Specification of an Open Architecture for Interactive Storytelling

Nicolas Szilas[1], Thomas Boggini[1], Monica Axelrad[1], Paolo Petta[2], Stefan Rank[2]

[1] TECFA, FPSE, University of Geneva, CH 1211 Genève 4, Switzerland
{Nicolas.Szilas, Thomas.Boggini, Monica.Axelrad}@unige.ch
[2] Austrian Research Institute for Artificial Intelligence OFAI,
Freyung 6/6, A 1010 Vienna, Austria
{Paolo.Petta, Stefan.Rank}@ofai.at

Abstract. This article introduces *OPARIS*, an OPen ARchitecture for Interactive Storytelling, which aims at facilitating and fostering the integration of various and heterogeneous Interactive Storytelling components. It is based on a modular decomposition of functionalities and a specification of the various messages that different modules exchange with each other.

Keywords: Interactive Digital Storytelling, Interactive Narrative, Architecture, Narrative Engine, Behaviour Engine, Animation Engine.

1 Motivation for an Open Architecture

This last decade, a significant number of Interactive Digital Storytelling (IDS) systems have been implemented [1]. In order to provide the end-user with influence on the storytelling process, these systems include a core narrative engine, that is responsible of choosing/generating appropriate actions during the experience, and various peripheral yet essential technologies: menu-based interfaces text recognition/ generation technologies, behaviour engines, camera engines, light engines, etc. Most systems above integrate their own set of technology, within an in-house architecture, but there has been limited integration *between* these systems to date.

This lack of integration is problematic because it limits the ability of each research team to experiment/validate/demonstrate their core contribution. Furthermore, with limited integration, the spectrum of available technologies ready for applications remains limited. More precisely, our motivation is twofold:

From an engineering point of view, it is desirable that, within a given architecture, a module could be replaced by another one with limited effort. Furthermore, a well-defined API enables modules to be easily distributed and supports the development of new compatible modules.

From a more theoretical and narrative viewpoint, our goal is to formalize the dependencies between the narrative level and the medium, by studying which narrative phenomena can be described within the Narrative Engine, which one cannot and finally which information needs to circulate between modules to generate these phenomena. More generally, an open architecture gives the opportunity to formalize what circulates between the architecture's components in narrative terms, rather than in raw Computer Science terms, without being confined to a specific theory of IDS.

2 General Design of OPARIS

2.1 Design Principles

Flexibility: OPARIS aims at both creating new IDS modules and integrating existing ones. Therefore, a key principle of OPARIS is to provide several levels of integration, from an ideal case, where each module works according to the documented API, to cases where modules have followed different specifications.

Simplicity: our users are not system engineers but researchers in IDS and advanced media designers. Therefore, the architecture specification must remain simple, avoiding too much abstraction and not be over-constrained.

Narrativity: We aim at fostering a narrative point of view on IDS via the architecture. Therefore, OPARIS uses an existing set of already defined IDS concepts [2] to specify and name the data circulating between modules.

2.2 Modular decomposition

The modular decomposition is inferred from the various existing IDS architectures developed within each system [3,4,5,6]. This decomposition cannot be unique. The granularity of modularity varies between systems. This implies that *modules* should be differentiated from *functionalities*. Modules are isolated software components that perform one or more functionalities (Table 1).

Table 1. Modules in the proposed architecture, with the associated typical functionalities. The two last modules are composite modules.

Module	Functionalities
Narrative Engine (NE)	Calculates actions that must be displayed to the end-user.
Behaviour Engine (BE)	Transforms behaviours into series of elementary animations.
Animation Engine (AE)	Animates a 3D or 2D model of a character and displays other information to the end-user.
User Interface (UI)	Takes input from the end-user.
Music Engine (ME)	Plays background music according to the unfolding of the story.
Theatre (TH)	Displays the story events and descriptions and manages the end-user's interaction.
Small Theatre (ST)	Displays the story events and descriptions and manages the end-user's interaction. In case of 3D, it excludes the BE.

2.3 Physical implementation

Each module is an independent software component that communicates with the rest of the architecture with TCP sockets. Data circulating within the architecture are coded in XML. The specification of the API for each module is supported by an XML schema (XSD), which is provided to the users of the architecture.

In the architecture described so far, each module's output must be compatible with every other module's input. However, this is not always the case when working in a real integration case. In particular, the NE outputs narrative actions, which do not necessarily match the behaviours of the BE. A conversion must be performed at the NE or the BE level, specifically for each integration. This hinders the interoperability of the whole architecture. To solve this issue, a new module is introduced, the **Director**, which follows the design pattern of a *Mediator*. It performs three functions: (1) It receives all messages from the other modules and redirects them to the proper module(s), based on the content of the messages; (2) It performs a semantic conversion between modules (e.g. it translates the scenario-specific vocabulary outputted by the NE to the behaviour's catalogue terms in use by the BE); (3) If a module is not compliant with the API, it can perform a syntactic conversion of data from and to this module. The Director plays a role of both routing and conversion. For the conversion, taking advantage of the XML framework, the Director uses an XML transformation language, XSLT, dedicated to transforming an XML document into another XML document. A single ".xslt" file enables to manage all the conversions needed. This enables the building of a specific architecture in an easy manner, without entering into the Director.

In numerous situations, data from one module to another does not need to be transmitted directly as a command but rather should be shared in a central place, accessible to all modules. Therefore, a module called the ***Shared Narrative States Repository*** **(SNSR)** is introduced. It enables any module to store narrative data that is useful to one or more other modules. These modules can either take these data when they need it (pull method) or subscribe to certain data and receive notifications when it changes (push method). It is recommended that the shared data follow the conceptualisation of IDS concepts mentioned above [2].

To further increase the openness of OPARIS, an optional module is introduced, an **adapter**, that is inserted between a module and the Director, to cover cases where a module does not use XML and/or is a TCP server itself. Several adapters can be used within the same specific architecture.

3 Modules' API

Each module described in Table 1 is specified via its API. Since OPARIS is language independent, the API is not described in a specific language (Java, C#, etc.). Instead, it focuses on communication protocols, i.e., all messages sent and received by modules are specified, first in a reference document, then via examples of XML messages and the corresponding XML schema. The specification is accessible online [7] and specifies 15 types of messages. Here follows an example of a message from the NE to the BE:

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
    <ACTION_TO_PERFORM>
        <act>
            <id>34</id>
            <actionType>talk</actionType>
            <actor>Amanda</actor>
            <actionParameters>
                <actionParameter parameterName="addressee">John</actionParameter>
            </actionParameters>
        </act>
    </ACTION_TO_PERFORM>
</message>
```

The above message corresponds to an action named “talk”, with the character *Amanda* as actor and the character *John* as a parameter of the talk action, the parameter being named “addressee”. (See [7] for the corresponding XML schema).

4 Conclusion

In order to foster the collaboration between researchers, we have proposed OPARIS, an open architecture for IDS. While not yet tested at a large scale, OPARIS has been used to connect one Narrative Engine (IDtension [5]) to four different Theatres.

Acknowledgements

This work has been funded (in part) by the European Commission under grant agreement IRIS (FP7-ICT-231824) and the Swiss National Science Foundation (J. Dumas and N. Szilas, principal investigators).

References

1. Interactive Storytelling Systems, http://tecfalabs.unige.ch/mediawiki-narrative/index.php/IS_Systems
2. Interactive Storytelling concepts, http://tecfalabs.unige.ch/topincs/
3. Mateas, M. and Stern, A.: Integrating Plot, Character and Natural Language Processing in the Interactive Drama Façade. In: Göbel et al. (eds.) TIDSE’03, 139--151. Frauenhofer IRB Verlag (2003)
4. Young, R.M., Riedl, M.O.: Towards an architecture for intelligent control of narrative in interactive virtual worlds. In: IUI 2003, pp. 310—312. ACM, New York (2003)
5. Szilas, N., Barles, J., Kavakli, M.: An implementation of real-time 3D interactive drama. Computers in Entertainment 5(1) (Jan 2007)
6. El-Nasr, M.S.: Interactive Narrative Architecture based on Filmmaking Theory. Int. J. Intell. Games & Simulation 3(1), 29--36 (March 2004)
7. Oparis API, http://tecfa.unige.ch/~szilas/iris/oparis/