Design Document – Competence Assessment Component

TUGraz – T2.2C – Apache 2.0 – Client side

About this component

This component will be implemented as a client-side component. Based on a Domain Model and information from the game it assesses the competence state (= set of possessed competences) of a player. The component architecture will prevent multiple component creation; only one component per game is needed.

Component mechanics

Evidences (events or tasks in a game) speaking either for or against possession of a competence are used to update the probability of competence possession. For this purpose a competence structure represented in the domain model is used. One can work directly with evidences by sending them to the component or send actions, expressed as identification strings, performed in the game and use a mapping between these actions and evidences. For the update procedure we allow multiple significances, meaning that the evidence of possessing/lacking a competence may be stronger or weaker for some actions. All competences with a probability over a certain threshold are assumed to be available for a player.

Component interfaces

- The component will be able to return the set of possessed competences linked to a player . A C# representation in the component interface could be:

  List<string > getCompetenceState()

- The component will be able to update the player's current competence state by receiving evidence-tuples consisting of a competence identification, significance and an indicator if the player showed possession of the corresponding competence or not. A C# representation in the component interface could be:

  void updateCompetenceState(List<string> evidences, List<string> significance, List<Boolean> type)

- Another way of updating the competence state is to use actions performed by the player for this purpose. Therefore, a mapping located in the domain model is used to map the actions/gamesituations represented by identification strings to competences and evidences for or against possession of them. A C# representation in the component interface could be:

  void `updateCompetenceStateAccordingToActivity` (String action)

  void `updateCompetenceStateAccordingToGamesituation`(String gs, Boolean success)

- In order to repeat a game the following interface resets the competence state:

  void resetCompetenceState()

Component dependencies/requirements

- The component depends upon the Domain Model Component for receiving a domain model.
- It also depends on the client-side tracker for sending the current competence state to the server.
- The storage component is used to store the competence state locally.

Milestones

Milestone 1

- t1.1: Creating the first version of the design document, defining the API and creating a dummy component with the API implemented

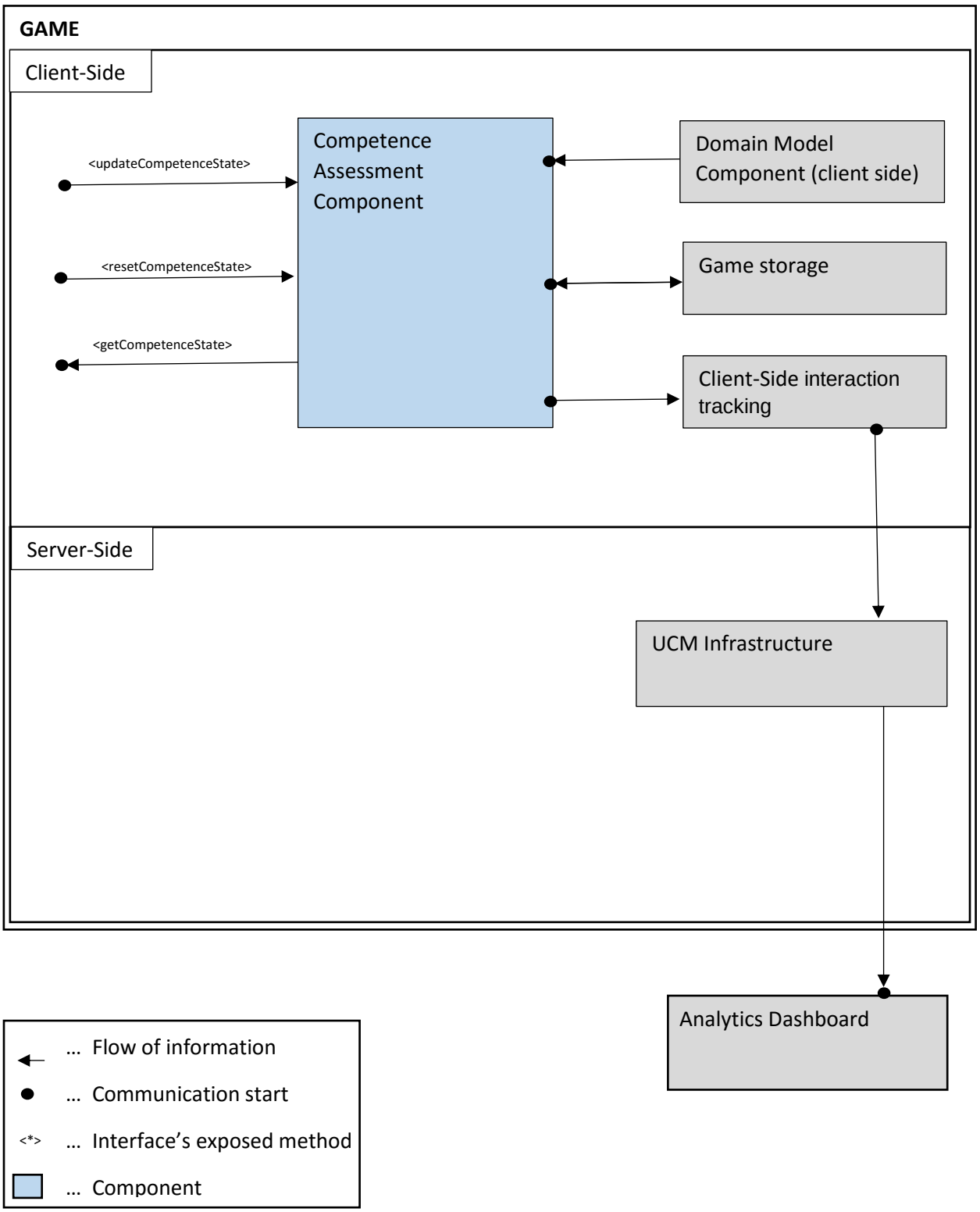
Milestone 2

- t2.1: Create Software component in line with Component -Manager infrastructure.
- t2.2: Elaborate Settings-structure within the Component -Manager infrastructure.
- t2.3: Integrate the tracker.
- t2.4: Include different update strengths.
- t2.5: tUpdate Competences via activities.
- t2.6: Integrate Game Storage functionality into the component-functionality.

Milestone 3

- t3.1: testing the component with a game
- t3.2: instructions and scripts for building and deploying

Graphical representation



Set up the Component

For the Competence Assessment Component, there are two thing to do (additionally to creating the component) when setting it up:

- The Domain Model Component needs to be in place, supplying a domain model adopted to the desired update method.

Use the Component

There are three different ways of using this component when it comes to input new information about the competence state and one for accessing the current competence state.

- Updating competences directly. Therefore a list of competences to update, a list of information of which quality the update is (true means increase, false means decrease) and a list of information how strong the update should influence the possession probability. For one competence, this can be done like shown in the following code snipped:

```
CompetenceAssessmentAsset caa = CompetenceAssessmentAsset.Instance;
List<String> compList = new List<string>();
List<Boolean> evidenceList = new List<Boolean>();
List<EvidencePower> evidencePowers = new List<EvidencePower>();
compList.Add("C1");
evidenceList.Add(true);
evidencePowers.Add(EvidencePower.Medium);
caa.updateCompetenceState(compList, evidenceList, evidencePowers);
```

- Updating competences based on activities. Therefore only the activity needs to be supplied, all other information needs to be persistent in the loaded domain model.

```
CompetenceAssessmentAsset caa = CompetenceAssessmentAsset.Instance;
caa.updateCompetenceStateAccordingToActivity("doThingA");
```

- Updating competences according to a game situation. Therefore the game situation id needs to be supplied and the information if the game situation was mastered successfully (true) or not (false), all other information needs to be persistent in the loaded domain model.

```
CompetenceAssessmentAsset caa = CompetenceAssessmentAsset.Instance;
caa.updateCompetenceStateAccordingToGamesituation("gamesituation1",true);
```

- For accessing the competence state the following method is used. The result is interpreted as the pairs of competence name and probability of possession for the player.

```
Dictionary<string, double> cs = caa.getCompetenceState();
```

Deployment

For the source code the following GitHub-link can be used https://github.com/RAGE-TUGraz/CompetenceBasedAssets - it contains the Visual Studio solution of the competence based component. 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.