WHITE PAPER

Sybase® PowerDesigner® Extensibility

SYBASE®

TABLE OF CONTENTS

INTRODUCTION

Why Extensibility?

It is almost certain that whatever metamodel you use, it will not have everything you want once you start to build real-world software. You'll have to extend it, usually by creating additions that express information the metamodel could not express in its original version. To achieve your special application domain goals without the need for radical changes, you need to have extensibility of your metamodel. Extensibility mechanisms allow particular projects to extend the metamodel for their application at a low cost. Extensibility mechanisms allow you to define and implement your own methodologies, standards, and develop software your way.

When selecting a tool from any tool vendor, you must check whether the user will have a well-defined way of extending the metamodel. What can you extend? Can you add new concepts? Can you add new relationships between concepts? Provide more information per concept? Is the extensibility mechanism solid and useable for complex extensions, or just for very simple ones?

Extensibility is key to all methods and tools. From the start, one of the goals of the UML has been to provide extensibility and specialization mechanisms to extend its core concepts. There are 3 kinds of UML extensions: constraints (semantic restrictions on design elements), tagged values (allow the addition of new attributes to elements) and stereotypes (named grouping of constraints and tagged values).

PowerDesigner has taken the concepts of the UML extensibility features far beyond their definition to transform them into a powerful and complete metadata and tool extension mechanism for all types of models. The purpose of this white paper is to present the various capabilities of model extensions and to give you some ideas about using all the power you have on hand with PowerDesigner.

POWERDESIGNER EXTENSION CAPABILITIES

PowerDesigner has native support for a set of models representing the most commonly used methodologies: E/R modeling, Object modeling (UML), Business Process Modeling (BPM) , and more. These models have proven themselves to be efficient enough on their own, but you may always get more from your models if you extend their default metamodel to support your own methodology, operational requirement or development standard.

PowerDesigner allows you to go beyond the built-in model definition. You can refine the metamodel by adding properties to existing concepts, define new concepts to complement existing ones or even define a new kind of concept (a new kind of diagram).

Not only does PowerDesigner allow you to define your own extensions, you can do that in a very easy-to-use interface. The most common extension can be defined in a few mouse clicks. PowerDesigner's new features are often delivered as model extensions direct from PowerDesigner engineering, which proves this mechanism is very robust, and actually the best way to extend PowerDesigner.

All of this capability is provided through the "Extended model definition". The extended model definition will contain all the details of your extensions that you want to apply to your models. It is easily shared throughout the company.

The more common extensions are:

- Add new attributes to any concept
- Create new tabs in property sheet to enter value for these new attributes
- Customize symbol based on values of some properties
- Add new constraint on the model by adding new check model rules
- Add new concepts using Stereotypes
- Add new relationships between new or existing concepts
- Perform new code generation tasks or whole new language/artifact output
- Implement Model Driven Architecture (MDA™) style transformations

ADDING NEW ATTRIBUTES

Let's consider the situation where you want to add two new properties to all the entities of your conceptual data models: a functional domain and a business owner. Each entity in your CDM will need a set of values for these new properties.

Adding new properties to an existing metaclass in the metamodel is done using extended attributes. The extended attributes are added to the definition of the metaclass; in our case, the entity. In order to let an analyst easily enter values for the new properties you can create also a custom form that will become a new tab in corresponding metaclass property sheet; again in our case the entity property sheet.

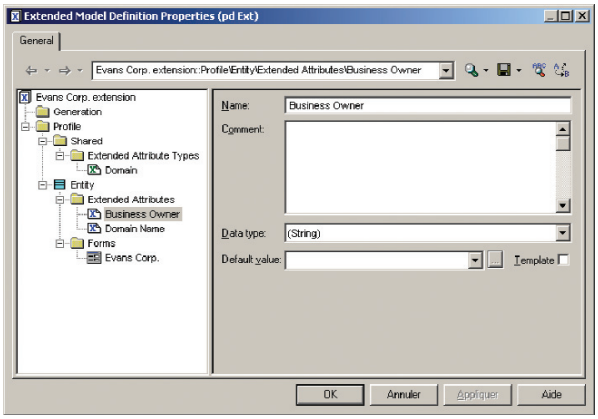

Fig 1: Model extension with extended attributes and custom forms

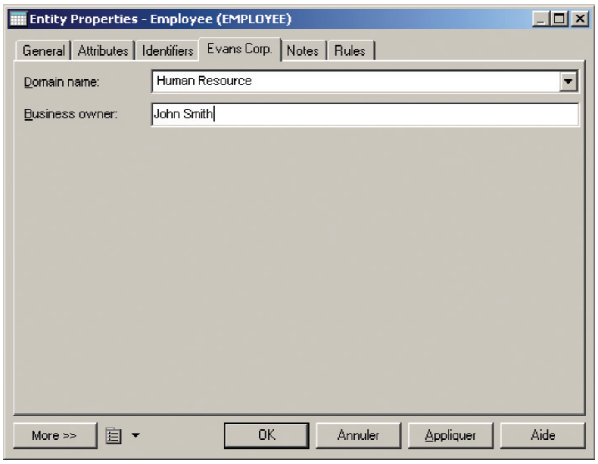

Fig 2: Custom for on entities with extended attributes

As shown in the figure above, these attributes are very easy to define and the form is completely managed by PowerDesigner once you have indicated which attributes you want to add on it.

CUSTOMIZING THE DISPLAY

Let's see another very common and pretty simple case: you want to change the symbol of your objects. This can be done based on a value of a property, or based on a more complex test of several values for several attributes. Here again, you simply define each test (criteria) that will be the basis for a new symbol and define in the section beneath it the symbol you want to use. The expression of the criteria is defined in PowerDesigner's own Generation Template Language (GTL), a simple yet powerful built-in language.

In our example, let's say you want to change the color of the entities belonging to the Human Resources domain. Under the Entity metaclass, you will define a criteria testing the value of the extended attribute "Domain". Then, add a custom symbol below the criteria and the new symbol will be applied to all entities where the criteria is satisfied.

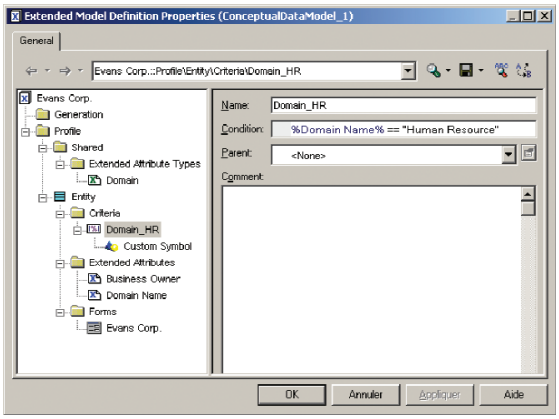

Fig 3: Criterion definition in model extension

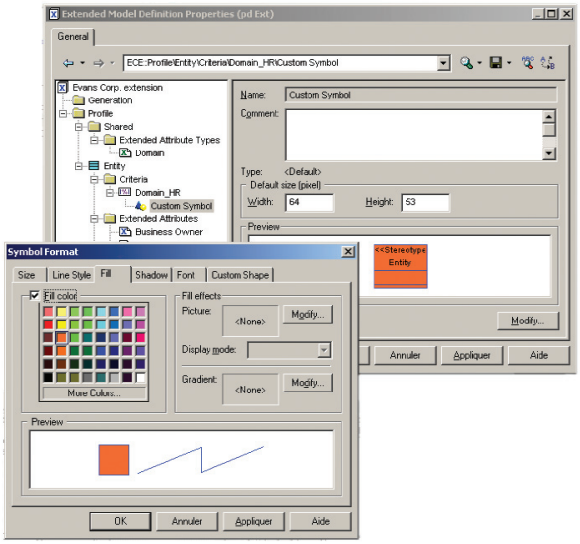

Fig 3b: Custom symbol on a criterion

All standard PowerDesigner format choices for a symbol are available here. This feature allows you to make major or minor changes to the basic symbol of the object.

ADDING NEW CONCEPTS AND CONSTRAINTS

In more complex cases, you may want to define your own concepts to complement the existing modeling semantic. Further, you may want to define a set of concepts that support a new kind of diagram.

To illustrate this point, will look at the Robustness diagram Extended Model Definition as implemented in the Object Oriented Module. The Robustness Diagram is a high-level class diagram, where the following different types of classes are defined:

- **Actors.** This is the same concept as actors on a UML use case diagram.
- **Boundary elements.** These represent software elements such as screens, reports, HTML pages, or system interfaces with which actors interact. These may also be called interface elements.
- **Control elements.** These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.
- **Entity elements.** These are entity types that are typically found in your conceptual model, such as Student and Seminar.

Here is a typical sample of a Robustness diagram:

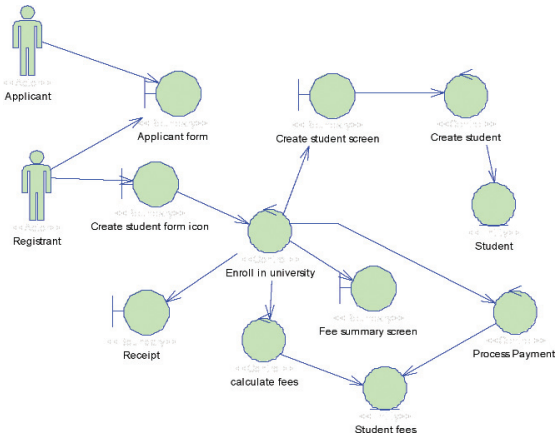

Fig 4: Robustness diagram sample

These new concepts are close to classifiers in the UML. In order to create this new type of diagram in PowerDesigner, you will complement the definition of class and use a class diagram to represent Robustness diagrams. In order to define these new concepts, you will define four stereotypes on the metaclass Class that will represent each a new concept. Thanks to the "stereotype used as a metaclass" feature, PowerDesigner will add a specific list to the Model menu, a specific "New" entry, a new tool in the tool palette to graphically create the objects and a customized symbol for each new concept.

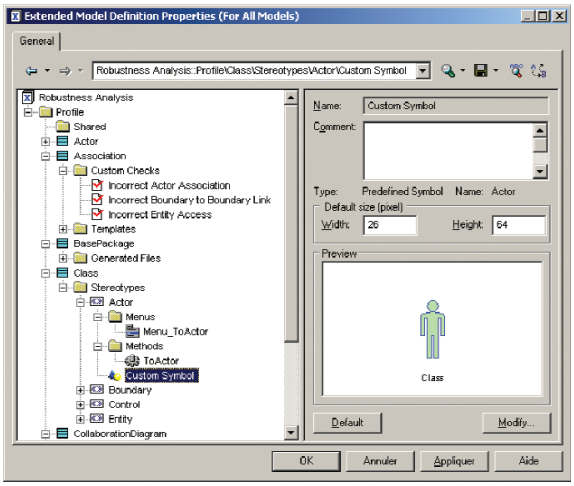

Fig 5: Stereotype and custom checks to implement a concepts

The Robustness diagram definition includes some constraints. You will need to some custom checks to the model extension that instructs PowerDesigner to verify automatically that the new constraints are respected. These custom checks are run every time you run a check model process.

In case no existing objects are a good candidate for the new concept you want to add, you can always use the "extended object", which is present in any model and allows you to define virtually any customization.

All new added concepts will behave the same way predefined concepts behave inside PowerDesigner. All common features like Property sheets, Reports, etc... will automatically take the new concepts into account.

ADDING NEW RELATIONS BETWEEN CONCEPTS

Sometimes adding new concepts is not enough. Sometimes you also have to add new relationships between new and/or existing concepts. PowerDesigner provides two complementary ways of creating links between concepts.

New Link Object

If the new relation you want to add has graphical representation or has some specific attributes to set, you should use an extended link. In this case you will create a stereotype and add a custom symbol to determine link appearance (end points, line style, etc). You must also validate that the end points of the link are correct based on the object definition you want. To enforce the validity of the creation of a link (ie: do I believe the start and end points of the link are objects I deem this link is valid between) you may code an initialization event handler in VB script that will check the validity of the object and only authorize its creation if it respects the constraint you have defined.

New Collection Between Objects

If the new link you want to add has no need to be displayed on the graphic, then an extended collection between the two objects involved in the link can be sufficient.

IMPROVE INTER-OBJECT NAVIGABILITY

Another extension capability of PowerDesigner can be used to ease the navigability through related objects. Using existing known dependencies between objects, you can calculate new collections of related objects and display it directly in the properties dialog.

For example, you will make use of the UML sequence diagram. In this case, for each operation you will want to display the list of sequence diagrams where a message is linked to the operation.

This can be done very easily using a calculated collection. A short VB Script will calculate the list of objects and PowerDesigner will display this collection in the list of dependencies.

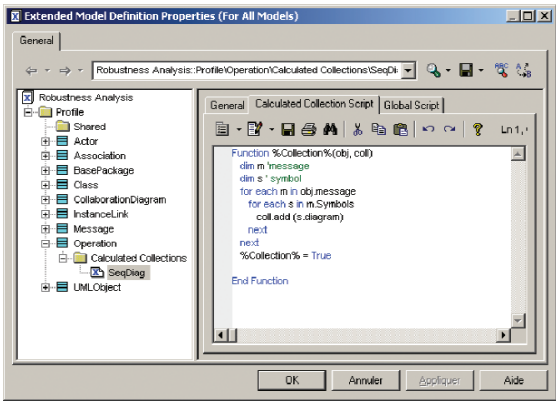

Fig 7: Calculated collection on Operation

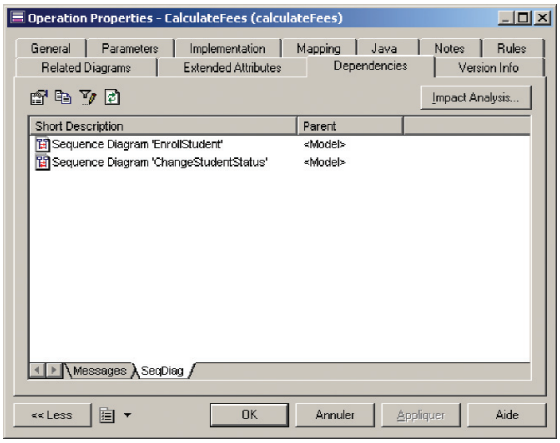

Fig 8: Calculated collection displayed in Operation property sheet

ADDING NEW CODE GENERATION CAPABILITIES

All these new extensions, new concepts and relationships between them, may eventually be used for code generation. You may also just want to extend the existing code generator without having made any metamodel definition changes. All code generation extensions are defined using PowerDesigner's own Generation Template Language (GTL) in an extended model definition.

Most of PowerDesigner's generation capabilities are implemented using GTL and all new features are developed based on these principles. For example, the support for Hibernate has been defined as an overloaded version of the Java generation for a UML model and uses the O/R mapping information that can be defined in PowerDesigner.

In order to generate your own files, you will define a "Generated files" element for each metaclass that must produce a file. The Generation Template Language (GTL) allows you to use all PowerDesigner objects, properties and collections. You can browse the model and define what should be the output, or the generated code. You can define pieces of code in a "template" that you can refer to in any other piece of GTL.

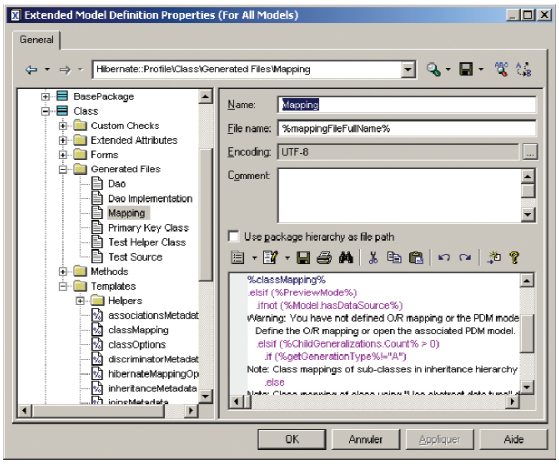

Fig 9: Generated files and templates in a model extension

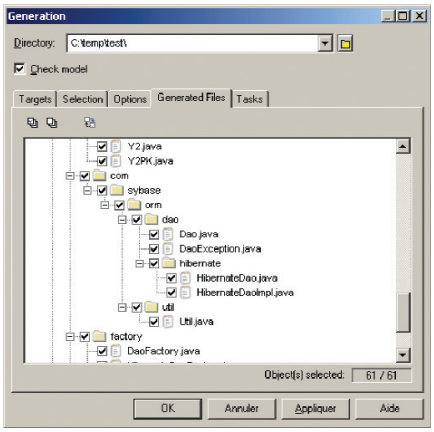

Fig 10: New generated files visible just before generation

ADDING NEW BEHAVIOR

To extend PowerDesigner for even greater automation of common tasks, like performing some mass change throughout a model, calling an external program that will extract metadata from the model to export something to another tool, etc., you will use the plug-in capabilities of PowerDesigner.

The whole PowerDesigner metamodel is accessible through a COM interface and you can call any PowerDesigner function, reference any object or collection. Using the COM interface allows you to build more complex logic, using virtually any language including Visual Basic script, Visual Basic, C# or Java. You may also use a mix, for example, you can choose Visual Basic scripting for features that do not require a complex user interface and Visual Basic, C# or Java when a user interface is required.

All these new functions will be plugged into PowerDesigner menus, in object contextual menus or global menus and will appear very naturally to the PowerDesigner end user.

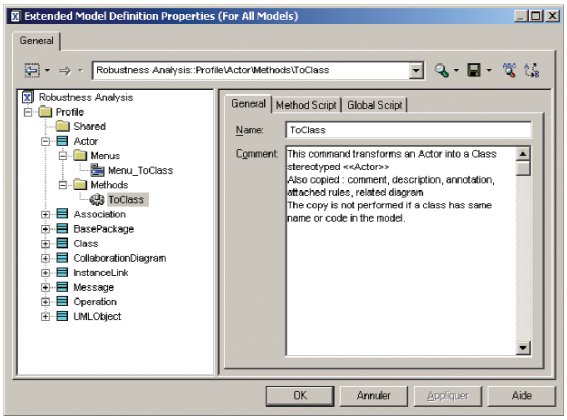

Fig 11: Add a menu to the Class contextual menu

IMPLEMENTING MDA™ TECHNIQUES IN POWERDESIGNER

The principals of the Object Management Group (OMG)'s Model Driven Architecture (MDAtm) tell us to separate the fundamental logic behind a system specification from the specifics of the particular middleware or target environment that implements it. This allows rapid development and delivery of new interoperability specifications that use new deployment technologies but are based on proven, tested business models. Organizations can use principals like MDA to meet the integration challenges posed by new platforms, while preserving their investments in existing business logic based on existing platforms.

Implementing an approach that follows the principals of MDA will broaden the benefits of using models. Techniques like PowerDesigner's model-to-model transformation will give users the ability to drive through layers of abstraction without loosing the more general level definitions. By combining modeling with model transformation, PowerDesigner gives users the ability to generate entire applications without writing a single line of code.

To ensure that applications do not become obsolete as platforms and infrastructures evolve, and to allow easy change of implementation, MDA like approaches demand that software be defined independently of the infrastructure or platform on which it is executed. This goal is achieved by creating Platform-Independent Models (PIMs). To take advantage of a specific platform or infrastructure, PIMs must be transformed to Platform-Specific Models (PSMs),

PowerDesigner uses unique, built-in model-to-model generation capabilities between the PIM and PSM and allows you to define the model to model transformations through extended model definitions. Using PowerDesigner you will be able to apply MDA concepts and easily transform the independent models into the best implementation models based on the definitions you build for your application.

The generation capability already described in a previous paragraph complements the model-to-model transformation to drive complete code from PIMS. By creating your own code patterns and templates and leveraging extra metadata added or generated (or calculated) throughout the model-to-model generation, and matching those patterns to the deployment needs for a specific platform, PowerDesigner can create virtually all the code needed to implement a system without deviating from the original high-level, business-centric platform independent view.

Here a small sample of transformation that will be applied during a generation from an Object Model at the analysis level (Analysis selected as the language target) into a Object Model targeting a Java implementation. This transformation is to change concrete and persistent classes automatically into J2EE EJB code. The reverse transformation is also available to delete the component when reverse engineering the Java model into an analysis model.

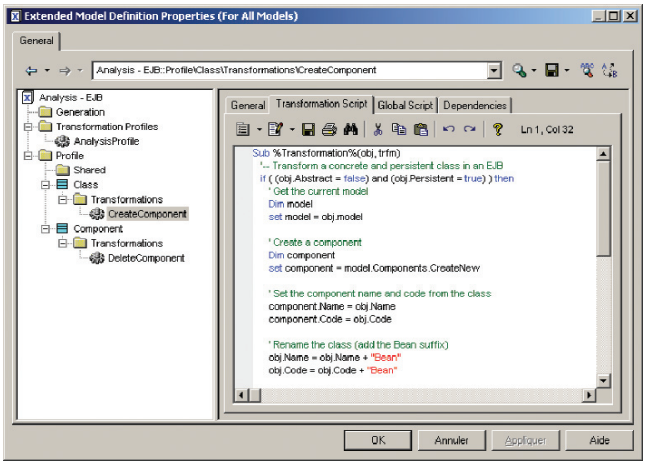

Fig 12: Transformation used in MDA style implementation

DEPLOYING AN EXTENDED MODEL DEFINITION

PowerDesigner model extensions are easy to deploy to a group of analysts or designers. Extensions are saved in an XML file. Just copy the file into the extensions directory of the PowerDesigner installation and it is now available to that user. Alternately, you can use a shared network drive, and configure all PowerDesigner installations to reference the shared location, if you want to centralize the administration and management of all model extensions. Several extensions can be added to a single model. You can also tell PowerDesigner that a model extension is to be automatically attached to all new model when they are first created. The automatic attachment can be based on a specific target language. This happens behind the scenes so that you can ensure that all designers will use it without even noticing that it has been attached.

WHERE TO LEARN MORE ABOUT POWERDESIGNER EXTENSIONS

The PowerDesigner's Advanced User documentation has more detailed information and samples about all the extensions described in this paper. You can find it in the PowerDesigner set up, or on the documentation download site (available at www.sybase.com/powerdesigner under Product Manuals).

CONCLUSION

By combining all the mechanisms described in this document, you can extend the basic metamodel of PowerDesigner to support any extension need you may have. . As you have seen; you can define new properties, define new concepts including their graphical symbol, define new constraints, define new capabilities and features, add, extend or change code generation, and drive an MDA like implementation. PowerDesigner provides a metamodel that is easy to extend. You gain flexibility, rapid response to change, and to true alignment between your modeling environment and all your business challenges, now and in the future.