OPARIS

Open ARchitecture for Interactive Storytelling

Technical Documentation

v1.0

Dec. 15, 2011

Nicolas Szilas

Table of Contents

1. OPARIS general description

Oparis is a specification of an architecture for Interactive Digital Storytelling (IDS).

It consists in specifying a series of well defined Application Programming Interfaces (API) for each module in a global architecture. Within such an architecture, it will be possible to easily add/replace a module and therefore produce a variety of specific IDS systems with minimal effort.

Main technical choices:

– Module can be implemented in any language and hardware. Communication between them occurs via sockets.

– Message are defined according to narrative concepts. In particular, the following list of concepts has been used: http://tecfalabs.unige.ch/topincs/

– Messages are coded in XML

– The API consists in an XML Schema (.xsd), available here: http://http://tecfa.unige.ch/~szilas/iris/oparis/oparis.xsd

– Modules can communicate directly, but a preferred protocol of communication consists in using a central module called a **Director**, for passing (and converting, see below) messages between modules. The Director is a socket server, while all other modules are socket clients.

– The modules do not need to specify the target of each messages: routing is carried out by the Director, based on the name of each message. Routing information is coded in a separate file.

– The Director can also process a *transformation* of messages, in case a module is not compatible with the Oparis API. This conversion is specified via an XML transformation sheet (.xsl).

– If a module needs to connect to the Director but is a socket server instead of a socket client, a specific module called an **Adapter** performs the conversion.

– An alternative way to communicate consists in using a shared repository rather than the direct exchange of messages. This repository is called the *Narrative State Repository* (NSR).

2. Specification of the API

2.1 Introduction

This part describes the API for each module in the open architecture. It includes all input and output messages that circulate between them.

Messages are specified via a *message name* and several *parameters*. Messages are emitted by one module, and have one or more receiver modules (destination). Parameters are typed, from simple types (string, integer, boolean) to complex types.

In the following, the messages are sorted alphabetically. Each message is described by:

- Its name: The name of the message (in bold capitals).
- Its transmission: from which module to which module(s) the message is sent. This information is given regardless of the existence of the Director in the architecture. Note that the list of destinations is not final, since other modules might need the message too.
- Its description: A text explaining the message.
- Its parameters (optional): The list of parameters, if there are any. Each parameter has a type and some optional attributes specifying if the parameter is optional (it can be omitted) or multiple (there can be two or more instances of this parameter).
- An example of a portion of XML that codes the message.

Before describing the messages, a certain number of *types* are introduced, that intervene in the message specification.

Important notice: The structuring of parameters below does not fullly specifies the way the XML messages are coded (for example, coding as attribute vs value, a classical XML issue, is not specified). Readers need to refer to either the example or the .xsd file to obtain all information.

2.2 Types

Type: **ActionParameterType**

Description: Parameter that specifies the action.

Parameters:

Name	Type	Attribute	Description
parameterName	string		Name of the action parameter
value	string		Value of the action parameter

Type: **DescribedAction**

Description: An action described textually.

Parameters:

Name	Type	Attribute	Description

id	int		Unique identifier of the action
description	string		Text describing the action

Type: **PerformanceConstraintType**

Description: Constraint that modifies how an action must be performed.

Parameters:

Name	Type	Attribute	Description
constraintName	string		Name of the constraint. Known examples of constraints names are: – perceived: Boolean specifying if the action is perceived by the end-user. – priority: Integer defining the level of priority of the action. The value 0 is defined as the normal priority. – Expressivity: Qualifies the action from an expressive point of view
value	string		Value of the constraint

Type: **StructuredAction**[1]

Description: An action described structurally (with an action type and parameters).

Parameters:

Name	Type	Attribute	Description
id	int		Unique identifier of the action.
actionType	string		Type of the action (usually a verb)
actor	string		Character causing the action
actionParameter	ActionParameterType	optional, multiple	Parameters of the action. They depend on the action type.

[1]The case of narrative happenings, that is when no actor is attached to a narrative event (“the sun rises”, “it rains”) has not been included in the API for the sake of simplicity. It can be accounted for two different ways that are up to the IDS designers and authors. Either an empty string in the actor field denotes such a happening or an impersonal actor is introduced (“someone”).

2.3 Messages

Message name: **ACTION_RETURN**

Transmission: Behaviour Engine or Theatre →Narrative Engine

Description: Message describing the status of an action initially sent by the Narrative Engine.

Parameters:

Name	Type	Attribute	Description
actionId	integer		Unique identifier of the action.
status	string		Status of the action. Possible values for status are: “ongoing”, “failure”, “success” and “successOngoing”. Other status values might be added in a future version of the API.

XML Example:

```
<ACTION_RETURN>
      <actionId>6</id>
      <status>success</status>
</ACTION_RETURN>
```

Message name: **ACTION_TO_PERFORM**

Transmission: Narrative Engine → Behaviour Engine or Theatre

Description: An action that must be performed by the BE or Theatre

Parameters:

Name	Type	Attribute	Description
act	StructuredAction		Action to be performed.
constr	PerformanceConstraintType	optional, multiple	Constraints that modify the way the action must be performed.

XML Example:

```
<ACTION_TO_PERFORM>
      <act>
            <id>6</id>
```

```
            <actionType>dialog1default</actionType>
            <actor>Paul</actor>
            <actionParameters>
                  <actionParameter
parameterName="listener">Julia</actionParameter>
                  <actionParameter parameterName="content">What a nice day!
Let's enjoy this lovely day together!</actionParameter>
            </actionParameters>
      </act>
      <constr constraintName="perceived">TRUE</constr>
</ACTION_TO_PERFORM>
```

Message name: **ANIMATION_RETURN**

Transmission: Animation Engine →Behaviour Engine

Description: Message describing the status of an animation sent by the Behaviour Engine.

Parameters:

Name	Type	Attribute	Description
animationId	integer		Unique identifier of the animation
status	string		Status of the animation. Values for status are: “failure”, “success”.

XML Example:

```
<ANIMATION_RETURN>
      <animationId>13</id>
      <status>failure</status>
</ANIMATION_RETURN>
```

Message name: **ANIMATION_TO_PERFORM**

Transmission: Behaviour Engine → AnimationEngine

Description: Animation to be performed by the Animation Engine.

Parameters:

Name	Type	Attribute	Description
animation	StructuredAction		Animation to be performed. Structurally, it is an action.
attribute	string	optional, multiple	Any attribute that specifies how the

			animation is performed. It depends on the type of Animation Engine.

XML Example:

```
<ANIMATION_TO_PERFORM>
      <act>
            <id>5</id>
            <actionType>moveTo</actionType>
            <actor>Paul</actor>
            <actionParameters>
                  <actionParameter
parameterName="target">Julia</actionParameter>
            </actionParameters>
      </act>
</ANIMATION_TO_PERFORM>
```

Message name: **ASSOCIATION**

Transmission: Narrative Engine → menu-based User Interface (or Theatre which embeds it) - optional

Description: The association between a fictional data (its name) and an action to be selected by the end-user (its ID). Assuming that the User Interface handles these fictional data by their name, it enables to access actions by these data. For example, accessing an action via the characters involved in this action.

Parameters:

Name	Type	Attribute	Description
fictDataName	string		Name of fictional data
actionId	int		Unique identifier of the action associated to the fictional data.

XML Example:

```
<ASSOCIATION>
      <ficDataName>Julia</ficDataName>
      <actionId>6</actionId>
</ASSOCIATION>
```

Message name: **CONTEXTUAL_FILTER**

Transmission: Narrative Engine → menu-based User Interface (or Theatre which embeds it) - optional

Description: Provides a contextual information that enables the User Interface to present the available options differently.

Parameters:

Name	Type	Attribute	Description
type	string		Type of the contextual filter. Types are: – "addressee": character the end-user is interacting with. – "place": location of the action. – "userState": description of the estimated user state.
value	string		Value of the filter.

XML Example:

```
<CONTEXTUAL_FILTER type="addressee">Paul</CONTEXTUAL_FILTER>
```

Message name: **DISPLAYED_ACTIONS**

Transmission: menu-based User Interface → Music Engine - optional

Description: List of actions that are currently shown or focused on the graphical user interface, so that the user is able to choose among them. It is relevant for menu-based interfaces only.

Parameters:

Name	Type	Attribute	Description
actionId	integer	multiple	Unique identifier of the shown action.

XML Example:

```
<DISPLAYED_ACTIONS>
      <actionId>6</id>
      <actionId>9</id>
</DISPLAYED_ACTIONS>
```

Message name: **DISPLAY_INFORMATION**

Transmission: Narrative Engine → User Interface (or Theatre which embeds it) - optional

Description: Provides additional information useful for the User Interface. It depends on the type and sub-type of User Interface. For example, in the history-based User Interface, the annotated lines of history need to be sent to the User Interface.

Parameters:

Name	Type	Attribute	Description
type	string		Type of the additional information. The identified display information types are: “historyLine” and “location”.
value	string		Value of the display info.

XML Example:

```
<DISPLAY_INFORMATION type="historyLine">You enter the living
room.</DISPLAY_INFORMATION>
```

Message name: **FRAGMENT**

Transmission: Narrative Engine → Animation Engine or Theatre

Description: A descriptive text, possibly formatted in HTML, that must be displayed by the Animation Engine or Theatre.

Parameters:

Name	Type	Attribute	Description
fragment	string		Text string or html string, that describes a set of narrative events or states.
fragmentType	string	optioanl	Type of the fragment. Types are: “foreword”, “introduction”, “ending”, “intertitle”.

XML Example:

```
<FRAGMENT fragmentType="introduction">This is an interactive game in which you
will play the role of Frank.</FRAGMENT>
```

Message name: **NO_ASSOCIATION**

Transmission: Narrative Engine → menu-based User Interface (or Theatre which embeds it) - optional

Description: Invalidates all associations previously sent.

Parameters: None.

XML Example:

```
<No_ASSOCIATION/>
```

Message name: **PARANARRATIVE_COMMAND**

Transmission: User Interface or Theatre → Narrative Engine

Description: paranarrative commands, that is commands that are not part of the story itself but still play a role in the interactive narrative experience. They correspond to the narrative concept of *paratext* as introduced by G. Genette.

Parameters:

Name	Type	Attribute	Description
type	string		Name of the command. Names include: “quit”, “restart”.
value	string	optional	A value associated with the command. This anticipates further evolution of the API.

XML Example:

```
<PARANARATIVE_COMMAND type="quit"/>
```

Message name: **POSSIBLE_USER_ACTION_ATT**

Transmission: Narrative Engine → attribute-based User Interface (or Theatre which embeds it)

Description: An action that the end-user can choose to perform. This message is used for attribute-based User Interface, that is an interface that enables the user to select the action according to one or more attributes of the action. For example, selecting an action according to it’s emotional attributes, or the associated speech act.

Parameters:

Name	Type	Attribute	Description
id	int		Unique identifier of the action
description	string	optional	Description of the action
attribute	string	multiple	Attribute of the action.

XML Example:

```
<POSSIBLE_USER_ACTION_ATT>
      <act>
            <id>3</id>
            <attribute>Negative-Active</attribute>
      </act>
</POSSIBLE_USER_ACTION_ATT>
```

Message name: **POSSIBLE_USER_ACTION_MNU**

Transmission: Narrative Engine → menu-based User Interface (or Theatre which embeds it)

Description: An action that the end-user can choose to perform. This message is used for a menu-based User Interface.

Parameters:

Name	Type	Attribute	Description
act	DescribedAction		Action to be selected.

XML Example:

```
<POSSIBLE_USER_ACTION_MNU>
      <act>
            <id>3</id>
            <description>Talk to Paul to calm him down</description>
      </act>
 fragmentType="introduction">This is an interactive game in which you will play
the role of Frank.
</POSSIBLE_USER_ACTION_MNU>
```

Message name: **USER_ACTION**

Transmission: User Interface or Theatre → Narrative Engine (and Music Engine, optionally)

Description: The action that the end-user has selected (explicitly or implicitly).

Parameters:

Name	Type	Attribute	Description
id	int		Unique identifier of the action

XML Example:

```
<USER_ACTION>
      <id>3</id>
</USER_ACTION>
```

Message name: **USER_ACTIVITY**

Transmission: User Interface or Theatre → Narrative Engine - optional

Description: An information related to the physical activity on the interface, beside the selected action itself. For example, it can be related to the pace of clicks.

Parameters:

Name	Type	Attribute	Description
type	string		Name of the user's activity that is transmitted. Types are not defined.
value	string	optional	A value associated with the user activity.

XML Example:

```
<USER_ACTIVITY type="mouse_velocity">5.4</USER_ACTIVITY>
```

2.4 Communication with the NSR

The Narrative State Repository uses a publish/subscribe mechanism to exchange data. It is also possible to consult data directly (query/result mechanism). This is performed via the 7 following messages:

subscribe (→ NSR): Any module can subscribe to a given data.

Message example:

```
<subscribe>
      <data id="data1"/>
      <data id="data2"/>
</subscribe>
```

status(NSR →): Returns the value of a subscribed data.

Message example:

```
<status>
      <data id="data1">42</data>
</status>
```

unsuscribe (→ NSR): To cancel a subscription.

Message example:

```
<unsubscribe>
```

```
      <data id="data1"/>
</unsubscribe>
```

publish (→ NSR): Inform the NSR of the value of a data. Any module that has subscribed to this data will receive a *push* message.

Message example:

```
<publish>
      <data id="data1">67</data>
      <data id="data2">69</data>
</publish>
```

push (NSR →): value of a data that has changed.

Message example:

```
<push>
      <data id="data1">67</data>
</push>
```

query (→ NSR): Any module can consult a given data. It receives a *result* message.

Message example:

```
<query>
      <data id="data1"7>
</query>
```

result: (NSR →): value of a data that has been queried.

Message example:

```
<result>
      <data id="data1">67</data>
</result>
```

The full specification of the NSR can be found here: http://tecfa.unige.ch/~szilas/iris/oparis/nsr_manual.html

3. Installations

3.1 The Director

The Director's distribution contains:

– director.jar: the java program itself

– director.ini: the file spexifying the routing information

– a xslt file, for conversion.

– A 'lib' directory containing two libraries.

To launch the director, the command line is:

```
java -jar director.jar <socket_port> <xslt_FileName>
```

For example:

```
java -jar director.jar 2468 transf1.xslt
```

3.2 The NSR

To launch the NSR, the command line is:

```
java -jar sse.jar <socket_port> <print_output>
```

For example:

```
java -jar sse.jar 56789 true
```