. Softw. Eng. **30**(8), 535–546 (2004). DOI http://dx.doi.org/10.1109/TSE.2004.44
2. Benford, S., Greenhalgh, C., Craven, M., Walker, G., Regan, T., Morphett, J., Wyver, J., Bowers, J.: Broadcasting on-line social interaction as inhabited television. In: Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work, p. 179 (1999)
3. Bulterman, D., Grassel, G., Jansen, J., Koivisto, A., Layaïda, N., Michel, T., Mullender, S., Zucker, D.: Synchronized multimedia integration language (smil 2.1). http://www.w3.org/TR/2005/REC-SMIL2-20051213/ (2005)
4. Campos, P., Nunes, N.: Principles and practice of work style modeling: Sketching design tools. In: Proceedings of HWID06 - Human-Work Interaction Design, IFIP International Federation for Information Processing, pp. 203–219. Springer (2006)
5. consortium, W.W.W.: Extensible Markup Language (XML). World Wide Web, `http://www.w3.org/XML/` (2001)
6. Constantine, L.L.: Canonical abstract prototypes for abstract visual and interaction design. In: Proceedings of DSV-IS 2003, no. 2844 in LNCS, pp. 1 – 15. Springer, Funchal, Madeira Island, Portugal (2003)
7. Coppens, T., Trappeniers, L., Godon, M.: Amigotv : towards a social tv experience. In: Proceedings of EuroITV 2004 (2004)
8. Dubinko, M., Klotz, L.L., Merrick, R., Raman, T.V.: Xforms 1.0. W3C, World Wide Web, http://www.w3.org/TR/2003/REC-xforms-20031014/ (2003)
9. Dusseldorp, M.V.: Video-phone feeds getting into mainstream media. E-Media TidBits, `http://www.poynter.org/column.asp?id=31\&aid=81683` (2005)
10. Gawlinski, M.: Interactive Television Production. Focal Press (2003)
11. Green, T.R.G., Petre, M.: Usability analysis of visual programming environments: A 'cognitive dimensions' framework. Journal of Visual Languages and Computing **7**(2), 131–174 (1996)
12. Jensen, J.F.: Interactive television: new genres, new format, new content. In: IE2005: Proceedings of the second Australasian conference on Interactive entertainment, pp. 89–96. Creativity & Cognition Studios Press, Sydney, Australia, Australia (2005)
13. Luyten, K., Thys, K., Huypens, S., Coninx, K.: Telebuddies: Social stitching with interactive television. In: Accepted for publication for CHI 2006 (Extended Abstracts) (2006)
14. Michahelles, F.: Innovative application development for ubiquitous and wearable computing. Ph.D. thesis, Ludwig-Maximilians-Universität München (2004)
15. Repenning, A., Ioannidou, A.: End-User Development, chap. What Makes End-User Development Tick? 13 Design Guidelines, pp. 51–85. Springer (2006)
16. Saint-Andre, P.: Extensible messaging and presence protocol (xmpp): Core. `ftp://ftp.rfc-editor.org/in-notes/rfc3920.txt` (2004). ©The Internet Society
17. Saint-Andre, P.: Extensible messaging and presence protocol (xmpp): Instant messaging and presence. `ftp://ftp.rfc-editor.org/in-notes/rfc3921.txt` (2004). ©The Internet Society
18. Ursu, M.F., Cook, J.J., Zsombori, V., Zimmer, R., Kegele, I., Williams, D., Thomas, M., Wyver, J., Mayer, H.: Conceiving shapeshifting tv: A computational language for truly-interactive tv. In: Interactive TV: A Shared Experience. 5th European Conference, EuroITV 2007, *LNCS*, vol. 4471, pp. 96–106. Springer (2007)
19. Van den Bergh, J.: High-level user interface models for model-driven design of context-sensitive interactive applications. Ph.D. thesis, Hasselt University (transnationale Universiteit Limburg) (2006)
20. Van den Bergh, J., Bruynooghe, B., Moons, J., Huypens, S., Handekyn, K., Coninx, K.: Model-driven creation of staged participatory multimedia events on tv. In: Interactive TV: A Shared Experience. 5th European Conference, EuroITV 2007, *LNCS*, vol. 4471, pp. 21–30. Springer (2007)
21. Van den Bergh, J., Huypens, S., Coninx, K.: Towards Model-Driven Development of Staged Participatory Multimedia Events. In: Interactive Systems. Design, Specification, and Verification, 13th International Workshop, DSVIS 2006, *LNCS*, vol. 4323, pp. 81–94. Springer (2007)
22. Van den Bergh, J., Luyten, K., Coninx, K.: High-Level Modeling of Multi-User Interactive applications. In: Task Models and Diagrams for User Interface Design, 5th International Workshop, TAMODIA 2006, *LNCS*, vol. 4385, pp. 153–168. Springer (2007)