HAT Asset Design Document

Document version: 1.1
Document date: 2016.03.17
Document author: Enkhbold Nyamsuren (Enkhbold.Nyamsuren@ou.nl)

Asset author information
Name: Enkhbold Nyamsuren
E-mail: Enkhbold.Nyamsuren@ou.nl
Organization: OUNL

Accompanying documents:

Document	Description
"HATUsabilityDocument.docx"	Provides a layman's description of mechanics underlying HAT ratings. Provides descriptions of use cases showing example applications of the HAT asset.
"HATAccompToolsManual.docx"	A manual providing a step-by-step guidelines of how accompanying tools can be used for data management, analysis and visualization.
"adaptations.xsd"	XML schema describing the structure of the XML data file for storing adaptation parameters.
"gameplaylogs.xsd"	XML schema describing the structure of the XML data file for logging gameplay data.

Delivery deadlines:

Deadline	Functionality IDs	Note
March 1st, 2016	F1, F2, F3, F4	see Table 1
May 1st, 2016	F5, F7	see Table 2
September 1st, 2016	F6, F9	see Table 2
December 1st, 2016	F8	see Table 2

NOTE: Except Table 2, the rest of the document describes design solutions for the deliverables for March 1st, 2016. This content is likely to be a subject of change in consecutive deliverables.

Asset information:
Current version: 1.0
Date: 2016.03.17
Deployment side: client-side
Currently supported programming language: C#
Required libraries: Microsoft .NET 3.5 Framework or higher
Recommended platform: Windows OS

Overview of the HAT deliverables:
Figure 1 provides an overview of the software components and accompanying tools that will be delivered by March 1st, 2016. HAT asset package is a library written in C# that provides asset's core functionality. It can be integrated at the client-side. The core functionality includes game difficulty adaptation to the player skill (F1 in Table 1). This adaptation functionality relies on a stealth assessment of players (F2 in Table 1). Therefore, the asset can be used for assessment only without using its

adaptation functionality. The asset manages player and game data in locally stored XML files. Please, refer to the "*Local data storage*" section for more details. Two accompanying tools are designed to work with these files (F3 and F4 in Table 1). The data management tool provides a convenient UI for managing (adding, removing and editing) player and game scenario data. The data visualization tool provides UI for doing predefined analysis of player's performance and a learning rate. Analysis results are visualized as graphs. Accompanying tools are designed to work as standalone applications without need of integration with game engines or the HAT asset package. Please, refer to the separate manual ("HATAccompToolsManual.docx") and HAT use case descriptions ("HATUsabilityDocument.docx") for more information on accompanying tools.

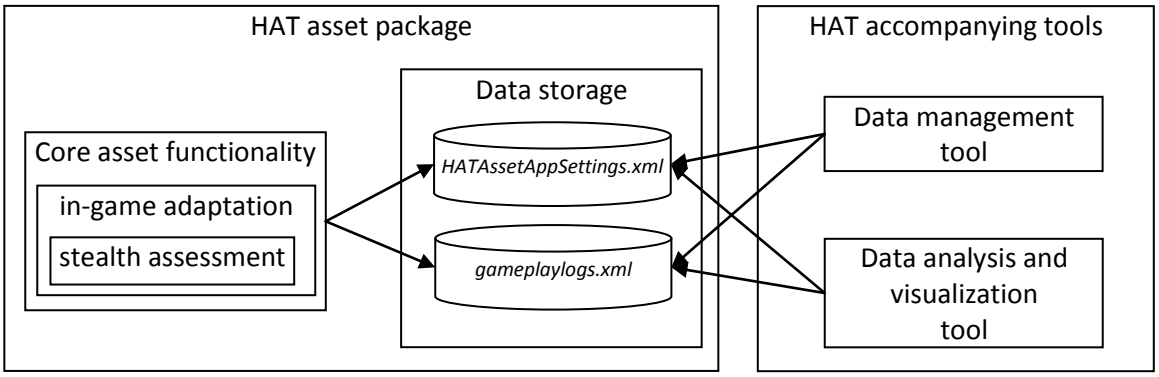

Figure 1: Software components and tools within asset deliverables.

Table 1: Functionalities currently supported by the asset and accompanying tools

Func. ID	Functionality description	Provided by
F1	Player skill to game difficulty adaptation.	Asset package
F2	Learning/performance tracking and assessment	Asset package
F3	Client-side XML data management.	Accompanying tools
F4	Visualization and analysis to aid assessment by teachers and students.	Accompanying tools

Table 2: Future functionalities to be added to the asset and accompanying tools

Func. ID	Functionality description	Provided by
F5	Having an option to update only player ratings.	Asset package
F6	Accepting accuracy values between 0 and 1.	Asset package
F7	Having options for calculating rating based on (1) both time duration and accuracy or (2) accuracy only.	Asset package
F8	Preliminary design and prototype of the server-side version	Asset package
F9	Calculating ratings on the player group level	Asset package or Accompanying tool

Setting up and using the HAT asset for integration with your game:

1. Integrate the HAT asset package with your game.
2. Make sure that information about games, game scenarios and players is added to the XML files (refer to the "*Local data storage*" section and "HATAccompToolsManual.docx" document for how-to).
3. Implement *IBridge* and *IDataStorage* interfaces from the RAGE architecture.

4. Instantiate HATAsset class (refer to the "UML Class diagrams of HAT asset" section).
5. Call relevant methods from your game when necessary (refer to "Using HAT asset for both adaptation and assessment" and "Using HAT asset for assessment only" sections).
For an example, you can refer to the TestApp project inside the HATAsset solution.

UML Class diagrams of HAT asset:
The figure below shows a UML class diagram for the HAT asset. The main class is the HATAsset class. It should instantiated by the game. The HATAsset class also provides two methods that can be called by the game:

- TargetScenarioID - This method should be called if the game needs a recommendation from the HAT asset regarding a game scenario that should be played by a player. The method returns a unique identifier for a game scenario with difficulty level that matches the player's skill level. It returns NULL value if a recommendation was not produced for some reason.
- UpdateRatings - This method should be called after the player finishes playing a game scenario. The method updates ratings of both the scenario and the player based on the player's performance.

Details about above method can be found in Table 3. These two methods are used for adapting game difficulty and for assessment (F1 and F2 in Table 1). As was mentioned earlier, adaptation implicitly includes assessment of both the game scenario and the player. However, it is also possible to use HAT's assessment functionality without adaptation functionality. These two use cases are shown in sequence diagrams in Figure 3 and Figure 4.

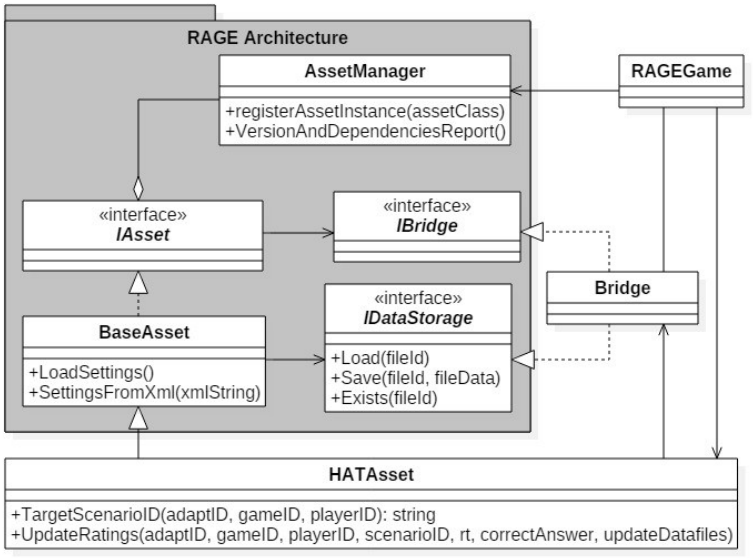

Figure 2: UML Class diagram for HAT asset architecture.

Table 3: Descriptions of methods that can be called by a game engine.

Parameter name	Type	Description
HATAsset.TargetScenarioID		
adaptID	string	A unique identifier for an adaptation. Currently, the value of this parameter is fixed to "Game difficulty - Player skill". Pass this value when

		calling this method.
gameID	string	A unique identifier for a game. The value should be a valid string without any special characters.
playerID	string	A unique identifier for a player. The value should be a valid string without any special characters.
return	string	Returns a unique identifier for a recommended scenario.
HATAsset.UpdateRatings		
adaptID	string	A unique identifier for an adaptation. Currently, the value of this parameter is fixed to "Game difficulty - Player skill". Pass this value when calling this method.
gameID	string	A unique identifier for a game. The value should be a valid string without any special characters.
playerID	string	A unique identifier for a player. The value should be a valid string without any special characters.
scenarioID	string	A unique identifier for a game scenario. The value should be a valid string without any special characters.
rt	double	The duration of time the player spent playing the scenario. The duration is measured in milliseconds and should be any positive non-0 number.
correctAnswer	double	The value should be either 1 or 0. The value is 1 if a player finished the scenario successfully and 0 otherwise.
updateDatafiles	bool	Set to true if XML files should be updated at the end of the method call.
return	*void*	

Using HAT asset for both adaptation and assessment (F1 and F2):
Figure 3 shows a sequence diagram depicting how the HAT asset can be used for adaptation. When a player starts a new game (step 1), the game requests the HAT asset to recommend a scenario with a difficulty level that matches player's skill level (step 2). The HAT asset returns a unique identifier of the scenario that matches to player's skill level (step 3). The game invokes the scenario identified by the HAT asset and starts a gameplay session (step 4). After the player finishes the scenario (step 5), the game request the HAT asset to update player and scenario ratings (step 6). For this step, the game needs to provide player's performance metrics (see Table 3). If the player chooses to play another scenario, then the steps 2-7 are repeated.

Figure 3: UML Sequence diagram showing steps involved in adapting the game difficulty to the player's skill.

Using HAT asset for assessment only (F2):
Figure 4 shows a sequence diagram depicting how the HAT asset can be used for assessment only. When a player starts a new game (step 1), the game itself decided which scenario should be given to the player (step 2). After the player finishes the scenario (step 5), the game request the HAT asset to update player and scenario ratings (step 4). For this step, the game needs to provide player's performance metrics (see Table 3). If the player chooses to play another scenario, then the steps 2-5 are repeated. The ratings can be later used for self-assessment by players or for assessment by teachers.

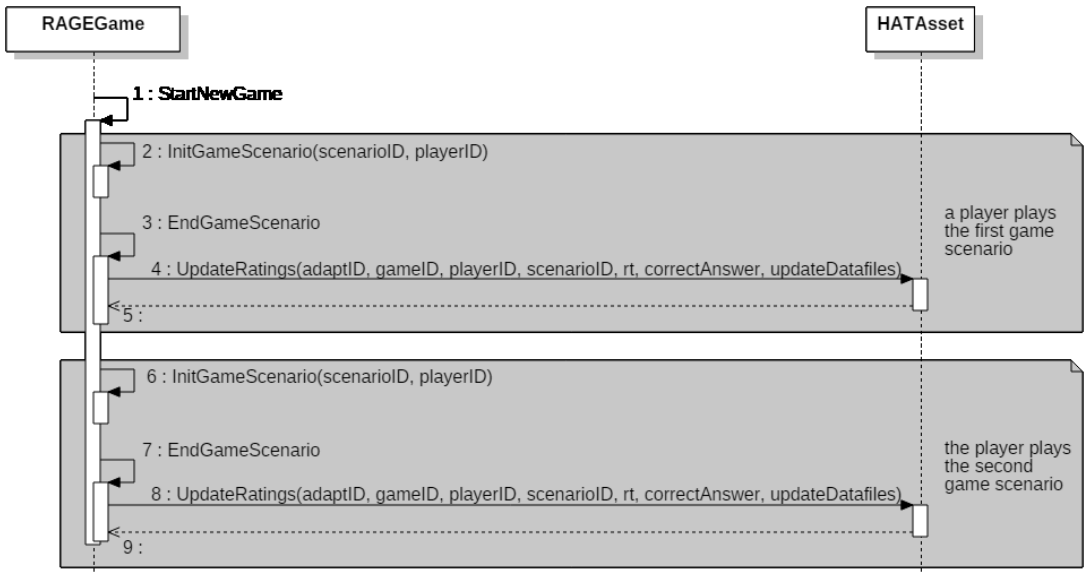

Figure 4: UML Sequence diagram showing steps involved in using the HAT asset as an assessment tool only.

Local data storage:
The client-side version of the asset uses XML-based files for local data storage. There are two xml files: "*HATAssetAppSettings.xml*" and "*gameplaylogs.xml*". This section describes structures of these XML files. However, an accompanying tool provides UI for editing XML files. Refer to "HATAccompToolsManual.docx" document for instructions on how to use the accompanying tool.

NOTE: The asset relies on abstract interface "IDataStorage" for loading local files. The game developer will need to implement this interface for the asset to be able to load xml files.

Within HATAsset solution, both files are located at "HatAsset\Resources\". Table 4 shows what kind of operations are performed by "TargetScenarioID" and "UpdateRatings" on these XML files. The "*HATAssetAppSettings.xml*" file contains the most recent updated ratings (and relevant parameters) of all players and all game scenarios. These ratings are used by the "TargetScenarioID" method to match a scenario to a player. These ratings are also updated by the "UpdateRatings" method. The "*gameplaylogs.xml*" contains a history of all ratings that were calculated after each scenario played by a player. This history is not used for providing the adaptation functionality, but is required for the assessment functionality. This file is updated by the "UpdateRatings" method.

Table 4: Operations performed by two methods on the xml files.

	TargetScenarioID	UpdateRatings
HATAssetAppSettings.xml	read only	read/write
gameplaylogs.xml	-	read/write

The structure of "*HATAssetAppSettings.xml*":

Please, refer to the accompanied "*adaptations.xsd*" for an XML schema that provides a full and detailed description of the structure of the "*HATAssetAppSettings.xml*"**.** This section provides a succinct description of the XML file.
Code Snippet 1 provides a compressed view of the overall structure of the "*HATAssetAppSettings.xml*". Since the HAT asset can be used with one or more games, information about each game is encapsulated by a "*Game*" element. The element has "*GameID*" attribute that stores a unique identifier for the game. Each "*Game*" element must have two child elements: "*ScenarioData*" and "*PlayerData*". "*ScenarioData*" element stores a list of available scenarios in the game. Information about each scenario is encapsulated inside a "*Scenario*" element. It has "*ScenarioID*" attribute that stores a scenario identifier that is unique within the scope of the parent "*Game*" element. "*PlayerData*" element stores a list of available players in the game. Information about each player is encapsulated inside a "*Player*" element. This element has a "PlayerID" attribute that refers to a player identifier that is unique within the score of the parent "*Game*" element.

Code Snippet 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<AdaptationData>
    <Adaptation AdaptationID="Game difficulty - Player skill">
        <Game GameID="ExampleGameID">
            <ScenarioData>
                <Scenario ScenarioID="ExampleScenarioID">
                    <!-- contains scenario's rating and other parameters -->
                </Scenario>
                <!-- add more scenario elements here -->
            </ScenarioData>
            <PlayerData>
                <Player PlayerID="ExamplePlayerID">
                    <!-- contains player's rating and other parameters -->
                </Player>
                <!-- add more player elements here -->
            </PlayerData>
        </Game>
        <!-- add more game elements here -->
    </Adaptation>
</AdaptationData>
```

Code Snippet 2 shows an example of the "*Scenario*" element with its child elements. The game developer should be aware of following elements. The element "*Rating*" stores the difficulty rating of the scenario. "*PlayCount*" element refers to the number (integer) of times the scenario was played (by any player) during its lifetime. "*LastPlayed*" refers to the date and time when the scenario was last played. It should have a format "yyyy-MM-ddThh-mm-ss". The "*TimeLimit*" element refers to the maximum amount of time within which the player should finish the scenario. It is measured in milliseconds. Except for the "*TimeLimit*", the values shown in the code below are default values that are recommended to be used in any game.

Code Snippet 2:

```
<Scenario ScenarioID="ExampleScenarioID">
    <Rating>0.01</Rating>
    <PlayCount>0</PlayCount>
    <KFactor>0.0075</KFactor>
    <Uncertainty>1.0</Uncertainty>
    <LastPlayed>2015-07-22T11:56:17</LastPlayed>
    <TimeLimit>900000</TimeLimit>
</Scenario>
```

Code Snippet 3 shows an example of the "*Player*" element with its child elements. The game developer should be aware of following elements. The element "Rating" stores the skill rating of the player. "*PlayCount*" refers to the total number of scenarios played by the player (repeatedly played scenarios are included in the counting). "*LastPlayed*" refers to the date and time when the player finished the most recent scenario. It should have a format "yyyy-MM-ddThh-mm-ss". The values shown in the code below are default values that are recommended to be used in any game.

Code Snippet 3:

```
<Player PlayerID="ExamplePlayerID">
    <Rating>0.01</Rating>
    <PlayCount>0</PlayCount>
    <KFactor>0.0075</KFactor>
    <Uncertainty>1.0</Uncertainty>
    <LastPlayed>2015-07-22T11:56:17</LastPlayed>
</Player>
```

The structure of "*gameplaylogs.xml*":

Please, refer to the accompanied "*gameplaylogs.xsd*" for an XML schema that provides a full and detailed description of the structure of the "*gameplaylogs.xml*"**.** This section provides a succinct description of the XML file.

Code Snippet 4 provides a compressed view of the overall structure of the "*gameplaylogs.xml*". "*Adaptation*" and "*Game*" elements play the same roles as in the "*HATAssetAppSettings.xml*" document. "*Game*" element contains a list of "*Gameplay*" elements. A "*Gameplay*" element encapsulates information about a single gameplay. In this context, the term gameplay refers to any scenario played by any player. The element has three attributes. The "*ScenarioID*" attribute refers to the ID of the scenario that was played. The "*PlayerID*" attribute refers to an ID of the player who played the scenario. The "Timestamp" attribute refers to the date and time at the end of the gameplay. The "*Gameplay*" element has four child elements. The "*RT*" element refers to the duration of the gameplay in milliseconds (the duration of time the player spent playing the scenario). The "*Accuracy*" element has a value of 1 if the player successfully finished the scenario, and 0 otherwise. "*PlayerRating*" and "*ScenarioRating*" refer to player and scenario ratings that have been updated based on values of "*RT*" and "*Accuracy*" elements.

Code Snippet 4:

```
<?xml version="1.0" encoding="UTF-8"?>
<GameplaysData>
    <Adaptation AdaptationID="Game difficulty - Player skill">
        <Game GameID="GameID">
            <Gameplay Timestamp="2015-07-29T09:01:20"
                      ScenarioID="ScenarioID1" PlayerID="PlayerID1">
                <RT>115657</RT>
                <Accuracy>1</Accuracy>
                <PlayerRating>0.0774275394923526</PlayerRating>
                <ScenarioRating>0.960022196614398</ScenarioRating>
            </Gameplay>
            <Gameplay Timestamp="2015-07-29T09:03:46"
                    ScenarioID="ScenarioID1" PlayerID="PlayerID2">
                <RT>143132</RT>
                <Accuracy>1</Accuracy>
                <PlayerRating>0.0399778033856024</PlayerRating>
                <ScenarioRating>0.922572460507648</ScenarioRating>
            </Gameplay>
            <Gameplay Timestamp="2015-07-29T09:18:13"
                    ScenarioID="ScenarioID2" PlayerID="PlayerID1">
```

```
                <RT>226569</RT>
                <Accuracy>0</Accuracy>
                <PlayerRating>0.0694275394923526</PlayerRating>
                <ScenarioRating>0.992572460507648</ScenarioRating>
            </Gameplay>
            <!-- add more gameplay elements here -->
        </Game>
        <!-- add more game elements here -->
    </Adaptation>
</GameplaysData>
```

Managing XML files:

Data management tool is provided as a separate application in the RAGE repository. With this tool, a developer or instructor can use UI to easily manage data inside XML files. Please, refer to the accompanying manual on the management/visualization tool. However, if one chooses to manually edit XML files then following operations need to be performed.

If a new game is to be registered with the HAT asset then respective game elements should be created in both "*HATAssetAppSettings.xml*" and "*gameplaylogs.xml*" files as show in Code Snippet 5.

Code Snippet 5:

```
    <!-- adding new Game element to the HATAssetAppSettings.xml -->
    <Adaptation AdaptationID="Game difficulty - Player skill">
        <Game GameID="NewGameID">
            <ScenarioData />
            <PlayerData />
        </Game>
    </Adaptation>


    <!-- adding new Game element to the gameplaylogs.xml -->
    <Adaptation AdaptationID="Game difficulty - Player skill">
        <Game GameID="NewGameID" />
    </Adaptation>
```

To register a new game scenario with the HAT asset, a new "*Scenario*" element should be created in "*HATAssetAppSettings.xml*" file. An example of the new scenario element is shown in Code Snippet 2. To register a new player with the HAT asset, "*Player*" element should be created in "*HATAssetAppSettings.xml*" file. An example of the new "*Player*" element is shown in Code Snippet 3.