Micro Focus Enterprise Analyzer 3.5 Update 1

Using Architecture Modeler

2014-12-11

Contents

Introducing Architecture Modeler

Use Architecture Modeler to add support for unsupported languages and frameworks to Enterprise Analyzer (EA). You can also use it to extend support for already supported languages and frameworks. Architecture Modeler's:

- Graphical user interface, with error checking and metamodel lookup, makes it easy to define new entity and relationship types in EA.
- Regular expression generator simplifies the task of specifying the search patterns the EA parser uses to extract entities and relationships from source code.

Based on your input,Architecture Modeler creates a plug-in that defines an extension, or *add-on*, to the EA metamodel. The add-on specifies the entity and relationship types of interest in the newly supported source files.

The plug-in also references a *universal parser* configuration file that defines the search patterns EA uses to extract entities and relationships from newly supported files. For text files, search patterns are mapped from regular expressions. For XML files, search patterns are mapped from XPath queries.

Here in schematic form are the tasks you perform in Architecture Modeler:

1. Load an existing metamodel.
2. Save the metamodel with the name of your extension.
3. Add a sample source file of the type you want to support in EA.
4. Define the entity types for the add-on.
5. Define the relationship types for the add-on.
6. Map search patterns for the entity and relationship instances you want the parser to extract.
7. Export the model to EA.
8. Reconfigure EA.

Before turning to these tasks, users who are new to EA may want to familiarize themselves with the concepts underlying the EA metamodel. Experienced users can skip straight to the tasks.

Note: Architecture Modeler currently supports the level-1, or *application-level*, metamodel only. Support for the level-2, or *object-level*, metamodel (Interactive Analysis) will be available in a future release.

Entity extension mode is currently not available for the following entities: EZT, CA7, JSP, JSX, NetronSPC, NetronSpec, TLD, TS, and WebConfig.

Important: If you are using Windows 7 or Windows 8, make sure you are running the Architecture Modeler with administrator rights. Otherwise the some operations might fail.

Opening Architecture Modeler

1. Open the Enterprise Analyzer installation directory
2. Double-click `\bin\Architecture Modeler.exe`. The Architecture Modeler window opens.

   Note:
   - Choose **Tools > Confirmations** to specify whether you want to be prompted to confirm deletion of attributes, entities, relationships, or source files.
   - Choose **Tools > Samples view font** to specify the font in which source file samples are displayed.

   Important: If you are using Windows 7 or Windows 8, make sure you are running the Architecture Modeler with administrator rights. Otherwise the some operations might fail.

Understanding the Application-Level Metamodel

The *object model* for an application defines the relationships between the objects that comprise the application. These can be physical objects, like program source files or JCLs, or logical objects that identify abstract program elements: entry points, data stores, screens, jobs, and so forth.

The *relationships* between objects describe the ways in which they interact. In the figure below, the source file GSS.CBL *defines* the GSS program. The program, in turn, *reads* the data file GSS.FCSTSEC.

```
Cobol File
  GSS.CBL
    Defines Program
      GSS
        Reads File
          GSS.FCSTSEC
```
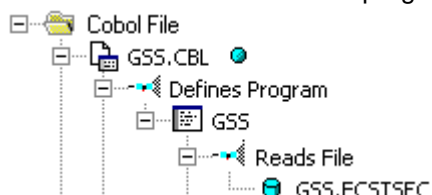
Relationships are conceived as having a left and right end. In the relationship CobolDefinesMap, for example, the left relationship end is DefinesMap, while the right relationship end is IsDefinedInCobol.

Each object or relationship the parser generates is modeled as an entity or relationship in an *Entity Relationship Diagram (ERD)*. Each entity or relationship is an instance of an entity or relationship *type*. The *application-level metamodel* defines these types.

Entity Type Properties

The properties of an entity type define its characteristics: its internal and external names, icon in Enterprise Analyzer, flags, and so on. The table below shows the properties of an entity type.

Note: Only properties editable in Architecture Modeler are shown in the table.

Property	Description
Name	The internal name of the entity type: COBOL, for example.
Description	The display name of the entity type: Cobol File, for example.
Source Name	The name of the entity attribute that holds "reference-resolving" information for an entity: Source, for example. See the subtopics below for more information.
Display Name	The name of the entity attribute that holds the display name for the entity. A Java File's display name derives from the value of its ClassName attribute, for example.
Flags	Flags that define the nature of the entity type: LEGACY, for example. See the subtopics below for more information.
Registration	The name of the class used to perform registration, unregistration, and invalidation tasks. Specify Default, unless the entity type should not have a folder in the project, in which case, specify NoRegistration.
Source Type	The type of "reference-resolving" information for the entity: File, for example. See the subtopics below for more information.
Icon	The name of the file that contains the entity type's icon for display: COBOL.gif, for example.

Source Name Property

The Source Name property of an entity type defines the name of the entity attribute that holds "reference-resolving" information for an entity. This attribute name is not the same for all entity types.

Consider a copybook. If a Cobol source file references a copybook, the parser creates a copybook object with the name referenced in the source file. And it does so whether or not the copybook actually exists in the EA workspace.

Later, when the system resolves references to the copybook, it looks in the Source attribute of the copybook for its location in the workspace file system. If the attribute is empty, the system flags the copybook as missing. The system knows it should look in the Source attribute because the Source Name property of the copybook entity type is set to "Source."

For a program entry point, by contrast, the sourcename property is set to "HCID," meaning that the system looks in the HCID (HyperCode ID) attribute when it attempts to resolve a CALL statement that references the entry point. An empty HCID attribute indicates that the called program does not exist in the workspace or has not been parsed.

Source Type Property

The Source Type property of an entity type defines the type of reference-resolving information for the entity: a file name in the case of a copybook, for example, a record in a table in the case of an entry point. The Source Type also determines whether EA deletes the reference-resolving item when the corresponding object is deleted from the workspace. The table below shows the types.

Type	Description
File	A source file: a copybook, for example. If the object is deleted, the source file is deleted.
Container	A file containing a list of source files: a TARGETMODEL, for example. If the object is deleted, the list file is deleted.
Folder	The folder for the Folder entity, which identifies an EA project. If the project is deleted, the folder is deleted.
Global	An item in a global file: a record in a table, for example. If the object is deleted, the item is not deleted.

Entity Flags

The Flags property of an entity type defines the nature of the entity: whether it is a physical source object or a logical object extracted from a source object, whether it defines a program, and so forth. An entity type can have multiple flags. A Cobol source file, for example, has LEGACY, PROGRAMCODE, and SEED flags. The table below shows the flags for an entity type.

Note: Only flags selectable in Architecture Modeler are shown in the table.

Flag	Description
COMPOSITE	n/a
EXTRACT	An entity extracted from a seed entity. A program, for example, is extracted from a Cobol file. An extract entity must be unique in the repository.
GENERATED	An entity generated in the transformation process, a TargetXML file, for example.

Flag	Description
LEGACY	A source file that can be loaded in the repository, a Cobol file, for example.
KNOWLEDGE	A business function or logical component.
PROGRAMCODE	An entity that defines a program, a Cobol file, for example.
SEED	An entity from which another entity can be extracted. A Cobol file is the seed for a program, for example.
SYSTEM	An entity referenced in the application but not loaded in the repository, a data store, for example.

Entity Type Attributes

The attributes of an entity type define the characteristics of the entity instance: its name, the name of the source file from which it derives, its complexity metrics, and so forth. The default attributes and attribute values supplied by Architecture Modeler are sufficient for most entity types. You can add or delete attributes and edit their values as necessary.

Note: You can specify additional default attributes in the file \bin\ArchitectureModeler.exe.config.

The properties of an attribute define its characteristics: its name, data type, whether the attribute value persists after invalidation, and so on. The table below shows the properties of an attribute.

Property	Description
Name	The internal name of the attribute: Name, for example.
Description	The display name of the attribute: Name, for example.
Type	For attributes other than Name, the data type of the attribute: CHAR, for example.
Size	For CHAR type attributes, the length of the attribute.
DefaultValue	Whether the attribute should be displayed with a value of "N/A" until it receives a value at verification: 0 for true, -1 for false.
Persistent	For an attribute of a LEGACY object, whether its value persists after the object is invalidated.

Relationship Type Properties

The properties of a relationship type define its characteristics: its name, flags, cardinalities, and so on. The table below shows the properties of a relationship type.

Property	Description
Name	The full relationship name: CobolDefinesMap, for example.
Flags	Flags that define the nature of the relationship type: PRODUCE, for example. See the subtopics below for more information.
Left Entity	The internal name of the entity on the left relationship end, COBOL, for example.
Left End Name	The internal name of the left relationship end: DefinesMap, for example.
Left End Description	The display name of the left relationship end: Defines Screen, for example.

Property	Description
Left End Cardinality	The cardinality of the left relationship, 1 or M. 1 means that the left entity can have one such relationship; M means that the left entity can have multiple such relationships. A Cobol file can define only one map, for example.
Right Entity	The internal name of the entity on the right relationship end, MAP, for example.
Right End Name	The internal name of the right relationship end: IsDefinedInCobol, for example.
Right End Description	The display name of the right relationship end: Is Defined In Cobol File, for example.
Right End Cardinality	The cardinality of the right relationship, 1 or M. 1 means that the right entity can have one such relationship; M means that the right entity can have multiple such relationships. A map can be defined in multiple Cobol files, for example.

Relationship Flags

The Flags property of a relationship type defines the nature of the relationship: whether the relationship is one between a seed and an extract entity, for example, or between extract entities or legacy entities. A relationship can have multiple flags. The FolderIncludesDbSchema relationship, for example, has the flags GROUPING and PRODUCES.

The relationship flag also determines how the product behaves when one of the entities in the relationship is invalidated, modified, or deleted. In a relationship between a seed and an extract entity, for example, deleting the seed requires deleting the extract. The table below shows the flags for a relationship type.

Flag	Description
GROUPING	A relationship between a folder entity, which identifies an EA project, and a legacy entity, FolderIncludesCobol, for example.
GENERATES	A relationship between a legacy entity and a generated entity, CopybookGeneratesTargetXml, for example. If the legacy entity is modified or deleted, the generated entity is deleted.
LINKS	Future use.
PRODUCES	A relationship between a seed and an extract entity, CobolDefinesProgram, for example. If the seed entity is modified or deleted, the extract entity is deleted.
REFERS	A relationship between extract entities, ProgramCallsProgramEntry, for example. If the left entity is invalidated or deleted, the right entity is deleted.
USES	A relationship between legacy entities, CobolIncludesCopybook, for example. If the right entity is modified or deleted, the left entity is invalidated, and every entity with a relationship of type PRODUCES with the left entity is deleted.

Defining an Extension with Architecture Modeler

Perform the following tasks to define a metamodel extension with Architecture Modeler:

1. Load an existing metamodel.
2. Save the metamodel with the name of your extension.
3. Add a sample source file of the type you want to support in EA.
4. Define the entity types for the add-on.
5. Define the relationship types for the add-on.
6. Map search patterns for the entity and relationship instances you want the parser to extract.
7. Export the model to EA.
8. Reconfigure EA.

Before turning to these tasks, users who are new to EA may want to familiarize themselves with the concepts underlying the EA metamodel. Experienced users can skip straight to the tasks.

Loading a Metamodel

You can load an existing metamodel to use as a shell for your extension, or create a new metamodel. Loading an existing metamodel is probably a better choice, because it ensures against duplicating existing entity and relationship names.

- To load an existing metamodel, choose **Model > Open**. An Open dialog is displayed, where you can browse for `\Model\Repository\Common.Repstry.xml` in the Enterprise Analyzer installation directory. Click **Open** to load the metamodel.
- To create a new metamodel, choose **Model > New > Model**. Nothing is displayed in the **Architecture Modeler** window until you add files, entities, or relationships.

Saving the Extended Metamodel

Whether you load an existing metamodel or create a new one, you must save the extended metamodel to preserve your changes. To save an extended metamodel, choose **Model > Save**. A Save As dialog opens, where you can save the extension with a new or an existing file name. Click **Save** to return to the Architecture Modeler window.

Adding a Sample Source File

When you add a sample source file to Architecture Modeler, you define the entity type for the source file in the extended metamodel, and make available the code you will use to generate regular expressions. To add support for Objective C, for example, you would define an entity type that represents an Objective C source file, then add one or more Objective C source files to Architecture Modeler.

Choose a sample that contains the kinds of entities and relationships you want the parser to extract. If you want the parser to model calls to Objective C functions, make sure your sample contains a function call!

> **Note:** Architecture Modeler supplies default values for the properties and attributes of the source file type you are adding. Except for the value of the Description property, which holds the entity type

name displayed to the end user, the default values usually are sufficient. You can change the Description (and any other property or attribute values) when you edit the source file type.

1. Click the **Files** button in Architecture Modeler and choose **Add New** in the drop-down menu. The Add New File window opens.

2. In the **Name** field, enter the name of the entity type for the source file. For an Objective C source file, you might choose the name "OBJC" for the entity type.

   Note: If you are extending support for an already supported source file, the name you specify cannot be the same as the name of the already supported source file. No entity type is created for the specified name. The extension is simply a mechanism for naming the universal parser configuration file that contains the search patterns with which you are extending support.

3. In the **Type** drop-down, select the type of the source file you are adding, XML or Text. If you select Text, click the **Format** button to view formatting options.

   Note: Think of Text as any kind of file other than XML.

4. Select **as an extension to** if you are extending support for an already supported source file, then choose the source file type in the adjacent combo box.

   Note: The Metrics and Name Pattern areas are disabled in entity extension mode.

5. In the Metrics group box, select each metric you want EA to calculate for the source file.

6. In the Source samples group box, click **Add new**. An Open dialog is displayed, where you can browse for the sample source file. Click **Open** to add the file. Repeat this procedure for each sample you want to add. To delete a file, select it and click **Delete**.

7. In the Name Patterns group box, click **Add new**. An editable text field is displayed. Click inside the field and enter the extension, including the dot (.), for the source file type. You can use wildcard patterns allowed in LIKE statements by Visual Basic for Applications (VBA). Click outside the field to add the extension. Repeat this procedure for each extension you want to add. To delete an extension, select it and click **Delete**.

8. When you are satisfied with your choices, click **OK**.

Architecture Modeler displays the new source file type in the Name area in the central pane of the window, and the text of the sample source file in the righthand pane. If more than one source file type is listed, click the name of the type to display the corresponding sample.

   Note: Select a source file type and click **Files > Edit** to modify its file characteristics. Select a source file type and click **Files > Delete** to delete the type.

Specifying Formatting Options for Text Files

Comments, strings, and fixed formats may create "noise" in parser extractions. The more specific you are in defining how you want the parser to handle these and similar text file characteristics, the cleaner your results will be.

1. To specify formatting options for a text file, click the **Format** button in the Add New File or Edit File window. The Input Source Format window opens.

2. In the Input Source Format window, click the button ⊞ to sort the list by category, click the ↓ button to sort the list alphabetically.

3. Edit the formatting options as necessary:

   • For Boolean values, choose from the drop-down list inside the value field.
   • For string values, enter the text of the string in the value field.
   • For the StringLiteralSymbol option, click the browse button in the value field. The Char Collection Editor window opens. Click the **Add** button to add a symbol for the start of a string literal. Architecture Modeler displays the new collection member in the lefthand pane. Select the member,

then enter its value in the righthand pane. Repeat this procedure for each symbol you want to define.

To delete a collection member, select it and click **Remove**. Click the button ⊞ to sort the list by category, click the 🔤 button to sort the list alphabetically. Click **OK** to confirm your changes.

The table below shows the formatting options in alphabetical order. Option values are modified automatically depending on your choices in other fields.

Option	Description
EndingColumn	The last valid column of the source.
HasEndingColumn	Whether the source has an ending column.
HasLiteralContinuationSymbol	Whether the source has a literal continuation symbol.
HasMultiLineComment	Whether the source has multiline comments.
HasSingleLineComment	Whether the source has single line comments.
HasSingleLineCommentContinuation	Whether the source has single line comment continuations.
HasStartingColumn	Whether the source has a starting column.
HasStringLiteralSymbol	Whether the source has symbols used to start string literals.
IgnoreColumns	Whether to ignore column information before the starting column and after the ending column.
IgnoreComments	Whether to ignore comments.
IgnoreStringLiterals	Whether to ignore string literals.
MultiLineCommentEnd	Symbol for the end of a multiline comment.
MultiLineCommentStart	Symbol for the start of a multiline comment.
SingleLineCommentContinuationSymbol	Symbol for the continuation of a single line comment.
SingleLineCommentStart	Symbol for the start of a single line comment.
StartingColumn	The first valid column of the source.
StringLiteralContinuationSymbol	Symbol for the continuation of a string literal.
StringLiteralSymbol	Symbols for the start of string literals.

4. When you are satisfied with your choices in the Input Source Format window, click **OK**.

Defining Entity Types

Each object the parser generates is an instance of an entity *type*. You need to define these types in the extended metamodel. If you are adding support for Objective C, for example, you would define an entity type that represents an Objective C function.

For each entity type, you specify two kinds of information:

- The *properties* of the entity type define its characteristics: its internal and external names, icon in Enterprise Analyzer, flags, and so on.
- The *attributes* of the entity type define the characteristics of the entity instance: its name, the name of the source file from which it derives, its complexity metrics, and so forth. The default attributes and attribute values supplied by Architecture Modeler are sufficient for most entity types. You can add or delete attributes and edit their values as necessary.

1. Click the **Entities** button in Architecture Modeler and choose **Add New** in the drop-down menu. The properties definition page of the Add New Entity window opens.

2. On the properties definition page, fill in the properties of the entity type. Architecture Modeler flags any errors with a ⚠ symbol.

   ✎ **Note:** Do not enter spaces in the Name property.

3. When you are satisfied with your entries on the properties definition page, click the ➡ button. The attributes definition page opens.

4. On the attributes definition page, add, edit, or delete attributes as necessary:

   • To add an attribute, click the **Add New** button. The Add New Attribute dialog opens, where you can specify the values for each property of the attribute. Architecture Modeler flags any errors with a ⚠ symbol. When you are satisfied with your entries, click **OK**.

   • To edit an attribute, click the **Edit** button. The Edit Attribute dialog opens, where you can specify the values for each property of the attribute. Architecture Modeler flags any errors with a ⚠ symbol. When you are satisfied with your entries, click **OK**.

   • To delete an attribute, click the **Delete** button.

5. When you are satisfied with your entries on the attributes definition page, click the ✔ button. Architecture Modeler flags any errors with a ⚠ symbol.

   ✎ **Note:** The errors may not be immediately visible. If you are on the attributes definition page, errors may be flagged on the properties definition page, and vice versa.

Assuming there are no errors in your entries, Architecture Modeler saves the entity type definition, and displays the entity type in the list of entity types in the righthand pane of the window.

✎ **Note:** Select an entity type and click **Entities > Edit** to modify its definition. Select an entity type and click **Entities > Delete** to delete the definition.

Defining Relationship Types

Each relationship the parser generates is an instance of a relationship *type*. You need to define these types in the extended metamodel. If you are adding support for Objective C, for example, you would define a relationship type that represents the relationship between an Objective C source file and an Objective C function.

1. Click the **Relations** button in Architecture Modeler and choose **Add New** in the drop-down menu. The Add New Relation window opens.

2. In the Add New Relation window, fill in the properties of the relationship type. Architecture Modeler flags any errors with a ⚠ symbol.

   ✎ **Note:** Do not enter spaces in the Name property.

3. When you are satisfied with your entries, click **OK**.

Assuming there are no errors in your entries, Architecture Modeler saves the relationship type definition, and displays the relationship type in the list of relationship types in the righthand pane of the window.

✎ **Note:** Select a relationship type and click **Relations > Edit** to modify its definition. Select a relationship type and click **Relations > Delete** to delete the definition.

Architecture Modeler Internal Language Functions Description

string.substr(pos1,pos2)

`string.substr(pos1,pos2)` – returns a substring, derived from `pos1` to `pos2` of `string`.

Examples:
```
expressionid.0.substr ( 5, 4 )
expressionid.0.substr(expressionid.0.length() - 5, 4 )
expressionid.0.substr( 4 , expressionid.0.length() - 5 )
expressionid.0.substr(expressionid.0.length() - 9 , expressionid.0.length() -
5 )
expressionid.sourcefilename.substr(1,5)
expressionid.sourcefilename.substr(expressionid.0.sourcefilename.length() -
11 , 3 )
expressionid.sourcefilename.substr( 0, expressionid.0.sourcefilename.length()
- 4 )
expressionid.sourcefilename.substr(expressionid.0.sourcefilename.length() –
15 )
```

string1.append(string2)

`string1.append(string2)` - appends `string2` to `string1`.

Examples:
```
expressionid.0.append( \"._abcd_\" )
expressionid.0.append(expressionid.0 )
expressionid.0.append(expressionid.0.sourcefilename )
expressionid.sourcefilename.append( \"_001\" )
expressionid.sourcefilename.append(  expressionid.0  )
```

string.length()

`string.length()` - returns the length of the `string`.

Examples:
```
expressionid.sourcefilename.length()
expressionid.sourcefilename.length()-3
expressionid.sourcefilename.length()+13
expressionid.sourcefilename.length( )
```

concat(string1, string2)

`concat(string1, string2)` – returns concatenation of `string1` and `string2`

Examples:
```
concat(\"ab_\", \"cd__\" )
concat( expressionid.0,expressionid.0)
concat( expressionid.0, \"_test_\" )
concat( \"_test_\" , expressionid.0 )
concat( expressionid.sourcefilename , expressionid.0 )
concat(expressionid.0, expressionid.sourcefilename )
concat(expressionid.sourcefilename.substr(0,
expressionid.sourcefilename.length()-3), expressionid.0)
concat(expressionid.sourcefilename.append( \"...\"), expressionid.0)
concat(expressionid.sourcefilename, \"_abc_\" )
concat( \"_abc_\" , expressionid.sourcefilename )
concat( \"_abc_\" , concat( \"_efg_\" , \"_xy_z_\") )
```

```
concat( concat( \"_efg_\" , \"_xy_z_\"), \"_abc_\" )
concat( concat( \"_efg_\" , concat(\"_val1_\", \"_val2_\")) , \"_abc_\" )
concat( concat(expressionid.sourcefilename.append( \"...\") , \"_val_\" ) ,
expressionid.0)
```

replace(string1,string2,string3)

replace(string1,string2,string3) – replaces all occurrences of string2 in string1 with string3.

Examples:

```
replace( \"ab_\", \"_\", \"__\" ) // equals "ab__"
replace( \"ab_\", \"ab\", \"cd\" ) // equals "cd_"
replace(expressionid.sourcefilename.substr(1,5), \"a\", \"b\" )
// replaces 'a' with 'b' in the first 5 letters of expressionid.sourcefilename
replace(expressionid.sourcefilename, '/' . '\' ) // converts backslashes to
slashes
```

getpath(string)

getpath(string) – returns a string, trimmed starting from the last occurrence of '\' or '/' . Returns the absolute path as string before the last occurrence of \ or /.

Examples:

```
getpath("c:\path1\path2\filename.ext") // returns c:\path1\path2\
getpath("..\path1\path2\filename.ext") // returns ..\path1\path2\
getpath(expressionid.sourcefilename) // returns the path of the sourcefilename
```

getfilename(string)

getfilename(string) – .returns the filenames in paths as a string after the last occurrence of the \ or /?

Examples:

```
getfilename ("c:\path1\path2\filename.ext") // returns filename.ext
getfilename ("..\path1\path2\filename.ext") // returns filename.ext
getfilename (expressionid.sourcefilename) // returns filename of the
sourcefilename
```

unifypath(string)

unifypath(string) – returns transformed string by the following rules:

- Converting all backslashes to slashes.

  Example:

  /path1/path2 is transformed into \path1\path2
- Removing duplicate slashes.

  Example:

  //path1/\path2 is transformed into \path1\path2
- Removing references to current directory

  Example:

  \.\path1/.\path2 is transformed into \path1\path2
- Removing references to parent directory (if not in the beginning) examples

  Examples:

  \path1\..\path2 is transformed into \path2

`..\path1\..\path2\path3` is transformed into `..\path2\path3`

Mapping Regular Expressions to Text File Searches

The Enterprise Analyzer parser extracts entity and relationship instances from text files using search patterns you define in Architecture Modeler. In schematic form, the parser uses these patterns to:

- Match relationship instances in the code.
- Form the names of entity instances.
- Form the names of relationship instances.

Architecture Modeler's regular expression generator simplifies the task of defining these patterns. Follow the steps below to map regular expressions to text file searches.

1. Click **Files** in Architecture Modeler. Architecture Modeler displays the defined source file types in the Name area in the central pane of the window, and the text of the corresponding sample in the right-hand pane. If more than one source file type is listed, click the name of the type to display the sample.

2. Generate the pattern the parser uses to match relationship instances:
   a) Select the code of interest in the sample source file.
   b) Choose **Pick** in the right-click menu.
   The Add Expression window opens.

3. Add an expression ID, e.g. "function" for the selected function call. All expression IDs appear in the middle-bottom left-hand side of the screen in a tree form.

4. Select the expression ID and choose **Edit** in the right-click menu. The Edit Regular Expression Mapping window opens. Expressions are represented in a hierarchical manner. Each expression contains its regular expression pattern and all direct children expressions.

5. Select the expression pattern and choose **Edit** in the right-click menu. The Edit Regular Expression window opens. The generated regular expression is displayed in the upper part of the left-hand pane in tree form. The expression does not require further editing.

   > **Note:** Do not manually edit the generated regular expression unless you have modified it inadvertently.

6. In the left-hand pane of the Edit Regular Expression window, define the regular expression the parser uses to form the names of entity instances:
   a) Expand the tree for the generated regular expression.
   b) Use a subexpression of the generated regular expression to match the names of entity instances. Follow the instructions in *Editing Subexpressions* to edit the subexpression.
   c) Click **Apply Changes** after each modification.

7. In the right-hand pane of the Edit Regular Expression Mapping window, map the regular expression for the names of entity instances. You do not need to map a regular expression for the source file name:
   a) In the Entity group box, select the entity type you want to match in the Type drop-down.
   b) Drag-and-drop the subexpression for the names of entity instances from the expression tree to the cell beside the Name attribute. The subexpression is mapped to a variable with a name reflecting the order of grouped expressions in the expression tree, counting from the regular expression for the matched relationship. For example if (\w+) is the only other grouped expression besides the expression for the matched relationship, the mapped variable is named `%[ExpressionID].2%`.
   c) Assuming you are defining a relationship between a source file and an extracted object, enter the system variable `%[ExpressionID].sourcefilename` in the cell beside the Source attribute. If you are defining another type of relationship, reference-resolving information may be contained in a different attribute.
   d) You can use functions to help you resolve entities' relationship ends. These functions are: `substr`, `append`, `length`, `concat`, `replace`, `getpath`, `getfilename`, `unifypath`. For more details on the usage of these functions, read *Application Modeler Internal Language Functions Description*.

8. Click **Save** in the Entity group box. The entity mapping is added to the list of mappings in the Mapping List group box. Repeat this procedure for each entity mapping you want to define.

9. In the right-hand pane of the Edit Regular Expression Mapping window, map the regular expression for the names of relationship instances:

   a) In the Relation group box, select the relationship type you want to map in the Type drop-down.

   b) Drag-and-drop the subexpression for the left entity name from the expression tree to the cell beside the Left Entity Name attribute. If you are defining a relationship between a source file and an extracted object, enter the system variable `%[expressionid].sourcefilename%` in the cell beside the Left Entity Name attribute.

   c) Drag-and-drop the subexpression for the right entity name from the expression tree to the cell beside the Right Entity Name attribute.

10. Click **Save** in the Relation group box. The relationship mapping is added to the list of mappings in the Mapping List group box.

   Note: To edit a mapping definition, select it in the Mapping List group box and click **Edit**. Follow the procedure you used to define the mapping. To delete a mapping definition, select it in the Mapping List group box and click **Remove**.

11. When you finish editing your entries in the Edit Regular Expression Mapping window, click **Save**. Architecture Modeler displays the edited expression ID in the central pane of the window.

   Note:

   • Select the expression ID and choose **Edit** in the right-click menu to modify the pattern.
   • Select it and choose **Delete** in the right-click menu to delete the pattern.

Editing Subexpressions

Editing subexpressions consists of two tasks:

• *Grouping and quantifying* subexpressions.
• *Providing Scope for recursive regular expressions.*

The subexpression for the entity you want to match must either be grouped or an exact match. Any other input is not accepted.

1. In the Edit Regular Expression dialog, edit the expression manually, or use the buttons to modify te expression.

   a) Click **Convert to Exact match** to match the Item Value to the Matched Value sample only.

   b) Click **Convert to Group** if you want to group the expression.

   c) Click **Revert Changes** if you want to cancel any last changes or **Apply Changes** to save them.

   Note: The subexpression for the entity you want to match must either be grouped or an exact match. Any other input is not accepted.

2. If you need to qualify the expression further, edit the regular expression in the **Item Value** field considering the following:

   • Plus equates to the quantifier +, which denotes one or more occurrence.
   • Star equates to the quantifier *, which denotes 0 or more occurrences.
   • Optional equates to the quantifier ?, which denotes 0 or 1 occurrence.

3. Click **Apply Changes** after each modification.

4. Click **OK** when you are done editing your entries in Edit Regular Expression dialog.

Architecture Modeler displays the edited subexpression in the expression tree.
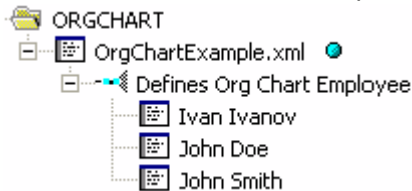
Mapping XPath Queries to XML File Searches

The EA parser extracts entity and relationship instances from XML using search patterns you define in Architecture Modeler. In schematic form, the parser uses these patterns to:

- Match relationship instances in the code.
- Form the names of entity instances:

- Form the names of relationship instances:

Architecture Modeler's XPath query generator simplifies the task of defining these patterns. Follow the steps below to map XPath queries to XML file searches.

1. Click the **Files** button in Architecture Modeler. Architecture Modeler displays the defined source file types in the Name area in the central pane of the window, and the text of the corresponding sample in the right-hand pane. If more than one source file type is listed, click the name of the type to display the sample.

2. Define the XPath query the parser uses to match relationship instances:

    - Select the node of interest in the sample, then use the right-click menu to form the XPath query. To match the relationship OrgChartDefinesOrgChartEmployee in the XML code shown below, the XPath query would be:

    ```
    /def:OrgChart/def:Employees/child::def:Employee
    ```

    To form the query, select the `<Employees>` node and choose **Select All <Employees> children elements > /child::def:Employee** from the right-click menu.

    - The XPath query is displayed in the **XPath Query** field. When you are satisfied with the query, click **Pick**.
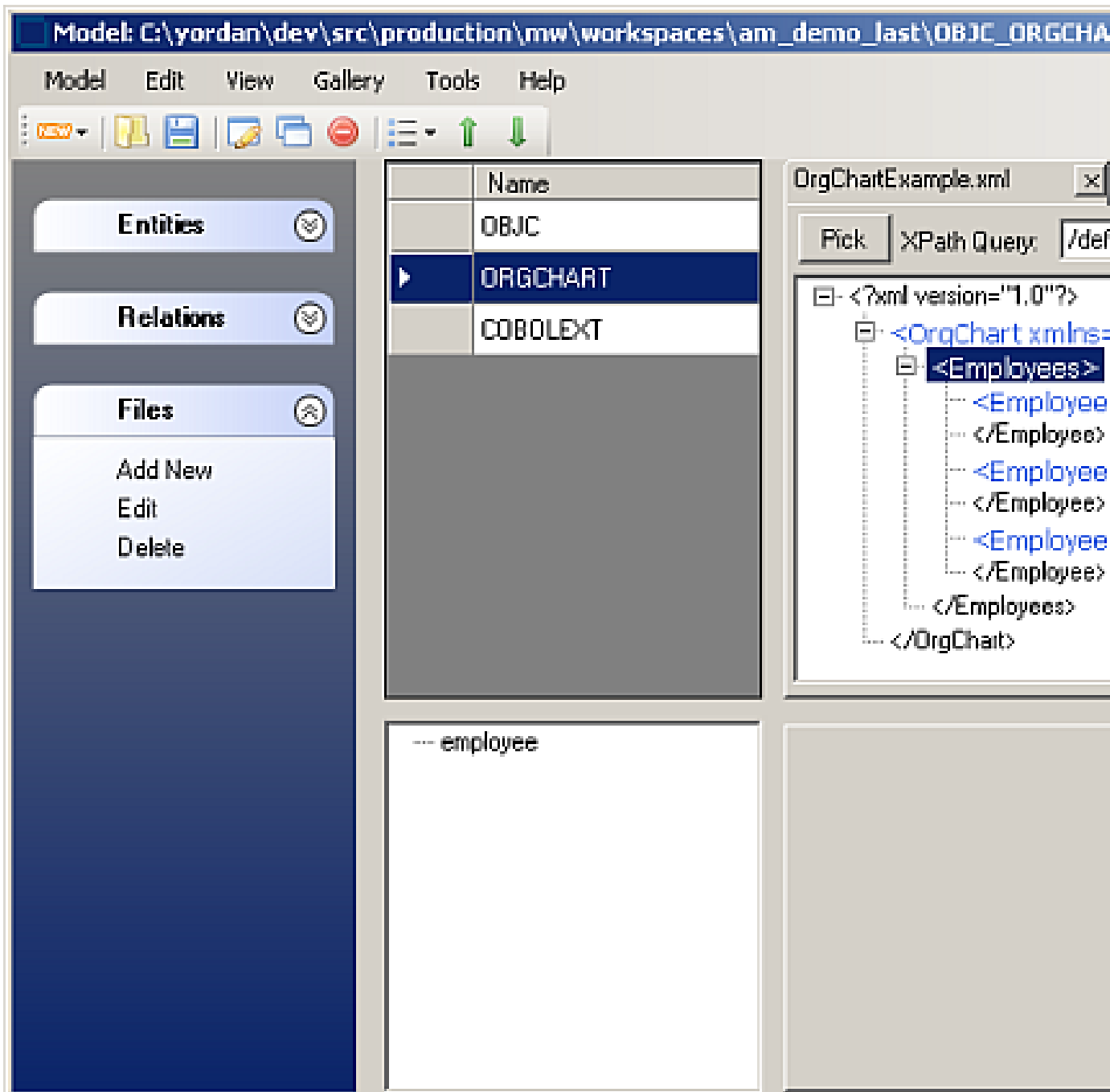
The Add Expression window opens.

3. Enter an expression ID, e.g. "employee" for the selected Employees xml element. All expression IDs are displayed in a tree on the middle-bottom left-hand side of the screen in.

4. Select the expression ID and choose **Edit** from the right-click menu. The Edit XPath Mapping window opens. The XPath Query for the matched relationship is displayed in the **XPath Main Query** field.

5. In the right-hand pane of the Edit XPath Mapping window, define the XPath query the parser uses to form the names of entity instances. You do not need to define a query for the source file name:

   • In the Entity group box, select the entity type you want to match in the **Type** drop-down, ORGCHARTEMPLOYEE in our example.
   • Use a subexpression of the XPath query you created in step 2 to match an attribute of the node. If you want to form the entity name from the `Name` attribute, enter `%./@name%` in the cell beside the Name attribute. That subexpression matches the employees "John Doe," "Ivan Ivanov," and "John

Smith" in our example. If you are matching a different attribute, `title`, for example, enter `%./`
`@title%` in the cell next to the `Name` attribute.

- Assuming you are defining a relationship between a source file and an extracted object, enter the system variable `%[ExpressionID].sourcefilename%` in the cell beside the Source attribute. If you are defining another type of relationship, reference-resolving information may be contained in a different attribute.

6. Click **Save** in the Entity group box. The entity mapping is added to the list of mappings in the Mapping List group box. Repeat this procedure for each entity mapping you want to define. The figure below shows the entity mapping for our example.
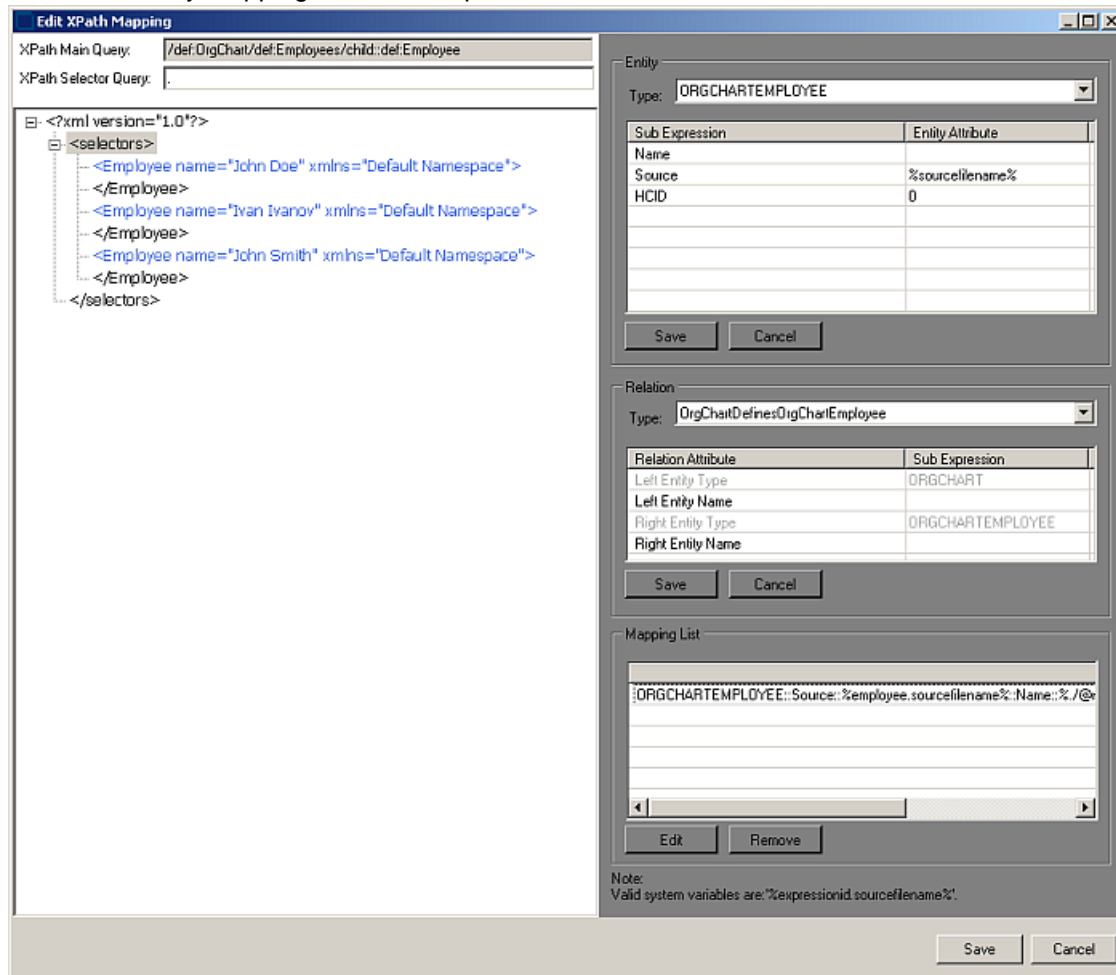
7. In the right-hand pane of the Edit XPath Mapping window, define the XPath query the parser uses to form the names of relationship instances:

- From the **Type** drop-down in the Relation group box, select the relationship type you want to map, OrgChartDefinesOrgChartEmployee in our example.
- Map the subexpression for the left entity name. If you formed the entity name from the `name` attribute of the node, enter `%./@name%` in the cell next to the Left Entity Name attribute. If you are defining a relationship between a source file and an extracted object, enter the system variable `%[ExpressionID].sourcefilename%` in the cell next to the Left Entity Name attribute.
- Map the subexpression for the right entity name. If you formed the entity name from the `name` attribute of the node, enter `%./@name%` in the cell beside the Right Entity Name attribute.

8. Click **Save** in the Relation group box. The relationship mapping is added to the list of mappings in the Mapping List group box. The figure below shows the relationship mapping for our example.

Note: To edit a mapping definition, select it in the Mapping List group box and click **Edit**. Follow the procedure you used to define the mapping. To delete a mapping definition, select it in the Mapping List group box and click **Remove**.

Edit XPath Mapping

XPath Main Query: `/def:OrgChart/def:Employees/child::def:Employee`

XPath Selector Query: `.`

```
<?xml version="1.0"?>
  <selectors>
    <Employee name="John Doe" xmlns="Default Namespace">
    </Employee>
    <Employee name="Ivan Ivanov" xmlns="Default Namespace">
    </Employee>
    <Employee name="John Smith" xmlns='Default Namespace">
    </Employee>
  </selectors>
```

Entit

Typ

Su
Na
So
HC

Rela

Typ

Su
Le
Le
Ri
Ri

Map

OF
OF

◄

Note:
Valid s

9. When you are satisfied with your entries in the Edit XPath Mapping window, click **Save**.

Architecture Modeler displays the search pattern for the matched relationship in the Match pattern area in the central pane of the window.

> **Note:** Select the search pattern for the matched relationship and choose **Edit** in the right-click menu to modify the pattern. Select it and choose **Delete** from the right-click menu to delete the pattern.

Exporting the Extended Metamodel

Based on your input, Architecture Modeler creates a plug-in that defines an extension, or *add-on*, to the EA metamodel. The add-on specifies the entity and relationship types of interest in the newly supported source files.

To export the plug-in to EA:

1. Select the source file type for the extension in the **Name** area in the central pane of the Architecture Modeler window.

2. Choose **Export to EA** in the right-click menu.

A Browse dialog opens.

3. Browse for the Plugins folder in the Enterprise Analyzer installation directory.

4. Click **OK** to export the extended metamodel. The plug-in has a name of the form `<SourceFileType>Plugin.xml`.

> **Important:** If you are using Windows 7 or Windows 8, make sure you are running the Architecture Modeler with administrator rights. Otherwise the some operations might fail.

Reconfiguring Enterprise Analyzer

EA After exporting an extended metamodel to Enterprise Analyzer, you need to reconfigure EA for the new source file type. To reconfigure EA after plug-in export,

1. Double-click \bin\rescan.exe in the Enterprise Analyzer installation directory.

The Configuration Manager window opens, with a check box in the Programming Languages folder for the new extension.

2. Select the check box and click **OK**.

Make sure to upgrade the configuration of any workspaces you want to use the new extension in.

Troubleshooting the Extended Metamodel

The plug-in for an extended metamodel references a *universal parser* configuration file that defines the search patterns EA uses to extract entities and relationships from newly supported files. EA issues the following errors for problems it encounters processing the universal parser configuration file at verification:

- *Error - 404 3 Error while processing XML file '%1'* . Logged when the Universal Parser configuration file is corrupt or missing.
- *Error - 407 3 Error while processing regular expression '%1'.* Logged when a regular expression defined in the Universal Parser configuration file is invalid.
- *Error - 409 3 Input source file '%1' not found.* Logged when the file to be parsed is not found.
- *Error - 411 3 Error creating expression with Id '%1'.* Logged when an expression with a given Id failed to be created.
- *Error - 412 3 Expression Id '%1' already exists.* Logged when an expression Id is not unique.

- *Error - 413 3 Error in attribute value '%1': Unexisting expression Id*. Logged when a non existing expression Id is found in an attribute value.
- *Error - 414 3 Error in attribute value '%1': Match not found*. Logged when a not existing match index is found in an attribute value.
- *Error - 415 3 Error in attribute value '%1': Match group not found*. Logged when a not existing match group was detected in an attribute value.
- *Error - 416 3 Error in attribute value '%1': Syntax error*. Logged when an attribute value is not syntactically correct.
- *Error - 417 3 Error in attribute value '%1': Type mismatch*. Logged when an attribute value contains type mismatching.

Using Galleries

In Architecture Modeller you can export or import model extensions to a common location known as Gallery.

1. To export to Gallery:
   a) Select the extension you want to export and click **Gallery>Export**. The Export to Gallery window opens.
   b) Write a short description for the Gallery item and click **OK**. The Select gallery for export window opens.
   c) Browse to the current Gallery file location and click **OK** .

   The selected model extension has been successfully exported to the selected gallery.

2. To Import from Gallery:
   a) Click on **Gallery > Import** menu item. The Import from Gallery window appears.
   b) Select the short description for the model extension you want to import from the gallery and click **OK**.