Design Document – Evaluation Component

TUGraz – T8.2A – Apache 2.0 and proprietary – Client- and server-side

This is a draft and needs discussion with the RAGE partners!

Evaluation Component

This document describes an approach to collect uniform log data from the games for evaluation purposes. In the first version the game directly sends information on relevant events and activities to the evaluation component and the evaluation component forwards this information to a service on the back-end. Later the information is probably sent to the Interaction Tracker where the evaluation component collects this information.

The evaluation component is a simple client-side application that collects the interaction data of the play, but does not analyse the data itself. However, it is able of sending the data to a back-end software, where the data can be analysed. In our example, Equalia is used as back-end software.

1) Which data should be collected by/from the game?

The following data should be sent to the Evaluation Component by the game (to be discussed with each game developer as some of the information depends on the individual game):

- game usage and general data
  - it should be tracked when the game is started, stopped (exited), paused, or resumed
- user profile
  - it should be tracked when the player makes changes on the user profile settings
  - for example the change of difficulty level or the change of appearance of the player character
- game activity
  - every time game-specific activity is performed by the player, this activity should be recorded. In addition, if this activity is performed with an in-game tool, then also the used tool should be recorded. Furthermore, it should be recorded, if this activity was useful for reaching the goal of the game (if this information is available)
  - for example, sending a message with the built-in chat tool would lead to the log data "message – chat tool".
- gamification activity
  - activities that are related to the meta level of the game (dealing with the scores or goal of the game)
  - for example, viewing the ranking/leaderboard, high scores, or group comparisons, overview of game goals or modules
- game state
  - each time when a change in the game state is done by the game (game situation, difficulty level, task completion)
  - for example, the player reaches the next difficulty level or the player completes a task
- support

    - each time the player requests support and explanation
    - for example, accessing the built-in help functionality or explanation of the game goals
- component activity
    - each time the result of an component is used (or not used) by the game
    - for example, assessment, intervention, recommendation

In detail, for each of these data types, the following parameters shall be tracked.

Game Event	Parameter	Remarks
gameusage	event= • gamestarted • gamestopped • gameresumed • gamepaused	The same for each game
userprofile	event= • e.g. difficultylevelchanged • e.g. appearancechanged • …	Depending on the game and the possibilities to change user profile settings, respective events should be sent, if a user profile setting is changed. This should be elaborated together, TUGraz and game developer
gameactivity	event= • e. g. messagetoplayer • e. g. messagetonpc • … goalorientation= • progression • regression • neutral • notavailable tool= • e.g. chattool • …	Depends on the game, which game activities can be sent; If an in-game tool is used for this activity (e.g. chat tool, then this information should also be logged) goalorientation means, if a the activity contributes to achieving the game gaol; This should be elaborated together, TUGraz and game developer
gamification	event= • e.g. personalscrore • e.g. groupscore • …	Depends on the game, which gamification elements are available
gameflow	type= • task • level • gamesituation id= • [task-id|level|…] completed= • success • failure	Depends on the game which possibilities to change the game flow it contains
support	event= • e.g. help • e.g. gamegoal • …	Depends on the game which type of support it provides. The examples indicate the help functionality or the explanation of the game goal was requested and viewed by the player
assetactivity	asset=[assetID]	Depends on the game which components are

	done=yes|no	included. This should be tracked if the component delivered a result or other information if that was used by the game

2) Sending the data from the game to the Evaluation Component

The game sends the data to the Evaluation Component by calling a method with two strings as argument.

The following information should be included in a log data record:

GAMEEVENT – the event type (see above)
PARAMETER – additional information depending on the game event (see above)

API of the Evaluation Component:

void sensorData (String GAMEEVENT, String PARAMETER)

Paramter are encoded to a single string (has to be split up for the XML format below):
key1=value1&key2=parameter2&…

Example:
sensorData ("gameactivity", "event=messagetoplayer&tool=chat&goalorientation=neutral");

At the beginning the component needs to get the following information (e.g. from the game settings):

GAMEID – a (not too long) string that specifies the game, e.g. "watercooler"
GAMEVERSION – the version of the game, e.g. 3.1
PLAYERID - an ID that specifies the user (unique within the game), e.g. "player123"

3) Data to be sent to Equalia

(not relevant for the game developer)

The client-side component collects the game data from the game and delivers it to the server-side evaluation component (Equalia) via a REST API in XML format. The uppercase words must be substituted by the evaluation component.

```
<sensordata>
	<context project="rage" application="GAMEID" version="GAMEVERSION" date="[ISO format]"/>
	<user id="PLAYERID" group="" ref=""/>
	<predicate tag="GAMEEVENT "/>
	<parameter KEY1="VALUE1" KEY2="VALUE2" … />
</sensordata>
```

Example (see above)

```
<sensordata>
```

```
        <context project="rage" application="watercooler" version="2.1" date="2016-06-19
13:32:33"/>
        <user id="player123" group="" ref=""/>
        <predicate tag="gameactivity"/>
        <parameter event="messagetoplayer" tool="chat"/>
</sensordata>
```

The XML string should be sent to Equalia via the REST call with POST method:
http://css-kti.tugraz.at:8080/equalia/rest/submitraswsensordata
(will be changed in the near future!!!)

Set up the component

For the Evaluation Component, there is one thing to do (additionally to creating the component) when setting it up:

- Define the component settings, i.e. define the evaluation server (PostUrl), the game identification (GameId), the game version (GameVersion) and the player (PlayerId). This can be done with the following code snipped in C#:

```
EvaluationAsset ea = EvaluationAsset.Insatnce;
EvaluationAssetSettings eas = new EvaluationAssetSettings();
eas.PostUrl = "http://css-kti.tugraz.at:8080/equalia/rest/submitsensorrawdata";
eas.GameId = "watercooler";
eas.GameVersion = "2a";
eas.PlayerId = "sven";
ea.Settings = eas;
```

Use the Component

For using this component tracks need to be submitted to the component, which will forward the to the evaluation service. At the moment this is possible by sending the traces via the component interface, using for example the following C# code:

```
ea.sensorData("gameactivity", "event=messagetoplayer&tool=chat)");
```

Deployment

For the source code the following GitHub-link can be used https://github.com/RAGE-TUGraz/EvaluationAsset - it contains the Visual Studio solution of the component's C# implementation. Furthermore, the broken links to external component DLLs need to be fixed for each project and the Bridge code need to be adopted to the new environment, e.g. changing the IDataStorage path.

For integration into Unity, the resulting DLLs need to put into a folder in the Unity working-directory.

Unit test

For executing unit tests, the source code need to be open in visual studio and all links need to be fixed. In the test-explorer all tests can be executed.