Situation-Adaptive Multimodal Dialogue Platform

Version 1.4.0

Tutorial

Contents

1. Introduction

About SiAM-dp

The SiAM-dp is intended to allow dialogue engineers to rapidly create multimodal dialogue applications with sufficient latitude to adapt and extend individual components for specific research issues in the context of multimodal dialogue systems. In the development, particular attention was paid to the fact that simple applications should be kept simple without losing the depth that is needed to implement more complex applications or components to examine relevant research questions in the intelligent user interface group. Thus, the system is suitable for both, the quick development of demonstrators that concentrate on very domain specific use case aspects and the use as a base platform for dialog system specific research tasks.

2. Installation

System Requirements

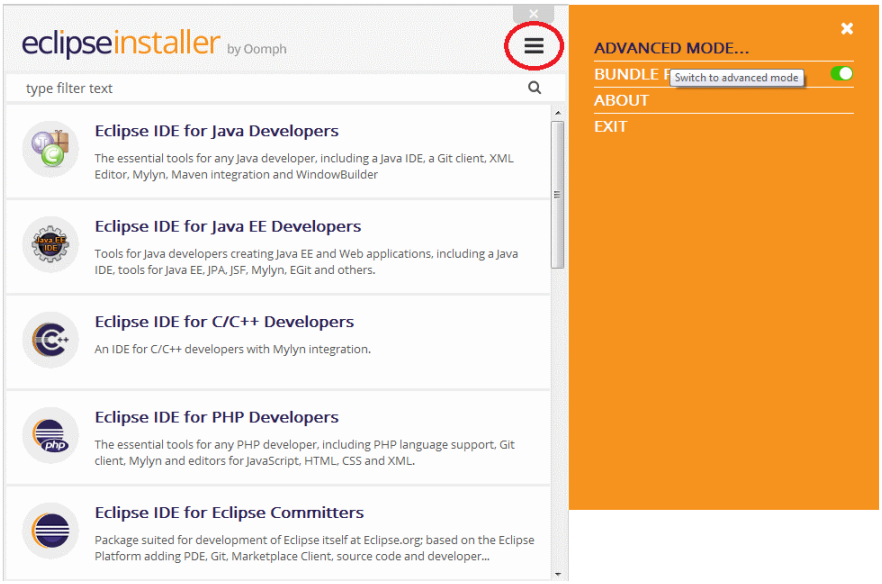
- Minimum Java Version: 1.8
- Eclipse Version: Luna Eclipse Modeling Tools

The SiAM-dp runtime environment runs in Equinox OSGi. The SDK provides development tools for the creation of new dialogue applications and extends the Eclipse RCP with editors and wizards. Model descriptions are modeled with EMF (Eclipse Modeling Framework).

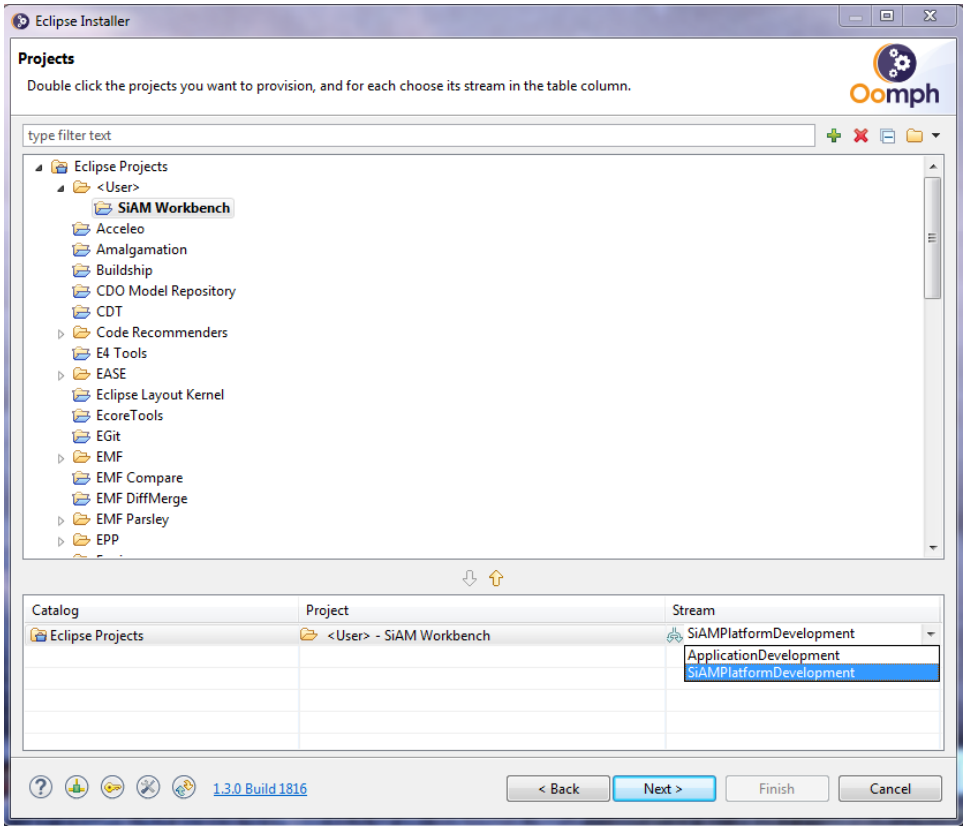
Before installing the SiAM-dp platform you need to download the **Eclipse Modeling Tools** package solution from the Eclipse Download page (https://www.eclipse.org/downloads/). Moreover**,** you need to create your own GitLab account by registering on the following website: www.forschungssoftware.de. Your account credentials will be needed later on during the installation process.

With the new version of Eclipse the **Eclipse installer** has been introduced, which allows to automate the process of downloading Eclipse, installing required plugins, setting necessary preferences, and creating a workspace. For the installation of SiAM-dp follow these steps:

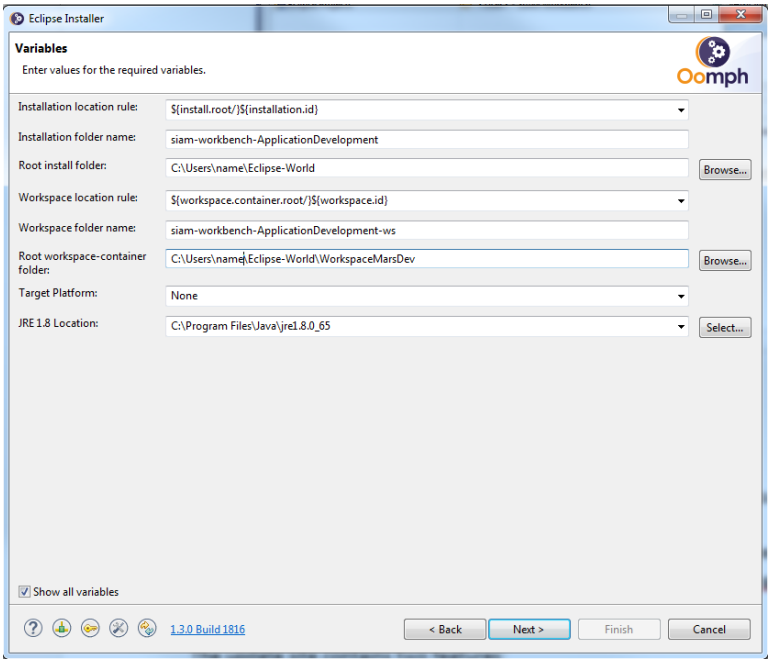
1. Download the Eclipse installer here: https://wiki.eclipse.org/Eclipse_Installer
2. After the download, start the installer and switch to the advanced mode



3. Select “Eclipse Modeling Tools” and continue to the next screen.
4. Press on the Plus Icon “Add user project”.
5. Insert the following URL: http://madmacs.dfki.de/siam_dp/updateSite/siamWorkbench.setup and select the Eclipse.org catalog.
6. Double-Click on the new entry in order to add the new entry to the table at the bottom of the window. The screen should now look like this:



7. As Stream of the update site you can select between two features: you can either select “ApplicationDevelopment” (no source code) or “SIAMPlatformDevelopment” (including source code).
    a. **ApplicationDevelopment**
    The runtime environment is necessary for running dialogue applications and contains all libraries of the dialogue application engine. Install this feature if you already have a finished dialogue application which you want to execute on the computer.
    b. **SIAMPlatformDevelopment**
    This SDK feature provides editors and wizards that allow the comfortable development of dialogue applications in Eclipse. Install this feature if you want to create a new or edit an existing dialogue application using the integrated tools.
8. Click on “Next”. Click on “Show all variables”. Here it is possible to define the directories for the Eclipse installation as well as the created workspace. Make sure you choose the correct Workspace location rule. The other settings should keep the default values. The entries should look similar to this:



9. Click “Next” and follow the installation process. If prompted, accept requirements during this installation process.
10. After the installation process Eclipse will automatically start (this might take some minutes). Wait until Eclipse has finished executing the setup tasks (process bar in the lower right corner). If prompted to enter username and password, enter the credentials of your previously created GitLab account (of the Forschungssoftware website). These setup tasks install required plugins and clone projects into your given workspace location.
11. Once the build has finished, close the “Eclipse Installer” window via clicking “finish”.
12. **NOTE**: In case the startup task cannot be completed due to a “cannot open git-upload-pack” error (“java.lang.Exception:org.eclipse.jgit.api.errors.TransportException: https://www.forschungssoftware.de/dfki/SiAM-dp.git: cannot open git-upload-pack”) you can resolve this by adding a git configuration entry in Eclipse as follows: Go to *Windows* -> *Preferences* and enter “git” in the text field. Under *Git* go to *Configuration*, click on *“Add Entry…”.* As *Key* enter “http.sslVerify” and as *Value* enter “false”. Click on OK -> Apply -> OK. Go to *Help* -> *Perform Setup Tasks* and click on Finish. This will restart the setup process.

In the remainder of the tutorial, we assume that you have installed the SDK **SIAMPlatformDevelopment** feature.

3. Building your first dialogue application project

This section will guide you through the process of building a dialogue system application from scratch. The final result can also be found in the downloaded workspace under "…/SiAM/demonstrators" or it can be downloaded as example here: http://madmacs.dfki.de/?page_id=26.

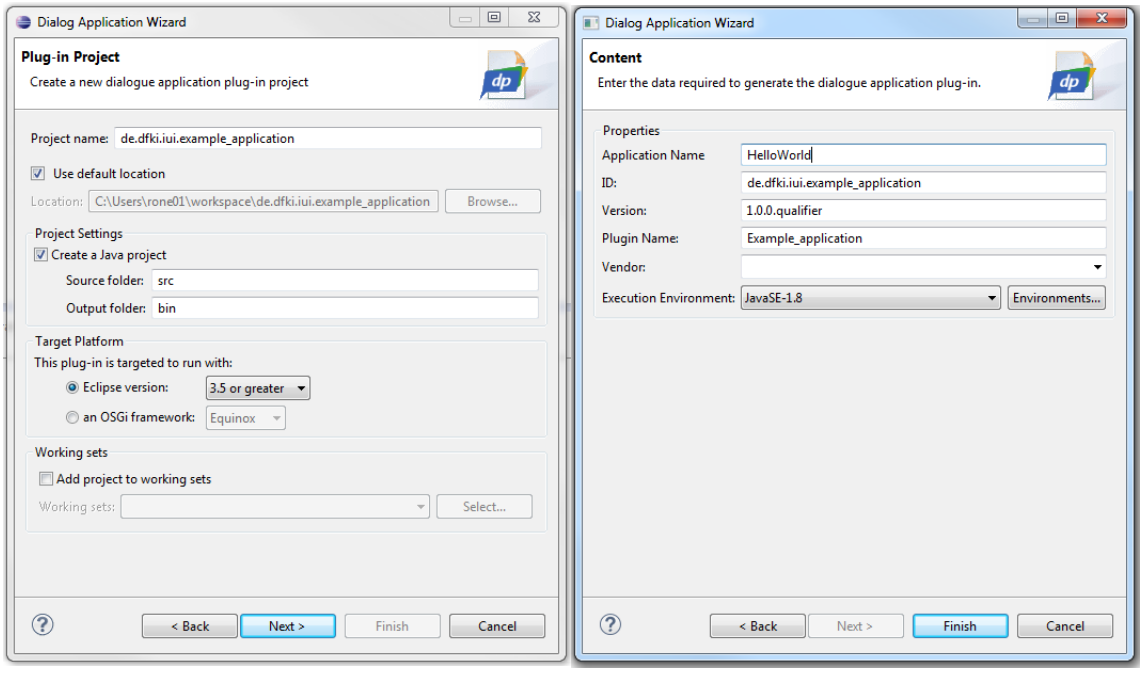
3.1. Creating a new dialogue application project

A project brings together all files and resources that make up your dialogue application, thus simplifying the design process. The modeling of a new project will take place in the *Dialogue Designer*. For running the designer follow these steps:

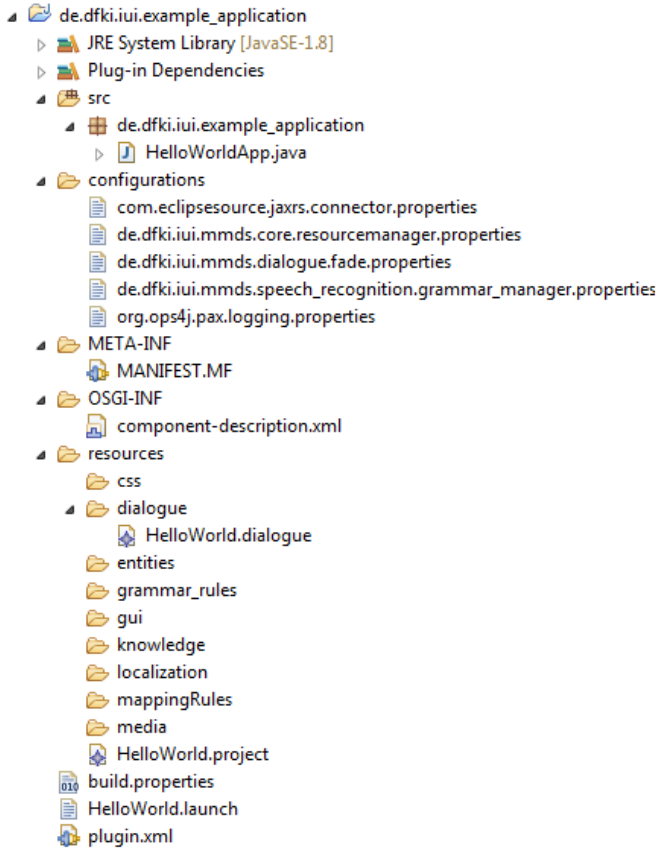
1. In Eclipse click on Run → Run configurations
2. Unfold the Eclipse Application → click on *Dialogue Designer* → Run
   This will open a new Eclipse environment.
3. In the newly opened *Dialogue Designer* click on Project -> Build Automatically should be inactive

In the *Dialogue Designer* the SiAM SDK contains a wizard for the creation of new dialogue application projects.

1. Open the wizard: *File → New → Project…*
2. Select *SiAM DP → SiAM Application*
3. In the next window you can set up the plugin project. For the *Project Name* choose *de.dfki.iui.example_application* and click on *Next*.
4. In the next window enter the application name *HelloWorld* and press the *Finish* button. The other settings on this window are general OSGi plugin information and added to the manifest file of the plugin.



The wizard automatically creates a new eclipse plugin project and complements some relevant dialogue application files and configurations. The final project will be stored in the workspace folder under “demonstrators” (Note: this is not the folder “../SiAM/demonstrators”). The following image shows the initial project contents:



In the file *component-description.xml* a new OSGi component is defined. The description declaratively binds the component to the *ProjectManager*-service of the dialogue platform. The component’s class is the generated *HelloWorldApp.java*. This class extends the class *AbstractDialogueApplication* which is responsible for loading a dialogue project declaration resource into the dialogue system. For this, the constructor of *the HelloWorldApp.java* calls its super constructor with the name of the file with the project description as argument.

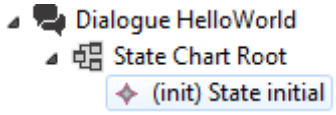
```
public class HelloWorldApp extends AbstractDialogueApplication {
	public HelloWorldApp() {
		super("HelloWorld.project");
	}
}
```

The project description *HelloWorld.project* is located in the *resources* directory. For many basic tasks it is sufficient to make changes to this project file and the referenced resources using the corresponding editors. For some advanced functionality, the dialogue application class can also directly be customized.

With the project description the project’s context and resources are specified. This includes the dialogue model that describes the program logic, rule sets for speech recognition grammar

specification, sessions, users, supported devices and domain specific ontology models. Furthermore resources for presentation like media content or style sheets are declared.

The *HelloWorld.project* file can be opened with the SiAM-dp model editor by simply double-clicking it in Eclipse. It only contains a reference to the dialogue model for our application at *resources/dialogue/HelloWorld.dialogue.* This model consists of a statechart with an empty initial state.



In the folder *configurations* configuration files for logging, the restful server and grammar management are located.

3.2. Launching the application

For launching the project it needs to be imported to the projects in Eclipse. For doing this, *right click on the package explorer → Import… → choose General → Existing Projects into Workspace → Browse to "YourWorkspace/demonstrators/de.dfki.iui.example_application" → Ok → Finish*.

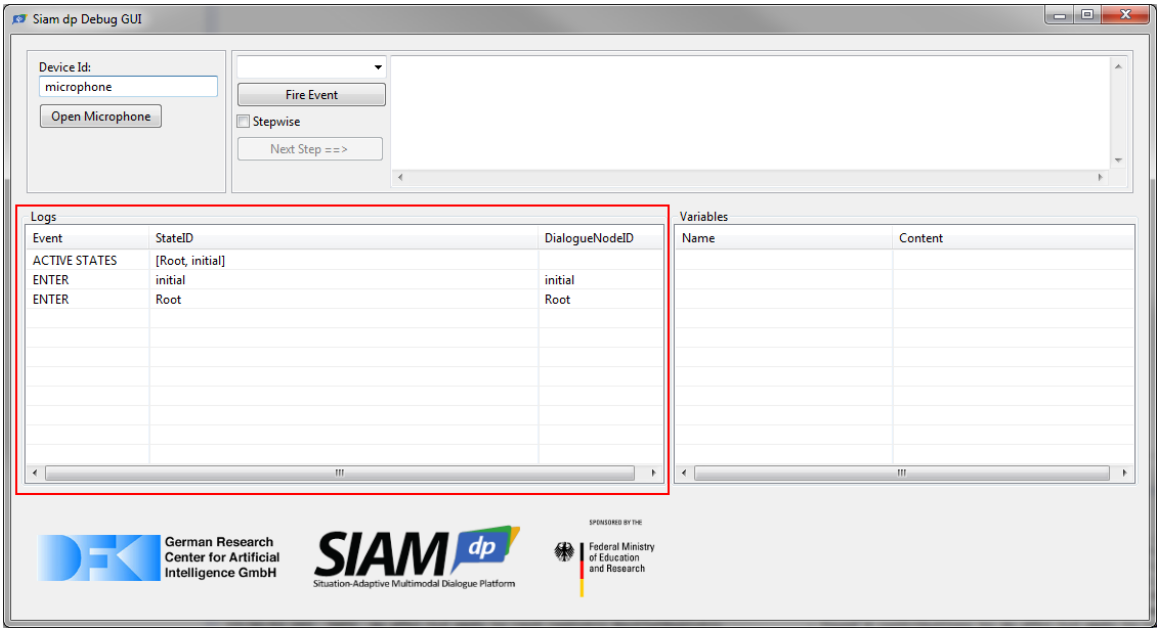
The wizard generates the equinox launch configuration *HelloWorld.launch*. It is necessary to add platform specific bundles before starting the application the first time. For this select the *HelloWorld* configuration in *Run as → Run Configurations.* Make sure that the following bundles in Workspace are selected*:*

- *de.dfki.iui.example_application* (with Start Level 0)
- *de.dfki.iui.mmds.application*
- *de.dfki.iui.mmds.cm*
- *de.dfki.iui.mmds.core*
- *de.dfki.iui.mmds.core.interfaces*
- *de.dfki.iui.mmds.core.model*
- *de.dfki.iui.mmds.dialogue*
- *de.dfki.iui.mmds.dialogue.fade*
- *de.dfki.iui.mmds.io.html*
- *de.dfki.iui.mmds.io.rest*
- *de.dfki.iui.mmds.knowledge_server*
- *de.dfki.iui.mmds.presentation_planner*
- *de.dfki.iui.mmds.scxml*
- *de.dfki.iui.mmds.scxml.engine*
- *de.dfki.iui.mmds.speech_recognition*
- *pax-logging-service-appender*

Click *Apply → Run*.

After starting the application the SiAM-dp Debug GUI is displayed. It gives an overview about active states, fired transitions and variable values in the dialogue engine. The red box in the figure below marks the statechart-engine log view that logs state changes. Since the example only contains an

empty *inititial* state in the statechart *Root*, the engine enters this state and nothing further happens. There are no variables defined in the initial dialog specification model.



3.3. Adding a graphical user interface

SiAM-dp supports the creation of graphical user interfaces with an own model. It follows a declarative GUI design approach, inspired by HTML and other application markup languages like XAML by Microsoft or the declarative GUI design for Android apps.

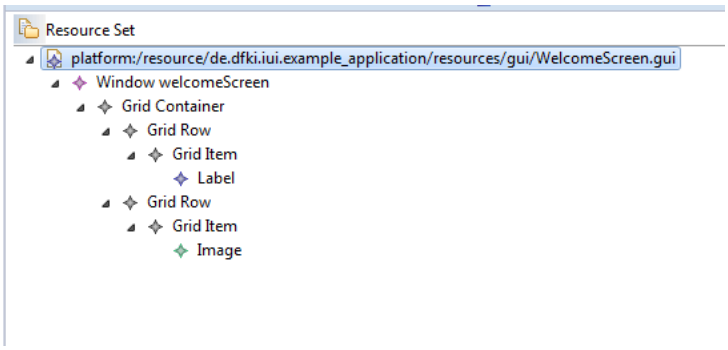
The first step is to add a send event to the dialogue project that sends an output act containing the view to be shown on the GUI device when entering the initial state.

1. Open the *HelloWorld.dialogue* file in the *resources/dialogue* directory
2. Right-click on *State initial* and select *New Child* → onEntry → *Send*
3. Right-click on (OnEntry) Send: New Child → outputRequest → Output Act
4. Right-click on Output Act: New Child → presentation → GUI Application
5. Set the device property of GUI Application to “WelcomeScreen”.
   (This tells the dialogue platform explicitly which of the registered devices should receive the output act.)
6. Save the dialogue model file.
   (Changes are only synchronized between editor windows when you save the file.)

At this point the GUI we are sending is still empty, so let us define a new window layout and add it to the send event.

1. Right-click on the *resources/gui* folder. (By convention, GUIs are put into this directory.)
2. Select New → Other → SiAM DP → SiAM DP Model
3. Choose “GUI Window” from the dropdown box
4. Set the filename to “WelcomeScreen.gui”
5. Open the new file in the editor

6. Set the *Id* property of *Window* to welcomeScreen
7. Set the *Title* property of *Window* to “Welcome”
8. Add a *Grid Container* child to the *elements* slot of the *Window* instance.
   (A grid defines a layout based on rows and columns, which can easily resize to different window sizes.)
9. Add two *Grid Rows* elements to the slot *row* of the *Grid Container*
   (You can easily find the entry in the list by typing the first character in your keyboard)
10. Add a *Grid Item* child to the *item* slot of the first *Grid Row*
11. Add a *Label* child to the *Grid Item*
12. Set the *Text* property of the *Label* to “Hello World”
13. Add a *Grid Item* child to the second *Grid Row*
14. Set the *Alignment* property of *Grid Item* to *Center*
15. Add an *Image* element to the second *Grid Item*
16. Set the *Id* property of the *Image* to “worldImage”
17. Save the GUI model file.
18. Open the *HelloWorld.project* in the editor
19. Right Click anywhere and select *Load Resource* → *Browse Workspace*
20. Select the GUI instance *resources/gui/WelcomeScreen.gui* and press Ok.
    (This allows you to access the window model from within your project file)
21. Expand the *Dialogue HelloWorld* entry in the tree until you see the *GUI Application* entry, and select it.
22. In the property view, select the welcome screen for the *Window* property from the drop-down list.



The image we added to the window is still empty. We need to register an image resource and connect it to the GUI image.

1. Save the image at http://madmacs.dfki.de/siam_dp/downloads/world.png in the *resources/media* directory (create this directory if it does not exist).
2. Open the *HelloWorld.project* model
3. Add a *Picture* child to the *digitalResources* slot of *Project.*
4. Set the *Id* property of *Picture* to “world”.
5. Add a *Digital Location* child to *Picture.*
6. Set the *Url* property of *Digital Location* to “media/world.png”.
7. Expand the dialogue tree until you see the *Image* instance in the GUI model.
8. Select “Picture world” for the *Resource* property of this instance.

Launch the application out of Eclipse (not within the *Dialogue Designer*). The logging output in the Eclipse console now informs that an output act was fired with the previously defined GUIApplication as content. The SiAM-dp dialogue platform provides a generic HTML GUI renderer that creates webpages for all GUI devices in the dialogue application. You can access a list of all available GUI devices with graphical content at http://localhost:8188/mmds/html/gui/. Click on "gui" to see the rendered HTML representation of the GUI defined by our application. It should look similar to the following picture:

If you cannot see the page, make sure that you are running only one instance of the dialogue application. You can also try to restart the browser.

Set an event handler

We cannot interact with the GUI yet. The first step towards this goal is to define which user events should be reported to the dialogue platform.

1. Go back to the GUI model and unfold it until you see the *Image worldImage* instance.
2. Add a *Supported Event* child to *Image*
3. Choose for property *Event Type* the entry *ClickEvent*

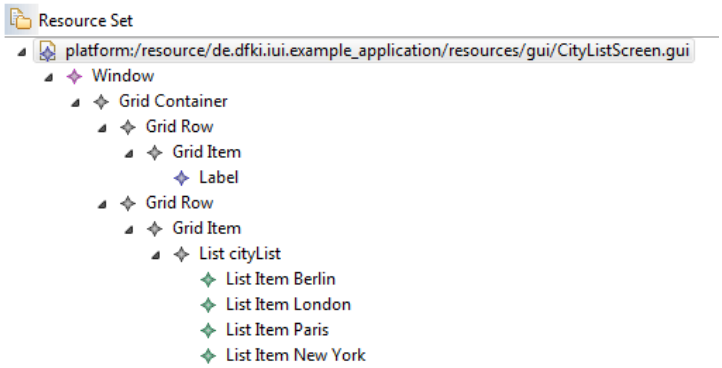
Launch the application and observe in the Eclipse console that a ClickEvent input act is sent to the dialogue platform if you click on the world image in the browser. Now we receive the event in the dialogue platform, but it still does not cause any visible effect in the dialogue. This event can be captured in the dialogue specification and be used to trigger transitions between states.

Define a second GUI window and a state that displays this screen

We define a second window that is displayed if we click on the world image.

1. Right Click on the *GUI* folder and select *New → Other → SiAM DP Model*
2. Select *GUI Window*
3. Set the file name to CityListScreen.gui
4. Set the Id property to "citySelectionScreen"
5. Set the title property to "City Selection"

6. Add a Grid Container to the Window instance
7. Add two Grid Rows as children to the Grid Container
8. Add a Grid Item to the first Grid Row
9. Add a label to the Grid Item
10. Set the text property of the Label to “Where are you now?”
11. Add a Grid Item to the second Grid Row
12. Set the Alignment property of Grid Item to Center
13. Add a List to the second Grid Item
14. Set Id property of List to *cityList*
15. Add four List Item children to List
16. Set the Label properties of the ListItems to “Berlin”, “London”, “Paris” and “New York”



17. Save the file.
18. Open the HelloWorld.dialogue file in the editor
19. Add the second gui-resource to the model: Right Click → *Load Resource* → *Browse Workspace...* → *gui/CityListScreen.gui* → *Ok*
20. Add a new *State* to the *states* slot of *State Chart Root*
21. Set the *Id* and *Display Name* property to “selectCity”.
    (Setting the display name is optional but simplifies handling in the editors.)
22. Add an *On Entry Send* child to this state
23. Add an *Output Act* child to *Send*
24. Add a *GUI Update* child to *OutputAct.*
    (A GUI Update output act will change an existing GUI rather than creating a completely new one. The difference between this and sending an new GUI Application presentation alternative is that only the window content is changed, information about used style sheets are retained.)
25. Set the *Device* property to “WelcomeScreen”
26. Add a *Window Update* child to *GUI Update*
27. Select “Window citySelectionScreen” for the *Window* property
28. Save the file.

Define a transition between two states that reacts on the click input event on the world image and updates the screen

Now we define the transition between both screens. The transition is defined for the state in which the transition starts, and is parameterized by the condition which needs to be met (in this case an InputAct of a specific type) and the target state to which to proceed.

1. In the dialogue model, add a *Transition* to *State initial*
2. Set target property to the *State selectCity*
3. Add a *Cond Event* to the slot *event* of the Transition
4. Add *InputAct* to the slot *pattern* of *CondEvent*
5. Add *representation* to *InputAct*
6. Add *GUIEvent* to *representation*
7. Add *eventData* to *GUIEvent*
8. Add *ClickEvent* to *eventData*
9. Add *targetId* to *ClickEvent*
10. Add *BString* to *targetId*
11. Add *PRestrictions* to *BString*
12. Add *PStringRestriction* to *PRestrictions*
13. Set property Value to “worldImage”

The condition which we created essentially attempts to match input acts which are of type GUIEvent. Within these events, it matches Click events where the targetId (i.e. the ID of the clicked element) corresponds to the string “worldImage”.

Launch the application again. Now after clicking on the world image the next screen with a list of cities is displayed. Note: Before launching the application in Eclipse, refresh the project either with F5 or right click on the project: Refresh.

3.4. Defining Style Sheets for the GUI

The layout of the graphical user interface can be adapted by style sheets.

1. Set the property Id of the Label in WelcomeScreen.gui to welcomeLabel
2. Set the property Id of the Label in CityListScreen.gui to citySelectionLabel
3. Create the file *helloWorld.css* in the *resources/css* directory
4. Enter the following content to the file

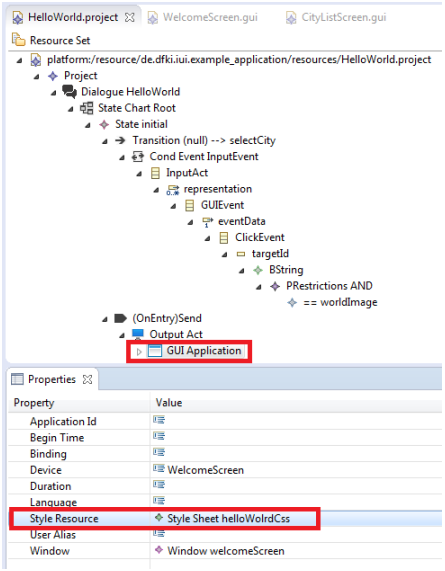
```
#welcomeLabel {
        display:block;
        text-align:center;
        font-size:35pt;
        color:#008000;
}

#worldImage {
        width:50%;
}

#citySelectionLabel {
        display:block;
        text-align:center;
        font-size:35pt;
        color:#800000;
}
```

```
#cityList {
        width:500px;
        font-size:large;
}
```

5. Open the *HelloWorld.project* file
6. Add a *Style Sheet* to the slot *digitalResources* of *Project*
7. Set the property Id of *Style Sheet* to “helloWorldCss”
8. Add a *Digital Location* to *Style Sheet*
9. Set the property *Url* of *Digital Location* to “css/helloWorld.css”
10. Unfold the Dialogue HelloWorld entry, until you see GUI Application
11. Choose “StyleSheet helloWorldCss” for the *Style Resource* property



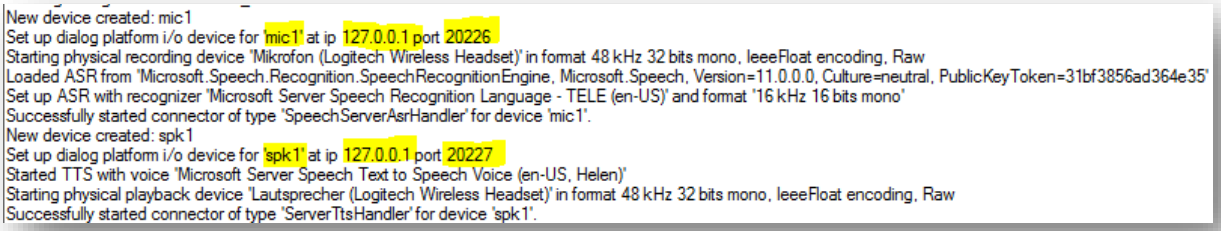
Relaunch the application to watch the layout changes.

3.5. Adding Speech Interaction

The SiAM-dp platform can easily be extended with other modalities or devices. Thus, speech synthesis and recognition are not limited to a single engine. However, for certain common modalities, SiAM-dp contains out-of-the-box support based on widely usable implementations for convenience. As such, a “default” provider for the speech modality, i.e. speech synthesis (TTS) and speech recognition (ASR) functionality is realized via the external Audio Manager for Windows component. In our example, we assume that you use Audio Manager for TTS, which works with the native Windows Speech API. You can obtain it from http://madmacs.dfki.de/secret/?page_id=30 (simply extract the archive to any directory – it contains a separate documentation as PDF as well).

For the example we need an Audio Manager configuration with a TTS and ASR plugin. The included sample configuration file (*default_en.xml*) provides exactly that. Use the file *start.bat* to run Audio Manager with this configuration. If the audio manager is configured correctly, ASR and TTS service should be accessible via TCP protocol. In this example, we will create a static association to the TTS (as output device) and ASR (as input device). For the connection with SiAM-dp it is necessary to specify the address and port where the audio devices are configured. Normally, Audio Manager assigns port

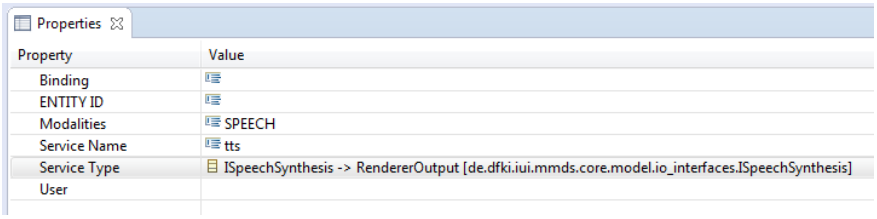
20226 to the first device and then increments by 1 for each successive device. To be sure, you can look at the Audio Manager console logs to find out the correct port numbers for both services.



Adding TTS / ASR Device Registration

All interactions in SiAM-dp are communicated through devices (either coming from input devices or going to output devices). In a first step we have to statically register a TTS and an ASR device for the dialogue application project. Their services will be defined as part of the entities:

1. First create a new entities model: In the *Project Explorer* of the *Dialogue Designer* unfold *resources* and right click on the folder *knowledge* → New → Other → Siam DP Model → select *Entity Resource* with the file name “start.entities”.
2. Secondly, load this new entities model into the project: In the *Dialogue Designer* open the *HelloWorld.project* file in the *resources* directory, right click somewhere → Load Resource → Browse Workspace → select *start.entities*
3. In the properties view of the *Project* (which is the root element of your project) add *Entity Resource* as *Entity Resource* property. Refresh the editor.
4. Right click on *Entity Resource* under *Project* and select *New Child* → content → Service Instance
5. In the *Properties View* of the new *Service Instance* set the *Description* and the *Service Instance Name* to “tts”
6. Right click on the Service Instance → NewChild → communicationinfo → TCP server
7. Set the property *Host* to “localhost” and the *Port* to “20227”
8. Right click on the Service Instance → NewChild → services → Simple Service
9. In the properties view of the *Simple Service* set the *Modality* to “SPEECH”, the *Service Name* to “tts” and the *Service Type* to “ISpeechSynthesis”



10. Right click on the project → New child → supportedServiceInterfaces → Siam Internal Service Interface
11. In the properties view of the *Service Interface* set the *Modality* to “SPEECH”, the *Service Instance Id*, the *Service Name* and the *Siam Id* to “tts” and the *Service Type* to “ISpeechSynthesis”.

For using the ASR functionality, i.e. for providing speech input, a service instance for the "asr" and a respective Siam internal service interface have to be added (as it was done for the "tts" functionality):

1. Add a *Service Instance* child of Entity Resource under *Project*
2. In the *Properties View* of the new *Service Instance* set the *Description* and the *Service Instance Name* to "asr"
3. Right click on the Service Instance → NewChild → communicationinfo → TCP server
4. Set the property *Host* to "localhost" and the *Port* to "20226"
5. Right click on the Service Instance → NewChild → services → Simple Service
6. In the properties view of the *Simple Service* set the *Modality* to "SPEECH", the *Service Name* to "asr" and the *Service Type* to "Speech".
7. Right click on the project → New child → supportedServiceInterfaces → Siam Internal Service Interface
8. In the properties view of the *Service Interface* set the *Modality* to "SPEECH", the *Service Instance Id*, the *Service Name* and the *Siam Id* to "asr" and the *Service Type* to "Speech".

Adding this "asr" speech device will automatically add this device as microphone in the SiAM-dp Debug GUI and the Audio Manager will automatically load the grammar that is specified within SiAM-dp.

Initially, this "asr" device has the device mode set to Speak-to-Activate. That means the microphone is constantly open and the Audio Manager will constantly detect speech input. You can change the mode as follows:

1. Add an *Update Device Mode* child to the *Siam Internal Device Interface* (as part of the *Project*)
2. Add a *Speech Recognizer Mode* child to the *Update Device Mode*
3. In the properties view of the *Speech Recognizer Mode* set the property *Mode* to "Open Microphone", "Close Microphone" or "Speak-to-Activate".

Adding TTS Synthesis to the dialogue model

1. Open the *HelloWorld.dialogue* file
2. Unwrap the *Send* action in *State selectCity* and add a *Speech Synthesis* child to *Output Act* (This will add a synthesis output that is read out essentially simultaneously with the change of the GUI.)
3. Set the *Device* property of *Speech Synthesis* to "tts"
4. Set the *Utterance* property of *Speech Synthesis* to "where are you now?"

Make sure that Audio Manager is running, and then relaunch the dialogue application. You should hear the text when you click the image. If you do not, check for warnings and errors in the Audio Manager console log.

3.6. Using variables

The dialogue definition model allows defining variables that are active in the scope of their parent's state. Values can directly be assigned to these variables by Assign Actions or with the pattern of a

conditional event. In our example we use the second method. We want to save the name of the selected city in a new variable and use it for speech feedback information. First we declare the variable.

1. Open the HelloWorld.dialogue file
2. Add a Variable *to State Chart Root*
3. Set the *Name* property of Variable to “cityName”
4. Choose BString for the Emf Type property.

In the next step we add a listener to the change event in the city list.

1. Open CityListScreen.gui
2. Unwrap the model to List cityList
3. Add Supported Event child to List cityList
4. Choose ChangeEvent for the Event Type property of Supported Event

In the console we now receive a change event log message, if we select a city in the list. We add a new transition that fires, if this event is received. The transition will set the active state as target state again, because we want to stay in this state. A transition must not be defined without a valid target state. In the pattern of the transition condition we bind the city name that is part of the input message to our variable.

1. Open the file HelloWorld.dialogue
2. Add a Transition to State selectCity
3. Set the Display Name property to “citySelected”
4. Choose “State selectCity” as “Target” property
5. Add “Cond Event” child to “Transition citySelected”
6. Add “Input Act” to “Cond Event”
7. Add “representation” to “Input Act”
8. Add “GUIEvent” to “representation”
9. Add “eventData” to “GUIEvent”
10. Add “Change Event” to “eventData”
11. Add “value” to “Change Event”
12. Add “BString” child to “value”
13. Set the “Var Name” property to “cityName”. This is the name of our variable.

The value entry of the change event is bound to the variable “cityName” whenever the pattern matching with the input event was successful and the transition is fired. In other words, each time the user selects a city in the list, the variable “cityName” will be updated with the name of this city. We can now reference to this value in a Speech Synthesis request.

1. Add *Send* child to *Transition citySelected*
2. Add *Output Act* child to *(On Trigger)Send*
3. Add *Speech Synthesis* child to *Output Act*
4. Set property *Device* of *Speech Synthesis* to “tts”
5. Set property *Utterance* to “$expr(cityName)”

The utterance is of type BString, a special data type that additionally allows setting the value of the String by a Script-Expression that is evaluated during runtime. Similar concepts are also available for the other atomic data types of Java (e.g., BInteger, BLong, BCharacter…). The keyword $expr(…) signals

that the content of the feature is not the text, that is entered, but the result of the script inside the brackets. SiAM-dp uses the Java Expression Language (JEXL) for scripting. You find a reference at http://commons.apache.org/proper/commons-jexl/reference/syntax.html. The Jexl-Engine has access to all variables that are active in the actual dialogue scope.

Note that the order of transition children matters: The variable must be assigned before it can be used in the output act. With the new transition the statechart enters the *selectCity* State again, if we select a city in the list. Since we don't want the system to update the screen and ask us about our location every time we select a city, we have to move the "(On Entry) Send" content of this state to the transition in the "initial" state that leads us to this state, i.e. it will be changed to "(OnTrigger) Send". You can do this by Drag&Drop.

After restarting the application you get a voice feedback when selecting a city.

3.7. Adding speech grammar and speech recognition

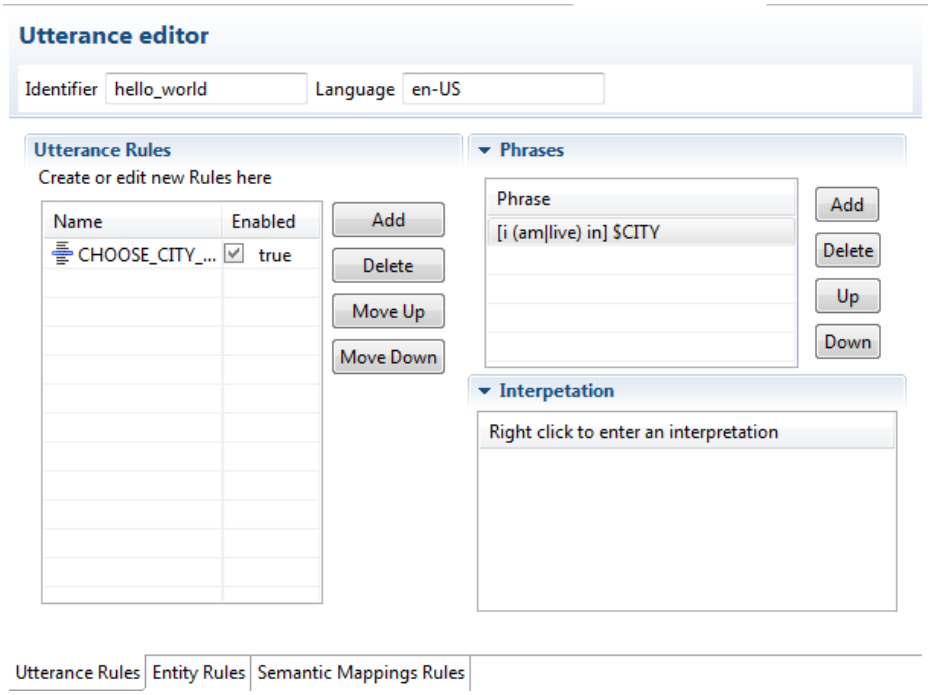
The next step is to add speech recognition to our application. SiAM-dp provides a rule based model that allows defining grammars for speech recognition. The model is automatically converted to GRXML (a standard for speech recognition grammars) and provided to the speech recognizer engine. We introduce a very simple grammar that allows the user to select a city by a speech command.

1. Right click on the folder *resources/grammar_rules* → *New* → *Other* → *SiAM DP Model*
2. Choose type *Grammar Ruleset*
3. Set filename to HelloWorld.grammar_rules

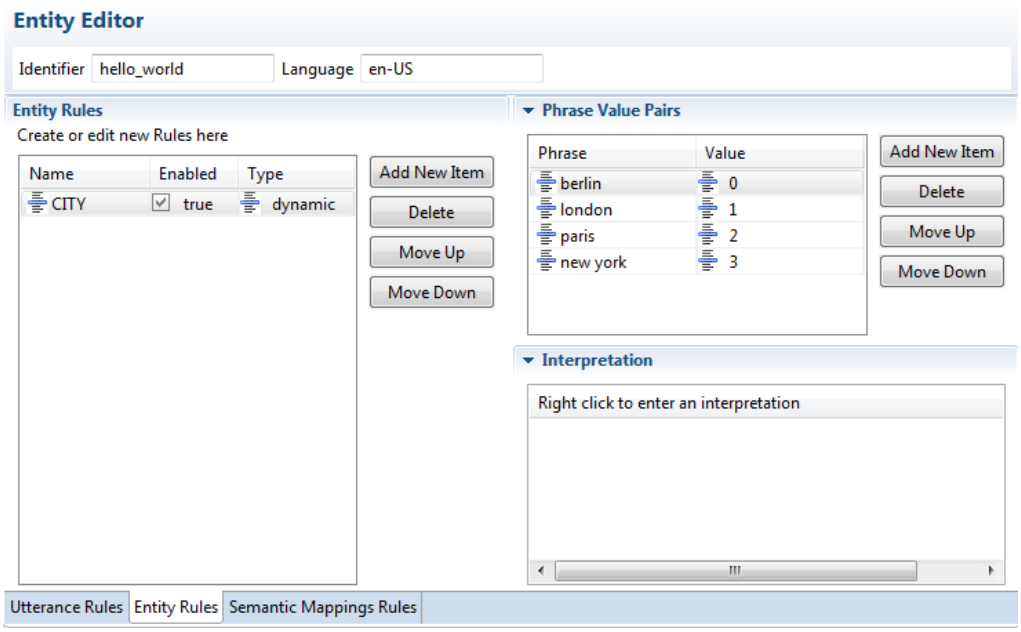
The file is opened in a special grammar editor. The grammar rule model provides three types of rules: **Utterances**, **Entities** and **Semantic Mappings**. Utterances are complete commands or sentences that are supported by the grammar. Entities describe a set of entities of the same content type that can be part of an utterance. Semantic Mapping Rules define sub-sentences that also can be part of an utterance. In our example we only define one utterance with a reference to an entity rule.

1. Set the Identifier of the ruleset to "hello_world"
   (The identifier allows to identify this rule set for activating/deactivating rules or changing the content of dynamic rules)
2. Set the language tag to "en-US"
   (If you were to specify a German grammar, you would use "de-DE" instead.)
3. Add a new Utterance Rule by clicking on the *Add New Item* button in the left part of the editor.
4. Set the name of the utterance rule (by clicking the new entry) to "CHOOSE_CITY_NAME"
5. Add a phrase by clicking the *Add New Item* button in the right part of the editor

6. Set the text of the phrase to " [i (am|live) in] $CITY"



7. The value *$CITY* is a reference to another rule. We create this rule by changing to the *Entity Rules* screen: Click on the *Entity Rules* tab at the bottom of the editor
8. Add a new entity rule
9. Set the name to "CITY"
10. Set the type to "dynamic" (this can be toggled by clicking the column or by using the properties window)
11. Add a *Phrase Value Pair* for every city, giving each phrase a numeric value (see screenshot). The numeric value serves as a unique identifier for the spoken entity phrase.



12. Save the grammar editor.

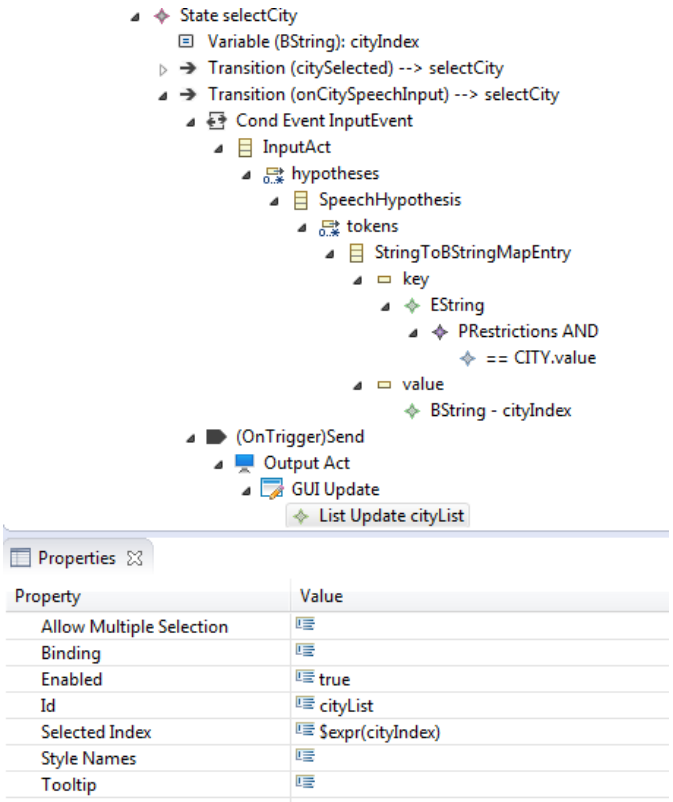
13. Open the file “HelloWorld.project”
14. Right Click on editor → Load Resource…
15. Browse Workspace… → Select the new grammar_rules file
16. Open the property view for *Project*
17. Press the “…” button in the Value field of the Grammar Rules property
18. Add the ruleset “hello_world”
19. Open the file “configurations/de.dfki.iui.mmds.speech_recognition.grammar_manager.properties.”
20. Add the line ”LANGUAGE = en-US” (if not there)

SiAM-dp supports multiple languages for one application. Thus it is possible to add grammars from different languages to a project resource. The language actually used (and therefore the active grammar rules) is specified by the language property entry in the configuration file.

Launch the application. You can open the microphone with the “Open Microphone” button in the SiAM-dp Debug GUI. Obviously, you need to have a microphone connected to your computer for this to work. (In more serious applications, it is possible to implement custom GUI buttons for push-to-talk, connect hardware buttons, or even change to speak-to-activate mode.) Give the speech input “I am in New York”. The recognizer now sends an input event to the dialogue platform. You can observe it in the console logging output.

We now add a transition that reacts on this input action and adapts the selected item of the city list.

1. Open the file “HelloWorld.dialogue”
2. Unwrap to the *State selectCity*
3. Add a variable to the state with EmfType: BString and Name: cityIndex
4. Add a *Transition* to “State selectCity”
5. Set the *Display Name* property to “onCitySpeechInput”
6. Set the property *Target* to “State selectCity”
7. Add *Cond Event* child to the transition
8. Add the following pattern: InputAct → hypotheses → SpeechHypothesis → tokens → StringToBStringMapEntry
9. Add a *key* child → EString → PRestrictions → PStringRestriction and set the *value* property to “CITY.value”
10. Add a *value* to the StringToBStringMapEntry
11. Add BString child to *value*
12. Set *Var Name* property of BString to “cityIndex”
13. Add a *Send* child to the Transition
14. Add *Output Act* → presentation → GUI Update
15. Set *Device* property of *GUI Update* to “WelcomeScreen”
16. Add *List Update* to *GUI Update*
17. Set *Id* property of List Update to “cityList”
18. As *Selected Index* property add the value “$expr(cityIndex)”



With the pattern in this condition we subscribe to speech input acts whose utterances contain the *CITY* sub rule. The received tokens provide tags with additional information about the used grammar rules that go beyond the simple utterance. In this case we retrieve the value we previously defined for a city entity in the grammar ruleset and bind it to the variable *cityIndex*.

Launch the application and test the speech input.

3.8. Using a Java Plugin

Imagine a scenario where the city list is dynamically updated by a service. We want to simulate this behavior by deploying a Java class that holds a list of city names. The dialogue model allows calling methods of java class instances as plugins directly from the dialogue model.

Fill the City List GUI element with content from a Java Plugin

1. Open the file “CityListScreen.gui”
2. Unwrap to “List cityList”
3. Remove all List Items from the list
4. Open the file “HelloWorld.project”
5. Add a *Java Plugin* child to “Project”
6. Set Class Name property to “de.dfki.iui.example_application.Plugin”
7. Set Namespace property to “helloWorldPlugin”
8. Add the Java class “Plugin.java” to the package “de.dfki.iui.example_application”
9. Add the following code to the class:

```
package de.dfki.iui.example_application;

import de.dfki.iui.mmds.core.emf.datatypes.BString;
import de.dfki.iui.mmds.core.model.io.gui.GuiFactory;
import de.dfki.iui.mmds.core.interfaces.IGrammarManagementService;
import de.dfki.iui.mmds.core.model.io.gui.List;
import de.dfki.iui.mmds.core.model.io.gui.ListItem;

public class Plugin {

	static String[] cityNames =  {"Berlin","New York","Tokyo","Paris","London", "Madrid"};

	public List getCityList() {
		List list = GuiFactory.eINSTANCE.createList();
		for(int i=0;i<cityNames.length;++i) {
			ListItem item = GuiFactory.eINSTANCE.createListItem();
			item.setLabel(new BString(cityNames[i]));
			item.setValue(new BString(cityNames[i]));
			list.getItem().add(item);
		}
		return list;
	}
}
```

[Note: In the dialogue designer this will lead to import errors since the plugin has no direct access to the required projects.]

10. Go to file “CityListScreen.gui”
11. Set the property Binding of “List cityList” to helloWorldPlugin.getCityList().

The result of the script code in the property *binding* is extended with the content of the object that defines the binding. Thus, the resulting object will be the List with the supported event as defined in the model combined with the list item content it retrieves from the plugin method call.

Now we also add the cities to the dynamic CITY entity rule in the speech grammar.

1. Open the file “HelloWorld.grammar_rules”
2. Delete all city entries in the rule: CITY
3. Set the enabled attribute of the rule CHOOSE_CITY_NAME to false
4. Open the Plugin.java class and change the getCityList()-method

```
public List getCityList() {
		IGrammarManagementService grammarManagementService =
HelloWorldApp.getGrammarManagementService();
		List list = GuiFactory.eINSTANCE.createList();
		for (int i = 0; i < cityNames.length; ++i) {
			ListItem item = GuiFactory.eINSTANCE.createListItem();
			item.setLabel(new BString(cityNames[i]));
			item.setValue(new BString(cityNames[i]));
			list.getItem().add(item);
```

```
                    grammarManagementService.addEntityToDynamicRule("hello_world",
"CITY", cityNames[i], Integer.toString(i));
                }
                grammarManagementService.enableRule("hello_world", "CHOOSE_CITY_NAME");
                grammarManagementService.commit();

                return list;
        }
```

Our HelloWorldApp.java-class provides a reference to the grammar management service of the SiAM-dp platform. This allows the manipulation of grammar rules during runtime. For each item of the city list we add an entity to our *CITY*-rule. Furthermore we now enable the *CHOOSE_CITY_NAME* rule that we previously disabled in the grammar model. Thus the grammar rule is only active in the second screen. All changes become effective with the grammerManagementService.commit() – method call. In the console you can see that with this command the changed grammar is sent to the speech recognizer devices.

Launch the application. Now the cities displayed in the list are the cities from the string-array in the Plugin class. They can also be addressed by speech input.