For integration purposes integrator gets think integration layer, called VWML2EWIntegration. For Java integration this layer is represented by only one jar file – VWML2EWIntegration.jar. Integrator can get this file either from directly VWML development team or from project’s MVN repository. This layer contains both components: abstract integration and examples. Process of integration consists from following steps:

1. The abstract classes which included into abstract integration component should be implemented and tested by integrator.
2. Implementation these classes should replace ‘debug fringes’ in VWML project
3. Add integration layer, VWML2EWIntegration.jar, into main project file (POM if exist)
4. Re-compile VWML project, into main project’s workspace, in order to get final sources of the model.
5. Build final project and run debug tests.

Let define general steps which describe how to implement VWML defined fringes which described in previous sections:

For each fringe’s object following steps should be passed:

1. Inherit from com.vw.lang.beyond.java.fringe.gate.IVWMLGate and to implement all abstract methods, all code will be concentrated in (other methods may have empty implementation)

@Override

**public** EWEntity invokeEW(Object commandId, EWEntity commandArgs)

This method we will consider as dispatcher method, where commandId argument means which method should be called (contains method’s name) and commandArgs – arguments which are passed to specific methods and where they are processed.

1. Implement methods with names defined in VWML project
2. Integrate methods into dispatching mechanism declared in 1.

**Fringe ‘Conf’**

For fringe’s object ‘Conf’ following methods should be declared with following signature:

protected EWEntity GetImagesList() throws Exception;

here EWEntity is complex entity which contains logical images’ names

protected EWEntity GetImagePlaces() throws Exception;

here EWEntity is complex entity which contains logic names of places where images should be placed

protected EWEntity GetCouples() throws Exception;

here EWEntity is complex entity which contains set of pairs; each pair corresponds to image’s name and its place, so image and place ids should coincide to names and places which returned by GetImageList and GetImagePlaces

**Fringe ‘UserAction’**

For fringe’s object ‘UserAction’ following methods should be declared with following signature:

protected EWEntity SelectImage() throws Exception;

here EWEntity is simple entity which contains name of user selected image

protected EWEntity SelectPlace() throws Exception;

here EWEntity is simple entity which contains name of place where image should be placed

All methods of given fringe object must be implemented in blocked mode.

**Fringe ‘GameNotify’**

For fringe’s object ‘GameNotify’ following methods should be declared with following signature:

protected void TurnResult(EWEntity result) throws Exception;

here ‘result’ is complex entity which contain game step’s result and error counter

protected void GameComplete(EWEntity errors) throws Exception;

here ‘errors’ simple entity which contains number of errors; the errors are represented by entity’s Id