Developing Web fully-integrated conversational assistant agents

Jean Paul Sansonnet  
CNRS-LIMSI  
Orsay, France  
jps@llimsi.fr

Daniel Werner Correa  
UNISINOS University  
Porto Alegre, Brazil  
danielwcorrea@gmail.com

Patricia Jaques  
UNISINOS University  
Porto Alegre, Brazil  
patricia.jaques@gmail.com

Annelies Braffort  
CNRS-LIMSI  
Orsay, France  
braffort@llimsi.fr

Cyril Verrecchia  
CNRS-LIMSI  
Orsay, France  
cyril.verrecchia@llimsi.fr

ABSTRACT

In this paper we describe a framework, called DiValite dedicated to the development of Web-based Embodied Conversational Agents (ECAs) for research purposes. This context prompts two specific requirements, entailing original features: (i) DiValite agents are implemented as Document Object Model (DOM) elements so they can interact dynamically and at a fine-grained level with the Web page content; (ii) DiValite agents are implemented in full client mode making it quick and easy to develop and deploy experimental studies. We present the architecture and the DOM-integration of the virtual characters. The resulting capacities are compared with those of similar tools and gains in terms of developing costs are measured.

Categories and Subject Descriptors

H.5 [Information Interfaces and Presentations]: Miscellaneous; D.2.6 [Web Engineering]: Programming environments

General Terms

Design, Experimentation

Keywords

Embodied Conversational Agents, Web-based Animated Characters, Interacting with DOM content.

1. INTRODUCTION

For some years now, the research domain of Embodied Conversational Agents (ECA) [5] has been rapidly growing, with two main complementary dimensions: 1) virtual characters endowed with rich graphical expression of emotions and the study of their impact on users [11],[8]; 2) assistant agents [13], which emphasizes on the symbolic reasoning capacities over the model of the assisted application [23]. These two dimensions are integrated into experimental studies on conversational assistant agents, appearing in many environments where human and software agents are in interaction in mixed-initiative environments such as: eLearning [8], games [24] etc. Once based on stand-alone applications, these activities are now spreading on the Web thanks to the growing attraction of rich internet application technologies, thus prompting the need to deploy conversational assistant agents on Web pages.

In the context of the Internet, each dimension mentioned above entails a specific issue:

1) character animation: recent tools for animated ECA make use of intensive graphical computation (e.g. GRETA [18]) and are more suited for stand-alone applications than deployment into Web pages. Independently, technologies have been developed for supporting animated characters on Web pages, mainly to attract consumers. Roughly, they have two main drawbacks: a small range of predefined expressions; no dynamic control upon the sequencing of animations;

2) agent interaction: animated characters used in Web pages are first designed as separate software and then integrated as external components (typically Flash™ objects) into the pages. Again this approach entails two extra drawbacks: they are often ‘encased in a box’, with little or null bodily interaction a) with users and b) with objects on the page.

These issues are particularly critical in the specific context or research upon Web-based conversational assistant agents where basic requirements are: agent bodily manipulations by the user, agent expressiveness and dynamicity in animation, and agent fine-grained interaction (in read/write/exec modes) with elements of the application [23]. This is the reason why we propose a framework, called DiValite, which attempts at fulfilling those requirements thanks to two close-related principles that are original wrt similar tools: 1) characters’ structure is part of the DOM structure of the Web page they assist, which allows them to directly manipulate objects in this Web page. For example, this can be particularly useful in applications such as eLearning where it is important for the agent to highlight the most important aspects of the material in order to reduce student’s search cognitive load [6]; and 2) all agents capabilities (animation,
reasoning, dialogue etc.) are implemented in full client mode (hence suffix ‘lite’ in DivaLite name), i.e. the character executes on the client browser. This turns the application lighter and faster to run (hence there is little communication with the server, when necessary). Besides, it also allows developing simple web pages embedded with an animated agent.

Outline: In Section 2, we describe the model and the DOM-integration of the agents; then in Section 3 an evaluation is given in terms of developing costs, followed by a comparative study with similar works.

2. AGENT MODEL

The general architecture of the DivaLite framework is based on the notion of software agent [4]. Given a Web page \( w \in W \), the DivaLite framework enables the programmer to integrate one or several agents into this page to produce a conversational page \( w^* \in W^* \). We have the definitions:

\[
W = \{D\} \\
W^* = \{D,T,A\}
\]

where:
- \( D \) is a set of DOM expressions that represent the application supported by \( W \).
- \( T \) is a set of topics. A topic \( t \in T \) is a symbolic representation over \( D \subset W \) that can be used by the agent. A topic is the symbolic model of an ‘entity of interest’ in the application. It is a symbolic structure, close to Minsky’s frames [15], which is defined as a set of \( attribute \to value \) pairs, such that: \( t = \{a_j \to v_j\}_{i=1,n} \).
- \( A \) is a set of agents. An agent \( a \in A \) is composed of a pair of entities \( a = \{\langle D \rangle, C(a)\} \) where:
  . \( \mathcal{T}(a) \) is the specific topic associated with an agent \( a \).
  . \( C(a) \) describes its attributes, its capacities and its mental model.
  . \( \mathcal{C}(a) \) is the representation of the graphical character of an agent \( a \). It describes how the agent is displayed and how it behaves on screen.

The graphical characters displayed on screen can interact with users through several modalities. Below, we focus on animation movies; bodily actions and textual dialogue.

2.1 Character animation

The basic entity of character animation is the movie. A movie is a sequence of frames displayed successively on screen. A movie \( \mu \) is defined by a triplet: \( \mu = (id_{\mu}, [F_0, F_{i=1,..,r}, F_0], z) \) where:
- \( \mu \in \mathcal{M} \), where \( \mathcal{M} \) is the set of all movies. \( id_{\mu} \) is the identifier of the movie.
- \( F_0 \) is a special ‘coarticulation frame’ starting/ending movies;
- \( F_{i=1,..,r} \) are the individual frames of the movie. A frame is a transparent .png picture of size \( z \) (small = 200 \times 200 pixels, normal = 500 \times 500, large = 1000 \times 1000). Movies are not interruptible (this is not an issue because \( s \) ranges from 6 to 30 frames only).

Movies can be concatenated to form sequences of movies. A sequence \( \sigma \in \Sigma \) is defined as: \( \sigma = \{id_{\sigma}, [\mu_{i=1,..,r} \in \mathcal{M}]\} \) where \( id_{\sigma} \) is the identifier of the sequence. Sequences are dependent on a given application. Sequences are interruptible at movie-ends only. Sequences can be computed dynamically and then played.

Coarticulation between movies is achieved by ensuring that all movies start from the same frame \( F_0 \) and finish with

\[
\begin{array}{|c|c|c|}
\hline
\text{Class} & \text{Function} & \text{Description} \\
\hline
\text{Actions at agent’s initiative} & & \\
\hline
\text{Moving} & \text{jumpTo}(x,y) & \text{C}^a \text{hides and reappears at screen pix x,y} \\
& \text{moveTo}(x,y) & \text{C moves diagonally to x,y} \\
\hline
\text{Size} & \text{shrinkBy}(\%) & \text{C size is shrunk by } \% \text{ (between min/max)} \\
& \text{enlargeBy}(\%) & \text{C size is enlarged by } \% \text{ (between min/max)} \\
\hline
\text{Visible} & \text{hide()} & \text{C is hidden from screen}^b \\
& \text{show()} & \text{C is displayed on screen} \\
\hline
\text{Speed} & \text{incrSpeed}(\%) & \text{Animation speed of C is increased by } \% \\
& \text{decrSpeed}(\%) & \text{resp. decreased (between min/max values)} \\
\hline
\text{Speech} & \text{doSpeech(s)} & \text{Prints string s in C textual ‘balloon’} \\
\hline
\text{Actions at user’s initiative} & & \\
\hline
\text{Moving} & \text{drag} & \text{U}^c \text{ drags C on screen} \\
\hline
\text{Framing} & \text{+/ - buttons} & \text{Four control buttons enable U to shift C in four directions within its frame}^d \\
\hline
\text{Size} & \text{pull} & \text{U can pull up or down the bottom frame border of C to change the size of C} \\
& \text{+/ - buttons} & \text{U clicks controls to change C size} \\
\hline
\text{Speed} & \text{+/ - buttons} & \text{U clicks controls to change speed} \\
\hline
\text{Body} & \text{clicks} & \text{Simple and double click on C frame can be linked to any appli.-dependent function} \\
& \text{mouse over} & \text{Idea for mouse-over events within C frame} \\
\hline
\text{Chat} & \text{textual input} & \text{U can enter textual requests in chat box} \\
\hline
\end{array}
\]

\( ^a \text{C} = \text{graphical character}, ^b \text{When hidden, the agent continues to exist}, ^c \text{U} = \text{User}^d \text{Here, frame is the graphical container}^e \text{Needs external dialog (Litetal)} \)

\( F_0 \). This principle is simple but in practice it has proved to be quite effective, even in the case study of French Sign Language utterances (see applications 11 and 12 in Figure 1), where precise coarticulation is required (although we had to define a specific body/hands position for \( F_0 \) – see [22]).

A special sequence of movies is the Stand-Still Loop (SSL) defined as: \( \text{SSL} = \{\mu_{i=1..r} \in \mathcal{M}\}^\infty \), where: sign \( \star \) means that SSL movies are not played in 1 to \( r \) order but rather \( \mu_i \) are randomly chosen according to their probability \( p_i \); resp. sign \( \infty \) means that SSL is played continuously, during a given session.

The purpose of SSL movies is to support the ‘background animation’ of a character, i.e. its behavior when it is idle. A character that stays still, showing no move, would appear very robotic for the user, decreasing agent’s believability [7]. For this reason, it is desirable that it presents some idle behaviors such as blink or tap the foot.

In fact, movies or sequences of movies are inserted into SSL when they are launched. At any SSL \( \mu_i \)-end, SSL can be interrupted to insert either: a) a single movie \( \mu \in \mathcal{M} \) or b) a single sequence \( \sigma \) of movies. For the time being, this operation is not recursive, i.e. one cannot insert a movie or a sequence into any other sequence than SSL.

2.2 Bodily actions

Characters bodily actions appear on screen as graphical movements distinct from animation movies (e.g. displacement, shrinking, hiding etc.). Each action is implemented by an individual algorithm, in terms of operations over the DOM structure of the character. These actions can be triggered a) internally at the initiative of the agent control-
ling the graphical character or b) externally at the initiative of the user, mainly through mouse and click actions. The Divalite’s main bodily actions are displayed in Table 1.

### 2.3 Natural language interaction

In previous works listed in Table 2, user/agent textual natural language dialogue and agent’s symbolic reasoning upon DOM page topics were supported by server technology, which has several drawbacks with regard to our objective of simplicity: not all applications require those capabilities; server technology is complex and it increases developing costs. Therefore, these considerations prompted two decisions: 1) dialogue and reasoning are now supported in a separate project, called Litetalk that can be easily integrated in Divalite; 2) Litetalk works in full client mode, hence no server technology is required when it is integrated into an application. Litetalk is freely available on the Web at http://perso.limsi.fr/jps/interviews/litetalkwebsite/litetalk.site.main.html.

### 2.4 DOM structure of the characters

The main objective of Divalite is to provide a framework to develop characters that are fully integrated into the W3C Document Object Model (DOM) structure of the Web page where they appear. This is the reason why the directories and files structures that compose a Divalite character are manipulated through the DOM. As the Divalite framework operates only HTML objects, it does not need the installation of any special plug-in in the browser, as it occurs in Flash-based agent frameworks, for example. In order to insert a Divalite character in a HTML Web page, it is necessary to create dynamic elements of class <div>, also called ‘containers’:

- The Main container is responsible for delimiting the area of the Web page where the character is displayed. It defines the \( C_{x,y} \) coordinates of the character and it implements methods such as Drag&Drop and Resize defined in Table 1.
- The CharPicture container contains pictures displayed to animate the character. Character animation is based on the ‘key-framing’ technique: each frame that composes the animation is generated a priori and stored in the ‘picture repository’. A given animation is generated by displaying the frames sequentially, intercalated with a pause interval time; this interval can be set dynamically hence implementing the animation speed feature. Four distinct characters are available: two are iconic (see 1 and 6 in Figure 1) and two are realistic (i.e., modeled from real people – see 5 or 12 in Figure 1). Each character can perform a set of 63 distinct movies, which have been developed by authors at LIMSI-CNRS over years (extra movies will be added in the future to enrich the library).
- The SpeechBalloon container is responsible for presenting above the head of the character agent’s speech textual output with several textual attributes (color, size, fonts...) enabling the expression of agent’s affects with conventions similar to cartoons.
- The Control container is composed of six buttons that allow setting the appearance of the character (size, position within the main container) and also a cursor controlling the animation speed. It is presented whenever the user positions the mouse over the main container (resp. disappears on mouse out event).

### 2.5 Graphical design of the Characters

Applications with animated virtual characters have been deployed in both Web pages and standalone environments. They rely on two main approaches: pre-synthesized animations and generated animations.

- Pre-synthesized animations are first synthesized, and then selected according to the context. Two main techniques are used: 1) motion capture [16] is often used in film applications. This technique requires a sophisticated hardware/software infrastructure. While efficient for bodily movements, they lack in precision for hand/head expressions. 2) rotoscopy [12] consists in copying an animation from a video model. This technique requires a basic equipment and it relies on the personal skills of graphical experts. However animations of very good quality can be synthesized in terms of realistic expressions, gestures and body movements.

- Generated animations are an alternative approach consisting in automatic and real-time generation of animations from description languages. It allows the generation of animations that provide flexibility, but at the cost of realism.

Divalite characters have two strong requirements:

1) a good level of realism in facial and gestural movements is necessary in the context or research with ECAs;
2) characters must be part of the DOM structure, not encased in a separate box. For those two reasons, pre-synthesized animations were preferred. In this context, rotoscopy was chosen because: 1) the limited number of animations (<100) makes it tractable and 2) rotoscopy enables very realistic animations.

### 2.6 History of Divalite toolkits

The Divalite toolkit and its documentation are freely available on the Web. This project is a sequel from two previously developed software toolkits, namely WebLEA\(^1\) and Diva\(^3\) developed in the context of the research upon ECAs at LIMSI-CNRS, within the INTERViews\(^4\) project [21, 19, 20] and the subsequent DAFT\(^5\) project [10, 23]. The three toolkits share two main features: 1) characters are placed in DOM <div> structures, not in external objects; 2) they are animated by a JavaScript-based engine (jQuery technology), working in full client mode, hence no extra server technology or external plug-ins are required. Their distinctive features are listed in Table 2. These toolkits are also available on the Web and freely usable for teaching and research purposes.

### 3. EVALUATION

In this section, we present a preliminary evaluation of the Divalite architecture. Our objective is to provide:

\(^{1}\)http://perso.limsi.fr/jps/online/divalitewebsite/divalite.site.html

\(^{2}\)http://perso.limsi.fr/jps/online/leaexamples/leawebsite

\(^{3}\)http://perso.limsi.fr/jps/online/diva/divahome

\(^{4}\)http://perso.limsi.fr/jps/interviews

\(^{5}\)http://perso.limsi.fr/jps/research/daft

---

Table 2: History of Divalite toolkits

<table>
<thead>
<tr>
<th>Toolkit</th>
<th>Year</th>
<th>Char.</th>
<th>(N_{char})</th>
<th>Natural language</th>
<th>Server technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebLEA</td>
<td>2006</td>
<td>C</td>
<td>4</td>
<td>Built-in</td>
<td>yes(^a)</td>
</tr>
<tr>
<td>Divalite</td>
<td>2008</td>
<td>C;I</td>
<td>4;2</td>
<td>Built-in</td>
<td>yes(^a)</td>
</tr>
<tr>
<td>Divalite</td>
<td>2011</td>
<td>I;R</td>
<td>2;2</td>
<td>External</td>
<td>no</td>
</tr>
</tbody>
</table>

Character class: C = cartoon-like; I = iconic; R = realistic

\(^{a}\) Server handles natural language session and reasoning over DOM page.
1) an intrinsic measure of the effort necessary when one is integrating a DivaLite agent into a Web page.
2) a comparative measure of the gain in developing costs with regard to similar toolkits listed in Table 2. A set of twelve DivaLite-based applications was selected and re-developed with DivaLite. They can be accessed through the DivaLite Web site where it is possible to inspect their code online (both HTML and JavaScript code).

Applications have been programmed by a single programmer, selected with two requirements: 1) no knowledge of the DivaLite internal code; 2) medium skills in Web and JavaScript programming. Such a profile can be considered typical of potential DivaLite toolkit users. This aspect was reinforced by the fact that the applications were programmed in increasing order of complexity, say from simple tutorial pages to more complex applications. Profiling the program duration was more difficult to assess with precision, via an afterward questionnaire. It is expressed in hours and it is estimated within a ±30% margin; however we are mainly interested here in the magnitude and in the differences across the applications (again, the comparison is facilitated with a single programmer).

3.1 Results

From the results summarized in Table 3 it is possible to draw some qualitative considerations about DivaLite:

- The number of code lines dedicated to DivaLite agents programming \( C_D \) is of same order of magnitude as the number of lines \( C_W \) (i.e. without the agent(s)); this is true even in the case of very simple applications \((N = 1, 7, 10)\) for example). Hence adding agents requires minor overhead.
- Adding extra visible agents on screen \((N = 2, 8)\) or switchable (for example between a boy and a girl, as in \(N = 11, 12\)) results in mere replication of code in the HTML page but it does not increase programming time to install the characters. However these applications tend to be more complex (for example the duo chat in \(N = 8\)) thus requiring specific developing efforts.
- Adding natural language interaction increases both code size and programming time but it is associated with an extra functionality (natural language handling) supported by the LiteTalk toolkit that should not be attributed to the DivaLite toolkit; hence it was separated with ‘|’ in applications \(N = 7, 8, 9\).

<table>
<thead>
<tr>
<th>Application</th>
<th>(N_a)</th>
<th>(C_W)</th>
<th>(C_D)</th>
<th>(T_W)</th>
<th>(T_D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Page</td>
<td>1</td>
<td>25</td>
<td>48</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Character List</td>
<td>1[4]</td>
<td>153</td>
<td>114</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Chart Pointing</td>
<td>4</td>
<td>39</td>
<td>80</td>
<td>1</td>
<td>8⁹</td>
</tr>
<tr>
<td>Mosquito Chase</td>
<td>2</td>
<td>14</td>
<td>110⁰</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Gaze Awareness</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Simple Chat</td>
<td>1</td>
<td>26</td>
<td>32/23³</td>
<td>1</td>
<td>1/4</td>
</tr>
<tr>
<td>Duo Chat</td>
<td>2</td>
<td>27</td>
<td>60/98</td>
<td>1</td>
<td>4/7</td>
</tr>
<tr>
<td>Counter Control</td>
<td>1</td>
<td>63</td>
<td>32/56</td>
<td>4</td>
<td>1/10</td>
</tr>
<tr>
<td>Simple Home Page</td>
<td>1</td>
<td>46</td>
<td>25</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Gestural Agents</td>
<td>1²</td>
<td>51</td>
<td>67</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>Sign Teacher</td>
<td>1²</td>
<td>179</td>
<td>191</td>
<td>20</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 3: Evaluation of developing costs

\(Id\) refers to screen shot numbers in Figure 1. \(N_a\) is the number of DivaLite agents available in the application, whether they are visible on screen or hidden. ⁸\(1[4]\) means that only one agent is visible, but it can be switched among 4 characters, resp. for \(N = 11, 12\). ⁹\(C_W\) programming time for the required XY-pointing algorithm (can be reused in other applications). ³\(object mosquito is actually programmed as a DivaLite agent moving on screen (displayed as an animated .gif). ⁴\(in LiteTalk-based pages we separate the DivaLite code and time (left) from LiteTalk code and time (right) by \(|\).
Because the applications were programmed as a sequence according to their growing complexity (N = 1, 10, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12), thus mimicking the process of a person who learns about DiveLite and progressively builds larger applications, the programming time TD was quite constant. There is an exception for pointing, awareness and tracking (N = 4, 5, 6) due to the developing of extra functionalities for characters that could be reused in further applications and could be integrated in the API of the next version of the DiveLite toolkit.

Finally, programmers having previously used the DIVA architecture, can be tempted to compare the results of applications in Table 3 with their corresponding developing efforts required with DIVA (when available). Unfortunately, time efforts were not recorded in DIVA so the comparison can only be done about code size; moreover DIVA applications were developed by several programmers in various circumstances, certainly not controlled. However to give an idea, values for the "Counter control" application (N = 9) in DIVA are: C\text{DIVA} = 46, C\text{D}\text{P} = 107 [179]; resp. for "Gestural agents" (N = 11): C\text{DIVA} = 99, C\text{D}\text{P} = 221. This data shows that the integration of DiveLite agents is significantly easier in terms of code size and (very probably) in terms of programming time, than in DIVA architecture.

4. RELATED WORKS

Projects based on virtual agents in the Internet

Many early projects were based on Microsoft Agents Technology [14], which enables the animation of cartoon-like characters in virtual worlds that target social chat activities in creating online 3D virtual environments. Online 3D virtual worlds like Second Life, Active Worlds, Twinity, IMVU. The feature is: available (+); not available (-); available with add-ons(x). Many researchers have studied virtual communities frameworks [17], and some have used them for educational purposes. For example, Arya et al. [2] address the issue of online education through 3D avatar-based virtual environments. Online 3D virtual worlds like Second Life, Active Worlds, Twinity, IMVU etc. are also examples of virtual worlds that target social chat activities in creating social meeting places on the Web (see Boulos et al. [3] for a review of their usage in health and medical education).

Chatterbots

In early chatterbots the user and the agent interacted through text-based dialog sessions. Nowadays, Web-based chatterbots tend to incorporate virtual characters, which are provided by corporate companies such as Oddcast\textsuperscript{\textregistered}, LaCantoche\textsuperscript{TM}, Charame\textsuperscript{TM} etc. Outstanding chatterbots are: Alicebot, Jabberwacky, Elbot, HAL etc. (see [25] for a comparison). There are two main drawbacks to these Web frameworks: 1) Virtual characters are physically encased into a 'box' (\textlt;iframe\gt, Flash\textsuperscript{TM} objects…), for instance at the top left-hand corner of the Web page. Hence their interaction with the rest of the page is minimalistic: no drag-and-drop over the page, no deictic gesture, etc. 2) Agents have little or no access to the DOM-structure and to the informational content of the Web page. This makes it very difficult for them to interact a) with the page content (no browsing and action on DOM objects of the page…) and b) with the users, e.g. for assisting purposes.

| Table 4: Comparing DiveLite features |

<table>
<thead>
<tr>
<th>Feature Description</th>
<th>M</th>
<th>F</th>
<th>V</th>
<th>W</th>
<th>DL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Animation</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Animation performed inside client page</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Multiple resolutions for characters dep. on devices</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Multiple levels of character realism</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Real-time 3D characters</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Head realism (precise lip-synch)</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Gesture realism (precise deictic gestures)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Legs realism (walk, salt etc.)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Complex emotions playing</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Complex movements performing</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Number of characters (≈ many)</td>
<td>4</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>4</td>
</tr>
<tr>
<td>2. Mouse and click operations</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>∞</td>
<td>+</td>
</tr>
<tr>
<td>Character is draggable freely on page</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Character is resizable with mouse</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Handling of click-events on character's body parts</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>∞</td>
<td>-</td>
</tr>
<tr>
<td>3. Programmable actions of the character on screen</td>
<td>-</td>
<td>-</td>
<td>∞</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Multiple characters on the same page</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Interaction between characters on the same page</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Same character 'following' on multiple pages</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Possible interleaving with DOM objects (e.g. z-index)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Attaching extra (resizeable) DOM objects to characters</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Character can hide/show/resize</td>
<td>+ + +</td>
<td>+ + +</td>
<td>≈</td>
<td>≈</td>
<td>+</td>
</tr>
<tr>
<td>Character can move on page</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Character can finger-point a static pixel x,y on page</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Character can gaze-track a moving pixel x,y on page</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Opens source availability</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Loading time/character (- long, + short)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>4. Natural Language Interaction</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Textual input chatbox and textual output balloon</td>
<td>+ + + +</td>
<td>+ + + +</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Natural Language request handling</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Natural Language control/command of DOM objects</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Text to Speech and Sound play</td>
<td>+</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Main features of DiveLite (DL) are analyzed against four technologies: Microsoft agents (M); Flash\textsuperscript{TM}-based agents (F); VRML-based agents (V); Virtual worlds (W). For V and W, four main platforms were analyzed: Second Life, Active Worlds, Twinity, IMVU. The feature is: available (+); not available (-); partially available (%), available with add-ons(x).

4.1 Feature comparison

Conditions In order to assess the originality of DiveLite architecture, it is necessary to compare its main features with the Web tools supporting virtual characters that were previously cited in this section.

Results Comparisons of features are listed in Table 4 under four main categories:

1. Animation: While VRML technology offers powerful graphical capacities, simple rotoscopy-based DiveLite characters can compete in realism of expression, not in lip-synch but in facial gestural and body movements expressions. Actually, DiveLite has restrictions (number of characters, real-time synthesized movements) but it is much easier to integrate and lighter to run than VRML-based applications. Besides, VRML is a 3D technology; it is much more adequate to develop agents inserted into a 3D virtual environment. On the other hand, DiveLite aims at allowing easy development of agents for Web pages and Web environments.

2. Mouse and click operations: are a strong point of DiveLite because being part of the DOM structure, characters are endowed with easy-to-program mouse and key events. Due to this integration of DiveLite with the DOM structure of a web page, a DiveLite agent can point to and manipulate objects in the web environment where it is inserted into. This is an important feature, since some stud-
ies about animated pedagogical agents, for example, have shown that the “image effect” (the presence of an animated pedagogical agent has more impact on the students’ learning than a narration) only occurs if the agent interact with the material that it teaches (points to, highlights more important aspects, etc.) [6]. Note that loading time of animations is a weak point versus VRML or even Flash\textsuperscript{TM}, but it can be overridden with browser caching and background loading techniques;

3. Programmable actions of the characters on screen: is the strongest point of DivAlite that offers a large set of basic actions and the possibility to combine them into complex behaviors on screen, in quick dynamic response to internal events or to user’s events. Again, this is due to the intrinsic DOM structure of the characters.

4. Natural language interaction: the external LiteTalk toolkit provides DivAlite with a separate but easy-to-integrate natural language processing technology.

5. CONCLUSION

DivAlite (and LiteTalk) toolkits are fully operational and freely available for easy developing of research studies upon conversational agents in the Internet. With regard to this purpose, the strong point of DivAlite is that characters are part of the DOM structure, which makes it quick to develop experiments where dynamic animations and rich interactions (both with users and page objects) are a key requirement. Actually, we think that this approach could apply to other classes of Web agents (e.g. for Sign language teaching, as in Fig. 1-12); hence in further work, we intend to extend its scope in Internet applications.

6. REFERENCES


