Software design document

This document provides an overview of the features that will be implemented in **2.2 Step-based assessment asset.** This asset consists of a parser and a reasoner.

Description

The following figure schematically describes an implementation framework for scenario based games. It shows the relation between the two assets from Utrecht University that will be delivered to RAGE, 3.3 Communication Scenario Editor & 2.2 Step-based assessment. It additionally shows how game developers may use the assets.

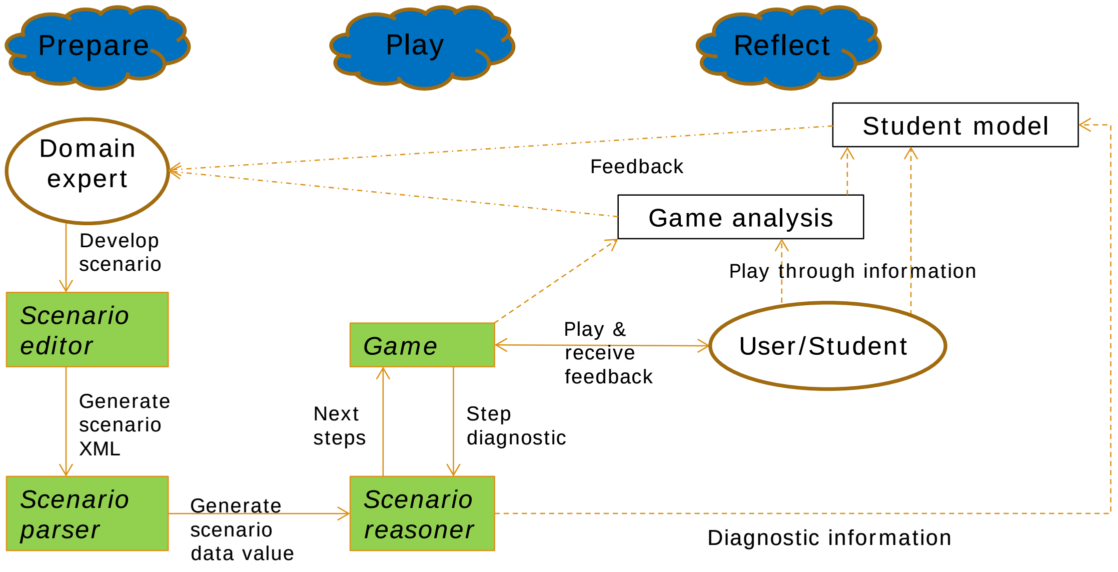

Figure 1: An implementation architecture of scenario-based simulations

A communication-skills teacher develops a scenario in the scenario editor as a graph of steps along with the respective scores and feedback per step. The editor generates valid scenarios in xml format.
We have designed the editor and the assessment assets to be loosely coupled using a REST architecture. The editor produces an xml that follows the schema: https://github.com/UUDSL/scenario/blob/v4.0.0/scenarioLanguage.xsd
The reasoner asset processes any valid scenario produced according to this schema, using any valid configuration (single VC /multiple VC's; configurable scores, parameters, properties etc).

How to use the Step-based assessment asset

A scenario that follows the schema can be parsed by the method ParseScenario, which is for a one-time conversion from XML (output from the editor) to a binary representation. This representation need not change unless the XML changes (i.e. there is a changed scenario/dialogue). This method returns the ID of the binary that are needed for the other API calls.

The scenario reasoned is a web-service that offers its services through JSON-RPC. A game interacts at run-time with the scenario reasoner (with the ID of the specific parsed scenario).

The reasoner has the following methods:

- ScenarioInfo: returns info about the scenario like metadata, definitions and properties of the scenario and the characters.
- Examples: returns an initial state that contains the initial values for the parameters that persist throughout the playthrough like the initial emotion of a character.
- AllFirsts: takes a state from the list of states returned by this method or the initial state. The list of states returned contains a state per next step (a node in the editor). A step has a type (player, computer or situation) and can be handled accordingly. A step also contains the statement text and the property values and prospected parameter values after that step has been done. If the returned list is empty the dialogue scenario has ended.

The JSON input and output of the methods listed above are detailed in the following:
https://github.com/UURAGE/ScenarioReasoner/tree/master/doc/schemas

The two assets 3.3 Communication Scenario Editor & 2.2 Step-based assessment are typically used together in a game. It is of course possible to use the XML output from the editor; however in that case the game-developer needs to develop their own “parser” and “reasoner” modules.

Technical and quality aspects
The asset is implemented in Haskell on top of the Ideas framework from Utrecht University. The Ideas framework has been tested extensively both functionally and non-functionally and used in diverse domain reasoners (DME secondary math education, Math-Bridge, MathDox, Logic tool, Ask-Elle, tutor for Haskell) in addition to the step-based assessment asset.

- Repository location: (https://github.com/UURAGE/ScenarioReasoner), including:
    - Running and building instructions.
    - Deploy instructions.
    - Documentation.
  - Load and reliability tests using locust.io (https://github.com/UURAGE/ScenarioReasoner/blob/master/doc/Analysis%20reliability%20and%20load%20test.xlsx)
- Technical Report Ideas (UU-CS-2014-005) including test report available on request.