Introduction

This document provides an architectural assessment of the SDK Eclipse IDE plugin. Readers of this document will gain an understanding of the components of the SDK plugin and their interaction through the model exploring and artifact generation process. Technology considerations are limited to the Java SDK platforms 1.5 and up, as they are supporters of the critical Java scripting support capabilities that the SDK Eclipse IDE plugin will depend on. Support for the JDK 1.4 environment is not required.

Principal Design Components

The SDK Eclipse IDE plugin (referred to from here on as the plugin) provides a development environment that fosters the generation of application artifacts from an Ecore model. The plugin helps developers convert application models created in a variety of meta-model representations into an Ecore model. From this ecore model, pluggable artifact generators can produce artifacts. The actual artifact production process is described in the accompanying SDK Core Generator Design document. In this document, the overall plugin interface is discussed. Of importance are the following components:

- Model Converter Manager
- Model Explorer
- Generator Manager

The Model Converter Manager provides a graphical user interface (GUI) for the management and use of model converters. Model converters are utility components that can convert a meta-model of a specific type to an Ecore model, or an Ecore model to a meta-model of a specific type. Model converters are designed to be pluggable in nature, using the same mechanisms used to facilitate pluggable generators in the SDK Core Generator Manager.

The Model Explorer supports the viewing of the individual meta modelling facets of a model. Using the Model Explorer, users may view the meaning, persistence, presentation, validation, security, and object representation aspects of a meta model.

The Generator Manager is used to execute pluggable generators against the Ecore model. the actual functionality of this component is described in the SDK Core Generator Design document. In this document, treatment is limited to how this component interfaces with the plugin.

Model Tagged Values

Outside of the meaning and persistence of a model, many meta models do not provide the ability to model presentation, validation, representation, and the security aspects of the model.

Indeed, the Ecore representation the SDK team will be using as an intermediate form does not provide this capability as a first class modeling feature. Consequently, we are relying on model annotations to express these other aspects. These annotations can be represented in other meta modelling languages as well. For instance, these annotations could be represented as object and data properties belonging to a class in OWL. In the Java language, the annotation construct provides a direct comparison to the EAnnotation in Ecore. In UML these annotations could be approximated using the tagged value construct.

The SDK team has come up with a string based representation format for model annotations. The following table indicates all the annotations that will be supported by the SDK application:

Annotation	Format	Description	Comments
#.*, #*	not applicable	Annotation comment	Use this to negate the use of an annotation. As part of the level 0 validations annotations that are not recognized will be identified as a warning by the core generator.
package.*	not applicable	Annotations of this form are expected to be applied to all classes, properties, operations, relationships and concepts of the package.	
package.class.*	not applicable	Annotations of this form are expected to be applied to all classes of the package.	
package.prop.*	not applicable	Annotations of this form are expected to be applied to all properties of the package.	
package.oper.*	not applicable	Annotations of this form are expected to be applied to all operations of the package.	

package.rel.*	not applicable	Annotations of this form are expected to be applied to all relationships of the package.	
class.*	not applicable	Annotations of this form are to be applied to this class.	
class.prop.*	not applicable	Annotations of this form are expected to be applied to all properties of the class.	
class.oper.*	not applicable	Annotations of this form are expected to be applied to all operations of the class.	
class.rel.*	not applicable	Annotations of this form are expected to be applied to all relationships of the class.	
prop.*	not applicable	Annotations of this form are to be applied to this property.	
oper.*	not applicable	Annotations of this form are to be applied to this operation.	
rel.*	not applicable	Annotations of this form are to be applied to this relationship.	
class.per.table.name	string	The name of the table corresponding to this class	
class.prop.per.immutable	boolean	Makes all properties belonging to this class immutable.	
prop.per.column.name	string	Then name	

		of the column corresponding to this property.	
prop.per.primary.key	number	Indicates this property to be a part of the primary key. The value determines the order of this property in a composite key.	
prop.per.immutable	boolean	Makes this property immutable.	
rel.per.foreign.key	number	Specifies that this relationship is a foreign key relationship. The value determines the order of this property in a composite key.	Paired with the property validation rule required can set the referential integrity rules for this relationship.
rel.per.join.table.name	string	Specifies that the name of the join table representing this relationship.	
rel.per.inh.single.table	string	Indicates the the single table idiom for representing the inheritance relationship should be used. The value is the name of the descriminator field.	
rel.per.inh.no.root.table	not applicable	Indicates the the no root table idiom for representing the inheritance relationship should be used.	
rel.per.inh.root.table	not applicable	Indicates the the root table idiom for representing the inheritance relationship should be used.	
class.mea.desc	string	Text describing this	

		class.	
prop.mea.desc	string	Text describing this property.	
oper.mea.desc	string	Text describing this operation.	
rel.mea.desc	string	Text describing this relationship.	
prop.val.min.length	number	Value representing the minimum length allowed for a property of type string.	
prop.val.max.length	number	Value representing the maximum length allowed for a property of type string.	
prop.val.format	regex	Regex expression that describes the format allowed for this string.	
prop.val.required	boolean	Indicates whether or not a null value is allowed for this property.	
prop.val.range	string	Indicates the lower and upper bound allowed for values of this property.	
prop.val.allowed.values	string	Indicates a finite list of values accepted by this property.	
class.pre.label	string	Indicates a label that should be used to identify this property during presentation.	
class.pre.plural.label	string	Indicates a label that should be used to identify more than one of this class during presentation.	

class.pre.field.set.label	string	Indicates the label that describes this class' properties as a single group.	
class.pre.button.label	string	Indicates the label that should be on the button for a form interface for this class.	
class.pre.style	string	Indicates a presentation style that should be applied to this class.	
class.prop.pre.readonly	boolean	Indicates that this class's properties are all readonly.	Not to be confused with property immutability, which merely indicates that property should never be changed once created.
class.prop.style	string	Indicates a name of the presentation style that should be applied to this property.	
prop.pre.label	string	Indicates the label that should be used to identify this property during presentation.	
prop.pre.plural.label	string	Indicates the label that should be used to identify many of these properties for presentation.	
prop.pre.display.order	number	Indicates the order these properties should be displayed during presentation.	
prop.pre.navigation.order	number	Indicates the order these properties should be navigated during presentation.	Usually implemented as a tab value.

~~prop.pre.display.length~~	~~number~~	~~Indicates how much~~ of this property's value should be displayed during presentation.	
prop.pre.input.type.text	not applicable	Indicates that the input mechanism of this property should support single line text input.	
prop.pre.input.type.text.area	not applicable	Indicates that the input mechanism of this property should support multiple line text input.	
prop.pre.input.type.radio	not applicable	Indicates that the input mechanism of this property should support single selection of an allowed value of this property.	
prop.pre.input.type.checkbox	not applicable	Indicates that the input mechanism of this property should support multiple selection of an allowed values of this property.	
prop.pre.input.type.select	not applicable		
prop.pre.input.type.multi.select	not applicable		
prop.pre.input.type.hidden	boolean	Indicates whether this property should be shown to the user. the property is still required to be in the form however.	
prop.pre.input.type.password	not applicable	Indicates this text input should include password protection measures.	

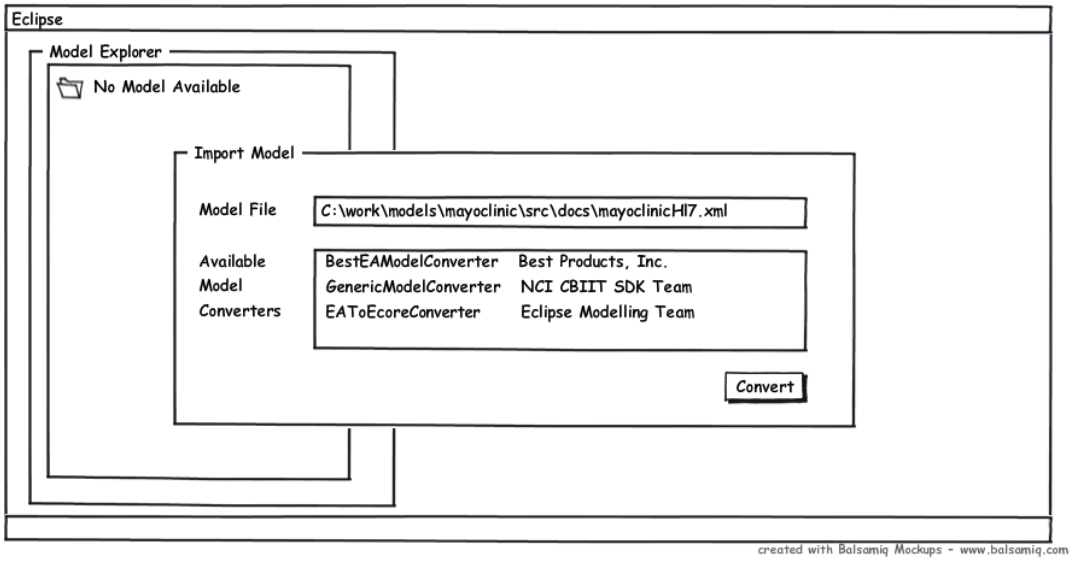
prop.pre.default.value	string	Sets the default value for a property	
prop.pre.readonly	boolean	This property cannot be changed via the presentation medium.	
prop.pre.style	string	This annotation sets the presentation styles for a property	
package.prop.sec.encrypt	boolean	This annotation specifies that the property should be encrypted when stored.	
package.oper.sec.role	string	This annotation indicates all the roles in the package that are allowed to execute this operation.	
class.prop.sec.encrypt	boolean	This annotation indicates that access to all the properties in this class should be secured in order to prevent input interception.	
class.oper.sec.role	string	This annotation indicates all the roles that are allowed to execute all the operations in this class.	
prop.sec.encrypt	boolean	This annotation indicates the the information in this property should be stored in an encrypted format.	
oper.sec.encrypt	boolean	This annotation indicates that the operation should be secured in order to prevent interception	

		of its input and returns.	
oper.sec.role	string	This annotation indicates the roles that are allowed to execute this operation.	
class.rep.rep	string	This annotation indicates the supported representation formats for objects of this class.	
prop.rep.date	string	This annotation indicates the supported representation format for a property of type date.	
prop.rep.currency	string	This annotation indicates the supported representation format for a property of type currency.	
prop.rep.unit.of.measure	string	This annotation indicates the supported representation format of a unit of measure associated with this property.	

Model Converter Manager Model Conversion Interface

The Model Converter Manager (MCM) model conversion interface will allow users to identify the list of model converters available for the conversion of meta-models from one format to the other. Users may choose to try to convert a model by using the file dialog to locate the serialized model on the file system. After choosing to open the model, the available model converters will be queried by the MCM, and asked whether or not they can convert this meta-model. Only those converters that can convert the meta model that was opened by the user will be displayed on the interface screen. If no model converters are identified, the user will be told that no model converters for this

conversion are available. The following mock gives a representation of how this screen will look to the user:

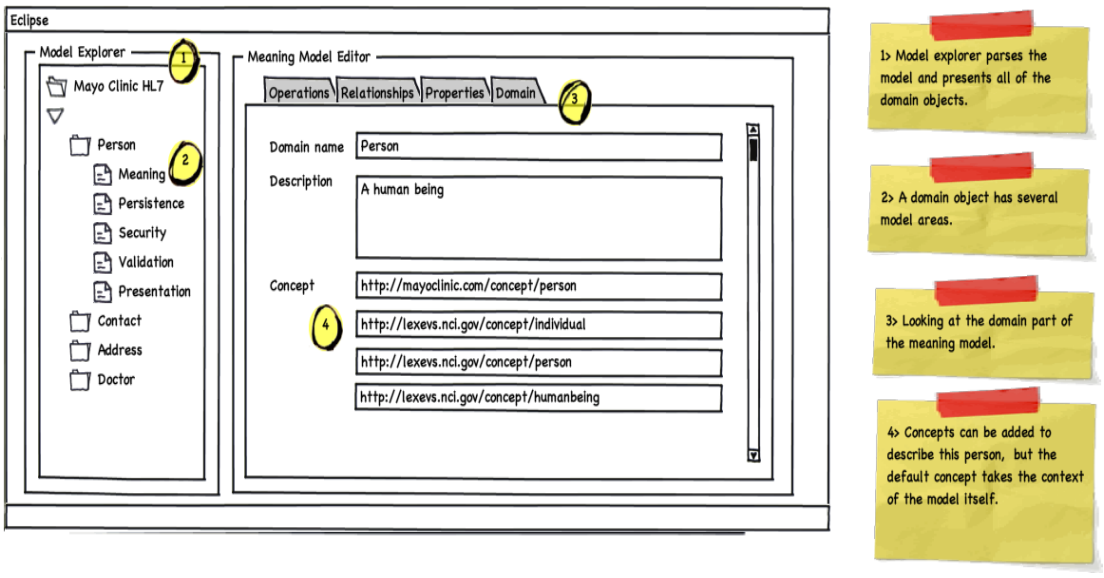


After the model converter is chosen, the user may press the convert button. Model conversion will then commence, with the model conversion progress reported to the Eclipse console. If the model conversion fails the user will be presented with the empty Model Explorer interface, and the Eclipse console will indicate the failure, along with the appropriate warning and error messages that can be used to diagnose the issues uncovered by the model converter.

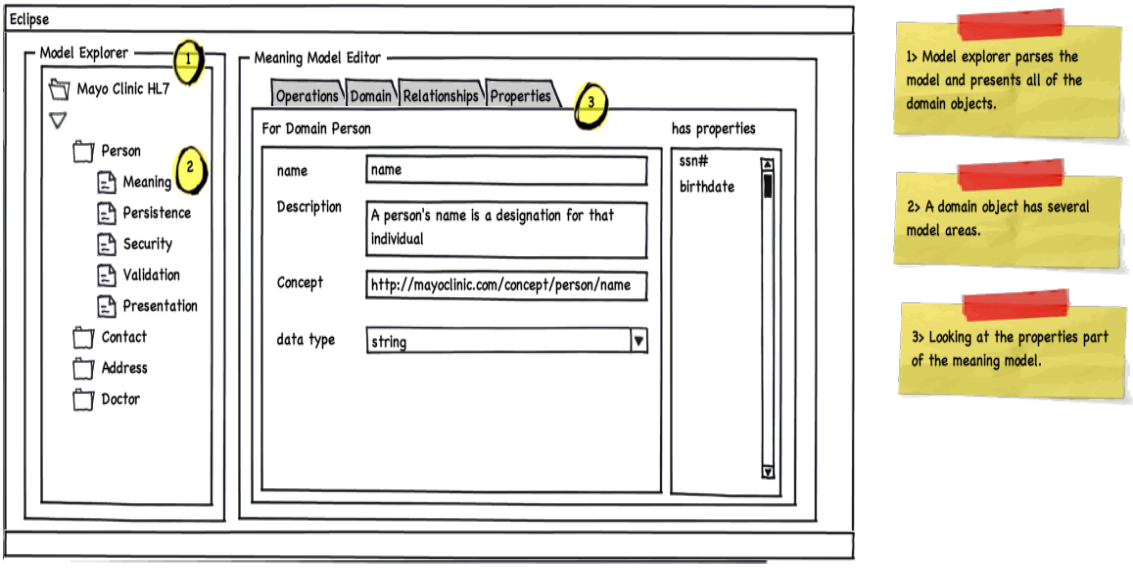
If conversion succeeds, the user will be presented with the loaded Model Explorer interface. The user will be able to browse the converted data model and view the six different aspects of that model from different screens. In the next chapter, the parts of the Model Explorer will be explored.

Model Explorer

The Model Explorer allows the user to discover the aspects of the model that was converter. The following mockup describes how this interface will look:

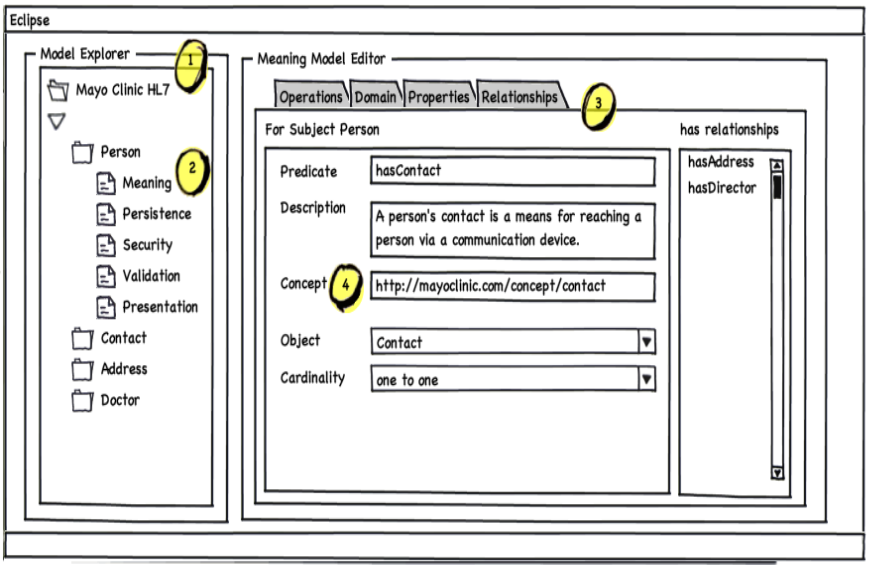


In this illustration, the meaning part of the meta model is displayed. This information on this screen is backed by the intermediate form of choice. On the left is the model explorer panel. This scrollable tree based view displays the models classes as domains. Clicking on a domain exposes the six different facets of models underneath it. By default the meaning screen is presented to the user first. The panel on the left is tabbed to represent four of the five components of the meaning facet of a model. The user may choose to look at the domain, property, operation, or relationship facets. The concept facet is not represented as a tab. Instead, concepts can be applied to either of these facets on the particular facet screen. In the above mockup, pointer 4 highlights how concept URIs are associated with a the model element Person domain. The following mockup show how this idea is extended to the property aspect of the meaning part of the model:



This tab screen looks similar to the domain model tab, except as there are multiple properties per domain model, the properties list panel is included on the right side of the screen. A user may select a particular property for viewing in the center panel. Property aspects peculiar to the meaning part of the model are then displayed for this property. Of note is that a property may have concepts associated with it as well as datatype, name, and other information.

The relationship and operations tabs follow the same pattern. The following diagrams provide a picture of these aspects of the meaning model. Note the similarity in the user experience for these two tabs. Here is the relationships tab:



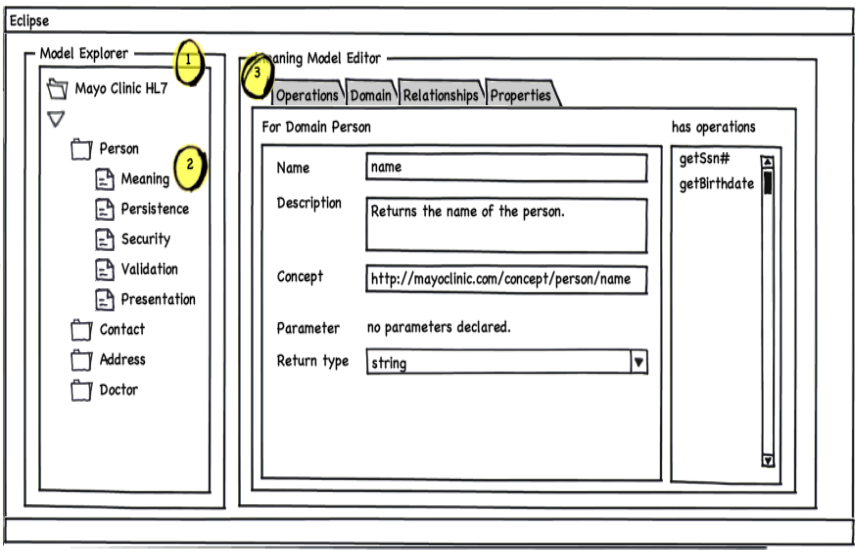
1> Model explorer parses the model and presents all of the domain objects.

2> A domain object has several model areas.

3> Looking at the relationships part of the meaning model.

4> Concepts can be added to describe this relationship, but the default concept takes the context

and the operations tab:

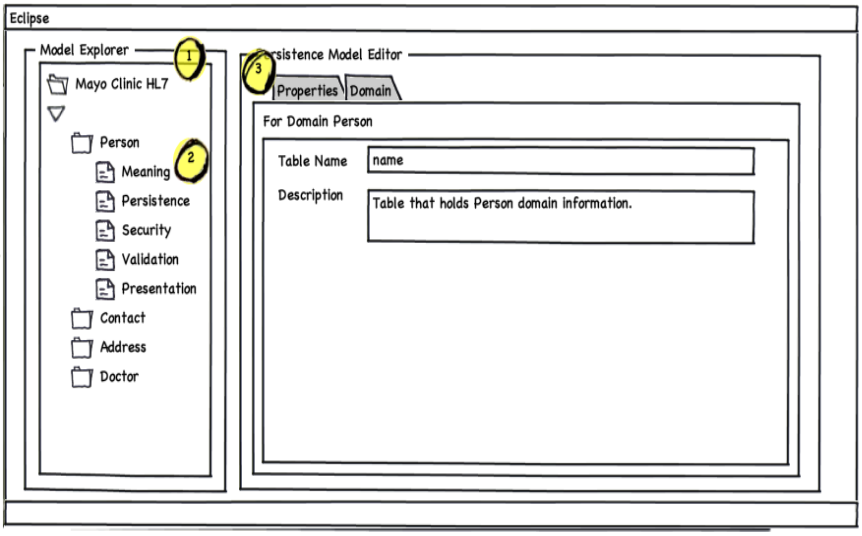


1> Model explorer parses the model and presents all of the domain objects.

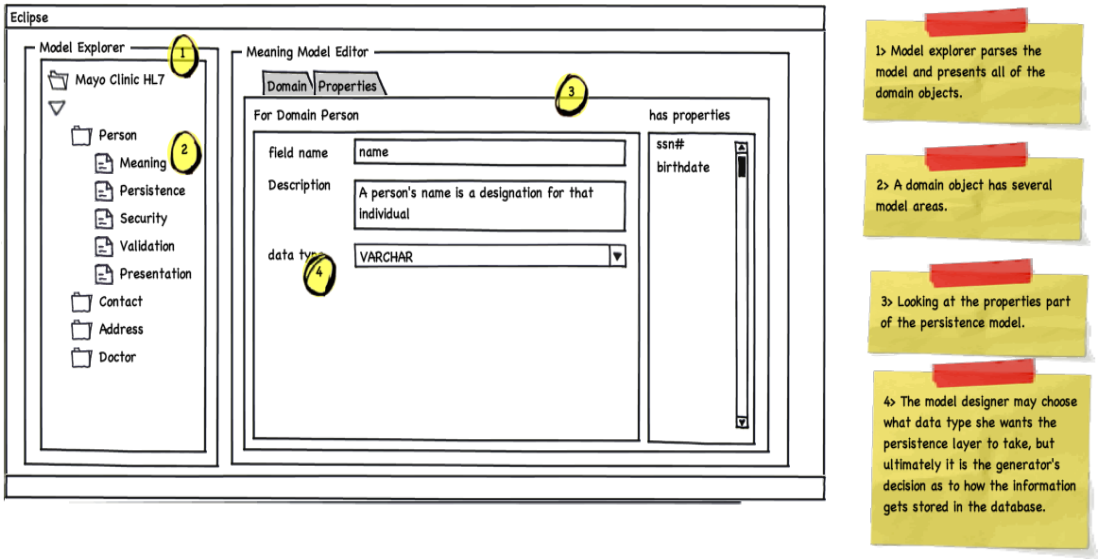
2> A domain object has several model areas.

3> Looking at the operations part of the meaning model.

The other portions of the model are displayed in similar ways. For modelling persistence the following screens show examples of how this will work. Again an effort is made to keep the user interface look and feel consistent across the six aspects of the model. First a look at the persistence model for domains screen. This screen is shown by default when the “Persistence” folder is opened.



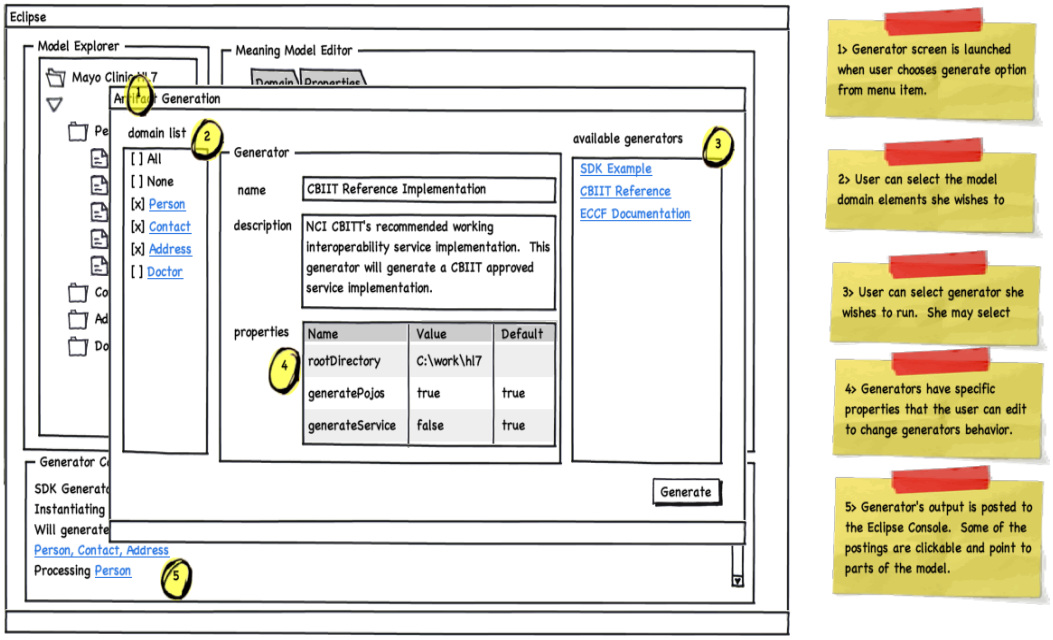
Choosing the Properties tab on this screen will render a view similar to the following:



Artifact Generation

Once satisfied with the model, users of the plugin can choose a particular generator to generate the appropriate artifacts from this model. The plugin sports a generator choosing screen to facilitate this. The user may select from a collection of installed

generators, and also select the which domains from the model should be used to create generated artifacts. The following illustrates the user interface for artifact generation. To get this screen, users are expected to right mouse click the model explorer to get to the “Generate Artifacts” floating menu item.



As the Artifact Generation screen is presented to float over the Model Explorer screen. This was considered to be ideal as generation is a relatively short lived activity in which the user chooses the relevant domain elements to be used for generation, as well as the generator she wants to use. A user may choose to use only one generator at a time, but may decide to choose any number from zero to all of the domains from the model to be fed to the generator.

On the center panel, the name of the generator and the text describing the generator are displayed at the top. Also the user sees the properties the generator is expected to made available for its proper execution. The user may choose to edit these properties, or leave them as they are. Once the user has selected the domains to be processed, and the generator to process them, and is satisfied with the generator properties displayed, she may press the “Generate” button to begin generation.

At this point the Artifact Generation screen will disappear, and the Eclipse console will be displayed on the bottom portion of the screen. The domain by domain progress made by the plugin generator will be displayed here. Clickable status will scroll past as the generator makes progress. Once the generation progress is completed, the console with indicate a successful run, or display the list of errors that caused the run to be unsuccessful. The user may click on these errors to be taken to the part of the model

that caused the issue. If the issue is addressable, then the new model can be used to feed the next generation sequence.