MQSeries Everyplace - Administration tool

User Interface Archiecture Design

Version 1.0

17 August 2000

Phill van Leersum
MQSeries Development
IBM United Kingdom Laboratories
Hursley Park
Hursley
Hampshire, SO21 2JN
UK

phillvl@uk.ibm.com

Take Note!

Before using this report be sure to read the general information under "Notices".

First Edition, August 2000

This edition applies to Version 1.0 of *MQSeries Everyplace - Administration tool* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

Trademarks and service marks

Summary of Amendments

Date	Changes
17 August 2000	Initial release

Preface

This document describes the architecture of a platform-independent, low footprint, extensible admin tool.

It makes use of User Interface Architecture (UIA) recommendations to avail itself of UIA compliant exchangeable renderers.

The architecture uses a layered approach to maximise reuse of components, and facilitate customisation, by IBM or its customers.

Bibliography

List any supporting publications here otherwise delete this page. Use following format:

- *MetaModel XML format*
- *MQSeries Everyplace - Administration tool design*

1 Introduction

This document describes the architecture of a platform-independent, low footprint, extensible admin tool.
It makes use of UIA recommendations to avail itself of UIA compliant exchangeable renderers.
The architecture uses a layered approach to maximise reuse of components, and facilitate customisation, by IBM or its customers.

2 Architecture

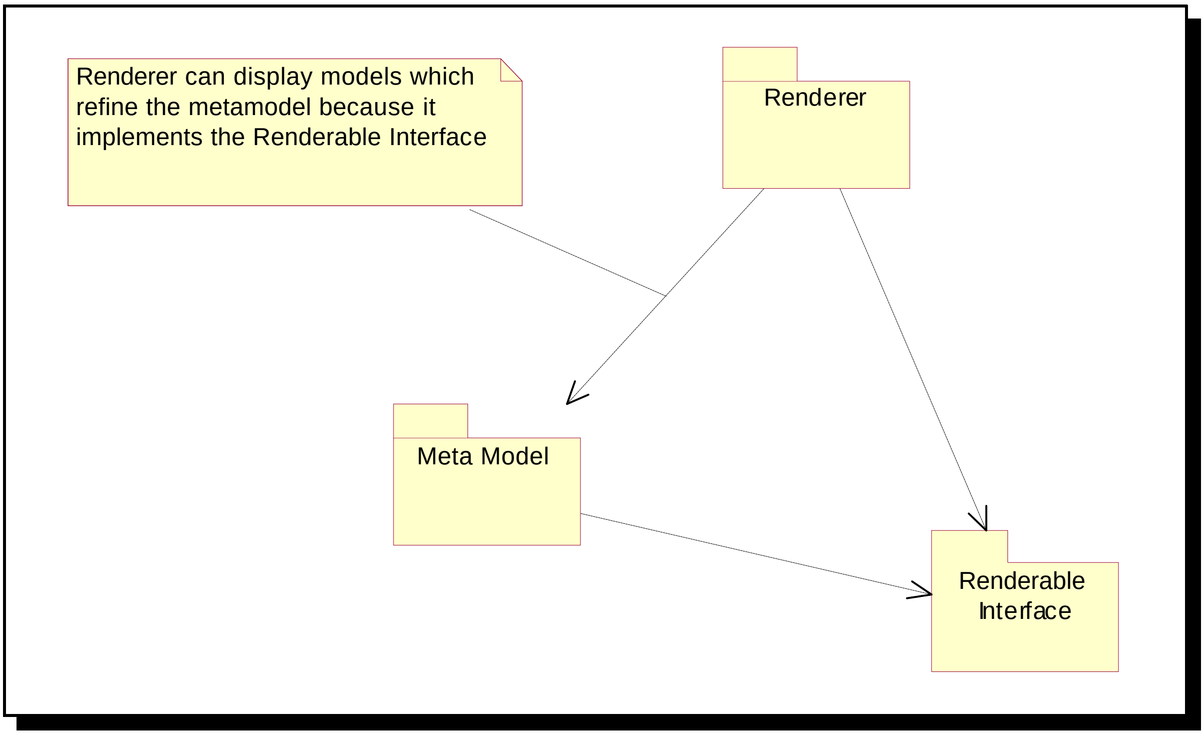
Overall concept: Provides a layered implementation of RendereInterface. Complies with UIA recommendations. Reusable layers. Use any renderer. The advantage of this approach is that a Renderer, once written, can be reused for many tools. This gives a common look-and-feel to al such tools, and ensures compliance with the UIA design guidelines.

It frees the implementor of the business logic from GUI considerations. It forces a clear separation between the GUI and the underlying logic, which leads to more reusable and portable code.

The design is layered for maximum reuse, minimum defects, and maximum maintainability. Portions of the code are specifically designed to support the User Interface Architecture (UIA) recommendations currently being defined by IBMs Ease of Use team. When the recommendations are made formal IBM-wide compliance will be made mandatory. This code already complies with these requirements as currently known, and will easily adapt to changes in the UIA requirements.

The designed has been prototyped in fully-portable Java 1.1 compliant code. This means that it will work on any platform that provides a Java 1.1 compliant JVM. Swing is not required.
Although the architecture is layered, and designed for reuse, it does not imply a large footprint. The entire prototype, including the simple renderer but excluding MQe.jar, is a little less than 100k bytes.

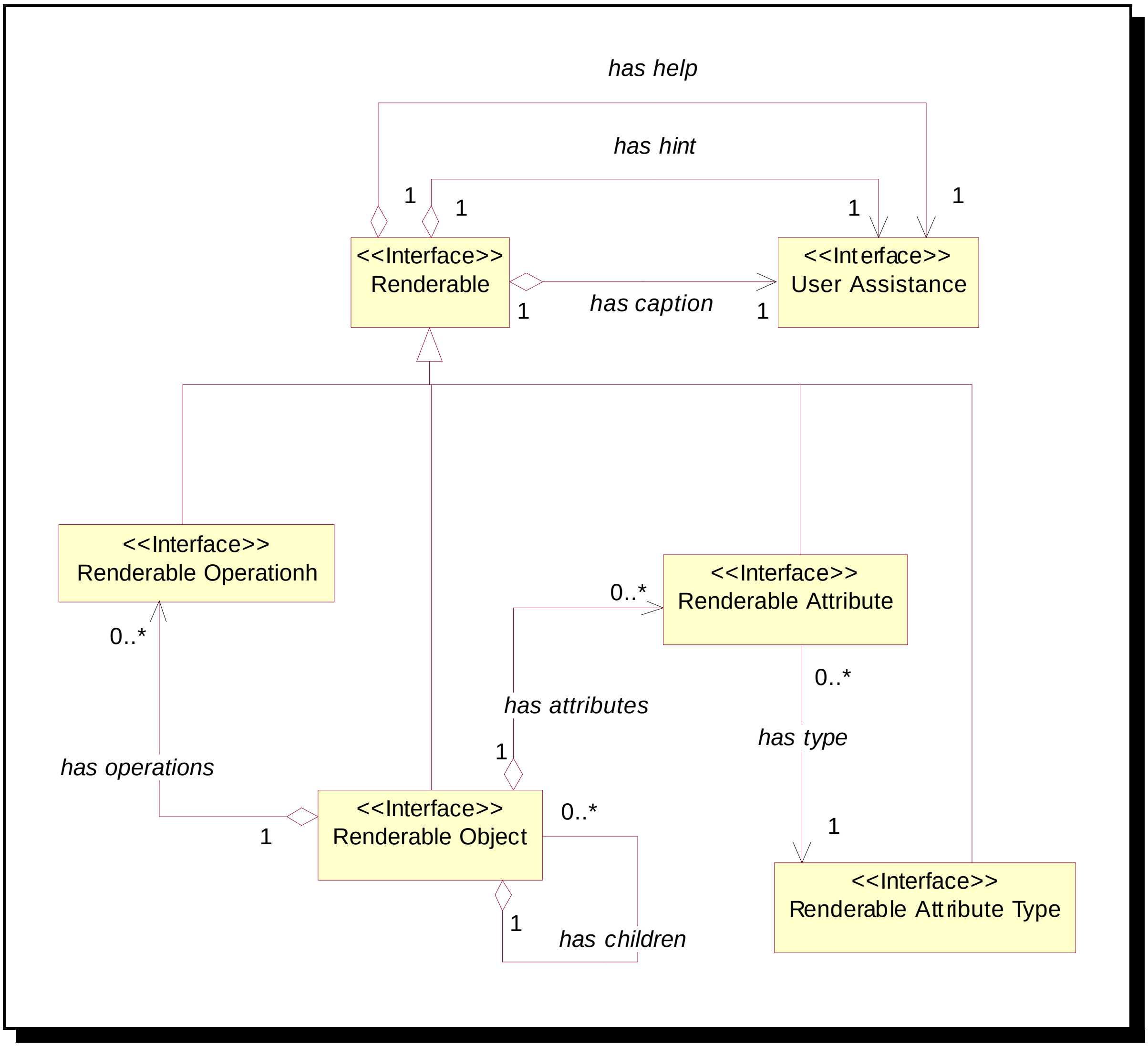
The factory pattern ***(reference to the Gang Of Four book)*** for object creation is used throughout this architecture, to allow customisation of its behaviour through subclassing. Essentially, the factory pattern allows an instance of one class (A 'factory class') to act as a factory for instances of another class. This is achieved by the factory object displaying an instance method that creates an instance of the desired object (sometimes caled a 'factory method'). This factory method can then be overriden by the implementor of a subclass of the factory class. The overide can then create instances of its desired class (which must be a subclass of the original created class). The pattern allows large amounts of reuse of code to be integrated seamlessly with specialisations of the same code.


Renderer can display models which refine the metamodel because it implements the Renderable Interface
Renderer
Meta Model
Renderable Interface

- Renderable Interace: provides a standard interface through which GUIs may render an exposed model, without detailed knowledge of the underlying model. Decouples the GUI from the underlying model. Allows any GUI to render any compliant model.
- MetaModel Framework: Provides a framework that facilitates the implementation of the Renderable interface. Allows much/most of the GUI representation to be encoded in XML. Provides xml rendering capabilities for the Model objects to allow them persistence/storage.
- Model Layer: exposes a subset of the functionality of the Stub Layer to the Renderer by implementing the RendererInterface (indirectly, by extending the metamodel).
- Renderer Implementation: A simple renderer has been implemented. This unsophisticated because it is built entirely using the highly portable AWT 1.1 code. More sophisticated GUIs are expected to be provided by (amongst others) IBM Ease Of Use organisation.

3 Renderable Interface

Provides a standard interface through which GUIs may render an exposed model, without detailed knowledge of the underlying model.
Decouples the GUI from the underlying model.
Allows any GUI to render any compliant model.
Compliance with (implementation of) the interface ensures complance with the UIA Designers Model recommendations.
Designed to be used/implemented by any project wishing to avail itself of the multiple-renderer capability, and as such is not strictly part of this project. Implemented as part of UIA work. (Reference to the UIA documents)



3.1 User Assistance

This is a general term given to data that aids the user. It can contain text, audio, image etc data.

3.2 Renderable

Any user perceivable entity must implement this interface. It is a marker for the renderability of the entity. A renderable contains three User Assistance data, one for each of hint, help, and caption. Each of these has a different level of detail of the help used by the renderer in different circumstances.

3.3Renderable Object

Renderable Object marks an entity that forms the smallest unit of manipulation by the user. Typicaly an implementation of this interface will represent a discrete object in the designers/users model, for example a file, or a folder.

Renderable Objects have zero or more Renderable Attributes.

Renderable Objects have zero or more Renderable Operations.

Renderable Objects have zero or more Renderable Objects as children.

3.4 Renderable Operation

Renderable Operations represent actions that the user is allowed to invoke upon a Renderable Object or upon a Renderable Model. These actions consist of a verb-like command, and an optional path, that allows the operations to be rendered in a hierarchical manner.

Renderable operations are like signals, in that they have no concept of arguments.

Renderable Operations are typically rendered as menu items or buttons.

3.5 Renderable Attribute

Renderable Attributes represent fragments of state that are inseparable from their owning Renderable Object.

Renderable Attributes have a name which is unique within their owning Renderable Object.

Renderable Attributes have a single value each, which is considered a primitive data type, and is always passed around as a strng representation.

Renderable Attributes have a single Renderable Attribute Type each, which constrains the values allowed for the attribute.

Renderable Attributes can have default values. If not explicitly set, then the default value is the same as the default value for the Renderable Attribute Type.

3.6 Renderable Attribute Type

Renderable Attribute Types are constraints on allowed values.

Renderable Attributes prvide the functionality to check the legality of a value, and to coherce a given value nto a legal value.

There are (curently) three broad categories:

- String: any arbitrary text, constrained by an optional length attribute.
- Integer: any legal decimal representaion of an integer
- Enumerated: The value is allowed to be one of a limited, predefined set.

4 MetaModel Framework

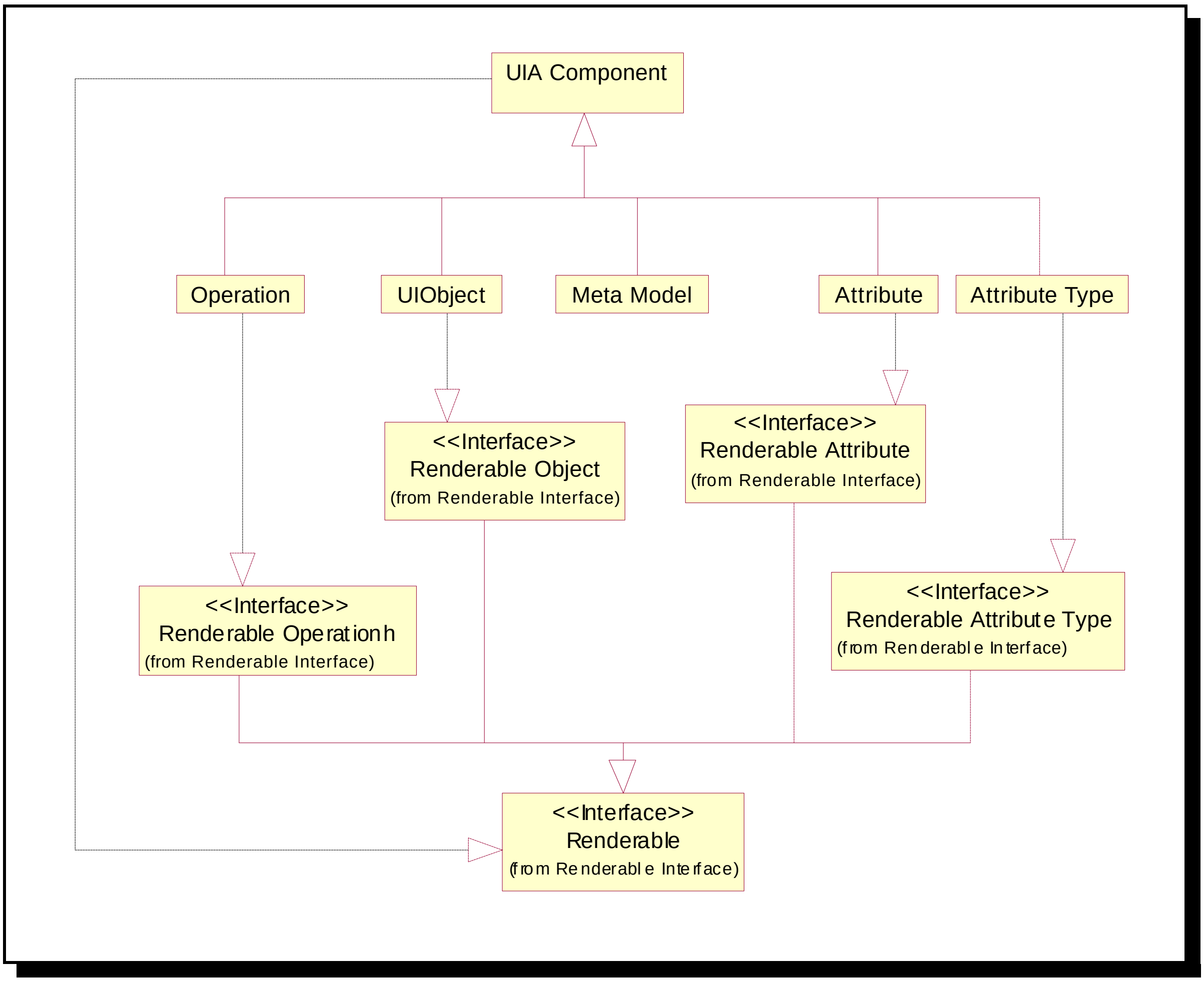
Provides a framework that facilitates the implementation of the Renderable interface.
Allows much/most of the GUI representation to be encoded in XML.
Provides xml rendering capabilities for the Model objects to allow them persistence/storage.

Designed/implemented for use by any wishing to implement RenderableInterface. As such not strictly part of this project (eventually). Part of UIA work.

The Meta Model Framework includes an XML parser, unparser, and parse-tree representation.

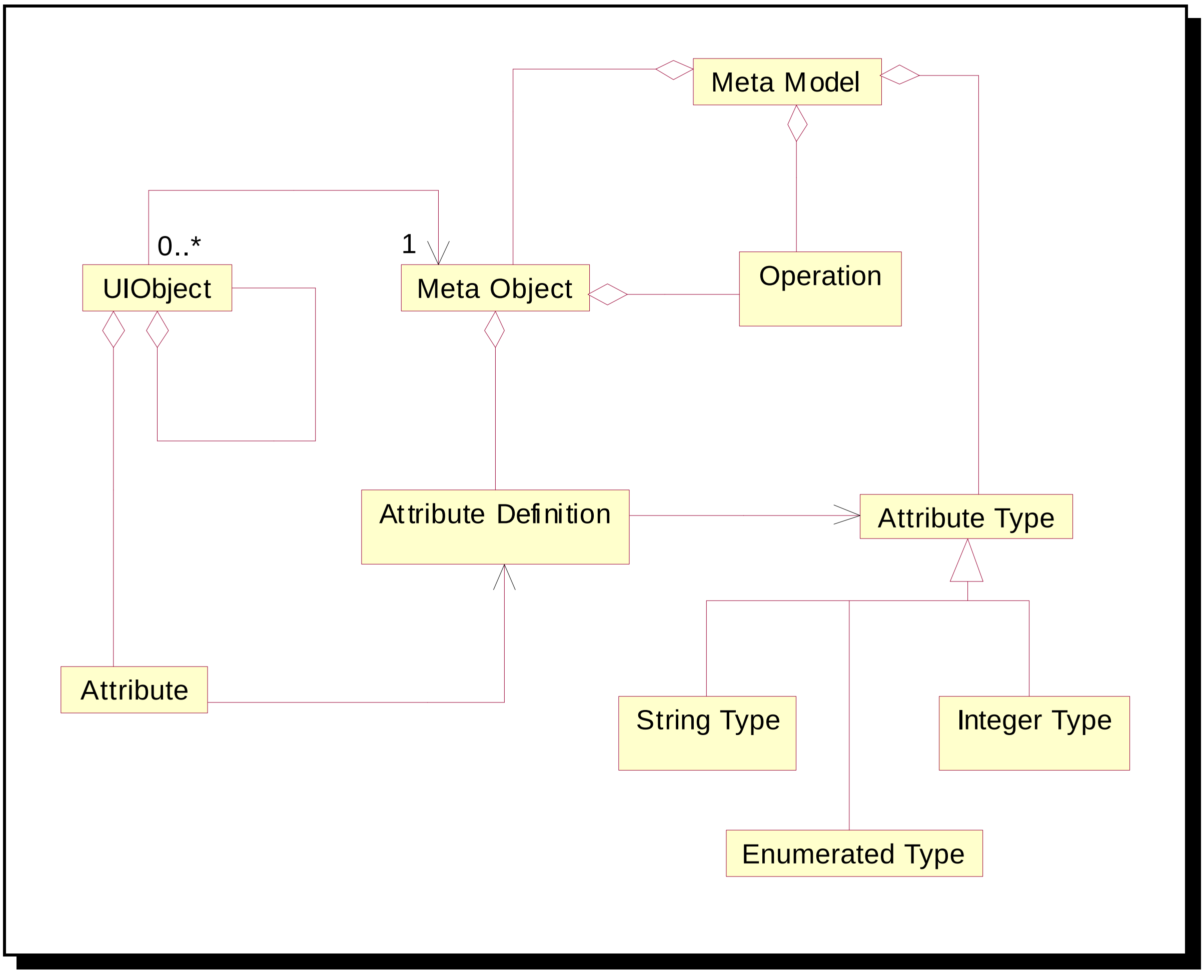
4.1 Implementation Of Renderable Interface

Diagram below shows how the Renderable interface is implemented by the MetaModel Framework. Each of the interfaces in the Renderable Interface is implemented by a class in the Meta Model Framework. The individual classes in the Meta Model Framework are described in detail later in the section.



4.2 Internal Associations

Diagram below shows how the components of the Meta Model Framework interact to create a functional model.



4.3 UIA Component

Implements Renderable.

Implements the interfaces that allow conversion to and from the XML parse-tree representation. The methods that perform the transformations to and from XML are overidden as required in subclasses so that all elements of the Meta Model Framework, and all subclasses are capable of both rendering themselves as XML, and of being restored from XML. This has two benefits. Firstly most of the UI specific information can be stored in flat-file format, allowing easy modification and internationalisation. Secondly, any subclasses of the model objects are immediately serialisable in human and computer readable form.

When reading itself from XML, a UIA Component will automaticaly create any dependent objects as neccesary. When producing XML the same pattern of recursive descent ensures the complete tree is translated.

4.4 Meta Model

Implements Renderable Model.

Acts as factory for Meta Objects.

Acts as factory for Attribute Types.

4.5 Meta Object

Represents a ‘type’ of UIObject.

Acts as factory for UIObjects.

Acts as factory for Attribute Definitions.

Acts as factory for Operations.

The Meta Object is a repository for all data common to a ‘type’ of UIObject: Operations and Attribute Definitions. In this way reduces the memory footprint.

4.6 UIObject

Implements Renderable Object.
Has a reference to the Meta Object that defines its ‘type’.
The Operations of a UIObject are the Operations held by the UIObjects Meta Object.

4.7 Operation

Implements Renderable Operation.

4.8 Attribute Definition

Provides a place to store data common to all attributes of the same name associated a Meta Object. In this way reduces memory footprint.
Acts as factory for Attributes.
Attribute Definitions are held by the Meta Object for which they are defined.
Attribute Definitions have a defined Attribute Type which constrains the values of all attributes referencing the Attribute Type.
Attribute Types can be marked as ‘essential’ to indicate that a legal value is required for validity of attributes referencing the Attribute Type.
Attribute Types can be marked as ‘read only’ to indicate that a user editing of of attributes referencing the Attribute Type is a meaningless activity.

4.9 Attribute

Implements Renderable Attribute.
Maintans a reference to an Attribute Definition, from which comes all the information no specific to a particular instance.
Has a current value.
Is considered ‘essential’ if the referenced Attribute Definition is marked ‘essential’.
Is considered ‘read’ if the referenced Attribute Definition is marked ‘read only’.
Constrains its values by use of the Attribute Type held by the referenced Attribute Definition.

4.10Attribute Type

Implements Renderable Attribute Type.
Acts as a constraint upon the values allowed to an attribute.

4.11 String Type

Implements the logic associated with ‘string’ Attribute Types (curently only truncation).

4.12 Enumerated Type

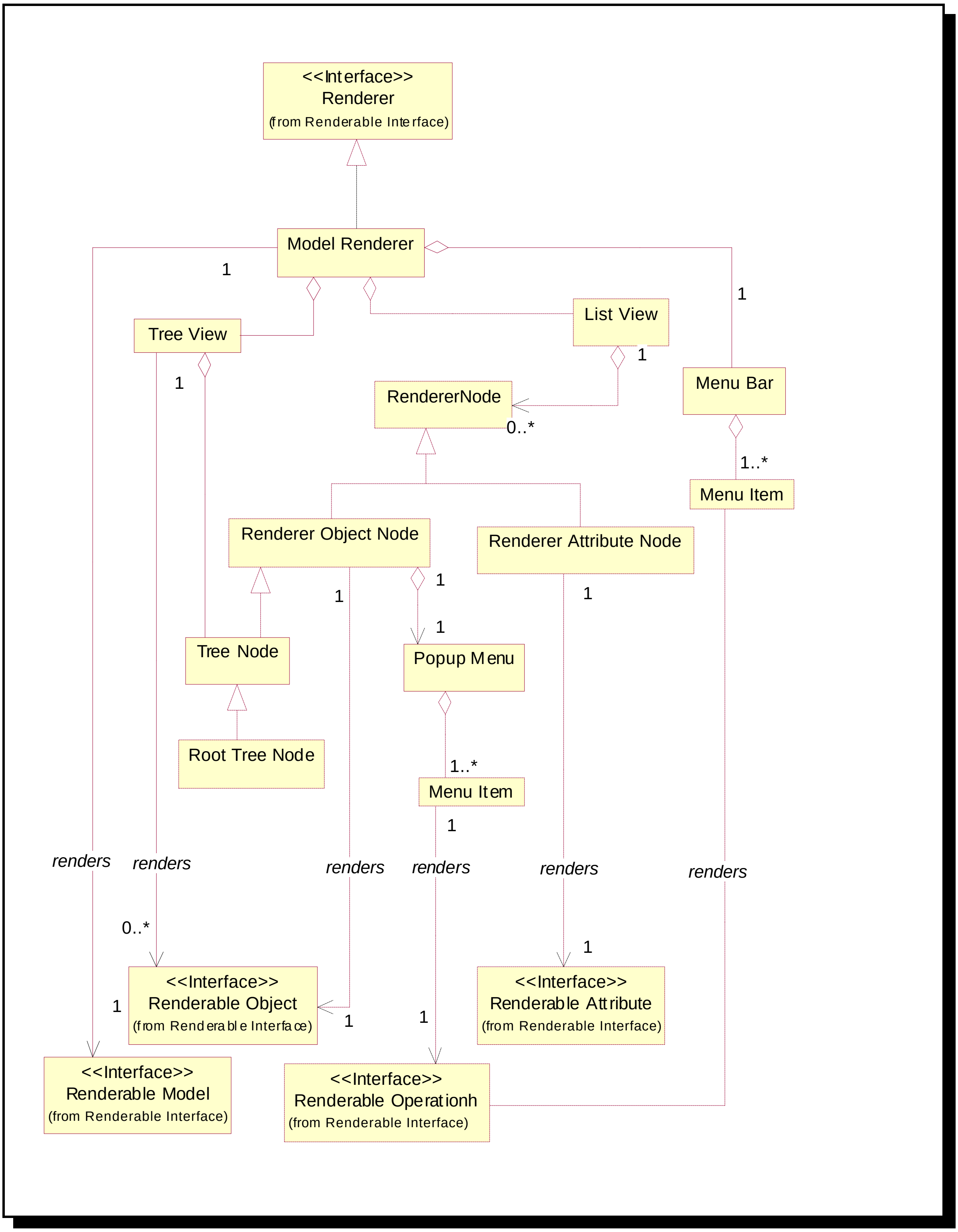
Implements the logic associated with ‘enumerated’ Attribute Types.
Holds a list of valid values.

4.13 Integer Type

Implements the logic associated with ‘integer’ Attribute Types.
Can parse and unparse strings of decimal digits.

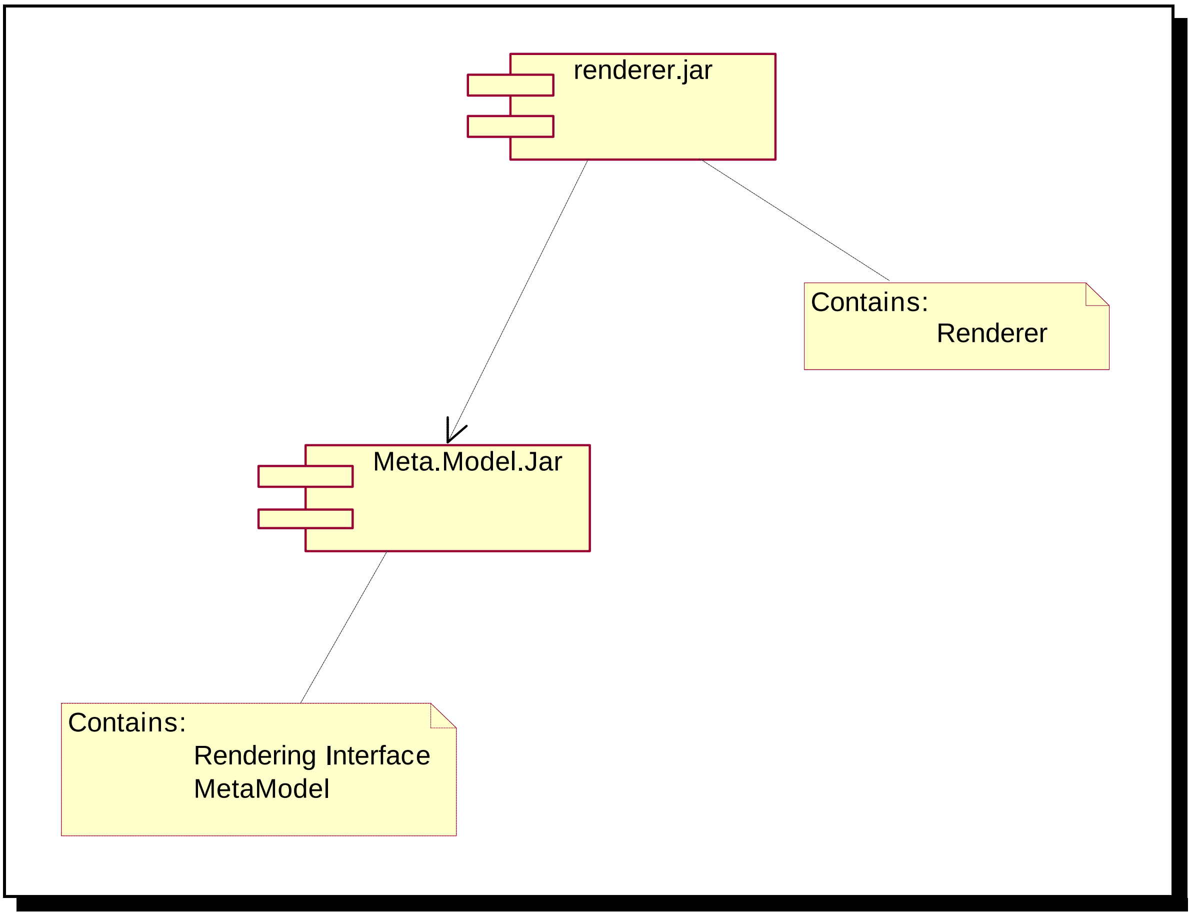
5 Renderer

A simple renderer has been implemented. This unsophisticated because it is built entirely using the highly portable AWT 1.1 code. More sophisticated GUIs are expected to be provided by (amongst others) IBM Ease Of Use organisation.



6 Packaging for Delivery

The Packages are be combined into four jar files for delivery:



Third parties can replace the renderer.jar with their own renderer