# IMPRESS: Narrative Integration through FAtiMA toolkit

José Bernardo Rocha, Rui Prada
INESC-ID Lisbon, Portugal

May 28th, 2020

## 1 Introduction

Storytelling has great value for learning scenarios. It improves the learning experience, as it creates meaningful contexts for the actions and gives flavour to the experience. It supports the flow of progression and improves the (emotional) engagement of users. Storytelling can be enhanced with Artificial Intelligence (AI) to increase the interactivity and adapt to users' actions. AI can also improve the believability of the characters by using models of emotion and generate behaviour that are coherent with the situation users face.

Our approach to deliver the Storytelling experience in IMPRESS is by adding characters, to the games, that engage users through dialogue and emotional display. The characters will pay attention to the events in the game and choose speech acts they believe are appropriate to the users' situation. These will follow the structure of the tasks that users are performing to keep the experience coherent.

We make use of the FAtiMA toolkit[1] to execute the approach. We developed a REST API for the FAtiMA toolkit to facilitate the integration. The toolkit drives the behaviour of AI characters and includes tools to edit dialogue options and behaviour rules. The game is responsible for presenting the character to the user (see Figure 1).

In this document we propose a narrative model that maps a common game structure to narrative opportunities. The narrative will be delivered by a Narrator Character powered by the FAtiMA toolkit. We describe how to use FAtiMA

[1] https://fatima-toolkit.eu/

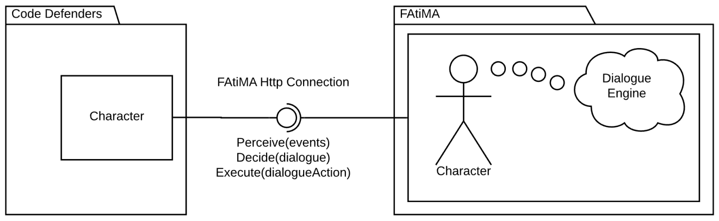


Figure 1: FAtiMA web integration.

Toolkit and its dialogue system in order to create the character and add it to the game.

## 2 Narrative Model

We propose a Hierarchical Narrative Model that uses a common game structure to identify opportunities to introduce narrative elements to a game. We follow a game structure that can be found in many games, particularly in old platform games. We defined the model to include the most complex structures, but it can even be applied to games that don't follow the structure strictly, as we can use only the parts of the hierarchical model that are relevant the concrete game we are addressing. For example, some of the levels of the hierarchy may be ignored if they do not make sense in the case of the game.

### 2.1 Hierarchical Narrative

Games are typically organized in hierarchical content (e.g. worlds, levels, sections of levels, etc.). We can have different levels of narrative based on that hierarchy. The the overall progress of a player in the game will be mapped in the hierarchical structure. Hence, the player's success and failure can be seen both at a macro and micro level. Table 1 shows an overview of the narrative model proposed.

A common way to track a player's progress in games is by having different game playthroughs, we will call this the Game level of the hierarchy. At this level you track a player's overall progress throughout the game, and have events that relate to that progress (starting, continuing, winning and losing a game).

An often-used structure in games is having different discrete levels where players are challenged and through which players must progress in order to advance in the game. Quite often these levels are grouped together in worlds, typically by themes. As such, in order to make it easier to integrate narrative to a game, we propose the following corresponding hierarchical levels in our Narrative Model: World and Level. At a the World level you track a player's progress throughout a world (grouping of levels), and we narrate events that relate to that progress (starting, continuing and completing a world). Whereas at the Level part of the hierarchy we narrate events that relate to the progress of a player on that level (starting, continuing, winning or losing a level). Worlds can be seen as volumes in the narrative and the levels can be seen as chapters. But, this depends on the granularity of the levels. A world can also be seen as a chapter and each level represent a scene (or section) in the chapter.

At the core of the gameplay experience we have the actions that players attempt in order to play the game, these tend to be different according to the genre of game, and across specific games. Nevertheless, these actions occur within a level, but in order to make the model more understandable, and its ability to be generalized, we define ans explicit layer for the actions, the Action level in the narrative model. At this level, we narrate events that relate to the actions that a players take in order to overcome gameplay challenges (attempting, failing, or succeeding in an action).

Table 1: Hierarchical Narrative Model

| **Narative Level** | **Types of Events** | **Examples** |
|---|---|---|
| **Game** | Events that relate to the overall progress of a Game session | Starting a new Game, Continuing a Game, Winning a game, Losing a Game |
| **World** | Events that relate to the progress of a player in a World (group of levels) | Starting a World for the first time, Continuing a previously started World, Completing a World |
| **Level** | Events that relate to the progress of a player in a Level | Starting a Level for the first time, Continuing a previously started Level, Losing a Level, Winning a Level |
| **Action** | Events that relate to a players Actions (typically within a Level) | Attempting an Action, Fail an Action, Succeed an Action |

The narrative is build around the events that occur in each of the levels. Therefore, characters must be able to identify each event in order to take action. Again, the events that are included in the concrete narrative defined for the game, do not need to include all the events we define in the model, but all should be, at least, considered when defining the narrative structure.

#### 2.1.1 Game Narrative

As stated previously, this level of our hierarchical model deals with events that relate to the progress a player is making in the overall progress of the game. It is common for games to implement and allow different playthroughs at the same time, organized as "games". This level should be used to present the overall Narrative of the game (similarly to how a Television Show can have an overarching plot across seasons and episodes). We then can extract the following relevant Game events that allow us to introduce this narrative:

- **Start Game** - The start of the game, the introductory narrative of the game should be presented in response to this event. We can use it to narrate the introduction of the game, and do some exposition of the important story elements, such as, what is setting and context of the game, what is the player playing as, and what are the objectives of the game.
- **Continue Game** - A continuation of a previous game, the narrative presented in response to this event should be about catching up players to previous events and/or the current (relevant) state of the game. We can use it to add some re-contextualization information for the player, concerning the current status of the game and the player's current progress.
- **Game Win** - The player has finished the game successfully, the narrative presented in response to this event should be the successful end. We can use it to present the happy ending conclusion to the game's narrative.
- **Game Loss** - The player has lost the game (e.g: has no more lives left to continue), the narrative presented in response to be this event should the unsuccessful end. We can use it to present the sad conclusion to our narrative.

### 2.1.2 World Narrative

At this level, we can expose the narrative that ties a world (collection of levels, typically thematically connected) together. Typically, at this level we can present a smaller unit of narrative (similarly to how a Television Show can have a plot and theme across a single season that will be resolved at the end of the season). We can integrate a narrative that ties up the major themes and narrative of a world (collection of levels) by narrating these basic World events:

- **Start World** - When a player first enters a new world, an introductory narrative to the world can be presented in response to this event. This allows us to introduce the main story elements that related to that world and allow us to setup the narrative that will span across the multiple levels that compose it.
- **Continue World** - In games that allow players to have a non-linear progression between worlds, when the player returns to a world we can present a narrative that reminds players of previous events and/or the current (relevant) state that relates to the world the player has returned to. This can be done even if the game is structured such as that the progression between worlds is linear.
- **Complete World** - The player has completed a world, the narrative presented in response to this event should be the successful end to the narrative of the world. Here we present the conclusion to the story elements we first introduced when a player first started this world.

### 2.1.3 Level Narrative

Here we focus instead on narrative that would be adequate to present in a single level of game (similarly to how a Television Show will have a plot for a single episode that will be resolved by the end of it). This Narrative can be introduced by narrating the basic events that relate to the game's levels:

- **Start Level** - When a player begins a level, an introductory narrative to the level can be presented in response to this event. Since this is a game, this narrative can change according to the number of attempts a player has made. We should focus on introducing the relevant elements of narrative related to the level.
- **Continue Level** - When a player returns to a level that they have already started, we can present a narrative that reminds players of previous events and/or the current (relevant) state of the level the player has returned to. When a player returns to a level he has already started, the narrative presented should be a condensed form, just as a quick refresher of the plot points for the player.
- **Win Level** - The player has completed a level, the narrative presented in response to this event should be the successful end to the narrative of the level. We can show the narrative that relates to the positive resolution of the plot points presented at the start of the level.
- **Lose Level** - The player has failed to complete a level, the narrative presented in response to this event should be the unsuccessful end to the narrative of the level. If a player loses a level he can be shown the negative resolution of the plot points presented at the start of the level.

### 2.1.4 Action Narrative

Narrating a player's action allows us to introduce micro narratives that describe what the player is doing and/or attempt doing. We can integrate this type of narrative by narrating a simple set of events that relate to player's actions:

- **Attempt Action** - When a player attempts an action (within a level for example), a narrative to the level can be presented in response to this event. We can consider this to be the build-up phase of a player's action, and have the narrator describe what the player is attempting to do. Since, quite often players will attempt the same action, it is possible to implement this so that the action can be narrated differently according to the number of attempts
- **Fail Action** - When a player fails in executing an action (within a level for example), a narrative to the level can be presented in response to this event. As mentioned previously, due to players attempting an action until they succeed, it is possible to take into account the number of attempts in order to modify the narration provided.

- **Succeed Action -** When a player succeeds in an action (within a level for example), a narrative of the successful execution of the action can be presented in response to this event. Here, we can also use the number of attempts in order to modify the narration that is provided.

It may be important to define more concrete action events that capture the nature of the action of the players are performing. We suggest that an taxonomy for the actions that are relevant for the narrative should be defined. This is highly dependent on the gameplay actions available to the player. A simple approach is to include all action that can be performed by players as events to send to the character, however, it may be the case that only a few of the actions really advance the sate of the game in a meaningful way that should advance the narrative as well.

The narrative model does not specify the number of characters we use to present the story to the players. We can have multiple characters, different at each level, or different in each game world, or we can have a single character that narrates the overall story.

## 2.2 Example of Use of the Narrative Model

We will now approach a practical example of using our Narrative model to implement narrative into a game. We will use the example of the Code Defenders[2] game. This game focuses on a simple metaphor of attackers modifying code that exploiting the faults in testing code so that, despite the code being incorrect, it still passes all tests, while defenders need to write tests that can catch the modified code.

The game's main mode is a multiplayer mode where players have take on the attacker and defender role for each game session, but the game also provides a single player puzzle mode where players solve challenges in groups of levels (a world), where they may alternate roles between challenges.

**Game Narrative** Player progress is saved throughout sessions, so we can apply the Game level of the Narrative Model. The game's main mechanics (attackers vs defenders) suggests that a possible narrative that we can introduce in this game is of a black hat hacker, who is attacking and exploiting flaws in the code (in this case modifying the existing code base so that it seems correct, but isn't) vs a white hat hacker who is attempting to fix the code (in this case by writing better tests that catch the modified code).

The game should send the events: star game, continue game and game win. We are assuming that the world is not necessarily finished in a single gameplay session. Therefore, the continue world event is important to bring back the context of the narrative to the players once they come back.

[2]https://www.code-defenders.org/

**World Narrative** Since we have groups of levels (a world) and they can be organized themeatically, we can write a narrative that encompasses the theme of the world. The game should send the events: star world, end world and continue world. As take the same assumption of a multiple gameplay sessions to finish a world. Therefore, the continue world event is important to recall the context of the narrative.

**Level Narrative** At the beginning and end of each level the character should make a comment. In the beginning it can serve as a introduction of the level problem and at the end it will depend on the players' success. A different message should be sent accordingly (i.e. if the player fails or succeeds). The character can keep track of the number of levels attempts and total failures and take that in consideration as well. The game should send the events: star level, level failed and level succeed. If the level has a score, this can be communicated to the character as well.

**Action Narrative** In each level, players attempt to attack or defend code, which means we can use the Action Level of our Narrative Model. Relevant actions are: sending a mutation, and checking for mutations once new code is added to the test. The relevant results are: a mutation is killed or survived.

## 2.3 Technical Execution

### 2.3.1 Overview

Integrating with FAtiMA can be facilitated by having a specialized component that manages all the FAtiMA REST API requests. We propose implementing a simple component that receives event notifications and redirects them to the appropriate FAtiMA REST API endpoint.

### 2.3.2 Events

There are 8 basic types of events we need to respond to in order to integrate with FAtiMA:

- Start (Game) , Start (World), Start (Level, Number of Attempts)
- Continue (Game), Continue (World)
- Win (Game), Win (Level)
- Loss(Level)
- Complete (World)
- Attempt (Action, Number of Attempts)

- Success (Action, Number of Attempts)
- Fail (Action, Number of Attempts)

We elaborate on these types of events in the next sections.

### 2.3.3 Start Event

In response to the Start event, the following pseudo-algorithm should be followed:

1. Send an HTTP Request to FAtiMA to Create instances of the characters that are required for the narrative (or simply Load the Scenario) if it is the first time starting, or simply load the correct session number.
2. Register the IDs of the characters created
3. Send an HTTP Request to each of the characters through FAtiMA in order to Perceive the Start event
4. Send an HTTP Request to each of the characters through FAtiMA in order to get the Dialog

### 2.3.4 Continue Event

In response to the Continue event, the following pseudo-algorithm should be followed:

1. Fetch the registered IDs of the previous created characters
2. Send an HTTP Request to each of the characters through FAtiMA in order to Perceive the Continue event
3. Send an HTTP Request to each of the characters through FAtiMA in order to get the Dialog

### 2.3.5 Win Event

In response to the Win event, the following pseudo-algorithm should be followed:

1. Fetch the registered IDs of the previous created characters
2. Send an HTTP Request to each of the characters through FAtiMA in order to Perceive the Win event
3. Send an HTTP Request to each of the characters through FAtiMA in order to get the Dialog

### 2.3.6 Loss Event

In response to the Loss event, the following pseudo-algorithm should be followed:

1. Fetch the registered IDs of the previous created characters
2. Send an HTTP Request to each of the characters through FAtiMA in order to Perceive the Loss event
3. Send an HTTP Request to each of the characters through FAtiMA in order to get the Dialog

### 2.3.7 Complete World Event

In response to the Complete World event, the following pseudo-algorithm should be followed:

1. Fetch the registered IDs of the previous created characters
2. Send an HTTP Request to each of the characters through FAtiMA in order to Perceive the Complete World event
3. Send an HTTP Request to each of the characters through FAtiMA in order to get the Dialog

### 2.3.8 Action Atttempt Event

In response to the Action Attempt event, the following pseudo-algorithm should be followed:

1. Fetch the registered IDs of the previous created characters
2. Send an HTTP Request to each of the characters through FAtiMA in order to Perceive the Action Attempt event
3. Send an HTTP Request to each of the characters through FAtiMA in order to get the Dialog

### 2.3.9 Action Success Event

In response to the Action Success event, the following pseudo-algorithm should be followed:

1. Fetch the registered IDs of the previous created characters
2. Send an HTTP Request to each of the characters through FAtiMA in order to Perceive the Action Success event
3. Send an HTTP Request to each of the characters through FAtiMA in order to get the Dialog

### 2.3.10 Action Fail

In response to the Action Fail event, the following pseudo-algorithm should be followed:

1. Fetch the registered IDs of the previous created characters
2. Send an HTTP Request to each of the characters through FAtiMA in order to Perceive the Action Fail event
3. Send an HTTP Request to each of the characters through FAtiMA in order to get the Dialog

## 2.4 Using the FAtiMA Toolkit

The FAtiMA Toolkit[3] is an emotion engine for Artificial Intelligence characters and robots. It has several components for implementing Artificial Characters and an Authoring Tool to facilitate the development, particularly in the domain of Emotion Generation. Despite the relevance of these components for implementing characters with social and emotional intelligence, in our particular use case, the introduction of Narrative by means of a Narrator, we will focus on only one of FAtiMA's components - The Dialogue System. The dialogue system is highly flexible and can easily be used to implement our Narrative Model and this can be achieved my using FAtiMA's Integrated Authoring Tool.

FAtiMA currently allows easy integration with C# applications (since it was developed natively in C#) and particularly with the Unity game Engine. However, since we wish to integrate FAtiMA with browser based games, we modified FAtiMA to have a RESTful API so that it can easily be integrated with any development platform as long as it supports HTTP calls.

In order to implement our simple narrative model, we need to have an Artificial Intelligence Character, in this case, a Narrator. This character will perceive the events we have previously defined and return the appropriate "dialogue". We implement our scenario, and character by using the FAtiMA Integrated Authoring Tool.

### 2.4.1 Creating a scenario

We start by opening up the FAtiMA Integrated Authoring tool, when we do we start with a new scenario as shown on Figure 2. We should fill in the Scenario text field with the scenario's name, fill in the description field with a short description of the scenario. We can now take the time to do our initial save of the scenario (this is important, as we alter the scenario we should be saving constantly), simply go to File - Save As. As a recommendation, we recommend that the filename ends with -scenario so that you can tell the scenario file, from

[3]Further information and tutorial resources and examples can be foun at the website: https://fatima-toolkit.eu/home/get-started/

the assets file that will be required at a later time, alternatively, just store a single scenario and single assets file per directory.

Next, we should create a new Cognitive Rules File, we do this by clicking on the **New** button under the header Cognitive Rules File (see Figure 2). This will be the previously mentioned asstes file, and as a recommendation we suggest that the filename ends with -assets so that you can tell it appart from the scenario file.

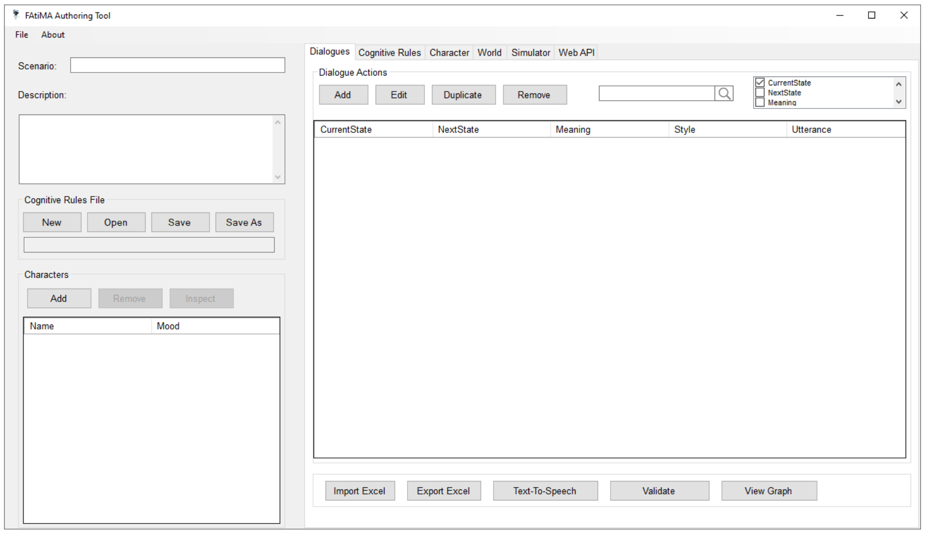


Figure 2: FAtiMA Integrated Authoring Tool Start Screen

We recommend saving the scenario file at this time by using File ¿ Save.

### 2.4.2 Creating a Narrator

We can now add a character to the scenario, in this case we are adding a Narrator character so simply click the **Add** button under the Characters header shown in Figure 2. In the popup that appears simply introduce the name for the character (Narrator, in this case). Take the time to save your scenario again.

We can now setup a simple World Action that allows our Narrator character to speak. Select the Narrator character we just created and that is listed under the Characters header (as shown on Figure 3). Then on the right hand side of the screen select the **World** tab.

### 2.4.3 Adding as Speak Action to the Narrator

We will now add an action by clicking the **Add** button that is under the Action header in that tab (see Figure 4). You will be greeted with a popup as is shown in Figure 5. We wish to setup a Speak action that all characters in the world can respond to so we start by filling the **Action** field with the following: Speak([cs],

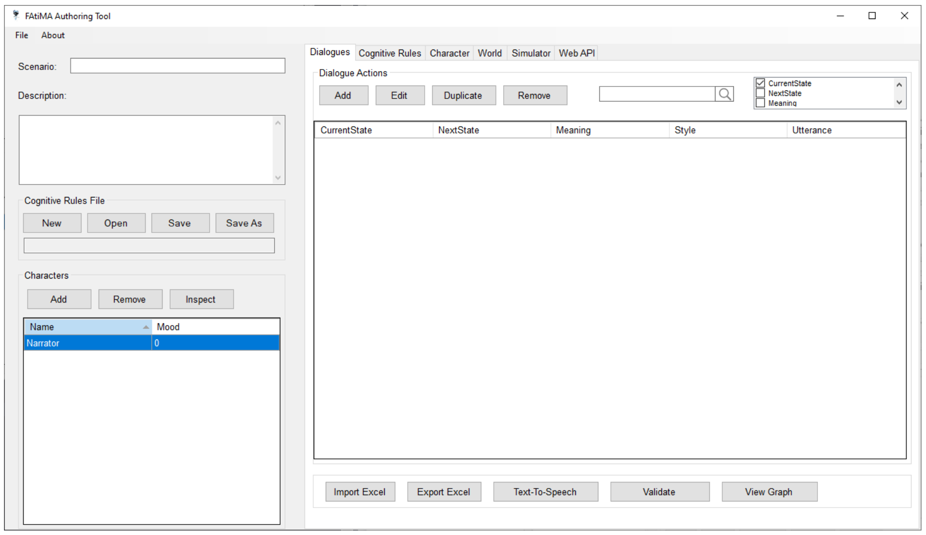


Figure 3: Selecting the Narrator

[ns], [m], [style]). This means that a speak action will take a current state (the [cs] specified in the string), a next state (the [ns] specified in the string), a mood (the [m] specified in the string) and the style (the [style] specified in the string) as parameters. We define the **Subject** as [s], the **Target** as [t] and the **Priority** as 1. We then click the **Add** button present in the popup in order to create the new speak action.

While in this tab we will also add an effect to the action we just created. Select our **Speak** action, then under the header **Effects** click the **Add** button. a popup will appear (see Figure 6). Fill in the **Property Name** field with the string DialogueState([t]), the **New Value** field with [ns], and the **Observer Agent** Field with * (this represents that this action will have the effect of changing the dialogue state of the target [t] to the next state [ns] for any agent that observes it).

### 2.4.4 Adding the dialogue

We can now add the dialogue to our Narrator character. Click the Dialogues tab (the resulting screen is shown in Figure 7). Then under the **Dialogue Actions** header click the **Add** button (A popup will appear as shown in Figure 8). We can now fill in: the **Current State** field, which specifies the state that the character needs to be in order for the dialogue to be triggered, we suggest using a text token to identify it; the **Next State** field, which specifies the state that the character will be in after the dialogue is triggered, we suggest using a text token to identify it; the **Meaning** field, which specifies the meaning associated with the dialogue, we suggest using a text token to identify it: the **Style** field, which specifies the style of that dialogue ; and the **Utterance** field, which

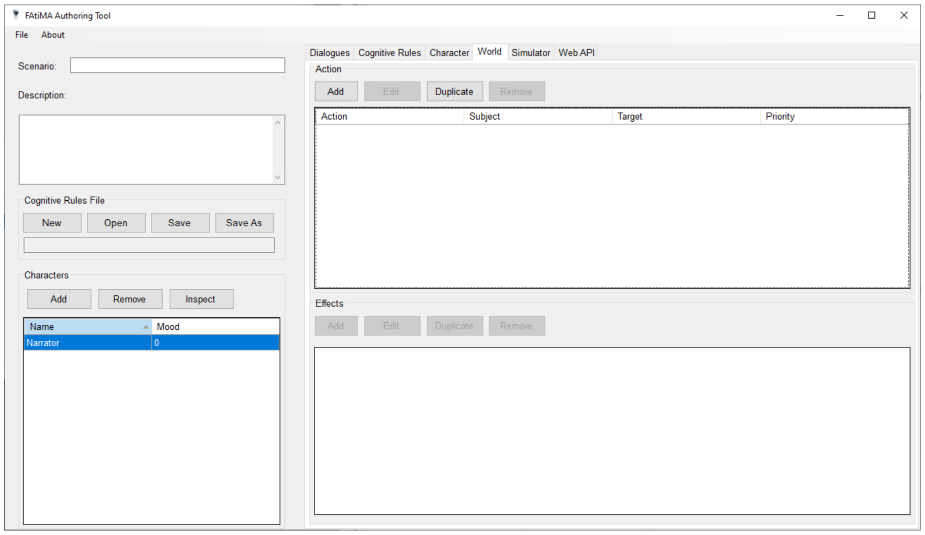


Figure 4: World Actions Tab

specifies the text that will be uttered when this dialogue is triggered.

### 2.4.5 Running the FAtiMA HTTP server and loading a scenario and its respective assets

The FatiMA HTTP server can be run in any machine that supports the .NET Core environment (Microsoft currently supports Windows, Linux and macOS). Running the dotnet FAtiMA HTTP server is achieved by, while in the directory that contains the FAtiMAHTTPServer.dll file, running the command

```
$ dotnet ./FAtiMAHTTPServer.dll port scenario assets
```

where the port is the port you want to run the server on, scenario is the path to the scenario JSON file that was created in the FAtiMA Integrated Authoring Tool, and assets is the path to the assets JSON file that was also created in the FAtiMA Integrated Authoring Tool.

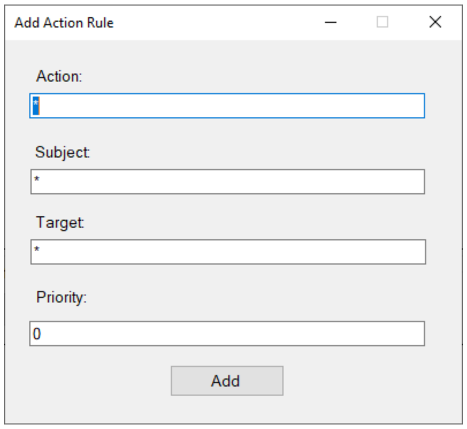
As an example, the command

```
% dotnet ./FAtiMAHTTPServer.dll 5000 ../scenario.json ../assets.json
```

will run the FAtiMA HTTP server on port 5000, loading the scenario file scenario.json contained in the parent directory of the current directory, and the assets file assets.json contained in the parent directory of the current directory.

### 2.4.6 How to integrate a web application with the FAtiMA HTTP server

Integrating a web application with the FAtiMA HTTP server is a simple matter of making the simple HTTP requests. It is important to note, that, at this time,


Add Action Rule

Action:
*

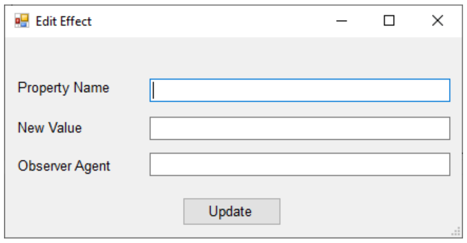
Subject:
*

Target:
*

Priority:
0

Add

Figure 5: Add Action Popup


Edit Effect

Property Name

New Value

Observer Agent

Update

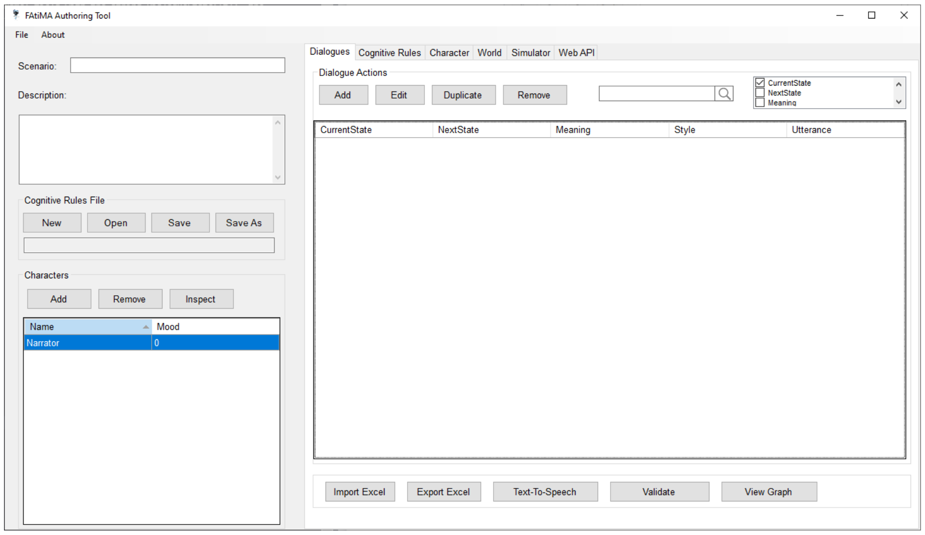
Figure 6: Add Effect Popup

the current implementation of the FAtiMA HTTP server does not support Cross-Origin Resource Sharing, and in order to work around it, we advise installing a request forwarding proxy that supports Cross-Origin Resource Sharing and will forward the requests from the integrated web application to the FAtiMA HTTP server. This means that while we will exemplify the required requests for getting the dialog in this section by assuming that we are sending them directly to the FAtiMA HTTP server, it will be necessary to send them to the request forwarding proxy.

The overview of the required requests to get a character's text is quite simple, we simply make the character perceive an event, and then get the character's dialogue. As an example, in order to send a perception the Narrator character that is running at the server address http://localhost:5000, we send an HTTP Post request to http://localhost:5000/perceive?c=Narrator, with a the following JSON body:

```
["event(property-change, world, DialogueState(Player), newState")"]
```

In this case, we are sending the character a single property-change event that will update its DialogueState to the newState (FAtiMA supports additional event types and perceiving more than one event). After the character has



| Name | Mood |
|---|---|
| Narrator | 0 |

Figure 7: Dialogues Tab

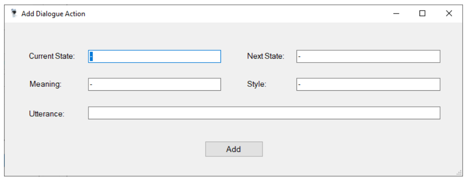


Figure 8: Add Dialogue Popup

perceived the event, we can simply get the Narrator's text by sending an HTTP Get Request to http://localhost:5000/decide?c=Narrator and getting the text contained in the first position of the response in the Utterance node.

**Example Code in Javascript:**

```
function getEventUtterance(event) {
    // Send the POST request that makes the narrator perceive an event
    $.post("http://localhost:5000/perceive?c=Narrator",
      JSON.stringify([
          "event(property-change, world, DialogueState(Player)," + event + ")"
        ])
    // Then get the character's dialogue
    ).then(getUtterance());
}

function getUtterance() {
    // Send the Get request that allows us to fetch the dialgoue
    $.get("http://localhost:5000/decide?c=Narrator").done(function( data ) {
        $("#narratorText").text(data[0].Utterance)
    });
}
```