

# Scripting Model Documents

---

*Die Vorzüge und Nachteile von Model Documents und wie man durch Modelle und Scripting die Nachteile in Vorteile verwandelt.*

**Dr. Horst Kargl**

*All material © Sparx Systems GmbH – Central Europe 2010 - Version 1.0*

*[www.sparxsystems.de](http://www.sparxsystems.de)*

## **Abstract**

In Enterprise Architect können HTML und RTF Dokumente aus dem Modell generiert werden. Beide Ansätze sind templateorientiert. Um mehr Flexibilität beim Generieren von RTF Dokumenten zu erlangen, kann die Struktur der zu generierenden Dokumente modelliert werden. Diese Modelle sind allerdings „konkret“ und können nicht beliebig auf eine beliebige Struktur angewendet werden. Dieser Artikel beschreibt wie „generische“ *Model Documents* erstellt werden und diese ähnlich wie „normale“ RTF Templates auf ausgewählte Modellteile angewendet werden können.

## Generieren von RTF Dokumenten

Alle Modelle in Enterprise Architect können über *Generate RTF Documentation* als RTF Dokument ausgegeben werden. Der Wizard *Generate RTF Documentation* ist eine Zusammenfassung aller

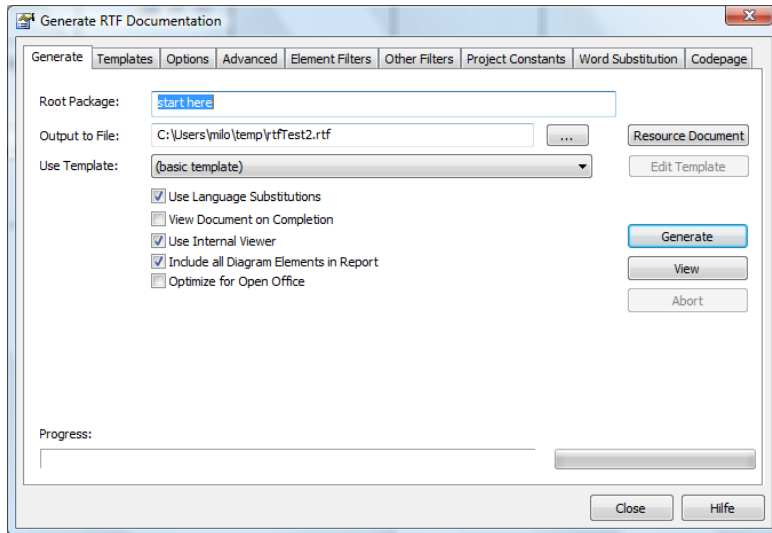


Abbildung 1: Generate RTF Document Wizard

Konfigurationen, um mit Hilfe von RTF Templates Dokumente zu erzeugen. Es können vordefinierte Templates ausgewählt werden, eigene Templates erzeugt und verwaltet werden, generelle Generierungsoptionen definiert werden, generelle Filter (alle Elemente mit bestimmten Eigenschaften) definiert werden, sowie projektspezifische Substitute und spezielle Wortsustitutionen (Actor = Akteur) definiert werden.

Wichtige Einstellungsoption ist in den generellen Settings der Punkt:

„*Include all Diagram Elements in Report*“. Diese Option erlaubt es, Elemente, die nur als „link“ in ein Diagramm eingefügt wurden und nicht direkt in dem Paket (oder Element), welches gerade im Dokument ausgegeben wird „physisch“ enthalten sind, trotzdem auszugeben. Dies führt zwar zu Redundanzen im Dokument, da, für jede Verwendung des Elements als „link“ in einem Diagramm, dieses Element im Dokument angeführt wird, bietet aber die Möglichkeit alle nötigen Informationen pro Diagramm auszugeben.

## Das RTF-Template

Das RTF-Template besteht aus einer Baumstruktur und einem RTF WYSIWYG<sup>1</sup> Editor. Der Baum beinhaltet alle im Modell verfügbaren Informationen (linker Teil nebenstehender Grafik). Der RTF-Editor erlaubt es beliebigen Text zu schreiben, sowie *Kontext-Tags* und *Element-Tags* zu benutzen. *Kontext-Tags* werden durch Auswahl einer *Checkbox* der linken Baumstruktur an der Cursorposition im Dokument angelegt und sind schreibgeschützt. Wird innerhalb des öffnenden und schließenden *Kontext-Tags* ein Rechtsklick mit der Maus durchgeführt, kann über das Kontextmenü ein *Element-Tag* aus einer Liste von möglichen Modellinformationen ausgewählt und eingefügt werden. Der *Element-Tag* wird im RTF-Dokument grau hinterlegt und steht in

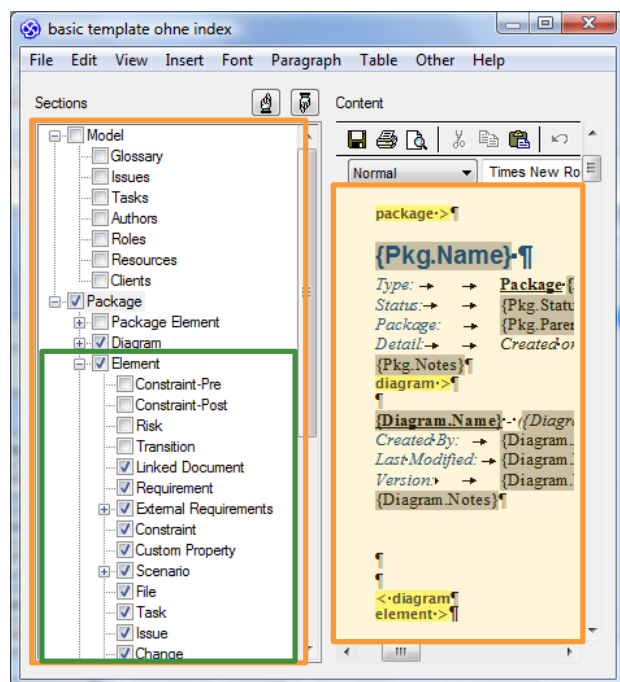


Abbildung 2: RTF Template

<sup>1</sup> What You See Is What You Get

geschweiften Klammern, z. B. {Pkg.Name} für den Namen eines Pakets.

Zum Generieren der Dokumentation muss ein Paket im *Project Browser* ausgewählt werden. Der Generator läuft, ausgehend vom selektierten Paket (z. B. *Start here* in folgender Abbildung), durch das Paket und alle enthaltenen Sub-Pakete mittels *depth-first* suche. Im RTF-Template kann definiert werden ob Sub-Packages mit ausgegeben werden sollen oder nur das aktuell ausgewählte Paket.

## Zusammengefasst

Das RTF-Template definiert WAS (welche Details), WO (and welcher Stelle im Dokument) und WIE (welche Formatierung) im RTF-Dokument ausgegeben werden soll.

Das selektierte Paket im Project Browser definiert, welcher Teilbaum des Projektes dokumentiert werden soll.

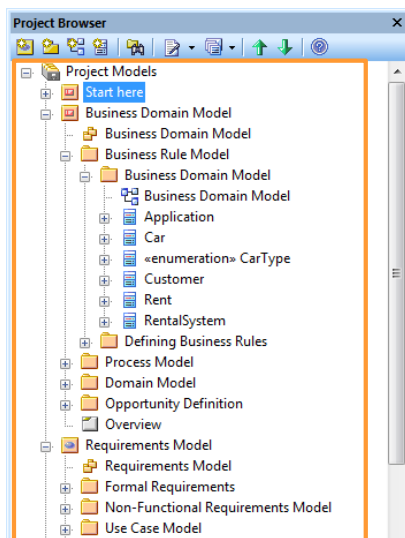


Abbildung 3: Project Browser

## Einschränkungen

1.) Im RTF-Template kann im Modellbaum (Linker Bereich in Abbildung 2) lediglich *Element* ausgewählt werden, jedoch nicht ein spezielles Element wie *UseCase* oder *Component*.

2.) Die Struktur des Project Browsers definiert die Struktur des generierten Dokuments. Dies kann erwünscht sein, falls jedoch verschiedene Sub-Pakete in verschiedenen Teilbäumen ausgegeben werden sollen, ist dies mit diesem Ansatz nicht möglich.

Diese zwei Einschränkungen bilden ein enges Korsett und verhindern eine flexiblere Auswahl der Inhalte eines Dokumentes. Um mehr Freiheitsgrade zu gewinnen, bietet Enterprise Architect zwei weitere Konzepte.

## Lösung für Einschränkung 1:

Um Einschränkung 1 entgegenzuwirken, gibt es die Möglichkeit Filter zu definieren. Diese Filter können an zwei unterschiedlichen Stellen definiert werden. Die erste Möglichkeit besteht in den Reitern *Element Filter*, *Other Filter* im Wizard *Generate RTF Document* (siehe Abbildung 1). Diese Filter können mit den restlichen Konfigurationen im Wizard für eine spätere Wiederverwendung gespeichert werden.

Alternativ können Filter auch pro RTF Template erstellt werden. Im RTF-Template Editor, im Menü: *File-> Document Options ...*

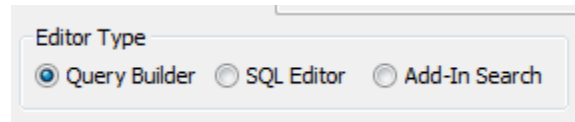
## Lösung für Einschränkung 2:

Um Einschränkung 2 entgegenzuwirken, gibt es die Möglichkeit *Model Documents* zu erstellen. *Model Documents* sind Container, in die beliebige Pakete aus dem Project Browser, per *drag and drop*, eingefügt werden können. Das Ergebnis ist eine flache Liste von Paketen, die in der modellierten Reihenfolge im generierten Dokument aufscheinen. Hat ein, im *Model Document* eingefügtes, Paket weitere Sub-Pakete, werden diese auch wie gewohnt, ausgehend vom Paket im *Model Document*, im RTF-Dokument ausgegeben. Voraussetzung ist wiederum, daß im RTF-Template die Option „child Packages“ ausgewählt wurde!

## Model Searches

In Enterprise Architect können drei verschiedene Arten von Suchen definiert werden:

- Query Builder Searches
- SQL Searches
- Add-In Searches




Am einfachsten erstellt man eine Suche mit dem *Query Builder*. Der *Query Builder* verfolgt den „query by example“ Ansatz. Dabei können beliebige Modellelemente durch Auswahl ihrer Properties und durch Angabe eines konkreten Beispielwertes aus dem Project Browser gefiltert werden. Suchen, welche über Beziehungen zu anderen Elementen hinweg reichen, können mit dem *Query Builder* nicht erstellt werden! Dafür bietet die *SQL-Suche* geeignete Mittel.

Die *SQL-Suche* erlaubt die Erstellung komplexer Abfragen auf das gesamte Daten-Repository von Enterprise Architect. Um *SQL-Suchen* erstellen zu können, benötigt man Kenntnisse über das Back-End von Enterprise Architect. Eine Beschreibung der wichtigsten Tabellen ist auf [blog.sparxsystems.de](http://blog.sparxsystems.de)<sup>2</sup> zu finden. Abfragen, welche das Modell rekursiv durchsuchen müssen, wie z. B. starte bei *Requirement 4711* und verfolge alle Realisierungsbeziehung bis keine weiteren Elemente mit einer Realisierungsbeziehung verbunden sind, können mit einer *SQL-Suche* nicht gefunden werden<sup>3</sup>. Für rekursive Suchen bietet die *Add-In Search* geeignete Mittel.

Die *Add-In Search* ist eine programmierte Suche. Das geschriebene Programm benützt das EA-Objektmodell und liefert als Ergebnis ein XML-Dokument mit allen gefundenen Elementen. Das XML-Dokument wird dem *EA.Repository* übergeben und genauso wie die Ergebnisse der anderen beiden Methoden im Enterprise Architect als Liste dargestellt.

In RTF Templates können Suchen verwendet werden um einen Filter für die Elemente, die bei der Generierung ausgegeben werden, zu definieren. Die Filter können im RTF-Template unter *File->Document Options...* ausgewählt, bzw. definiert werden. Ein RTF-Template mit einem Filter liefert nur noch einen bestimmten Ausschnitt an Modellelementen. Im Zusammenhang mit *Model Documents* wird die Mächtigkeit dieses Ansatzes erst sichtbar.

## Model Documents

Ein *Model Document* ist eine UML-Klasse mit dem Stereotyp «model document» und drei *TaggedValues* (*RTFTemplate*, *SearchName*, *SearchValue*). Im EA wird ein *Model Document* mit dem Stereotyp-Symbol  dargestellt.

Da die Struktur des Project Browsers bei Verwendung eines RTF-Templates primär die Struktur des generierten Dokumentes vorgibt, erreicht man mit dem *Model Document* neue Freiheitsgrade. Durch *drag and drop* von Paketen aus dem Projekt Browser kann dessen Struktur „aufgebrochen“ werden und eine beliebige Sicht auf den Projekt Browser gebaut werden (siehe Abbildung 4). Beinhaltet das im *Model Document* eingefügte Paket weitere Sub-Pakete, werden diese beim Generieren der Dokumentation ebenfalls in die Dokumentation ausgegeben. Das Paket im *Model Document* verhält sich beim Generieren der Dokumentation genauso wie das Paket im Project Browser.

<sup>2</sup> <http://blog.sparxsystems.de/2010/10/enterprise-architect-db-schema/>

<sup>3</sup> Rekursives SQL ist DB abhängig.

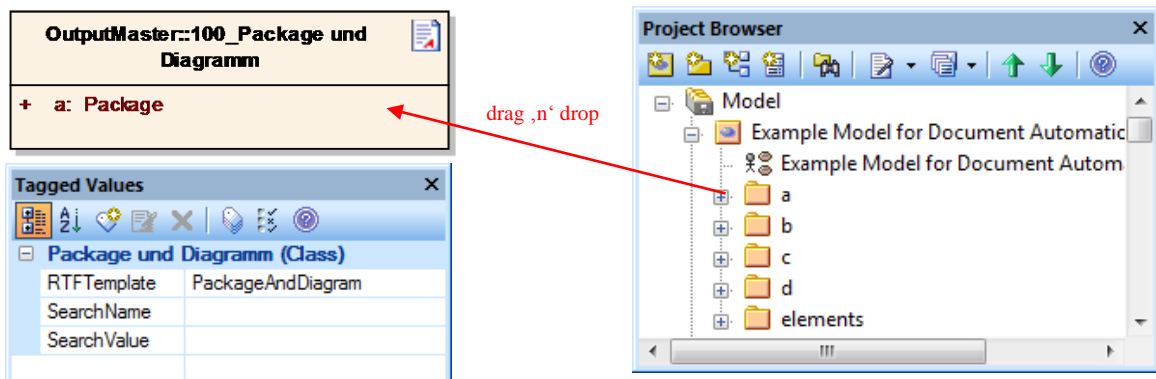
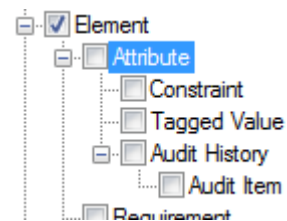


Abbildung 4: Model Documents

Pro *Model Document* kann nun ein RTF-Template ausgewählt werden. Befinden sich mehrere Pakete in einem *Model Document*, werden alle enthaltenen Pakete mit dem für dieses *Model Document* ausgewählten RTF Template generiert. Ein anderes *Model Document* kann wiederum ein unterschiedliches RTF-Template verwenden. Damit dient ein *Model Document* als Container, um Pakete neu zu strukturieren und durch verschiedene RTF Templates verschiedene Informationen aus den im *Model Document* enthaltenen Paketen auszugeben.

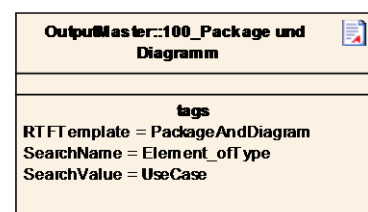
Dasselbe Paket kann auch in mehreren *Model Documents* verwendet werden. Durch Anwendung verschiedener RTF Templates an jedem *Model Document* mit demselben Paket gewinnt man wieder neue Freiheitsgrade! Durch diesen Ansatz ist es auch möglich, die im RTF-Template definierte Struktur aufzubrechen.



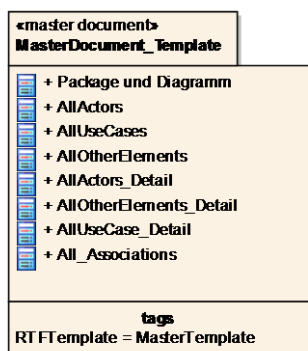
Ein *Model Document* kann z. B. ein Template enthalten, in dem alle Elemente ausgegeben werden, sonst nichts. Ein weiteres *Model Document* kann nun dasselbe Paket beinhalten und ein anderes RTF-Template verwenden, in dem alle Attribute ausgegeben werden, aber nicht deren Elemente. Dadurch wird die Containment Beziehung zwischen Element und dessen Features aufgebrochen. Das ist ein extremes Beispiel und wird wohl kaum Anwendung finden.

Ein wahrscheinlich interessanterer Anwendungsfall ist die Kombination von RTF Templates mit Filtern.

Ein Filter wird durch eine Modell Search definiert und kann wie oben beschrieben direkt in einem Template definiert werden (im RTF-Template: *File->Document Options*).



Alternativ dazu kann ein Filter auch direkt an ein Model Document geheftet werden. Das TaggedValue *SearchName* bietet eine Liste im Modell vorhandener Modellsuchen zur Auswahl. Wir im *Model Document* eine Suche ausgewählt, darf kein Paket im *Model Document* enthalten sein, sonst greift die Suche nicht und der gesamte Inhalt der im *Model Document* enthaltenen Pakete wird bei der Generierung der Dokumentation ausgegeben. Wenn es sich um eine „generische“ Suche handelt, kann diese durch Angabe eines SearchValue definiert werden.



Werden mehrere *Model Documents* erstellt, können alle *Model Documents* in ein *Master Document* gebündelt werden. Ein *Master Document* ist ein Paket mit dem Stereotyp «master document». Die Reihenfolge der *Model Documents* im *Master Document* ist

ausschlaggebend auf den Content des generierten Dokumentes. Alle Pakete werden pro Model Document von oben nach unten ausgegeben, ebenso wird beim ersten *Model Document* im *Master Document* begonnen, bis das letzte *Model Document* generiert wurde.

Das *Master Document* hat ebenfalls ein TaggedValue RTFTemplate. Mit dem RTF-Template im *Master Document* wird die „Hülle“ des generierten Dokumentes erstellt. Ein *Master Document* kann nun für ein oder mehrere Kapitel stehen oder viel fein granularer definiert werden und für einen Absatz in einem Kapitel stehen.

Werden pro *Model Document* mehrere Pakete eingefügt und sind diese Pakete in der Projekt Browser Struktur an oberer Ebene, sind diese wahrscheinlich „fix“ und werden sich wenig ändern. Mit diesem Ansatz können nun beliebig viele *Master Documents* erstellt werden. Die einzelnen *Model Documents* bilden eine unterschiedliche Sicht auf den Projekt Browser. Durch unterschiedliche RTF Templates an den *Model Documents* wird die gewünschte Information an der gewünschten Stelle ausgegeben.

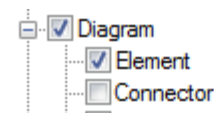
Da *Model Documents* lediglich Container mit keiner weiteren Semantik sind, können sie verwendet werden um die Struktur des Project Browser beliebig fein aufzuspannen. Durch die sehr feine Granularität der *Model Documents* verliert man allerdings Flexibilität bei der Anwendung der *Model Documente*, da Pakete aus dem Projekt Browser in das *Model Document* hineingezogen werden und somit fix verdrahtet sind.

## Einschränkungen von Model Documents

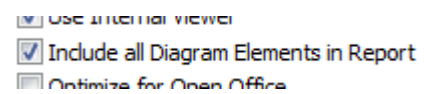
Durch die fixe Verdrahtung von *Model Documents* und Paketen im Project Browser verliert man Flexibilität beim Anwenden der definierten *Model Documents* auf unterschiedliche Teile des Project Browsers. Ein kleines Beispiel illustriert die Problematik.

Folgende Aufgabe ist zu lösen:

In einem Paket sind Views enthalten (Diagramme, welche Elemente aus anderen Paketen referenzieren). Das Paket selbst enthält keine Elemente. Durch RTF Templates ist es möglich auch Elemente, welche „nur“ im Diagramm verlinkt sind, aber nicht physisch im Paket enthalten sind, für das gerade die Dokumentation generiert wird.



Notwendige Konfiguration ist im RTF-Template die

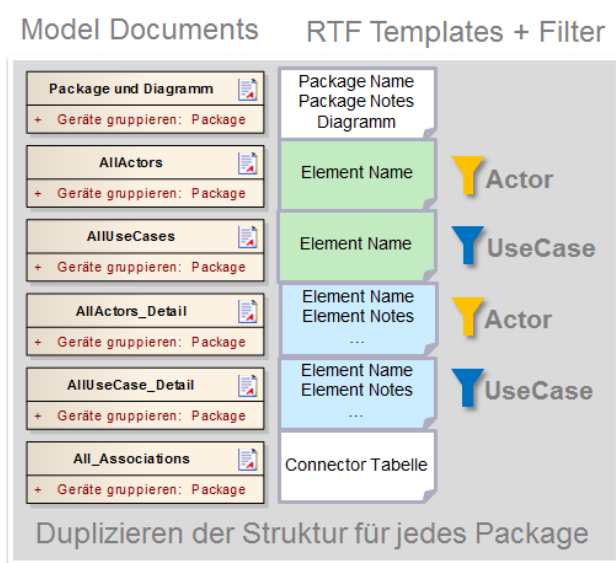


Auswahl der Checkboxen *Diagram::Element*. Damit werden alle Elemente die lediglich im Diagramm als link verfügbar sind ebenfalls in die Dokumentation ausgegeben. Zusätzlich zu der Konfiguration im Template ist die Einstellung „Include all Diagram Elements in Report“ im RTF-Generator notwendig.

Der Inhalt dieses Paketes soll nun folgende Struktur aufweisen: Der erste Abschnitt beinhaltet Informationen aus dem Paket sowie das Diagramm aus dem Paket.

Anschließend werden alle Akteure sowie alle Use Cases überblicksartig aufgelistet. Nach diesen zwei Abschnitten wiederholen sich beide. Dieses Mal allerdings mit allen Details. Abschließend werden alle Beziehungen zwischen den im Diagramm enthaltenen Beziehungen in Form einer Tabelle

ausgegeben. Also Beziehungen zwischen Akteure und Use Cases. Diese Struktur wiederholt sich nun für jedes Paket im Projekt Browser, in dem ein Diagramm liegt.



Durch die Flexibilität, die wir durch die *Model Documents* erlangt haben, können wir diese Anforderung lösen. Die korrespondierende *Model Document* Struktur sieht folgendermaßen aus:

Das erste *Model Document* verwendet ein Template indem lediglich der Name als Kapitelüberschrift, Paket Notizen als Content und das Diagramm ausgegeben wird. Die Übersicht aller Akteure und Use Cases wird durch ein Template realisiert. Das Template liefert lediglich den Namen des Elements (als Aufzählungsliste). Da der Filter am *Model Document* nicht den Inhalt der enthaltenen Pakete filtert, muß das RTF-Template

dupliziert werden und jeweils der verwendete Filter ausgetauscht werden. Dasselbe geschieht mit den Details der Akteure und Use Cases. Das verwendete Template gibt aber diesmal zusätzlich zum Elementnamen auch die Notizen aus. Wiederum muß das Template dupliziert werden und der Filter ausgetauscht werden. Abschließend werden alle Beziehungen zwischen den Elementen die in dem Diagramm sichtbar sind durch ein weiteres RTF-Template ausgegeben.

Die *Model Documents* dienen in diesem Beispiel als Container des identischen Paketes, lediglich die RTF Templates sind jeweils unterschiedlich. Die oben definierte Anforderung konnte somit mit *Model Documents* + RTF Templates + Filter realisiert werden. Ein „Schönheitsfehler“ fällt allerdings auf. Die *Model Document* Struktur funktioniert genau für ein Paket. Will man nun diese Struktur auf mehrere Pakete des Project Browsers anwenden, muss man die gezeigte Struktur für jedes gewünschte Paket im Project Browser duplizieren!

Das Duplizieren dieser Struktur kann manuell geschehen oder man verwendet ein weiteres Feature von Enterprise Architect. Model Scripts ist das Mittel der Wahl. Durch Scripting des EA Objektmodelles können beliebige Automatismen im EA etabliert werden. Alternativ zum Scripting wäre die Erstellung eines Add-In ebenfalls möglich. Dieser Artikel verfolgt den Ansatz zu Skripten, da die Skripte leichter anpassbar sind und somit noch mehr Flexibilität beim Ändern des Codes bieten als ein Add-In. Auf komfortable *Wizards* und GUIs muß bei der Verwendung von Skripten allerdings verzichtet werden.

Bevor nun ein Skript geschrieben werden kann, welches sich um die Automatisierung der Kopierarbeit kümmert, muss zuerst definiert werden Was, Wohin und Wie kopiert werden soll. Dies kann natürlich „hard-gecoded“ geschehen oder man verwendet einen modellgetriebenen Ansatz.

## Ein Schema für Model Documents

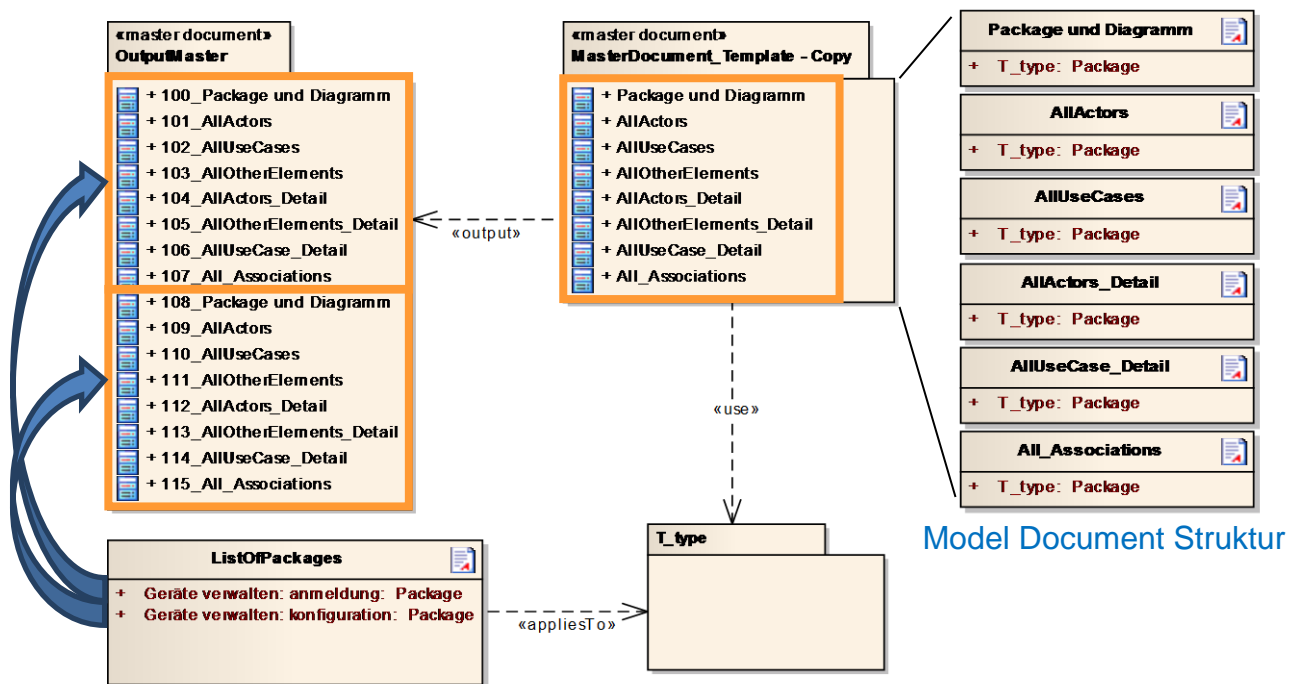
Um noch mehr Flexibilität zu erlangen, wird die Konfiguration Was, Wohin und Wie kopiert werden, soll in Form eines Modelles beschrieben. Das Script interpretiert nun das Modell und verwendet es als Konfiguration für den Kopiervorgang.

Das ausgangs *Model Document* muß nun „generisch“ werden. Dies ist realisierbar indem anstelle eines „konkreten“ Paketes aus dem Projekt Browser ein „dummy“ Paket aus dem Project Browser gewählt wird. Dieses T\_type Paket dient als Platzhalter in der zuvor definierten *Model Document* Struktur. Das

Master Document *MasterDocument\_Template – Copy* beinhaltet die generische *Model Document* Struktur.

Die anzuwendenden Pakete werden ebenfalls in einem *Model Document (ListOfPackages)* gesammelt. Diese werden wie gewohnt per *drag and drop* aus dem Project Browser in das *Model Document* gezogen. Durch Dependencies mit den Stereotypen *appliesTo* und *use* wird der Bezug zwischen *ListOfPackages* und dem generischen *Master Document* hergestellt. Das Script sucht, ausgehend vom *Master Document MasterDocument\_Template – Copy* alle nötigen Modellelemente (*T\_type*, *ListOfPackages*), sowie das *Master Document OutputMaster*, welches mit einer Dependency mit Stereotyp *output* verbunden ist, in das die kopierte Struktur eingefügt wird.

Der Algorithmus ist nun folgender: Für jedes Paket (Attribut im *Model Document ListOfPackages*) wird das *MasterDocument\_Template – Copy* Paket durchlaufen und jedes *Model Document* kopiert und das Paket (Attribut) *T\_type* durch das selektierte Paket der *ListOfPackages* substituiert. Die resultierende Struktur wird in das Paket *OutputMaster* kopiert. Damit die Reihenfolge der Pakete gewahrt bleibt, muss ein Zähler alle Pakete nummerieren.



Für jedes Package in der Liste wird ein „Model Document Struktur“ erstellt

Nachdem das *OutputMaster* Paket frisch gefüllt wurde, kann es selektiert werden und für dieses Paket der RTF-Generator aufgerufen werden (selektieren des Paketes + F8 drücken).

Durch den „Modellgetriebenen“ Ansatz und durch Verwendung von Model Scripts erhält man die Flexibilität, die Art und Weise der Generierung leicht zu beeinflussen. Das Modell könnte z. B. komplexer werden und aus mehreren „dummy“ Templates bestehen, bzw. zusätzliche Metainformationen können im Modell definiert werden um noch komplexere Strukturen und Dokumente zu erstellen. Das Script kann dann leicht den geänderten Modellen angepasst werden.

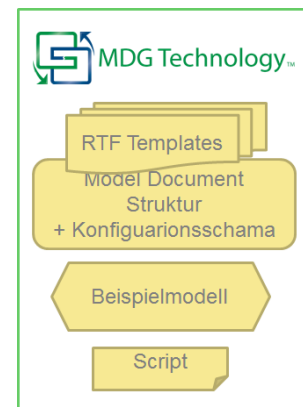


Vorstellbar ist z. B. die Definition einer Pfadangabe pro *Model Document* im Notizfeld, um zu definieren, daß ein weiteres „dummy“ Paket eingebunden werden soll. Dieses fungiert dann als Kapitelüberschrift und wird aus der Project Browser Struktur abgeleitet.

Dieser Ansatz funktioniert recht gut und bringt weitere Freiheitsgrade bei der Generierung von RTF Dokumenten mit Enterprise Architect. Ein Problem besteht allerdings noch. Wir haben viele verschiedene RTF Templates verwendet, eine *Model Document* Struktur + „Konfigurations-Modell“ erstellt, sowie ein doch schon umfangreiches Script. Um diesen Ansatz leicht auf beliebige Dokumente anwenden zu können, müssen viele Handgriffe getan werden. Doch EA bietet ein weiteres Feature, um auch diesem Problem zu begegnen.

## Bündeln in MDG Technology

In MDG Technologies können viele benutzerspezifische Anpassungen gebündelt werden. Die einzelnen Artefakte, welche in die Technologie aufgenommen werden sollen, können mit dem MDG Technology Wizard direkt aus dem EA-Projekt ausgewählt werden (RTF Template, Script, etc.) oder müssen zuvor als XML Dokument vorliegen. *Model Document* Struktur und Konfigurationsschema, sowie das Beispielmodell müssen als XMI exportiert werden. Diese XML-Dokumente müssen nun manuell in das vom MDG Technology Wizard generiert .MTS File eingetragen werden. Das .MTS File ist das Konfigurationsfile der Technologie. Die einzelnen Elemente werden als Model Template in die Technologie aufgenommen. Wie dies zu konfigurieren ist, steht detailliert in der Hilfe. Der Eintrag kann leicht gefunden werden wenn nach dem String „Incorporate Model Templates“ gesucht wird.



Nächste Herausforderung ist das Verteilen der erstellten MDG Technology, doch auf dafür gibt es eine Lösung.

## Verteilen von MDG Technologies

Eine MDG Technology kann bei jedem Start von EA dynamisch geladen werden. Dazu ist lediglich eine einmalige Konfiguration notwendig. Unter *Settings->MDG Technologies...->Advanced...* kann eine URL bzw. ein File-Path ausgewählt werden. Ist die URL/File-Path zugänglich und befindet sich dort die erstellte Technologie, dann wird sie beim Laden von EA automatisch mitgeladen. Best Practice Lösung zum Verteilen ist die Verwaltung der erstellten MDG Technology unter Versionskontrolle. Lokal hat jeder Benutzer ein Working-Directory indem die Technologie ausgecheckt wird. Das lokale Working-Directory ist immer verfügbar und daher wird die MDG Technology auch immer geladen. Im Fall einer URL/File-Path der vorübergehend nicht erreichbar ist, führt zu einer fehlenden MDG Technology.

