J Multimodal User Interfaces
DOI 10.1007/s12193-011-0077-1

ORIGINAL PAPER

Visual SceneMaker—a tool for authoring interactive virtual characters

Patrick Gebhard · Gregor Mehlmann · Michael Kipp

Received: 3 December 2010 / Accepted: 5 November 2011

Abstract Creating interactive applications with multiple virtual characters comes along with many challenges that are related to different areas of expertise. The definition of context-sensitive interactive behavior requires expert programmers and often results in hard-to-maintain code. To tackle these challenges, we suggest a visual authoring approach for virtual character applications and present a revised version of our SceneMaker tool. In SceneMaker a separation of content and logic is enforced. In the revised version, the Visual SceneMaker, we introduce concurrency and specific history structures as key concepts to facilitate (1) clearly structured interactive behavior definition, (2) multiple character modeling, and (3) extensions to existing applications. The new integrated developer environment allows sceneflow visualization and runtime modifications to support the development of interactive character applications in a rapid prototyping style. Finally, we present the result of a user study, which evaluates usability and the key concepts of the authoring tool.

Keywords Embodied conversational agents · Authoring tools · Modelling of interactive behavior

1 Introduction

The creation of believable interactive virtual characters involves expertise in many disciplines. It is a creative act which starts with application specific goals that are realized by the choice of appropriate models, techniques and content, put together in order to create an overall consistent application. Virtual characters can enrich the interaction experience by showing engaging and consistent behavior. This comes along with a whole range of challenges, such as interaction design, emotion modeling, figure animation, and speech synthesis [9]. Research is carried out in a range of disciplines, including believable facial expressions, gesture animations and body movements of virtual characters [14], modeling of personality and emotion [10] and expressive speech synthesis [21]. However, to what extent virtual characters actually contribute to measurable benefits such as increased motivation or even performance is still hotly debated (cf. [12, 18]). This makes it even more important that virtual character applications are carefully designed, in close interaction with users, artists, and programmers.

Over the past years, several approaches for modeling the content and the interactive behavior of virtual character have emerged. First, plan- and rule-based systems were in the focus of interest. Then, authoring systems were developed to exploit related expert knowledge in the areas of film or theater screenplay (e.g. dramatic acting and presentation). These systems are created to facilitate the authoring process and should enable non-computer experts to model believable natural behavior. One of the first authoring systems for interactive character applications is Improv [19]. Its behavior engine allows authors to create sophisticated rules defining how characters communicate, change their behavior, and make decisions. The system uses an "English-style" scripting language to define individual scripts. SCREAM [20] is a scripting tool to model story aspects as well as high-level interaction and comprises modules for emotion generation, regulation, and expression. Compared to our initial SceneMaker approach [8] which uses an author-centric approach with the primary focus of scripting at the story/plot level, the related projects use a character-centric approach in which the author defines a character's goals, beliefs, and

P. Gebhard (✉) · G. Mehlmann · M. Kipp
DFKI GmbH, Saarbrücken, Germany
e-mail: patrick.gebhard@dfki.de

Published online: 17 December 2011

Scene_en: Welcome **$name**
Sam: *Hi* **$name***! Come on,* [point_to_user] *let's play poker.* [Max nod]
Max: [camera upperbody] *My buddy an I have already gotten bored.*
Sam: [camera Sam] *Poker for two* [look_to_max][Max look_to_sam]*is just no fun.*
Max: [camera Max] *The game is draw poker.* [Sam nod] *Minimum bet is 5, maximum is 50.*

Fig. 1 The English example scene `Welcome` with a variable, gesture and system commands

attitudes. These mental states determine the behavioral responses to the annotated communicative acts it receives. SCREAM is intended as a plug-in to task specific agent systems such as interactive tutoring or entertainment systems for which it can decide on the kind of emotion expression and its intensity. There is no explicit support for scripting the behavior of multiple agents in a simple and intuitive way. Other author-centric approaches are the CSLU toolkit [17], SceneJo [22], DEAL [5], and CREACTOR [13]. The CSLU toolkit relies on a finite state approach for dialogue control while Scenejo relies on the A.L.I.C.E. chatbot technology [1] for the control of verbal dialogues which offers accessibility to non-experts in interactive dialogue programming. This technology offers AIML structures, which can be used to define textual dialogue contributions. A graphical user interface is provided for the modeling of structured conversations. DEAL is created as a research platform for exploring the challenges and potential benefits of combining elements from computer games, dialogue systems and language learning. A central concept is the dialogue manager, which controls the dialogues of game characters. The dialogue manager employs Harel statecharts [11] to describe an application's dialogue structure and to model different aspects of dialogue behavior (e.g. conversational gestures and emotional expressions). CREACTOR is an authoring framework for virtual characters that enables authors to define various aspects of an interactive performance through an assisted authoring procedure. The project focuses on the modeling of consistent narration and character behavior using a graph representation. None of the mentioned authoring systems support concepts for action history and concurrency on the authoring level. However, DEAL provides these concepts, but only at the level of statecharts. With the Visual SceneMaker, new authoring concepts are introduced: concurrency, variable scoping, multiple interaction policies, a run-time history, and a new graphical integrated development environment (IDE). Before we lay the focus on these, we give a brief overview on the existing concepts.

2 SceneMaker authoring concepts

SceneMaker's central authoring paradigm is the separation of content (e.g. dialogue) and logic (e.g. conditional branching). The content is organized as a collection of *scenes* which are specified in a multi-modal *scenescript* resembling a movie script with dialogue utterances and stage directions for controlling gestures, postures, and facial expressions. The logic of an interactive performance and the interaction with virtual characters is controlled by a *sceneflow*.

Scenescript A *scene* definition (see Fig. 1) starts with the keyword `Scene`, followed by a *language key* (e.g. `en` for English), a *scene identifier* (e.g. `Welcome`), and a number of variables (e.g. `$name`). Variables can be used to produce variations or to create personalized versions. The *scene body* holds a number of turns and utterances, which have to start with a character's name (e.g. `Sam` and `Max`). Utterances may hold variables, system actions (e.g. `[camera upperbody]`) and gesture commands (e.g. `[point_to_user]`).

A *scenescript* may provide a number of variations for each scene that are subsumed in a *scenegroup*. Different predefined and user-defined *blacklisting* strategies can be used to choose one scene of a scenegroup for execution. This selection mechanism increases dialogue variety and helps to avoid repetitive behavior of virtual characters.

Sceneflow A *sceneflow* is a hierarchical statechart variant specifying the logic and temporal order in which individual scenes are played, commands are executed and user interactions are processed. It consists of different types of *nodes* and *edges*. A *scenenode* (see Fig. 2 ①) has a name and an unique identifier. It holds scenegroup playback commands or statements specified in a simple scripting language format. These are type definitions and variable definitions, variable assignments, and function calls to predefined functions of the underlying implementation language (e.g. *Java*™ functions). A *supernode* (see Fig. 2 ②) extends the functionality of scenenodes by creating a hierarchical structure. A supernode may contain scenenodes and supernodes that constitute its subautomaton (marked graphically with the shaded area)[1] with one defined startnode (see Fig. 2 ③). The supernode hierarchy can be used for a *scoping* of types and variables. Type definitions and variable definitions are inherited to all subnodes of a supernode.

Nodes can be seen as little code pieces that structure the content of an interactive presentation while edges specify how this content is linked together. Different *branching*

[1] Within the IDE, the content of a supernode can be edited by selecting a supernode.

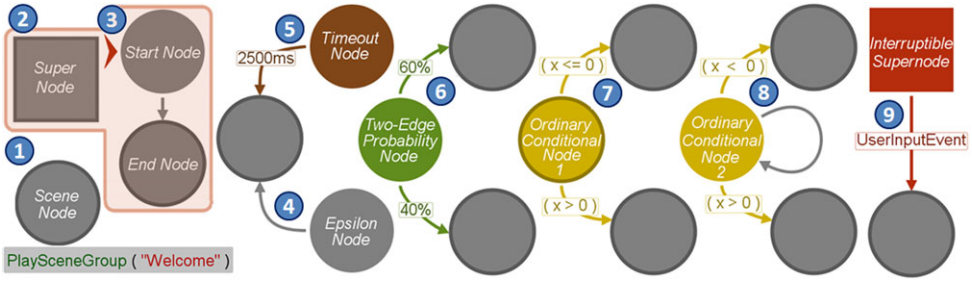

Fig. 2 Basic node types and edge types of sceneflows and the color code of nodes and edges

strategies within the sceneflow, e.g. logical and temporal conditions or randomization, as well as different *interaction policies*, can be modelled by connecting nodes with different types of edges [8]. The IDE supports the usages of edges. The first edge connected to a node defines the allowed types of edges and the color of that node. This color code helps authors structuring a sceneflow and helps keeping it well-defined.

Epsilon edges represent unconditional transitions (see Fig. 2 ④) and specify the order in which steps are performed and scenes are played back.

Timeout edges represent scheduled transitions and are labeled with a timeout value (see Fig. 2 ⑤) influecing the temporal flow of a sceneflow's execution.

Probabilistic edges represent transition that are taken with certain probabilities and are labeled with probability values (see Fig. 2 ⑥). They are used to create randomness and variability during the execution.

Conditional edges represent conditional transitions and are labeled with conditional expressions (see Fig. 2 ⑦+⑧). They are used to create a branching structure in the sceneflow which describes different reactions to changes of environmental conditions, external events, or user interactions.

Interuptive edges are special conditional edges that are restricted to supernodes (see Fig. 2 ⑨) and are used to interrupt ongoing activities.

Discussion The focus of the original SceneMaker is to enable authors to model typical interactive situations (e.g. with dialogue variations and distinct character behavior). This might be sufficient for some simple application setups, but there are limitations with regard to a more sophisticated modeling of dialogue, behavior of virtual characters, and interaction:

1. *Difficult dialogue resumption*. An external dialogue memory is needed to represent and store the current state of the dialogue topic or interaction. Without a dialogue memory coherent dialogue resumptions are not feasible.
2. *Coupled control of virtual characters*. An individual control and synchronization of multiple characters with separate sceneflows is not supported. With scene gesture commands multiple characters could be animated at the same time, but not independent of each other.
3. *Costly extension of virtual character behavior*. Adding new behavioral aspects, e.g. a specific gaze behavior in a multi-character application requires to add commands in each used scene for each character.

3 The Visual SceneMaker IDE

The Visual SceneMaker *IDE*, shown in Fig. 3, enables authors to create, maintain, debug, and execute an application via graphical interface controls.

The IDE displays two working areas in the middle of the window, the *sceneflow editor* (see Fig. 3 Ⓐ) and the *scenescript editor* (see Fig. 3 Ⓑ). In order to build an application, authors can drag *building blocks* from the left side and drop them on the respective editor. Building blocks for sceneflows (see Fig. 3 Ⓒ) are nodes, edges, scenes, and predefined *Java*™ functions. Building blocks for scenescripts (see Fig. 3 Ⓓ) are gestures, animations and system actions. All properties of nodes, such as type- and variable definitions as well as function calls, can be modified right to the sceneflow working area (see Fig. 3 Ⓔ). All defined types and variables are shown in a separate *variable badge* (see Fig. 3 ①) within the sceneflow editor. To facilitate maintaining and testing, the execution of an application can be visualized either by highlighting the currently executed nodes, edges, and scenes red at runtime (see Fig. 3 ②) or by highlighting all past actions and execution paths gray (see Fig. 3 ③).

The Visual SceneMaker is implemented in *Java*™ using an *interpreter* approach for executing sceneflows and scenes. This allows the modification of the model and the direct observation of the ensuing effects at runtime.

Figure 4 shows the system architecture, with the IDE in the modeling environment (see Fig. 4 Ⓐ) as well as the environments for the interpreter (see Fig. 4 Ⓑ), plug-ins (see Fig. 4 Ⓒ), and applications (see Fig. 4 Ⓓ). *Sceneplayer*s and *Plug-In*s represent the input and output interfaces to external applications and have to be implemented by programmers. The figure presented a typical application configuration showing the *Mary TTS*[2] for speech output (see

[2] http://mary.dfki.de

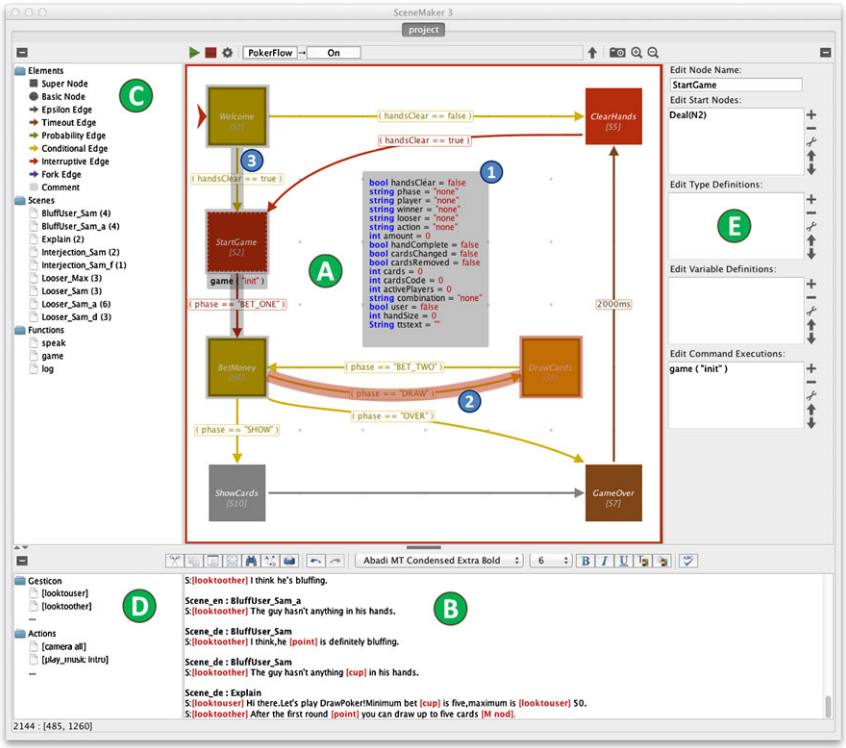

Fig. 3 The Visual SceneMaker IDE showing different working areas and highlighting modes

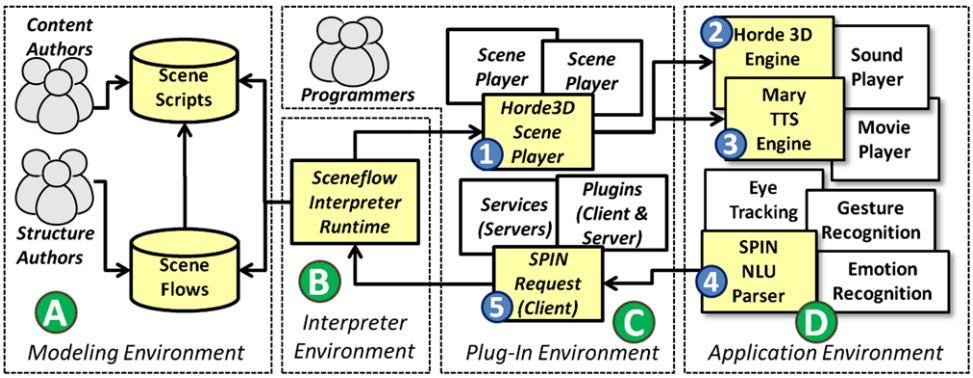

Fig. 4 The SceneMaker's system architecture with its sceneplayer and plug-in mechanism

Fig. 4 ③) and the *Horde 3D*[3] graphics engine for character rendering (see Fig. 4 ②) as well as the *Spin* [7] parser for NLU (see Fig. 4 ④). The *Horde3D Sceneplayer* (see Fig. 4 ①) is responsible for the translation of scenes to TTS and animation commands. The *Spin Request* (see Fig. 4 ⑤) forwards user input events to the sceneflow interpreter by changing the value of global sceneflow variables.

4 New Sceneflow elements

We extended the definition of sceneflows with concepts for a *hierarchical* and *parallel decomposition* of sceneflows, different *interruption policies* and a *runtime history* adopting and extending modeling concepts as they can be found in several statechart variants [3, 11].

Hierarchical and parallel decomposition By the use of a hierarchical and parallel model decomposition, multiple virtual characters and their behavioral aspects, as well as multiple control processes for event and interaction management, can be modeled as concurrent processes in separate parallel automata. Thus, individual behavioral aspect and control processes can be modeled and adapted in isolation without knowing details of the other aspects while previously modeled behavioral patterns can easily be reused and extended.

Sceneflows provide two instruments allowing an author to create multiple concurrent processes at runtime: (1) By

[3] http://horde3d.org

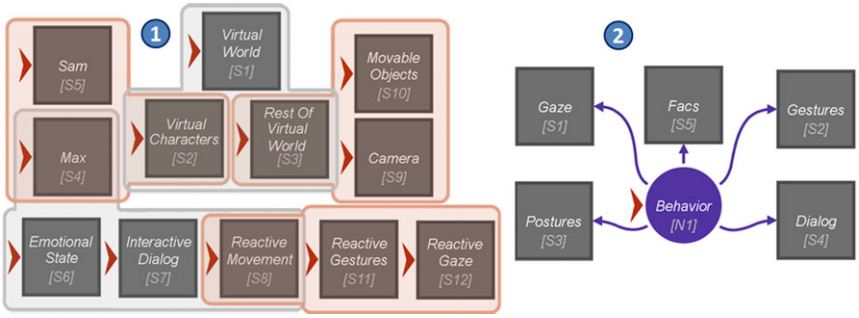

Fig. 5 ① The hierarchical and parallel decomposition of the virtual world with multiple startnodes and ② the parallel decomposition of behavioral aspects with multiple fork edges

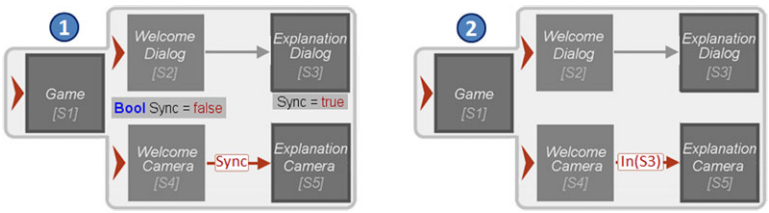

Fig. 6 ① Process synchronization over the variable `sync` and ② over a configuration query

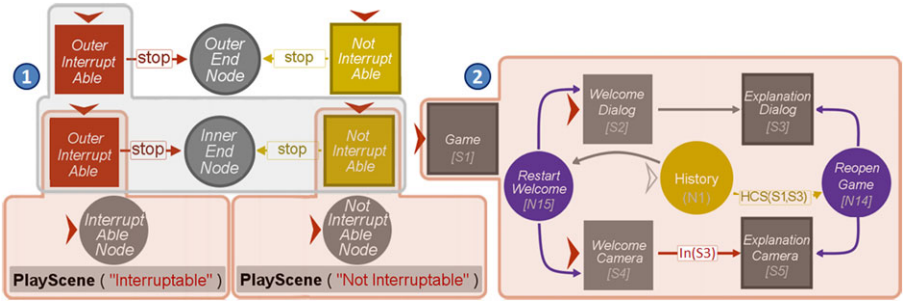

Fig. 7 ① Interruptive and ordinary conditional edges in a supernode hierarchy as well as ② an example of the use of a history node and a history condition for dialogue resumption

defining multiple startnodes, and thus, multiple parallel sub-automata for a supernode (see Fig. 5 ①) and (2) by using *fork edges* (see Fig. 5 ②), each of which creating a new process.

Communication and synchronization Individual behavioral aspects contributing to the behavior of a virtual character and the processing of events for interaction management are usually not completely independent. For example, speech is usually highly synchronized with non-verbal behavioral modalities like gestures and body postures. When modeling these aspects in parallel automata, the processes that concurrently execute these automata have to be synchronized by the author in order to coordinate them all with each other.

Sceneflows provide two instruments for an asynchronous non-blocking synchronization of concurrent processes, realized by a *shared memory model*. They allow the synchronization over (1) *shared variables* defined in the scope of some common supernode (see Fig. 6 ①) and (2) a more intuitive mechanism for process synchronization, a *state query* (see Fig. 6 ②) which allows to test if a certain state is currently executed by another concurrent process.

Interaction handling with interruption policies User interactions and environmental events can rise at any time during the execution of a model. Some of them need to be processed as fast as possible to assert certain real-time requirements, for example when interrupting a character's utterance in order to give the user the impression of presence. Other events may be processed at a later point in time allowing currently executed scenes or commands to be regularly terminated. These two interaction handling strategies are realized in sceneflows with the two types of conditional edges, having different *interruption-* and *inheritance policies*. In contrast to ordinary conditional edges, interruptive conditional edges directly interrupt the execution of any processes, and have priority over any other edges, lower in the supernode hierarchy and, thus, are used for handling events and user interactions requiring a prompt reaction.

Figure 7 ① shows ordinary and interruptive conditional edges in a supernode hierarchy. When condition `stop` comes true during the execution of the innermost scene playback command, then the outer interruptive edge to the outer end node is taken immediately, whereas the inner ordinary conditional edge is taken after the playback command has been terminated and the outer ordinary conditional edge is taken only after the inner end node has terminated.

Table 1 Feature comparison between the SceneMaker and the new Visual SceneMaker

Criteria	SceneMaker	Visual SceneMaker
Modeling Multiple Virtual Characters	scene based	no restriction
Modeling/Synchronization of Parallel Behavioral Aspects	scene based	no restriction
Concepts for Dialogue Resumption/Dialogue History	none	included
Extension Costs of Behavioral Aspects	high (scene based)	low
Integrated Developer Environment	no	yes

Dialogue resumption with runtime history During a sceneflow's execution, the interpreter maintains a *runtime history* to record the runtimes of nodes, the values of local variables, executed system commands, and scenes that were played back as well as nodes that were lastly executed. The automatic maintenance of this history memory releases the author of the manual collection of data, thus reducing the modeling effort while increasing the clarity of the model and providing the author with rich information about previous interactions and states of execution. This facilitates the modeling of reopening strategies and recapitulation phases for dialogues. The history concept is realized graphically as a special *history node* which is a fixed child node of each supernode. When re-executing a supernode, the history node is executed instead of its default startnodes. Thus, the history node serves as a starting point for authors to model reopening strategies or recapitulation phases. The scripting language of sceneflows provides a variety of expressions and conditions to request the information from the history memory or to delete it. Figure 7 ② shows the use of the history node of supernode `S1` and the history condition `HCS(S1, S3)` used to find out if node `S3` was the last executed subnode of `S1`. If the supernode `S1` is interrupted and afterwards re-executed, then it starts at its history node `N1`. If it had been interrupted during the explanation phase in nodes `S3` and `S5`, then the explanation phase is reopened, otherwise the welcome phase in nodes `S2` and `S4` is restarted.

Feature comparison The introduced new sceneflow elements support a more flexible creation of interactive applications with multiple virtual characters. Table 1 compares the original and the new version of the authoring tool.

With the new features authors are now able to model multiple virtual characters individually with separate sceneflows, which allow a fine-grained synchronization. Moreover, the use of parallel sceneflow elements allow modeling behavioral aspects with theses structures. While the old SceneMaker version supports an animation of characters through scene gesture commands (which probably have to be used in each scene, e.g. gaze behavior), the new version could process these commands in an additional parallel sceneflow. In addition, this enables an effortless general extension to existing SceneMaker applications. The task extending an application, by e.g. camera movements that focus on the current speaker, could easily fulfilled by modeling a new parallel sceneflow that executes the respective camera commands. Overall, this helps reducing modeling costs in several situations. The new history node structure liberates authors from the use of external dialogue- and context memories, which often come with their own interface language. By providing a simple integrated dialogue history memory, a fine-grained handling of dialogue resumption and dialogue interruption is feasible with "on-board" equipment. Finally, we belief that using the mentioned concepts in a visual programming paradigm together with an IDE that supports the manipulation and the execution of SceneMaker applications results in a powerful, easy to handle approach for the creation of interactive multiple virtual character applications.

5 Evaluation

At first, SceneMaker has been informally evaluated in two field tests (at the *Nano-Camp 2009* and the *Girls' Day 2010* [6]) with 21 secondary and high school students at the age of 12 to 16. Overall, the students describe SceneMaker as an easy and an intuitive tool, which supports a rapid creation of applications with interactive virtual characters.

We evaluated the Visual SceneMaker with a user study with 19 subjects aged 23 to 41 (average age is 30.89). The subject group consisted of 14 males and 5 females, 9 of the subjects were university students and 10 of them were researchers from multimedia and computer science. The subjects had to model an interactive sales conversation between a user and two embodied conversational agents. The participants were shortly introduced to the software and were given a textual description of the expected dialogue structure. Parts of the dialogue were pre-modeled and served as an orientation point for the participants to model the remaining dialogue without assistance. The dialogue specification required the employment of the new modeling concepts, such as different interruption policies, the history concept as well as hierarchical refinement and parallel decomposition. The modeling sessions lasted from 24 to 67 minutes with an average of 40.05 minutes. Afterwards the participants filled out a questionnaire on their previous knowledge and several five-point Likert scale questionnaires addressing Visual SceneMaker's modeling concepts and system usability. We used questionnaires (see Fig. 8) $Q1$ and $Q2$ for the

Q1	Modeling Dialogue and Interaction	Min	Max	Avg	Dev
1.	I think that it would be easy to reuse single parts of the model	4,00	5,00	4,58	0,49
2.	I think that it would be easy to adapt the model to changed requirements	3,00	5,00	4,32	0,65
3.	I think that it would be easy to extend the model with more virtual characters	2,00	5,00	4,16	0,87
4.	I think that it would be easy to add the reaction to more dialogue acts in the model	2,00	5,00	4,37	0,81
5.	I think it is easy to model a prompt reaction to the user's input	3,00	5,00	4,32	0,73
6.	I think it is easy to model a context-sensitive reaction to the user's input	3,00	5,00	4,21	0,77
7.	I think it is easy to model a coherent resumption of dialogues	3,00	5,00	4,11	0,64
8.	I think it is easy to model parallel behavior of multiple virtual characters	4,00	5,00	4,74	0,44
9.	I think it is easy to keep the model clearly structured	3,00	5,00	4,26	0,71
10.	I know a visual formalism that is better suited for dialogue modelling	4,00	5,00	4,74	0,44
	Overall Questionnaire Score	3,70	4,80	4,38	0,28
Q2	**Creating Multimodal Dialogue Content**	Min	Max	Avg	Dev
1.	I found the format for writing scenes intuitive	3,00	5,00	4,11	0,72
2.	I think that it is easy to create multimodal behavior with scenes	3,00	5,00	4,42	0,67
3.	I think that it is easy to create non-repetitive behavior with scene groups	2,00	5,00	4,42	0,82
4.	I found it useful to use parameters in scenes	3,00	5,00	4,53	0,68
5.	I think that it would be easy to reuse scenes	4,00	5,00	4,74	0,44
	Overall Questionnaire Score	3,20	5,00	4,44	0,53
Q3	**Testing and Visualization**	Min	Max	Avg	Dev
1.	I was able to directly comprehend the effects of my modeling actions	4,00	5,00	4,74	0,44
2.	I found the real-time and trace visualization of the execution helpful	3,00	5,00	4,58	0,59
3.	I think the software allows a rapid prototyping of interactive performances	2,00	5,00	4,42	0,75
	Overall Questionnaire Score	3,00	5,00	4,58	0,48
Q4	**System Usability Scale**	Min	Max	Avg	Dev
1.	I think that I would like to use this system frequently	2,00	5,00	4,11	0,97
2.	I found the system unnecessarily complex	1,00	4,00	1,68	0,98
3.	I thought the system was easy to use	4,00	5,00	4,37	0,48
4.	I think that I would need the support of a technical person to be able to use this system	1,00	4,00	2,21	1,00
5.	I found the various functions in this system were well integrated	3,00	5,00	4,21	0,61
6.	I thought there was too much inconsistency in this system	1,00	3,00	1,42	0,59
7.	I would imagine that most people would learn to use this system very quickly	3,00	5,00	4,16	0,67
8.	I found the system very cumbersome to use	1,00	2,00	1,26	0,44
9.	I felt very confident using the system	4,00	5,00	4,32	0,46
10.	I needed to learn a lot of things before I could get going with this system	1,00	3,00	1,74	0,71
	Overall Questionnaire Score	67,50	97,50	82,11	7,62

Fig. 8 The questionnaires of the evaluation sheet and the results of the user study

Results of User Group Comparison with U-Test and T-Test						High(> Avg) vs. Low (< Avg)				High(> 3) vs. Low (< 3)			
	Previous Knowledge and Experience	Min	Max	Avg	Dev	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
1.	Object Oriented Programming	3,00	5,00	4,42	0,67	○	○	○	○	not comparable			
2.	Model Oriented Programming	2,00	5,00	3,26	0,85	○	○	○	○	○	○	□	□
3.	Rule Based Programming	1,00	4,00	2,11	1,12	○	○	○	○	○	○	○	○
4.	Planning Based Programming	1,00	5,00	1,79	1,20	○	○	○	○	○	○	○	○
5.	Visual Programming Software	1,00	4,00	2,21	1,15	○	○	○	○	○	○	○	○
	Overall Programming Experience	1,80	4,20	2,76	0,64	○	○	○	□	○	○	○	○

□ significant ○ not significant

Fig. 9 The subject group comparisons with respect to their previous knowledge

assessement of SceneMaker's modeling concepts as well as $Q3$ and $Q4$ for measuring the system usability scale (SUS) from [4], to get a view of subjective assessement of general system usability. The SUS covers a variety of aspects of system usability, such as the need for support, training, and complexity and has been proven a very effective means to measure usability in a huge number of usability test. For example, Bangor and colleagues [2] applied it to a large range of user interfaces in nearly 3500 surveys with 273 usability studies. The SUS helps us to interpret the results of our experiment by comparing the scores of our study with the scores reported in those studies.

The results of the descriptive analysis of the questionnaire are shown in Fig. 8. To show that the mean rating value was significantly above the neutral value, we applied t-tests for one sample. These showed, that the participants got along very well with dialogue modeling and interaction ($t(18) = 21.02$, $p < 0.0005$), multimodal dialogue creation ($t(18) = 11.56$, $p < 0.0005$), testing and visualization features ($t(18) = 13.91$, $p < 0.0005$) and overall system usability ($t(18) = 44.04$, $p < 0.0005$). Furthermore, we compared different subject and experience groups. Rating comparisons of male subjects to female subjects and students to researchers as well as comparisons of different experience groups with respect to modeling time and age revealed no significant differences.

The results of the experience group comparisons, shown in Fig. 9, revealed that subjects with high experience in model oriented programming (N: 15, M_{Q3}: 4.73, SD_{Q3}: 0.29, M_{Q4}: 84.17, SD_{Q4}: 6.99) gave slightly better scores for $Q3$ and $Q4$ than those with low experience (N: 4, M_{Q3}: 4.00, SD_{Q3}: 0.72, M_{Q4}: 74.38, SD_{Q4}: 6.25; $t_{Q3}(17) = 3.26$, $p_{Q3} = 0.005$, $t_{Q4}(17) = 2.54$, $p_{Q4} = 0.021$). This was also observed for $Q4$ in the comparison of subjects with

high overall experience (N: 7, M: 87.50, SD: 5.95) to subjects with low overall experience (N: 12, M: 78.95, SD: 7.19; $t(17) = 2.65$, $p = 0.017$). Another comparison showed, that subjects with SUS score below average (N: 7, M: 49.43 min, SD: 12.15 min) needed more time to finish the task than subjects with a high SUS score (N: 12, M: 34.58 min, SD: 7.14 min; $t(17) = -3.38$, $p = 0.004$).

Although, subjects with less programming experience needed more time and gave slightly worse system usability scores, the overall results show that the users generally felt very confident with the tool. Generally, the SUS score obtained was very promising. Regarding the average results of around 70 on the SUS scale from 0 to 100 that Bangor and colleagues [2] obtained for a large variety of user interfaces they evaluated, Visual SceneMaker's performance (Min: 67.60, Max: 97.50, M: 82.11, SD: 7.62) is well above the reported average values and may be interpreted according to their usability classification as nearly excellent.

6 Summary and future work

In this paper, we have presented Visual SceneMaker, an authoring tool for the creation of interactive applications with multiple virtual characters. It extents the initial version providing creative, non-programming experts with a simple scripting language for the creation of rich and compelling content. The separation of dialogue content and narrative structure based on scenes and sceneflows allows an independent modification of both.

A powerful feature for the creation of new interactive performances in a rapid prototyping style is the reusability of sceneflow parts. This is explicitly supported by the sceneflow editor, which is part of the new Visual SceneMaker IDE. This graphical authoring tool supports authors with drag'n'drop facilities to draw a sceneflow by creating nodes and edges. The editor also supports new structural extensions of the sceneflow model. The improvements comprise concurrent sceneflows and a special history node type. With the use of concurrent sceneflows, large sceneflows can be decomposed into logical and conceptual components, which reduces the amount of structures but also helps to organize the entire model of an interactive performance. In addition, concurrent structures can be used to model reactive behavioral aspects (e.g. gaze). This will reduce the effort in creating pre-scripted scenes. History nodes are introduced for more flexible modeling of reopening strategies and recapitulation phases in dialogue. This new sceneflow structure provides access to the plot history in every node. This is a valuable help for the creation of non-static and lively interactive performances.

Visual SceneMaker IDE is build upon an interpreter for the execution of sceneflows. The graphical authoring tool allows the visualization of the execution progress of a given SceneMaker application. In addition, it provides an immediate observation of effects caused by the modification of the model even at runtime. This is a crucial help when debugging and testing a model in early development phases and in refinement and modification phases.

We tested the flexibility of the Visual SceneMaker approach first in field tests and second in a user study. As a result, we found that the visual programming approach and the provided modeling structures are easily comprehensible and let non-experts create virtual character applications in a rapid prototype fashion. This conclusion is especially justified by the excellent usability scores that were reached in the user study.

For future work, we plan to integrate the Visual SceneMaker with other crucial components (speech synthesis, nonverbal behavior generation, emotion simulation ...) with plan to integrate it into a component-based embodied agent framework [15]. This implies the use of standard languages for intention (FML), behavior (BML) and emotion (EmotionML) to make the Visual SceneMaker compatible with alternative components [16, 23]. We also plan to introduce a library of reusable behavior patterns for an easy creation of different behavioral aspects for multiple virtual characters in other interactive performances.

Acknowledgements The presented work was supported by the German Federal Ministry of Education and Research in the SemProM project (grand number 01 IA 08002), the INTAKT project (grand number 01 IS 08025B) and it was supported by the European Commission within the 7th Framework Programme in the IRIS project (project number 231824). The authors would like to thank Elisabeth André for supporting us with their expertise.

References

1. ALICE. Homepage of the alice artificial intelligence foundation. Online: http://www.alicebot.org
2. Bangor A, Kortum P, Miller J (2009) Determining what individual sus scores mean: Adding an adjective rating scale. J Usability Stud 4:114–123
3. Beeck Mvd (1994) A comparison of statecharts variants. In: Proceedings of the third international symposium organized jointly with the working group provably correct systems on formal techniques in real-time and fault-tolerant systems, London, UK. Springer, Berlin, pp 128–148
4. Brooke J (1996) SUS: A quick and dirty usability scale. In: Jordan PW, Weerdmeester B, Thomas A, Mclelland IL (eds) Usability evaluation in industry. Taylor and Francis, London
5. Brusk J, Lager T, Wik AHaP (2007) Deal dialogue management in scxml for believable game characters. In: Proceedings of ACM future play, New York, NY, USA. ACM, New York, pp 137–144
6. Endrass B, Wissner M, Mehlmann G, Buehling R, Haering M, André E (2010) Teenage girls as authors for digital storytelling—a practical experience report. In: Workshop on education in interactive digital storytelling on ICIDS, Edinburgh, UK

7. Engel R (2005) Robust and efficient semantic parsing of freeword order languages in spoken dialogue systems. In: Interspeech 2005
8. Gebhard P, Kipp M, Klesen M, Rist T (2003) Authoring scenes for adaptive, interactive performances. In: Rosenschein JS, Wooldridge M (eds) Proc of the second international joint conference on autonomous agents and multi-agent systems. ACM, New York, pp 725–732
9. Gratch J, Rickel J, André E, Cassell J, Petajan E, Badler NI (2002) Creating interactive virtual humans: some assembly required. IEEE Intell Syst 17(4):54–63
10. Gratch J, Marsella S, Wang N, Stankovic B (2009) Assessing the validity of appraisal-based models of emotion. In: International conference on affective computing and intelligent interaction, Amsterdam. IEEE, New York
11. Harel D (1987) Statecharts: a visual formalism for complex systems. In: Science of computer programming, vol 8. Elsevier, Amsterdam, pp 231–274
12. Heidig S, Clarebout G (2011) Do pedagogical agents make a difference to student motivation and learning? Educ Res Rev 6(1):27–54. doi:10.1016/j.physletb.2003.10.071
13. Iurgel IA, da Silva RE, Ribeiro PR, Soares AB, dos Santos MF (2009) CREACTOR—an authoring framework for virtual actors. In: Proceedings of the 9th international conference of intelligent virtual agents. LNAI, vol 5773. Springer, Berlin, pp 562–563
14. Kipp M, Neff M, Kipp KH, Albrecht I (2007) Toward natural gesture synthesis: evaluating gesture units in a data-driven approach. In: Proceedings of the 7th international conference on intelligent virtual agents. LNAI, vol 4722, pp 15–28
15. Kipp M, Heloir A, Gebhard P, Schröder M (2010) Realizing multimodal behavior: closing the gap between behavior planning and embodied agent presentation. In: Proceedings of the 10th international conference on intelligent virtual agents (IVA-10). Springer, Berlin
16. Kopp S, Krenn B, Marsella S, Marshall AN, Pelachaud C, Pirker H, Thórisson KR, Vilhjálmsson H (2006) Towards a common framework for multimodal generation: the behavior markup language. In: Proc of IVA-06
17. Mctear MF (1999) Using the cslu toolkit for practicals in spoken dialogue technology. In: University College London, pp 1–7
18. Miksatko J, Kipp KH, Kipp M (2010) The persona zero-effect: evaluating virtual character benefits on a learning task. In: Proceedings of the 10th international conference on intelligent virtual agents (IVA-10). Springer, Berlin
19. Perlin K, Goldberg A (1996) Improv: a system for scripting interactive actors in virtual worlds. In: Computer graphics proceedings, ACM SIGGRAPH, New York, pp 205–216
20. Prendinger H, Saeyor S, Ishizuk M (2004) Mpml and scream: scripting the bodies and minds of life-like characters. In: Life-like characters—tools, affective functions, and applications. Springer, Berlin, pp 213–242
21. Schröder M (2008) Approaches to emotional expressivity in synthetic speech. In: Emotions in the human voice, culture and perception, vol 3. Pural, San Diego, pp 307–321
22. Spierling U, Mueller SWaW (2006) Towards accessible authoring tools for interactive storytelling. In: Proceedings of technologies for interactive digital storytelling and entertainment. LNCS. Springer, Heidelberg
23. Vilhjalmsson H, Cantelmo N, Cassell J, Chafai NE, Kipp M, Kopp S, Mancini M, Marsella S, Marshall AN, Pelachaud C, Ruttkay Z, Thórisson KR, van Welbergen H, van der Werf RJ (2007) The behavior markup language: recent developments and challenges. In: Proc of IVA-07