# APEX Dynamic Controller (ADC) – Technical Documentation

Das APEX Dynamic Controller (ADC) ist ein APEX Dynamic-Action Plugin zur Erstellung dynamischer APEX Anwendungsseiten ohne den ansonsten notwendigen Overhead vieler Dynamic Actions zur Verwaltung des Status der Formularelemente.

Das Plugin kontrolliert die Anwendungsseite durch die Einrichtung von Anwendungsfällen (Use Cases) in einer separaten APEX-Anwendung. Diese Anwendung verwaltet die Anwendungsfälle für alle APEX-Anwendungen eines Workspaces, die mit ADC erweitert werden sollen. Ein Anwendungsfall folgt dabei einer spezialisierten Version des IFTTT-Patterns (IF THIS THEN THAT), indem es den Anwendungsbenutzer in den Fokus stellt:

Wenn der Anwender (IF THIS), dann (THEN THAT).

Für einen Anwendungsfall können beliebig viele Aktionen definiert werden, die Aktionen sowohl innerhalb der Datenbank, als auch auf der Anwendungsseite nach sich ziehen können. Eine Aktion kann also zum Beispiel den Sessionstatus der Anwendung innerhalb der Datenbank ändern, als auch die Sichtbarkeit von Anwendungselementen auf der APEX-Anwendungsseite kontrollieren. Hierfür stellt das Plugin eine Auswahl vordefinierter Aktionstypen bereit, die durch den Entwickler auf die eigenen Bedürfnisse hin erweitert werden können.

Ändert ein Anwendungsfall den Sessionstatus, kann dies zur Folge haben, dass ein weiterer Anwendungsfall ausgelöst wird. Auf diese Weise können die Anwendungsfälle passgenau und kompakt formuliert werden, ohne mehrfach zwischen Client und Server zu traversieren. Alle Anwendungsfälle werden rekursiv in einem Arbeitsgang abgearbeitet und eine kombinierte Antwort mit Anweisungen für die Anwendungsseite zusammengestellt.

Das Plugin hat folgenden Funktionsumfang:

* Verwaltung des Sichtbarkeitsstatus von Formularelementen (aktiv, deaktiv, sichtbar, versteckt)
* Validierung der Eingaben, Ausgabe entsprechender Fehlermeldungen
* Verwaltung des Session State, Ermittlung von Elementwerten basierend auf der Auswahl des Benutzers der Anwendung
* Rekursive Regelauslösung
* Einfache Administration: Das Plugin erfordert keinerlei Administration auf der Anwendungsseite, alle notwendigen Eventhandler, die Auswahl der zu übertragenden Elementwerte und alle weitere Konfiguration wird automatisch aus den eingegebenen Regeln abgeleitet

Alle Funktionen des Plugins werden dynamisch über AJAX-Aufrufe angeboten, ein erneutes Laden der Anwendungsseite erfolgt nicht.

## Konzept des Plugins

Das Plugin implementiert Anwendungsfälle mit Hilfe einer Entscheidungstabelle, die den aktuellen Sessionstatus (hier *Anwendungsstatus* genannt) gegen die definierten Anwendungsfälle auswertet. Der Anwendungsstatus enthält die aktuellen Benutzereingaben, zusammen mit Metainformationen über das ausgelöste Ereignis (welches Element hat das Ereignis ausgelöst, welches Ereignis wurde ausgelöst etc.). Beim Aufruf des Plugins wird der Anwendungsstatus an den Controller gesendet.

Die Anwendungsfälle sind als Regel der Entscheidungstabelle implementiert, in der der IF THIS-Anteil durch eine WHERE-Klausel ausgedrückt wird (*technische Bedingung*). Die technische Bedingung hat Zugriff auf den Anwendungsstatus, indem dieser als Spalten der Entscheidungstabelle zur Verfügung gestellt wird. Existiert also auf der Anwendungsseite ein Eingabefeld mit dem Namen P10\_ENAME, wird der Entscheidungstabelle eine Spalte P10\_ENAME mit dem aktuellen Sessionwert angeboten. Erforderliche Typkonvertierungen in Zahl- oder Datumswerte sind bereits erfolgt. Auf diese Weise kann eine Datumsspalte direkt gegen z.B. SYSDATE verglichen werden.

Beispiel:

Der Anwendungsfall »*Der Benutzer erstellt eine Firmenadresse*« wird durch die technische Bedingung P10\_ADDRESS\_TYPE = `J` ausgedrückt.

Das Plugin ermittelt den ersten, passenden Anwendungsfall aus dem gegebenen Sessionstatus und führt die Aktionen aus, die in diesem Anwendungsfall hinterlegt sind. Die Reihenfolge der einzelnen Anwendungsfälle kann kontrolliert werden, um speziellere Anwendungsfälle vor allgemeineren Anwendungsfällen für den gleichen Sessionstatus auszuführen.

Eine Anwendungsfall besteht, neben der technischen Bedingung, aus einer Sammlung von Aktionen, die ausgeführt werden sollen. Das Plugin stellt eine vordefinierte Menge von Aktionstypen zur Verfügung, die entweder innerhalb der Datenbank als PL/SQL-Aktionen, oder im Browser, als JavaScript-Aktionen, ausgeführt werden. Eigene Aktionstypen können durch Metadaten erstellt werden. Hierbei muss der Bezeichner des Aktionstyps, eine eventuelle PL/SQL-Aktivität und/oder eine JavaScript-Methode definiert werden. Etwaige Parameter des Aktionstyps lassen sich frei definieren (bis zu drei Parameter pro Aktionstyp) und durch Ersetzungszeichenfolgen in den PL/SQL- und JavaScript-Methoden referenzieren.

Das Plugin übermittelt bei jedem Aufruf nur den relevanten Anwendungsstatus. Als »relevant für den Anwendungsstatus« gelten die Seitenelemente, die in mindestens einer technischen Bedingung des Anwendungsfalls angesprochen werden. Ist also eine technische Bedingung definiert als P1\_VALID\_FROM < P1\_VALID\_TIL, erkennt das Plugin die beiden Elemente P1\_VALID\_FROM und P1\_VALID\_TIL als relevant. Extrahierte Elemente müssen auf der APEX-Anwendungsseite vorhanden sein, oder als Anwendungselemente zur Verfügung stehen. Wird ein Element verwendet, dass nicht vorhanden ist, validiert die technische Bedingung nicht.

Dass die relevanten Elemente extrahiert werden, hat zur Folge, dass ...

* diese Elemente durch das Plugin mit einem Eventhandler für den CHANGE-Event (oder CLICK-Event, wenn es sich um eine Schaltfläche handelt) versehen werden. Dadurch wird die Verarbeitung der geänderten Eingaben dieser Elemente oder der Klick auf die Schaltfläche durch das Plugin ausgelöst.
* beim Aufruf des Plugins die aktuellen Elementwerte dieser Elemente in den Anwendungsstatus kopiert werden. Dadurch stehen die aktuellen Elementwerte der SQL-Entscheidungstabelle zur Filterung durch die technischen Bedingungen zur Verfügung.

Sollte das Plugin den Sessionstatus ändern (Hier ist nicht nur der Anwendungsstatus, sondern alle Seitenelementwerte der aktuellen APEX-Anwendungsseite), werden diese in der Antwort des Plugins an die Anwendung übergeben und durch das Plugin in die entsprechenden Formularelemente eingefügt. Sind relevante Anwendungselemente von einer solchen Änderung betroffen, werden die zugehörigen Anwendungsfälle rekursiv abgearbeitet.

## Abgrenzung zu Dynamischen Aktionen (DA)

Auf den ersten Blick sind die Gemeinsamkeiten zu dynamischen Aktionen recht groß, doch bestehen fundamentale Unterschiede:

* ADC arbeitet serverseitig, DA clientseitig. Da die meisten Anwendungsfälle Daten aus der Datenbank benötigen (zum Beispiel zur Validierung oder zur Entscheidungsfindung), ist ADC in diesen Einsatzszenarien günstiger
* ADC integriert das Binden von Seitenevents, in DA muss ein passender Event gebunden werden. Dies reduziert das Risiko schwer zu findender Fehler durch fehlerhaftes Binden an Ereignisse
* ADC konzentriert sich auf die Geschäftslogik zur Verwaltung des Seitenstatus, DA sind eine Allzweckwaffe zur Vereinfachung clientseitigen Codes. Durch die Konzentration auf diesen Einsatzzweck ist ADC deutlich einfacher zu verwenden als DA
* ADC arbeitet mengenorientiert die Regeln ab und unterstützt rekursive Regelausführung, DA prüfen »one by one« und sind prinzipiell nicht imstande, Rekursion in der Datenbank auszuführen
* ADC benötigt im Regelfall keinerlei JavaScript-Programmierung und keinen clientseitigen Code durch den Anwender, DA müssen zumindest clientseitig eingerichtet und gebunden werden.

Aus dieser Gegenüberstellung kann keine Kritik an Dynamischen Aktionen abgeleitet werden, sie sind einfach für einen generischeren Einsatzzweck gedacht und schon von daher bei der konkreten Verwendung in einem gegebenen Szenario nicht immer einfacher oder besser als eine spezialisierte Lösung.

Einfache Dynamic Actions lassen sich aber letztlich durch ADC schneller und einfacher realisieren, denn es entfällt die Bindung des Events und die damit verbundenen Probleme, zudem ist das Erfassen einer Regel schneller und einfacher erledigt als die Anlage einer Dynamic Action.

## Verwendung des Plugins

Dieser Abschnitt beschreibt die grundsätzliche Arbeitsweise und Verwendung des ADC Plugins.

### ADC-Plugin

In der APEX-Anwendung, die durch ADC kontrolliert werden soll, ist die Installation eines Dynamic Action-Plugins erforderlich. Es enthält zwei Parameter auf Anwendungsebene, die vor der Verwendung eingestellt werden müssen.

#### Parameter »ADC Apexfunktionen«

Das Plugin unterstützt die Verwendung mit mehreren APEX-Versionen und eigenen oder geänderten Themes. Die erforderliche Adaptierung der ADC-Funktionen an die APEX-Version bzw. das verwendete Theme erfolgt über eine JavaScript-Datei für die entsprechende Kombination von Theme und Version. ADC liefert eine Adapterdatei für die Verwendung mit Theme 42 in mehreren Versionen mit. Die Versionen sind als Namensraumobjekte innerhalb dieser Adapterdatei realisiert und können über ihren Namensraum mit diesem Parameter ausgewählt werden. Verwenden Sie zum Beispiel APEX in Version 20.2, können Sie hier den Eintrag de.condes.plugin.adc.apex\_42\_20\_2 wählen. Als fortgeschrittene Verwendung des Plugins können Sie eigene Adapterdateien schreiben. Dies ist erforderlich, wenn Sie

* Für eigene Aktionstypen eine entsprechende Funktionalität in JavaScript vorhalten müssen
* Ein stark angepasstes Theme verwenden, das mit der Standardfunktionalität des Plugins nicht korrekt funktioniert.

Im Regelfall erben Sie ein möglichst passendes Namensraumobjekt und erweitern dieses um die geänderte oder neue Funktionalität.

#### Parameter »Kommentare ausgeben«

Dieser Parameter kontrolliert, ob Sie weitergehende Kommentare auf der Konsole des Browsers ausgeben möchten, wenn Sie mit SCT arbeiten. Diese Funktionalität ist insbesondere auf Entwicklungsrechnern sinnvoll, um die korrekte Funktion zu überwachen. Auf produktiven Servern kann diese Einstellung deaktiviert werden, um Bandbreite zu sparen.

#### Einbindung des Plugins

Das Plugin muss auf der Anwendungsseite lediglich beim Seitenladen aufgerufen werden, es ist keine Parametrierung erforderlich. Alternativ zur Einbindung des Plugins auf jeder zu kontrollierender Seite kann es auch auf Seite 0 eingefügt und somit für jede Anwendungsseite eingebunden werden. Eine serverseitige Bedingung kann die Ausführung des Plugins für eine Liste von Anwendungsseite unterbinden.

### Erfassung von Anwendungsfällen

Anwendungsfälle werden in der separat mitgelieferten APEX-Anwendung erfasst. Ist das Plugin auf einer Anwendungsseite installiert und wird diese Anwendungsseite zum ersten Mal aufgerufen, wird im Datenmodell von ADC ein Eintrag für diese Anwendungsseite hinzugefügt.

Auch ohne die Erfassung von Anwendungsfällen für eine Anwendungsseite verhält diese sich bereits anders als eine statische APEX-Anwendungsseite. Insbesondere werden alle Pflichtfelder (gekennzeichnet durch ein Labeltemplate, das die Bezeichnung REQUIRED enthält) bereits dynamisch geprüft. Wird ein Eingabefeld gefüllt und anschließend geleert, wird beim Verlassen des Eingabefeldes eine generische Fehlermeldung ausgegeben. Zudem werden Eingabefelder, die eine Zahl oder ein Datum enthalten, dynamisch daraufhin validiert, ob ein entsprechender Datentyp in ihnen enthalten ist. NULL-Werte werden toleriert. Die Konvertierung von Zahl- und Datumsfeldern erfolgt hierbei, falls vorhanden, durch die hinterlegten Formatmasken, im Fall eines Datumsfeldes alternativ durch das Datumsformat der Anwendung.

Hinweis:

Wenn eine dynamische Anwendungsseite erstellt wird, sollte diese vollumfänglich für die Validierung der Anwendungsseite verantwortlich sein. Stellen Sie insbesondere sicher, dass die Option, das ein Eingabefeld erforderlich ist, abgewählt ist, um ein unvorhergesehenes Verhalten zu vermeiden. Ebenfalls sollten diese Seiten keine Dynamischen Aktionen mehr enthalten, die den Status der Anwendungsseiten ändern. Verzichten Sie in diesem Zusammenhang auch auf kaskadierende Auswahllisten, Master-Detail-Berichte mit deklarativer Aktualisierung und ähnliche Einstellungen. Da diese Einstellungen über Eventhandler auf der Anwendungsseite ausgeführt werden, sind gegenseitige Beeinflussungen nicht auszuschließen und erschweren zudem das Verständnis der Vorgänge auf der Anwendungsseite.

Nun können für diese Seite Anwendungsfälle erfasst werden. Vordefiniert ist ein Anwendungsfall »*die Seite öffnet*« mit der technischen Bedingung INITIALIZING = C\_TRUE.

Hinweis:

Die Bezeichner der Anwendungsfälle sollten im Kontext »*Wenn der Anwender …*« benannt werden, um das IFTTT-Konzept umzusetzen.

Wie bereits beschrieben, umfasst der Anwendungsstatus Metainformationen zum aufgetretenen Ereignis. Beim Laden der Seite ist kein Benutzerereignis (etwa die Änderung eines Eingabefeldes oder der Klick auf eine Schaltfläche) vorhanden. Dies wird durch die Spalte INITIALIZING gekennzeichnet, die in diesem Fall den Wert C\_TRUE enthält. C\_TRUE und C\_FALSE sind weitere Spalten mit Wahrheitswerten, die Sie für Prüfungen verwenden können. Welche Werte diese Spalten repräsentieren, ist abhängig von der Installation des ADC (siehe Installation). Auf ähnliche Weise werden weitere Spalten angeboten, zum Beispiel je eine Spalte für die Schaltflächen, die den Wert C\_TRUE enthalten, wenn sie geklickt wurden, oder eine Spalte FIRING\_ITEM, die den Namen des auslösenden Seitenelements enthält. Im Fall der Initialisierung der Seite enthält FIRING\_ITEM den Wert DOCUMENT. Eine vollständige Liste der Spalten finden sie im Anhang.

Ein Anwendungsfall hat eine Bezeichnung und eine technische Bedingung, die eng miteinander verzahnt sind. Die technische Bedingung legt fest, woran ein Anwendungsfall aus den Daten der Anwendung zu erkennen ist. Dies kann ein einfacher Vergleich eines Eingabefeldes mit einem konstanten Wert sein, aber auch das Ergebnis eines Funktionsaufrufes in PL/SQL, das für den aktuellen Anwendungsstatus berechnet wurde.

Da die Syntax der technischen Bedingung einer WHERE-Klausel entspricht, sind beliebig komplexe boolesche Ausdrücke möglich. Komplexe Bedingungen sollten allerdings nicht hier ausgedrückt, sondern in einer Funktion gekapselt werden. Die üblichen Bedenken bezüglich des Aufrufs von PL/SQL-Funktionen in der WHERE-Klausel einer SQL-Anweisung kommen hier nicht in Betracht, weil sichergestellt ist, dass die SQL-Abfrage stets nur eine Zeile enthält, die durch die Regeln geprüft werden. Dies ist so, weil die Abfrage sich stets auf den Anwendungsstatus bezieht, der nur eine Zeile enthält. Umgebungswechsel sind also nur in geringer Anzahl zu erwarten.

Zu einem Anwendungsfall lassen sich beliebig viele Aktionen erfassen, die in der Reihenfolge der Sortierung ausgeführt werden. Die Aktionstypen unterstützen die Erfassung durch passgenaue Parameter sowie durch Hilfetexte, die für jeden Aktionstyp dynamisch eingeblendet werden und die verschiedenen Optionen erläutern.

Sind Anwendungsfälle und Aktionen erfasst, werden diese unmittelbar wirksam. Beim Erfassen eines neuen Anwendungsfalls kann es erforderlich sein, die APEX-Anwendungsseite neu zu laden, um sicherzustellen, dass alle Ereignishandler korrekt gebunden sind. Änderungen der Aktionen erfordern kein erneutes Laden der Seite, sondern werden unmittelbar berücksichtigt.

### Logische Bausteine und Datenmodell

ADC ist als eigenständige APEX-Anwendung implementiert und benötigt daher ein eigenes Datenmodell sowie ein separat zu installierendes Plugin für die APEX-Anwendungen, die durch ADC kontrolliert werden sollen.

#### Verwendete Namenskonventionen

Das Datenmodell verwendet eine Namenskonvention, die einerseits die zusammengehörigen Tabellen beieinander hält, zum anderen optimal für APEX-Anwendungen vorbereitet ist und zudem das Verständnis erleichtert.

Alle Tabellennamen beginnen mit dem Kürzel ADC. Jede Tabelle definiert ein Kürzel aus drei Buchstaben, dass jedem Spaltennamen vorangestellt wird. Die Kürzel werden bei der Besprechung der Einzeltabellen angegeben.

Jede Tabelle umfasst eine Spalte <Tabellenkürzel>\_ID als Primärschlüssel, sofern sinnvoll. Fremdschlüsselbeziehungen werden in der Form <Tabellenkürzel>\_<Referenziertes Tabellenkürzel>\_ID benannt. Auf diese Weise ist die Funktion am Spaltennamen abzulesen, zudem sind alle Spaltennamen eindeutig.

Constraints werden nach zwei Konventionen benannt:

* Primärschlüsselconstraints haben den Namen <Tabellenname>\_PK
* Alle anderen Constraints haben den Namen <Spaltenname>\_<Constraintkürzel>

Constraintkürzel können zum Beispiel CHK für Check-Constraints oder FK für Fremdschlüsselconstraints lauten.

#### Regelgruppe, Tabelle ADC\_RULE\_GROUPS (CGR)

Soll auf einer APEX-Anwendungsseite die Kontrolle des Status der Formularelemente, Regionen und Schaltflächen durch das Plugin ADC übernommen werden, muss dies in den Metadaten von ADC hinterlegt werden.

Hinweis:

Eine dynamische Anwendungsseite wird intern als *Regelgruppe* bezeichnet, die Anwendungsfälle als *Regeln*. Dies ist der technischen Umsetzung als Entscheidungstabelle geschuldet.

Eine Regelgruppe umfasst

* eine technische ID
* eine Referenz auf die APEX-Anwendung und -Seite, jeweils über deren ID
* eine Spalte zur Aufnahme der Decision Table-Logik
* eine Spalte zur Aufnahme etwaiger Initialisierungslogik
* Flags zur Kontrolle des Verhaltens, etwa, ob eine Regelgruppe ausgeführt werden soll oder nicht und ob eine rekursive Ausführung erlaubt ist.

Aufgabe der Regelgruppe ist ausschließlich die Gruppierung von Regeln. Sie stellt einen Zuständigkeitsbereich dar, innerhalb dessen Regeln und deren Abhängigkeiten analysiert werden. Regelgruppen werden in der Tabelle ADC\_RULE\_GROUPS gespeichert.

#### Regel, Tabelle ADC\_RULES (CRU)

Der Regelgruppe werden im folgenden Regeln zugeordnet, die wiederum aus einer technischen ID, einem beschreibenden Namen, einer Bedingung und einem Sortierkriterium bestehen. Die Regeln referenzieren die Regelgruppen, zu denen sie gehören, über deren technische ID und dienen gleichzeitig als Referenz für beliebig viele Aktionen, die ausgeführt werden sollen, wenn die Bedingung der Regel zu TRUE evaluiert und das Plugin diese Regel zur Ausführung auswählt.

Zusätzlich pflegt ADC für jede Regel eine Liste mit den Namen der Formularelemente oder -schaltflächen, die in der Bedingung referenziert werden. Die Liste dieser referenzierten Elemente stellt in ihrer Gesamtheit die Liste der »relevanten« Elemente dieser Regelgruppe dar. Regeln werden in der Tabelle ADC\_RULES gespeichert.

#### Aktionen, Tabelle ADC\_RULE\_ACTION (CRA)

Jeder Regel werden Aktionen zugeordnet, die ausgeführt werden sollen, wenn die Regel zu TRUE evaluiert und das Plugin diese Regel zur Ausführung auswählt. Die Aktionen referenzieren einen *Aktionstyp*, der in Tabelle ADC\_ACTION\_TYPE hinterlegt ist, sowie die Regel und die Regelgruppe, zu der sie gehören. Zudem referenzieren sie ein Seitenelement, auf das sich die Aktion bezieht. Soll also z.B. der Wert des Formularelements P1\_VALID\_TIL auf den Wert NULL gesetzt werden, referenziert diese Aktion das Element P1\_VALID\_TIL und ruft den Aktionstyp SET\_NULL auf, der wiederum definiert, wie ein Elementwert auf den Wert NULL zu setzen ist. Auch die Aktionen verfügen über ein Sortierkriterium, das steuert, in welcher Reihenfolge Aktionen ausgeführt werden sollen.

Aktionen enthalten zwei optionale Attributwerte in den Spalten ATTRIBUTE und ATTRIBUTE\_2. Diese Attribute können durch die Aktionstypen referenziert werden (siehe dort) und dienen der Übermittlung von Parametern an den PL/SQL oder JavaScript-Code.

Aktionen werden in der Tabelle ADC\_RULE\_ACTIONS gespeichert.

#### Aktionstypen, Tabelle ADC\_RULE\_ACTION\_TYPES (CAT)

Eine Aktion hat einen Aktionstyp. Dieser definiert, wie eine Aktion implementiert werden soll. Ein Aktionstyp umfasst einen technischen, alphanumerischen Schlüssel, eine Referenz auf ein übersetzbares Element zur Darstellung auf der Oberfläche, eine Implementierungsanweisung für die Sprache PL/SQL sowie eine Implementierungsanweisung für die Sprache JavaScript. Die Implementierungsanweisungen sind optional und können gar nicht, nur in PL/SQL, nur in JavaScript oder in beiden Sprachen gleichzeitig gesetzt sein. Anweisungen sind stets als Ausriss aus einem anonymen PL/SQL-Block oder einem JavaScript-Block zu schreiben, benötigen also insbesondere ein Semikolon an ihrem Ende.

Das Plugin sammelt alle PL/SQL-Anweisungen und führt diese innerhalb der Datenbank aus, anschließend werden die geänderten Elementwerte aus dem Session State geladen und mit den JavaScript-Anweisung an die APEX-Anwendung gesendet, wo sie ausgeführt werden. Daraus ergibt sich eine feste Ausführungsreihenfolge:

* Alle PL/SQL-Anweisungen in der Reihenfolge, die durch Spalte SORT\_SEQ der Tabelle ADC\_RULE\_ACTION definiert ist
* Alle JavaScript-Anweisungen in der Reihenfolge, die durch Spalte SORT\_SEQ der Tabelle ADC\_RULE\_ACTION definiert ist

Aktionstypen dürfen Ersetzungszeichenfolgen referenzieren, die bei der späteren Verwendung mit Bewegungsdaten gefüllt werden:

* #ITEM#  
  Dieser Ersetzungsanker wird zur Laufzeit mit dem Namen des referenzierten Objekts aus Spalte SRA\_SPI\_ID der Tabelle ADC\_RULE\_ACTION gefüllt
* #APARAM\_1# bis #PARAM\_3#  
  Diese Ersetzungsanker werden zur Laufzeit mit den Spalten CRA\_PARAM\_<n> der Tabelle ADC\_RULE\_ACTIONS gefüllt. Sie dienen der Übergabe von Parametern an PL/SQL- oder JavaScript-Funktionen, z.B. für Meldungstexte etc.

Aktionstypen werden in der Tabelle ADC\_ACTION\_TYPES gespeichert.

#### Seitenelemente, Tabelle ADC\_PAGE\_ITEMS (CPI)

Regelbedingungen und Aktionen referenzieren Seitenelemente. Die referenzierten Seitenelemente werden aus dem APEX Data Dictionary für die APEX-Anwendung, die in der Regelgruppe referenziert wird, ausgelesen und analysiert. Die aufbereiteten Beschreibungen dieser Seitenelemente werden anschließend in Tabelle ADC\_PAGE\_ITEMS gespeichert. Jedes Seitenelement wird beschrieben

* durch eine Referenz auf die Regelgruppe, von der das Seitenelement referenziert wird,
* durch die ID des Elements (Bei Formularelemente ist dies der Name des Elements, bei Schaltflächen und Regionen wird ausschließlich die statische ID, die auf der APEX-Anwendung vergeben wurde, referenziert),
* eine Referenz auf den Typ des Elements (Seitenelement, Anwendungselement, Schaltfläche oder Region), der in Tabelle ADC\_PAGE\_ITEM\_TYPES hinterlegt ist,
* sowie Angaben zur Konvertierung des Elements in die Datentypen TEXT, DATE oder NUMBER.

Die Tabelle umfasst alle Elemente der Anwendungsseite. Anwendungselemente, die in einer Regelbedingung referenziert wurden, sind durch den Spaltenwert CPI\_REQUIRED = C\_TRUE gekennzeichnet, denn diese Elemente werden für die Regelauswertung benötigt. Aktionen können alle Elemente der APEX-Anwendung ändern, doch führen diese Änderungen nur dann zu einer Reaktion des Plugins, wenn die geänderten Elemente in einer Regelbedingung verwendet wurden.

Die Konvertierung eines Elements in die Datentypen DATE oder NUMBER wird nur erkannt, wenn das Element auf der Anwendungsseite über eine Formatmaske verfügt oder über die Verwendung in einer Formularregion über die Metadaten diese Information enthält. Daher müssen folgende Regeln beachtet werden, wenn das Plugin problemlos mit einer Anwendungsseite kooperieren können soll:

* Elemente, die in NUMBER oder DATE konvertiert werden sollen, müssen über eine Formatmaske in der APEX-Anwendung verfügen oder über eine Formularregion verwendet werden.
* Verfügen Seitenelemente in APEX nicht über die Möglichkeit, Formatmasken zu definieren, müssen diese Elemente in den Regelbedingungen explizit in die Zieldatentypen überführt werden (z.B. SELECT-Listen oder Anwendungselementen)
* Schaltflächen oder Regionen, die durch Aktionen manipuliert werden sollen, benötigen auf der APEX-Seite verpflichtend eine statische ID, ansonsten werden sie vom Plugin nicht erkannt. Schaltflächen können darüber hinaus auch in Regelbedingungen verwendet werden, weil ihnen eine Spalte mit dem Bezeichner der statischen ID zugewiesen wird, die C\_TRUE enthalten, wenn sie geklickt wurden, Regionen (naturgemäß) nicht

Seitenelemente werden in der Tabelle ADC\_PAGE\_ITEMS gespeichert.

#### Seitenelementtyp, Tabelle ADC\_PAGE\_ITEM\_TYPES (CIT)

Seitenelemente können vom Typ APP\_ITEM, ITEM, BUTTON oder REGION sein. Diese Seitenelementtypen werden in einer eigenen Tabelle, ADC\_PAGE\_ITEM\_TYPES gespeichert. Zusätzlich zur technischen, alphanumerischen ID und der Referenz auf ein übersetzbares Objekt der Typen werden für Seitenelementtypen hinterlegt, ob diese Typen in APEX einen Session State besitzen und, ob und wenn ja, welchen Event diese Elementtypen binden sollen. Für die vorgegebenen Elementtypen ist festgelegt, dass nur die Typen ITEM und APP\_ITEM einen Session State besitzen und die Elementtypen APP\_ITEM, ITEM und BUTTON die Events CHANGE bzw. CLICK binden. Regionen können daher weder in Regelbedingungen verwendet werden, noch können sie einen Session State besitzen. Regionen können aber manipuliert (z.B. sichtbar oder unsichtbar, eingeklappt oder ausgeklappt oder aber aktualisiert) werden.

Als besondere Seitenelementtypen können auch Events hinterlegt werden. Damit ist es möglich, auf besondere Events der Anwendungsseite zu reagieren, etwa auf das Event apexafterclosedialog. Der Fokus dieser Funktionalität ist nicht, dynamische Aktionen zu imitieren, sondern auf Ereignisse der Anwendungsseite, die den Status der Seite ändern, reagieren zu können. Die Liste der Ereignisse ist möglichst kurz zu halten. Wird ein Ereignis auf diese Weise hinterlegt, bietet die Entscheidungstabelle eine entsprechende Spalte an, die das auslösende Element enthält, wenn das Ereignis ausgelöst wurde und ansonsten NULL.

### Verarbeitung einer Regel

Grundlage des Plugins ist die Überführung von Entscheidungslogik in eine SQL-Abfrage. Dies soll an einem Beispiel verdeutlicht werden.

#### Verarbeitungsbeispiel

Eine APEX-Anwendungsseite enthält die beiden Elemente P1\_VALID\_FROM und P1\_VALID\_TIL. Es soll festgelegt werden, dass wenn ...

* P1\_VALID\_FROM den Wert NULL enthält, Feld P1\_VALID\_TIL deaktiviert werden soll
* P1\_VALID\_FROM größer oder gleich P1\_VALID\_TIL ist, eine Fehlermeldung bei Feld P1\_VALID\_TIL ausgegeben werden soll
* P1\_VALID\_FROM einen Wert enthält, Feld P1\_VALID\_TIL aktiviert werden soll

Schließlich sind drei Regeln angelegt worden:

* Bedingung mit ID 5: P1\_VALID\_FROM is NULL  
  Aktion: P1\_VALID\_TIL: Deaktivieren
* Bedingung mit ID 6: P1\_VALID\_FROM >= P1\_VALID\_TIL  
  Aktion: P1\_VALID\_TIL: Zeige Fehlermeldung
* Bedingung mit ID 7: P1\_VALID\_FROM is not NULL  
  Aktion: P1\_VALID\_TIL: Aktivieren

Die Reihenfolge der Regeln ist hier von Bedeutung: Die Regel mit der ID 7 muss nach Regel 6 ausgeführt werden, da ansonsten Regel 6 nicht mehr berücksichtigt würde. Die Bedingungen müssen also vom Speziellen zum Allgemeinen angelegt sein. Typischerweise werden zunächst Initialisierungsregeln formuliert, anschließend Fehlerstatus und zuletzt das Verhalten im Normalfall.

Aus der Analyse der Einzelbedingungen ergibt sich, dass die Elemente P1\_VALID\_FROM und P1\_VALID\_TIL referenziert wurden. Daher ist es für eine Analyse der Regeln mit Hilfe von SQL wichtig, den aktuellen Wert dieser beiden Elemente zu kennen. Einzige Quelle für SQL-Anweisungen, um den Wert von Anwendungselementen zu erfragen, ist der Sessionstatus. Da dieser aber nicht notwendigerweise synchron zur Anwendungsseite ist, wird zunächst sichergestellt, dass die aktuellen Elementwerte der Anwendungsseite in den Sessionstatus kopiert werden.

Um aus einer Regel die relevanten Elemente zu filtern, wird ein regulärer Ausdruck benutzt, der alle in Frage kommenden Elemente (alle Elemente, die auf der APEX-Seite verfügbar sind, die Anwendungselemente und verfügbar gemachte Schaltflächen) in den Regelbedingungen sucht.

Anschließend kann mit folgender SQL-Anfrage der Wert der Anwendungselemente erfragt werden:

select to\_date(v('P1\_VALID\_FROM'), 'dd.mm.yyyy') valid\_from,  
 to\_date(v('P1\_VALID\_TIL'), 'dd.mm.yyyy') valid\_til  
 from dual;

Das Plugin erstellt diese Abfrage automatisch, basierend auf den Metadaten aus den Tabellen ADC\_PAGE\_ITEMS und ADC\_RULE. Zusätzlich nimmt die Abfrage noch weitere Informationen auf:

* Das auslösende Element der Anfrage wird durch die Spalte FIRING\_ITEM zugänglich gemacht und liefert die ID des auslösenden Elements. Wird das Plugin initialisiert, enthält diese Spalte den Wert DOCUMENT, ansonsten den Namen des auslösenden Elements oder die statische ID der auslösenden Schaltfläche, jeweils in Versalien.
* Ist das auslösende Element DOCUMENT, wird das Plugin also initialisiert, enthält die Spalte INITIALIZING den Wert C\_TRUE, ansonsten C\_FALSE.
* Die Elemente, die in einer Regel angesprochen werden, steht über die Spalte FIRING\_ITEMS in der Regelview zur Verfügung und werden verwendet, um nur die Regeln zu evaluieren, die eine Abhängigkeit zum auslösenden Element besitzen. Hierfür wird das auslösende Element (Spalte FIRING\_ITEM) mit INSTR() gegen die Spalte FIRING\_ITEMS gefiltert. Zusätzlich zu den angesprochenen Elementen enthält die Spalte FIRING\_ITEMS noch den Eintrag DOCUMENT, so dass beim Initialisieren der Seite alle Regeln evaluiert werden, da in diesem Fall FIRING\_ITEM den Wert DOCUMENT enthält.
* Schaltflächen werden ebenfalls über eine Spalte, die ihrer statischen ID entspricht, referenziert. Sie erhalten den Wert C\_TRUE, wenn sie auslösendes Element waren, ansonsten C\_FALSE.

Die zusätzlichen Spalten dienen der vereinfachten Formulierung von Regeln. So kann eine Regel, die die Seitenelemente P1\_VALID\_FROM und P1\_VALID\_TIL vergleicht, durch eine zusätzliche Referenz auf die Spalte FIRING\_ITEM so eingegrenzt werden, dass sie nur angewendet wird, wenn das Element P1\_VALID\_TIL sich ändert:

P1\_VALID\_FROM >= P1\_VALID\_TIL and FIRING\_ITEM = 'P1\_VALID\_TIL'.

Die Spalte INITIALIZING kann verwendet werden, um Regeln zu erstellen, die nur bei der Initialisierung der Anwendungsseite ausgeführt werden: INITIALIZING = 1.

Schaltflächen werden, wie gesagt, ebenfalls durch Spalten zugänglich gemacht. Existiert z.B. eine Schaltfläche mit der statischen ID B1\_SAVE, würde eine Spalte B1\_SAVE enthalten sein, die immer dann, wenn B1\_SAVE auslösendes Element war, den Wert C\_TRUE enthält, ansonsten C\_FALSE. Dadurch kann eine einfache Regel erstellt werden, die nur dann angewendet wird, wenn Schaltfläche B1\_SAVE auslösendes Element war: B1\_SAVE = C\_TRUE. Alternativ könnte die gleiche Bedingung mit Hilfe der Spalte FIRING\_ITEM formuliert werden: FIRING\_ITEM = 'B1\_SAVE'.

Die Daten des Session State stehen der Regelview in einer Inner View mit dem Namen SESSION\_STATE zur Verfügung, die im Anschluss durch die Bedingungen gefiltert wird. Die Regelbedingungen sind aufgebaut wie folgt: ((SRU\_ID = <Regel-ID>) and (<Regelbedingung>)). Alle Regeln werden durch eine OR-Verbindung in der WHERE-Klausel zusammengefasst. Daraus ergibt sich für unsere Beispielregelgruppe folgende WHERE-Bedingung:

with SESSION\_STATE as (...)  
select ...  
 from SESSION\_STATE  
 join ...  
 WHERE ((sru\_id = 5) and (P1\_VALID\_FROM is NULL))  
 or ((sru\_id = 6) and (P1\_VALID\_FROM >= P1\_VALID\_TIL))  
 or ((sru\_id = 7) and (P1\_VALID\_FROM is not NULL))

Je nach Zustand des Session State werden unterschiedliche Regeln ausgewählt, deren Bedingungen aktuell zu TRUE evaluieren. Die Abfrage wird, basierend auf dem Sortierkriterium der Regeln (Spalte SORT\_SEQ der Tabelle ADC\_RULE) die Regel mit der niedrigsten Sortierung auswählen und anwenden. Basierend auf der ausgewählten Regel stehen dann die Aktionen in PL/SQL und JavaScript über die Verbindung der Tabellen ADC\_RULE, ADC\_RULE\_ACTION und ADC\_ACTION\_TYPE zur Verfügung und können durch den Code zu einer Antwort zusammengefasst werden.

Alle verfügbaren Regeln werden der Regelview durch eine View ADC\_BL\_RULES zur Verfügung gestellt, die aus den einzelnen Basistabellen eine Liste aller verfügbarer Regeln und deren Aktivitäten bereitstellt. Um die Abfrageperformanz zu steigern, wird diese View über die Spalte SGR\_ID vorgefiltert. Durch den Join der View ADC\_BL\_RULES auf die Regelaktionen werden mehrere Zeilen zurückgegeben. Um sicherzustellen, dass alle Zeilen der ausgewählten Regel erkannt werden, wird über die Spalte SRU\_ID eine Rangfolge erstellt und nur die Zeilen ermittelt, die auf Rang 1 geführt werden.

#### Rekursive Regelausführung

Regelgruppen sind standardmäßig so eingestellt, dass Regeln rekursiv ausgeführt werden. Dies kann durch Setzen eines Flags bei der Regelgruppe auch unterdrückt werden. In diesem Fall wird stets und ausschließlich nur die Regel ausgeführt, die für die aktuelle Situation des Sessionstatus die höchste Gültigkeit hat (d.h. die erste Regel, deren Bedingung zu TRUE evaluiert).

Wird in einer Regel der Session State eines Elements verändert, wird dies, wie bereits beschrieben, im Plugin registriert, wenn die Methode PLUGIN\_ADC.SET\_SESSION\_STATE verwendet wird. Dies hat zunächst zur Folge, dass der geänderte Wert an die Anwendung zurückgemeldet und dort in das Seitenelement aktualisiert wird.

Zu diesem Verhalten existiert eine Erweiterung, die immer dann Berücksichtigung findet, wenn das Element

* nicht identisch mit dem auslösenden Element ist und
* das Element ein »relevantes« Element ist, sich also Regeln der Regelgruppe auf dieses Element beziehen.

In diesem Fall wird das Plugin *nach* der Bearbeitung des ursprünglichen Events, aber *vor* der Erstellung einer Antwort, für jedes in Frage kommende Element ein CHANGE-Ereignis simulieren und eine entsprechende Antwort des Plugin erfragen. Durch diesen Mechanismus ist es möglich, in einer Aktion mehrere Elemente mit Werten zu belegen und - vor der Antwort an das aufrufende Programm - alle geänderten Elemente, auf die sich eine Regel bezieht, zu prüfen, um festzustellen, ob diese Elemente z.B. angezeigt werden sollen oder nicht. Alle Datenbankaktionen, die sich durch diese Prüfungen ergeben, werden ausgeführt, alle JavaScript-Aktionen in einer Antwort gesammelt. Zur Unterscheidung, welche JavaScript-Anweisung durch welche Regel erzeugt wurde, enthält die Antwort einen Kommentareintrag vor jedem erneuten Aufruf mit Angabe zur Regel und zur Rekursionstiefe, die diese JavaScript-Anweisung erzeugt hat.

Da durch diesen Mechanismus Endlosschleifen auftreten können, wird im Plugin ein Rekursionszähler aktualisiert, der bei maximal 10 Rekursionen die weitere Verarbeitung abbricht. Die Rekursion wird für ein Element, das bereits bearbeitet wurde, nicht erneut aufgerufen, egal, auf welcher Rekursionstiefe dieser erneute Aufruf geschieht.

Als Anwendungsbeispiel für diese rekursive Regelausführung nehmen wir folgendes an:

Ein Eingabefeld auf dem Formular erlaubt die Eingabe einer Kunden-ID. Durch Änderung der Kunden-ID soll das Plugin mehrere Felder des Kunden mit Daten füllen. Ist der Kunde ein gewerblicher Kunde, soll das Feld P1\_FIRMA angezeigt werden, ansonsten soll dieses Element verborgen werden.

Eine Regel wird für das Feld P1\_KUNDE\_ID angelegt, die besagt, dass sie ausgeführt werden soll, wenn das Feld nicht leer ist. Die Aktion soll darin bestehen, eine PL/SQL-Methode aufzurufen, die mehrere Elemente im Session State mit Werten dieses Kunden belegt. Zwei weitere Regeln besagen, dass, wenn das Feld P1\_KUNDE\_ART den Wert J enthält, das Feld P1\_FIRMA gezeigt werden soll, ansonsten nicht.

Wird nun das Feld P1\_KUNDE\_ART durch die PL/SQL-Methode gesetzt, würde ohne rekursive Behandlung der Regeln der CHANGE-Event für das Element nicht ausgelöst und die verbundene Regel nicht berücksichtigt. Durch die rekursive Regelausführung erkennt das Plugin, dass das Feld P1\_KUNDE\_ART geändert wurde und dass sich eine Regel auf dieses Feld bezieht. Dadurch wird die Regelverarbeitung für dieses Element als auslösendem Element erneut ausgeführt und die JavaScript-Aktion der Antwort hinzugefügt. Da durch diese Regel keine weiteren Elemente geändert wurden, wird keine weitere rekursive Aktivität ausgelöst und die Antwort an die aufrufende Seite gesendet.

Das folgende Listing zeigt eine typische Antwort für eine relativ tief verschachtelte Seite. Hier ist das Szenario, dass nacheinander fünf Auswahllisten ausgefüllt werden müssen, um eine Regelgruppe in eine andere Anwendung zu kopieren. Zu Beginn sollen, wenn die oberste Auswahlliste nicht belegt ist, alle anderen Auswahllisten deaktiviert werden. Ohne Rekursion müssten die Regeln für die erste Auswahlliste den Zustand aller weiterer Auswahllisten definieren, ebenso die folgende für alle weiteren etc. Mit Rekursion ist das nicht nötig: Die erste Auswahlliste deaktiviert die zweite, die wiederum die dritte etc. Mittels Rekursion ergibt sich folgende (formatierte) Antwort:

<script>

de.condes.plugin.ADC.setItemValues(

[{"id":"P4\_SGR\_PAGE\_ID","value":""},

{"id":"P4\_SGR\_ID","value":""},

{"id":"P4\_SGR\_APP\_TO","value":""},

{"id":"P4\_SGR\_PAGE\_TO","value":""}]);

de.condes.plugin.ADC.setErrors(

{ "count":0,

"errorDependentButtons":"B4\_COPY",

"firingItems":"P4\_SGR\_PAGE\_ID,P4\_SGR\_ID,P4\_SGR\_APP\_TO,P4\_SGR\_PAGE\_TO",

"errors":[]});

//Recursion 1: RULE\_10 (Quellanwendung ist leer), Firing Item: DOCUMENT

apex.item('P4\_SGR\_PAGE\_ID').show;

apex.item('P4\_SGR\_PAGE\_ID').disable();

//Recursion 2: RULE\_30 (Quellseite ist leer), Firing Item: P4\_SGR\_PAGE\_ID

apex.item('P4\_SGR\_ID').show;

apex.item('P4\_SGR\_ID').disable();

//Recursion 3: RULE\_50 (Regelgruppe ist leer), Firing Item: P4\_SGR\_ID

apex.item('P4\_SGR\_APP\_TO').show;

apex.item('P4\_SGR\_APP\_TO').disable();

//Recursion 4: RULE\_70 (Zielanwendung ist leer), Firing Item: P4\_SGR\_APP\_TO

apex.item('P4\_SGR\_PAGE\_TO').show;

apex.item('P4\_SGR\_PAGE\_TO').disable();

//Recursion 5: RULE\_90 (Zielseite ist leer), Firing Item: P4\_SGR\_PAGE\_TO

apex.item('B4\_COPY').show();

apex.item('B4\_COPY').disable();

</script>

Rekursive Regeln erlauben, Regelwerke zu vereinfachen. Hierzu ein Beispiel: Auf einer Seite sollen Anwendungselemente initialisiert werden: Einige Schaltflächen und Eingabeelemente sollen solange deaktiviert bleiben, solange eine ID auf der Seite nicht gesetzt ist. Man wird nun - ohne rekursive Regelausführung - damit beginnen, den Status aller Schaltflächen zu definieren. Mit rekursiver Regelausführung reicht es, das Element ID auf NULL zu setzen und eine Regel zu definieren, die festlegt, welche Schaltflächen ausgeblendet werden sollen, wenn dieses Element keinen Wert enthält. Egal, wodurch nun dieses Element auf NULL gesetzt wird, wird diese Regel rekursiv immer aufgerufen und die entsprechende Aktion durchführen.

Durch Rekursion werden die Regeln modularer und können daher besser wiederverwendet werden. Im Beispiel oben sind lediglich einzelne Regeln für die verschiedenen Elemente definiert worden. Je nach auslösendem Element werden nun die jeweils abhängigen Elemente rekursiv aufgerufen und der Status stellt sich stets korrekt ein.

#### Initialisierungsregeln

Regeln können mit einem Flag »Beim Seitenladen ausführen« versehen werden. Standardmäßig ist dieses Flag FALSE, was bedeutet, dass die Regel nur ausgeführt werden kann, wenn das auslösende Elemente in der Regelbedingung angesprochen wurde. Ist das Flag gesetzt, wird die Regel in jedem Fall beim Initialisieren der Seite ausgeführt.

Diese Option kann dazu dienen, die Anzahl der Regeln zu reduzieren und entspricht grob der entsprechenden Option von Dynamic Actions. Eine Regel die durch dieses Flag ausgeführt wird, wird immer *zusätzlich* zu den ohnehin vorhandenen Initialisierungsregeln (initializing = 1) ausgeführt, und zwar *nach* diesen. Ist das Flag für mehrere Regeln gesetzt, werden mehrere Regeln ausgeführt, selbst dann, wenn für die Regelgruppe Rekursion untersagt wurde. Die Initialisierungsregeln werden nicht durch den Mechanismus der Rekursion ausgeführt, sondern parallel, auf Grund des Setzens dieses Flags. Initialisierungsregeln haben keine Rekursion zur Folge, auch wenn diese für die Regelgruppe erlaubt ist. Dadurch werden Rekursionsschleifen verhindert.

## Technische Umsetzung: Module

Das Plugin besteht aus mehreren Modulen:

* Kernfunktionalität  
  Implementiert in den Packages ADC\_INTERNAL und ADC\_API sowie den Tabellen ADC\_%
* Plugin  
  Das eigentliche APEX-Plugin, implementiert im Package PLUGIN\_ADC sowie den JavaScript-Dateien ADC.js und ADCApex.js
* PL/SQL-API  
  Dient dem Ermitteln von Aktionen durch PL/SQL-Codes. Dies ist eine Möglichkeit, sehr komplexe Regelwerke zu implementieren. Die API ist im Package ADC hinterlegt.
* Verwaltungspackage  
  Dient dem Export von Regelgruppen, aber auch der Erstellung und Validierung von Decision Table-Logik. Es ist im Package ADC\_ADMIN implementiert.
* APEX-Anwendung  
  APEX-Anwendung zur Verwaltung der Regelgruppen, Regeln, Aktionen und Aktionstypen, implementiert im Package ADC\_UI sowie der APEX-Anwendung selbst

### Kernfunktionalität: Packages ADC\_API und ADC\_INTERNAL

Das Package ADC\_INTERNAL enthält die zentrale Logik von ADC und ist nicht zur direkten Verwendung freigegeben. Die Methoden, die extern genutzt werden dürfen, sind in ADC\_API veröffentlicht. Diese Methoden werden durch die Packages PLUGIN\_ADC, ADC sowie ADC\_UI aufgerufen. ADC\_INTERNAL stützt sich in Teilen auf ADC\_ADMIN ab, um z.B. die Decision Table-Logik erstellen zu lassen. Diese wird auch beim Importieren von Regelgruppen benötigt.

Derzeit ist folgendes Verhalten implementiert:

* Wird eine Regelgruppe oder Regel angelegt oder verändert, wird eine SQL-Abfrage erstellt, wie im Beispiel erläutert, und als SQL-Abfrage in Tabelle ADC\_RULE\_GROUPS abgelegt.
* Wird eine Aktion erstellt, wird die SQL-Anweisung ausgeführt, die resultierende Regel ermittelt und aus den Metadaten dieser Regel die Antwort generiert und an den aufrufenden Code übermittelt
* Wird eine Regelgruppe exportiert, werden die Metadaten in Aufrufe der Administrationspackages überführt und diese als CLOB an den aufrufenden Code übermittelt
* Methode zur Analyse einer Regelgruppe und zur Erstellung der PL/SQL- und JavaScript-Skripte als Antwort des Plugins auf eine gegebene Anwendungssituation, Bereitstellung von Hilfsfunktionen für die Packages PLUGIN\_ADC und ADC\_UI

Es folge eine kurze Übersicht über einige relevante Funktionen:

#### Getter-Methoden GET\_FIRING\_ITEM, GET\_JS\_FUNCTION

Die Methoden liefern den Namen des auslösenden Elements sowie den Namen der JavaScript-Funktion, mit der das Plugin aufgerufen wird. Sie werden von der Regelview und vom Package PLUGIN\_ADC aufgerufen.

#### Methode CREATE\_ACTION

Diese Methode berechnet, welche Regel für eine gegebene Situation im Session State relevant ist, sowie die daraus resultierenden Aktionen für die Datenbank und die Anwendung. Aus der ermittelten Regel berechnet die Methode einen anonymen PL/SQL-Block mit den PL/SQL-Anweisungen sowie eine Liste mit JavaScript-Funktionen für die Anwendung. Die Ausführung des PL/SQL-Codes und die Aufbereitung der Antwort für die APEX-Anwendung erfolgen im Package PLUGIN\_ADC.

Die Methode greift auf die Werte des Sessionstates und alle hinterlegten Regeln zu, indem sie die vom Package bei der Pflege der Regeln erstellt Regelview mittels einer SQL-Abfrage auswertet. Diese Abfrage ermittelt alle gültigen Regeln und verknüpft diese über die Tabelle ADC\_ACTION\_TYPE mit allen PL/SQL- und JavaScript-Aktionen. Aus der ermittelten Ergebnismenge wird die erste Regel, die zu TRUE evaluierte, ausgewählt und die darin referenzierten Aktionen in einem PL/SQL- und einem JavaScript-Block zusammengefasst. Zudem werden alle Elemente ermittelt, die mit den auslösenden Elementen in einem Regelzusammenhang stehen. Die Liste dieser Elemente wird als kommaseparierte Liste an den aufrufenden Code übergeben, der daraus ableitet, für welche Elemente Fehlermeldungen auf der Oberfläche zu entfernen sind.

#### Methode SET\_SESSION\_STATE

Für die korrekte Funktion des Plugins ist es wichtig, Elementwerte im Session State ausschließlich über die Methode SET\_SESSION\_STATE zu setzen und nicht direkt über APEX\_UTL.SET\_SESSION\_STATE, um sicher zu stellen, dass das Plugin Kenntnis von der Werteänderung enthält und diese an die Anwendung weiterreichen kann.

Alle Elemente, die durch die Bearbeitung der Regelanfrage geändert werden, werden durch das Plugin registriert und in der Antwort mit ihren aktuellen Elementwerten aus dem Session State an die Anwendungsseite gereicht. Das Plugin setzt die Elementwerte, so dass Seite und Session State bezüglich der geänderten Elemente nachfolgend synchron sind.

#### Methode REGISTER\_ERROR

Die Methode REGISTER\_ERROR sammelt alle auflaufenden Fehler während der Verarbeitung und gibt diese an die JavaScript-Methoden des Plugins weiter, so dass diese dort bei den entsprechenden Elementen als Fehler angezeigt werden können. Ein Fehler wird über den Namen (ID) des Anwendungselements und den Fehlertext definiert. Durch die Angabe der ID des Anwendungselementes, dass den Fehler ausgelöst hat, ist das Plugin in der Lage, den Fehler bei dem betroffenen Element anzuzeigen und diesen selektiv von der Anwendungsseite zu löschen.

#### Methode REGISTER\_MANDATORY

Die Methode vermerkt zu einem Element, dass es ein Pflichtelement darstellt. Dieser spezielle Status wird in Spalte ADC\_PAGE\_ITEM.IS\_MANDATORY vermerkt. Der Status kann durch Regeln geändert werden, so dass es möglich ist, Elemente dynamisch zu Pflichtelementen zu erklären oder diesen Status zurückzunehmen.

#### Methode CHECK\_MANDATORY

Die Methode prüft alle Pflichtfelder, die durch diese Regelgruppe verwaltet werden und gibt Fehlermeldungen für jedes Element aus, dass keinen Wert enthält.

#### Methode REGISTER\_NOTIFICATION

Die Methode REGISTER\_NOTIFICATION sammelt, analog zur Methode REGISTER\_ERROR, Benachrichtigungen an den Anwendungsbenutzer, die während der Verarbeitung der Regel aufgelaufen sind. Nach Abschluss der Verarbeitung werden alle Regeln zusammengefasst und in einem Notification-Block auf der APEX-Seite dargestellt. Bei den Meldungen kann es sich um Bestätigungsmeldungen, Statusmeldungen oder ähnliche handeln.

#### Methode PROCESS\_RULE

Diese private Methode ist die Kernmethode des Plugins. Sie führt die Regellogik aus und kontrolliert den rekursiven Aufruf der Regeln. Die Arbeitsweise der Methode ist organisiert wie folgt:

* Vor dem Aufruf der Methode wurde der Fehlerstack und die Liste der rekursiv erforderlichen Aufrufe, der Rekursionsstack, bereinigt. Als einziges Element in der Liste der rekursiven Aufrufe befindet sich das auslösende Element, der Zähler der Rekursionstiefe steht auf 1.
* Die Methode vermerkt den aktuellen Stand der Rekursionstiefe und inkrementiert zu Beginn diesen Wert. Sollten weitere Elemente rekursiv bearbeitet werden müssen, würden diese mit ihrem Namen und der inkrementierten Rekursionstiefe im Rekursionsstack vermerkt. Würde also als Beispiel Element P1\_ID rekursiv bearbeitet werden müssen, wäre dieses Element als Eintrag P1\_ID mit dem Wert 2 in der Liste vermerkt.
* Nun iteriert die Methode über alle Elemente des Rekursionsstacks und sucht Elemente, deren Rekursionstiefe dem aktuellen Stand der Rekursionstiefe (zu Beginn 1) entspricht. Ist dies erfolgreich, wird ein Bearbeitungsflag gesetzt, um anzuzeigen, dass ein weiterer rekursiver Aufruf der Methode erforderlich sein wird. Ist dieses Flag nicht gesetzt, wird die Rekursion abgebrochen und die Methode beendet. Bevor die Methode rekursiv aufgerufen wird, werden alle Elemente des aktuellen Rekursionslevels bearbeitet.
* Hat die Methode einen Eintrag im Rekursionsstack gefunden, wird dieser als auslösendes Element betrachtet und für dieses Element die Regellogik ausgewertet.
* Die PROCESS\_RULE-Methode selbst enthält keine Entscheidungslogik und kann Regeln nicht evaluieren. Diese Aufgabe delegiert die Methode an ADC\_ADMIN.CREATE\_ACTION und enthält von dort Angaben darüber, welche Aktionen in PL/SQL innerhalb der Datenbank und in JavaScript im Browser auszuführen sind. Ist die Regelgruppe ausgewertet, wird ein PL/SQL-Script und ein JavaScript-Script an das Package geliefert. Zudem wird eine Liste der auslösenden Elemente (»FIRING\_ITEMS«) vermerkt. Damit sind die Elemente gemeint, die durch die evaluierten Regeln betroffen sind. Dies führt dazu, dass Fehlermeldungen, die sich auf der Oberfläche befinden könnten, von diesen Elementen entfernt und - falls vorhanden - neu gesetzt werden. Dies verhindert, dass Fehlermeldungen auf der Oberfläche entfernt werden, die durch die aktuellen Regeln nicht betroffen sind.
* Die Methode PROCESS\_RULE führt den PL/SQL-Code aus, er durch die Regelauswertung in ADC\_ADMIN.CREATE\_ACTION ermittelt wurde. Wenn als Teil der Ausführung des PL/SQL-Blocks die Methode SET\_SESSION\_STATE des Plugins aufgerufen und somit ein Elementwert im Session State verändert wurde, wird dieses Element zur rekursiven Bearbeitung vorgemerkt: Die Methode SET\_SESSION\_STATE ruft als Teil ihrer Ausführung die private Methode REGISTER\_ITEM auf, die das geänderte Element in den Rekursionsstack einfügt, falls sich das geänderte Element noch nicht dort befindet und die Rekursionstiefe noch nicht die maximale Rekursionstiefe, die durch die Packagevariable C\_RECURSIVE\_LIMIT festgelegt wurde, überschritten hat.
* Das aktuell verarbeitete Element wird nach Abschluss der Verarbeitung aus dem Rekursionsstack entfernt. Da bei der Bearbeitung des Elements das Bearbeitungsflag gesetzt wurde, um anzuzeigen, dass Elemente auf diesem Rekursionslevel verarbeitet wurden, wird nun die Methode PROCESS\_RULE rekursiv aufgerufen. Der Prozess beginnt von vorn, diesmal allerdings mit der inkrementierten Rekursionstiefe.

Nach Abschluss aller rekursiver Regelevaluierungen liefert die Methode eine kumulierte JavaScript-Anweisung zurück, die vom Plugin an die Oberfläche gesendet und dort ausgeführt wird. Die PL/SQL-Anweisungen sind als Teil der Regelevaluierung bereits ausgeführt worden und damit abgeschlossen.

Durch die Arbeitsweise der Rekursion, die zunächst alle Elemente einer Rekursionstiefe abarbeitet und danach Elemente der nächsten Ebene bearbeitet (»Breadth first«), ist sichergestellt, dass die nachfolgende Rekursionstiefe auf den geänderten Session State der vorhergehenden Rekursionstiefe zugreifen kann: Elemente, die auf Rekursionstiefe 1 gesetzt werden, stehen in Rekursionstiefe 2 in der Regelview zur Verfügung.

#### Methode REGISTER\_ITEM

Die Methode REGISTER\_ITEM ist eine private Methode, die aufgerufen wird, wenn ein Wert im Session State durch die Methode PLUGIN\_ADC.SET\_SESSION\_STATE geändert wird. Die Methode erhält den Elementnamen des Elements, das geändert wurde und versucht, dieses in der Liste der relevanten Elemente der Regelgruppe zu finden. Ändert sich das aufrufende Element selbst (Ist also der Elementname gleich dem aktuellen FIRING\_ITEM), wird das Element ignoriert, um Endlosschleifen zu verhindern.

Ist das geänderte Element relevant und nicht gleich dem auslösenden Element, wird versucht, das Element unter der aktuell geltenden Rekursionstiefe in den Rekursionsstack zu schreiben. Dies wird vom Stack abgelehnt, wenn sich das geänderte Elemente bereits dort befindet. In diesem Fall liefert das Plugin für das Element die Fehlermeldung *'Element hat rekursive Schleife erzeugt und wurde daher ignoriert.'* zurück. Die weitere Erfahrung mit dem Regelwerk wird zeigen, ob dies ein gangbarer Weg ist, oder ob das stillschweigende Ignorieren dieser Elemente sinnvoll ist.

Der Rekursionsstack lehnt die Aufnahme des Elements auch dann ab, wenn eine maximale Rekursionstiefe überschritten wurde. Die maximale Rekursionstiefe wird derzeit durch eine Packagekonstante C\_RECURSIVE\_LIMIT angegeben. Eventuell sollte diese Konstante als Parameter ausgelagert werden. Ist diese Grenze überschritten, wird für das aktuelle Element die Fehlermeldung *'Element hat Rekursionstiefe von <n> überschritten.'* zurückgeliefert.

### Administration: Package ADC\_ADMIN

Das Package umfasst mehrere Funktionsbereiche:

* Administration und Stammdatenpflege  
  Erstellung und Verwaltung von Regelgruppen, Regeln, Aktionen und Aktionstypen, Methoden zum Im- und Export von Regelgruppen zur Integration des Plugins in einen APEX-Deploymentzyklus
* Interne Verwaltung  
  Automatisierte Erstellung der Regelabfragen, Prüfung von Regeln

Die Metadaten des Plugins werden einerseits durch Assistenten-basierte Seiten der APEX-Anwendung direkt gepflegt, andererseits existiert eine API zur Erstellung von Regeln, zum Beispiel aus einem Export der Regelgruppe oder der Regelgruppen. Sie werden nicht von der APEX-Anwendung verwendet, sondern dienen dem Import von Regelgruppen in eine Anwendung. Die hierfür vorhandenen Methoden des Packages stellen einfache Wrapper um MERGE-Anweisungen dar, mit deren Hilfe die Stammdatentabellen gepflegt werden können.

Methoden, die auf diese Weise arbeiten, sind:

* MERGE\_RULE\_GROUP
* MERGE\_RULE
* MERGE\_RULE\_ACTION
* MERGE\_ACTION\_TYPE
* DELETE\_RULE\_GROUP
* RESEQUENCE\_RULE\_GROUP

Zudem bietet das Package Hilfsmethoden zum Löschen von Regelgruppen, Regeln etc. an.

#### Methoden EXPORT\_RULE\_GROUP und EXPORT\_RULE\_GROUPS

Die Methode EXPORT\_RULE\_GROUP ist überladen, um den Export einer Regelgruppe nach ID oder Regelgruppenname und Anwendungs-ID zu ermöglichen. Methode EXPORT\_RULE\_GROUPS ermöglicht den Export aller Regelgruppen einer Anwendung. Die Methoden liefern die Exportskripte als CLOB zurück, die Anwendung verfügt über eine Wartungsseite, mit deren Hilfe die Exporte ausgeführt und als Datei geladen werden können.

Beim Export einer Regelgruppe kann festgelegt werden, ob der Export für eine andere Anwendung und/oder Anwendungsseite erfolgen soll. Dadurch kann die Export-Funktion genutzt werden, um Regeln zwischen Anwendungen zu kopieren, oder aber auch, um Regeln beim Export einer APEX-Anwendung in den Skript zur Installation zu integrieren. Existiert die Regelgruppe für die angeforderte Anwendungs-ID und Seiten-ID, überschreibt der Export die existierende Regelgruppe.

#### Methode DELETE\_RULE\_GROUP

Die Methode entfernt die angegebene Regelgruppe aus den Stammdatentabellen.

#### Methode PROPAGATE\_RULE\_CHANGE

Die Methode PROPAGATE\_RULE\_CHANGE wird nach jeder Änderung einer Regel oder Regelgruppe aufgerufen. Durch den Aufruf der Methode wird veranlasst, dass ...

* Die Regel validiert wird
* Die Liste der relevanten Elemente der Regelgruppe neu kalkuliert wird
* Eine neue Version der Regelview erstellt wird

Die Methode wird durch die APEX-Anwendung zur Pflege des Plugins nach jeder Änderung aufgerufen. Die APEX-Anwendung basiert auf Assistenten-erstellen Seiten zur Pflege der Stammdaten, daher werden auch die Standardprozesse zum Anlegen und Speichern der Daten verwendet. Um einen Trigger zu vermeiden, der die Validierung der Regeländerung anstößt, wird auf den relevanten Seiten der APEX-Anwendung diese Methode explizit nach den Standardprozessen der Seite aufgerufen.

### Plugin: Package PLUGIN\_ADC

Das Package PLUGIN\_ADC implementiert die, durch APEX vorgegebene, Schnittstelle für ein Dynamic Action Plugin, namentlich die Methoden RENDER und AJAX.

#### Methode RENDER

Die RENDER-Methode des Packages initialisiert das Plugin, indem es für eine Regelgruppe eine Liste der relevanten Formularelemente erstellt, an die das Package im Anschluss Eventhandler bindet. Gebunden werden können CHANGE-Events an Formularelemente und CLICK-Events an Formularschaltflächen. Wie bereits beschrieben, erfolgt die Auswahl der Elemente, an die das Plugin bindet, automatisch über eine Analyse der definierten Regeln.

Der Ablauf des Renderprozesses ist wie folgt:

* Das Plugin wird, basierend auf den Parameterwerten des Plugins auf der Anwendungsseite, erstellt und auf der Seite eingerichtet
* Die Datenbank ermittelt alle relevanten Elemente der Regelgruppe und gibt diese als Parameter an das Plugin weiter
* Das Plugin bindet entsprechende Event-Handler an die Events, die aus der Datenbank für den entsprechenden Elementtyp hinterlegt sind (Formularelemente binden an den CHANGE-Event, Schaltflächen an den CLICK-Event, siehe Tabelle ADC\_PAGE\_ITEM\_TYPE)
* Zum Abschluss der Initialisierung ruft das Plugin erstmalig die AJAX-Funktion des Plugins auf und übergibt
  + Den aktuellen Elementwert aller relevanter Elemente
  + Die Angabe DOCUMENT als auslösendes Element

Anschließend wird mit diesem Aufruf verfahren wie im nächsten Abschnitt beschrieben.

#### Methode AJAX

Die AJAX-Methode implementiert das Verhalten des Plugins, wenn ein gebundener Event auf der Anwendungsseite ausgelöst wird.

Die JavaScript-Funktionalität des Plugins stellt die aktuellen Elementwerte der relevanten Elemente zusammen und persistiert diese im Session State und übermittelt den Namen des auslösenden Elements an die AJAX-Methode. Nach der Initialisierung des Plugins beim Seitenladen wird erstmalig diese Funktion mit dem auslösenden Elemente DOCUMENT aufgerufen, um sicherzustellen, dass die Anwendungsseite korrekt initialisiert wird.

In der AJAX-Methode werden folgende Schritte veranlasst:

* Ein interner Errorstack wird initialisiert.  
  Auf diesem Errorstack werden die eingehenden Fehlermeldungen durch die Methode REGISTER\_ERROR abgelegt
* Ein Rekursionsstack wird initialisiert.  
  Der Rekursionsstack ist eine PL/SQL-Tabelle, die als Schlüsselwerte die ID des auslösenden Elements verwendet und als Ladung die Rekursionstiefe, die bei 1 beginnt
* Die Regelgruppe wird verarbeitet.  
  Die Verarbeitung der Methode erfolgt durch die private Methode PROCESS\_RULE, die Beschreibung der einzelnen Verarbeitungsschritte findet sich bei deren Dokumentation.
* Ist die Bearbeitung der Regel abgeschlossen, werden die geänderten Elemente und die gesammelten JavaScript-Anweisungen in eine JSON-formatierte Antwort integriert und an die APEX-Anwendungsseite gesendet. Fehlermeldungen werden ebenso integriert wie eine Liste der Elemente, die durch die konkrete Regel tangiert wurde. Diese Liste wird verwendet, um auf der Seite vorhandene Fehlermeldungen zu entfernen.
* Das Plugin führt die Anweisungen der Seite aus und schließt den Bearbeitungszyklus damit ab.

### Plugin: JavaScript-Datei ADC.JS

Die JavaScript-Datei ADC.JS implementiert die JavaScript-seitige Logik des ADC Plugins. Die vollständige Implementierung ist auf zwei JavaScript-Dateien aufgeteilt. Datei ADC.js kümmert sich um die Logik des Plugins, exkludiert aber APEX-darstellungsspezifische Logik, die vom eventuell eingesetzten StyleSheet, dem Template etc. abhängt. Diese Darstellungslogik ist in der Datei ADCAPEX.js implementiert.

Die Datei ADC.js implementiert den Namensraum de.condes.plugin.adc (im Folgenden: ADC) und deklariert in diesem Namensraum eine Auswahl von Methoden, die der Bedienung des Plugins auf der Anwendungsseite dienen:

#### Methode INIT

Diese Methode dient der Initialisierung des Plugins. Die Datenbank sendet beim Seitenaufbau eine Datenstruktur, die von dieser Methode entgegengenommen und ausgewertet wird. Die Anweisung der Datenbank enthält zwei Attribute (neben dem AJAX-Identifier, der laut Vorgabe für alle APEX-Plugins generiert werden muss):

1. Ein JSON-Objekt mit der Liste der »relevanten« Seitenelemente. Jeder Eintrag enthält Angaben zur ID des Elements und zum Event, der durch das Plugin gebunden werden soll.
2. Ein kommaseparierte Liste der Seitenelemente, die durch das Plugin an die Datenbank gesendet werden sollen. Während der Initialisierungsphase sind alle relevanten Elemente und eventuell weitere Elemente in dieser Liste enthalten, während der weiteren Aufrufe wird die Liste aus der Liste der geänderten Seitenelemente generiert.

Da APEX die Attribute, die an ein Plugin gesendet werden, aus Sicherheitsgründen maskiert und ein maskierter Parameter andererseits von JavaScript nicht ohne Weiteres als JSON-Objekt interpretiert werden kann, werden die Steuerzeichen vor dem Versenden von der Datenbank maskiert und vom Plugin demaskiert. Erst anschließend wird die Antwort als JSON-Objekt geparst. Die Liste der Seitenelemente wird in eine globale Speicherstruktur des Plugins als Array übernommen.

Auf Basis des übermittelten JSON-Objekts mit den zu bindenden Seitenelementen werden anschließend die entsprechenden Eventhandler auf diese Elemente eingerichtet und das Plugin initial ausgeführt. Dies geschieht in der Methode EXECUTE.

#### Methode EXECUTE

Die Methode wird aufgerufen, wenn ein gebundenes Seitenelement einen Event geworfen hat, oder wenn das Plugin initialisiert wird. In diesem Fall wird als auslösendes Element DOCUMENT übergeben.

Diese Methode implementiert die Kernfunktionalität der JavaScript-Seite des Plugins. Der Ablauf besteht darin, alle »relevanten« Seitenelemente aus ADC.bindItems zu sammeln und einem AJAX-Aufruf zu übergeben, der für die aktuelle SessionState-Situation eine Antwort, basierend auf den Regeln, berechnet. Dies wird durch eine vorgegebene Option der APEX-Funktionalität apex.server.plugin erreicht: Beim Erstellen einer AJAX-Anfrage an die Datenbank existiert ein Parameter pageItems, dem eine Liste von Seitenelementen übergeben werden kann, deren Werte durch diese Methode automatisiert in den SessionState geschrieben werden. Als weiterer Parameter wird dieser Methode die ID des auslösenden Seitenelements (oder DOCUMENT) übergeben.

Diese Daten können eingesehen werden, wenn zum AJAX-Aufruf der POST-Anteil analysiert wird.

Wie bei AJAX üblich, erfolgt die Anfrage an die Datenbank asynchron. Um dies zu ermöglichen, wird eine Callback-Methode vereinbart, die aufgerufen wird, sobald die Antwort der Datenbank eintrifft.

Die Datenbank liefert als Antwort ein HTML-Fragment mit eingebettetem JavaScript zurück. Dieses HTML-Fragment wird durch die Callback-Methode in das Dokument eingefügt. Das Einfügen geschieht durch die JQuery-Methode append(), die die Eigenschaft hat, in HTML eingebettete JavaScript-Anweisungen direkt auszuführen. Da damit die Aufgabe des HTML-Fragments erledigt ist, kann es direkt anschließend wieder aus dem Dokument entfernt werden.

Das HTML-Fragment enthält Methodenaufrufe für das Plugin und übergibt entsprechende JSON-Objekte, die in den einzelnen Methoden ausgewertet werden. Das folgende Listing zeigt eine beispielhafte Antwort der Datenbank auf den AJAX-Aufruf:

<script>

de.condes.plugin.ADC.setItemValues(

[{"id":"P4\_SGR\_PAGE\_ID","value":""},

{"id":"P4\_SGR\_ID","value":""},

{"id":"P4\_SGR\_APP\_TO","value":""},

{"id":"P4\_SGR\_PAGE\_TO","value":""}]);

de.condes.plugin.ADC.setErrors(

{ "count":0,

"firingItems":"P4\_SGR\_PAGE\_ID,P4\_SGR\_ID,P4\_SGR\_APP\_TO,P4\_SGR\_PAGE\_TO",

"errors":[]});

//Recursion 1: RULE\_10 (Quellanwendung ist leer), Firing Item: DOCUMENT

apex.item('P4\_SGR\_PAGE\_ID').show;

apex.item('P4\_SGR\_PAGE\_ID').disable();

</script>

Zu erkennen sind drei Bereiche:

1. Ein Aufruf der Methode ADC.setItemValues  
   Diese Methode dient dazu, die übergebenen Elementwerte aus dem Session State, die innerhalb der Datenbank als Teil der Bearbeitung der Anfrage geändert wurden, auf der Anwendungsseite zu aktualisieren.
2. Ein Aufruf der Methode ADC.setErrors  
   Diese Methode enthält ein JSON-Objekt mit Angaben zu den Fehlern, die während der Bearbeitung aufgetreten sind. Zunächst werden die Fehler all der Seitenelemente entfernt, die im Eintrag firingItems vermerkt sind. Anschließend erhalten die Elemente, die im Array errors eingefügt wurden, eine Fehlermeldung.
3. Im letzten Bereich der Antwort ist der JavaScript-Skript enthalten, der als Aktion für die gewählte Regel vereinbart wurde. Dieser Eintrag kann rekursiv aufgerufen und zusammengestellt worden sein. Daher kann dieser Teil recht umfangreich sein. Zur erleichterten Fehleranalyse werden allen Teilskripten, die durch eine Rekursion erstellt worden, die Rekursionstiefe und die, für diese Rekursion, verwendete Regel sowie das auslösende Element vermerkt.

#### Hilfsmethoden

Das Plugin verfügt über weitere Hilfsmethoden, die Teilaufgaben übernehmen. Diese sind zum großen Teil trivial und sollen hier summarisch besprochen werden:

* ADC.setItemValues  
  Hilfsmethode, die über ein JSON-Objekt mit Seitenelement-IDs und -werten iteriert und die betroffenen Seitenelemente aktualisiert
* ADC.bindEvents  
  Der Methode wird ein JSON-Objekt mit den relevanten Seitenelementen und den Events, an die gebunden werden soll, übergeben. Die Methode iteriert über dieses Objekt und bindet die entsprechenden Events.  
  Eine Besonderheit dieser Methode besteht darin, dass die Events von Elementen, die durch einen APEX-Event apexrefresh aktualisiert werden, vor der Ausführung dieses Events vom Eventhandler entbunden und nach Abschluss des Events wieder gebunden werden. Dies vermeidet, dass die gebundenen Events durch eine APEX-Aktivität ausgelöst werden und so Endlosschleifen entstehen können.
* ADC.setErrors  
  Hilfsmethode, die den Fehlerstatus der Anwendungsseite aktualisiert. Da dies abhängig vom eingesetzten Theme ist, stellt die Methode lediglich einen Wrapper um die entsprechende Methode aus ADCAPEX.js dar.
* ADC.setMandatory  
  Wird ein Element durch ADC als verpflichtend deklariert, muss dies durch Anpassungen der Darstellung auf der Oberfläche kenntlich gemacht werden. Dies geschieht durch diese Methode, die ebenfalls einen Wrapper um ADCAPEX.js darstellt.
* ADC.submit  
  Auch diese Hilfsmethode stellt einen Wrapper um eine entsprechende Methode der Datei ADCAPEX.js dar. Die Aufgabe der Methode besteht darin, zu prüfen, ob die Seite noch einen Fehler anzeigt, und falls ja, eine entsprechende Meldung auszugeben. Falls kein Fehler auf der Seite existiert, wird die Seite mit der Methode apex.submit abgeschickt. Als Parameter kann der Methode ein Wert für die Variable REQUEST übergeben werden.
* de\_condes\_plugin\_adc  
  Diese Methode ist nicht Teil des JavaScript-Namensraums. Sie dient dazu, die Methode ADC.init aufzurufen. Aus nicht ganz nachvollziehbaren Gründen hat APEX offensichtlich Schwierigkeiten, eine Namensraummethode direkt aufzurufen. Diese Methode stellt also einen Workaround dar und könnte in Zukunft obsolet werden, wenn eine bessere Möglichkeit gefunden wird, das Plugin anzusprechen.

### Plugin: JavaScript-Datei ADCAPEX.js

Diese JavaScript-Datei implementiert die visuelle Darstellung der vom Plugin angeforderten Funktionalitäten. Grund der Zweiteilung ist, dass in dieser Datei Funktionalität implementiert ist, die von der konkreten APEX-Version, vom verwendeten Theme oder einer kundenspezifischen Erweiterung hiervon abhängig ist. Die Kapselung in einer eigenen Datei dient der besseren Trennung von Zuständigkeiten: Die Datei ADC.js ist prinzipiell von der Darstellung unabhängig.

Eine Kernfunktionalität der Datei ist die Darstellung von Fehlern, die durch das Plugin generiert wurden. APEX verfügt über einen Mechanismus zur Darstellung von Fehlern, dieser ist jedoch nicht ausreichend leistungsfähig, weil zum Beispiel das Entfernen gezielter Meldungen nicht unterstützt wird. Daher musste Funktionalität programmiert werden, um eine konsistente Darstellung der Fehler zu erreichen, egal, ob diese durch APEX oder durch das Plugin ADC erstellt wurden.

Die einzelnen Methoden sind weitgehend selbsterklärend, was ihren Einsatzbereich angeht. Im Detail sind aber zum Teil komplexe JQuery-Operationen erforderlich, um das gewünschte optische Erscheinungsbild zu erreichen. Da dies aber von der konkreten Designumgebung abhängig ist, soll dies hier nicht im Detail besprochen werden. Es bleibt bei einer summarischen Beschreibung:

* Methode <Namensraum>.setErrors  
  Die Methode erstellt Fehlermeldungen auf der Seite. Im Unterschied zu APEX-generierten Fehlern beinhalten die durch das Plugin generierten Fehlermeldung im Notification-Bereich immer auch eine CSS-Klasse, die der ID des Elements entspricht, auf die sich die Meldung bezieht. Dadurch können Fehlermeldungen im Notification-Bereich selektiv entfernt werden.  
  Zusätzlich werden den Fehlermeldungen im Notification-Bereich Links beigefügt, mit denen auf die entsprechende Stelle im Code verzweigt werden kann.
* Methode <Namensraum>.steuereFehlerSchaltflaechen  
  Die Methode kontrolliert den Zustand der Elemente, die durch das Plugin als »Fehlerabhängige Elemente« deklariert wurden. Ob Fehler durch das Plugin oder durch den Renderprozess der APEX-Seite auftreten, ist egal: Solange Fehler auf der Anwendungsseite angezeigt werden, sind die entsprechenden Elemente deaktiv, ansonsten aktiv.

### APEX-Anwendung: Anwendung

Die APEX-Anwendung des Plugins dient der vereinfachten Pflege von Regelgruppen, Regeln, Aktionen und Aktionstypen. Zusätzlich zu den Grundfunktionen, also der Anlage, Pflege und dem Löschen der beschriebenen Elemente, verfügt die Anwendung über eine Validierungsfunktion, die Regeln gegen das Data Dictionary der APEX-Anwendung, für die die Regel erstellt wird, validiert.

Die Anwendung selbst ist mit APEX-Standardmitteln erstellt, die Stammdatentabellen werden über eine Transaktions-API gepflegt. Zudem werden durch die Anwendung nach dem Speichern von Änderungen die Änderungen propagiert, d.h. es werden die Validierung sowie die Neuerstellung der Entscheidungstabellen veranlasst.

Die Anwendung verwendet das Plugin, um Schaltflächen einzublenden, Berichte zu aktualisieren und weiteres mehr. Die hierfür benötigten Regeln werden in der Übersicht der Regelgruppen ausgeblendet, um ein versehentliches Löschen dieser Regeln zu vermeiden.

### APEX-Anwendung: Package ADC\_UI

Das Package ADC\_UI implementiert die Anwendungslogik für die APEX-Anwendung zur Verwaltung der Regelgruppen, Regeln, Aktionen und Aktionstypen. Die eigentliche Logik delegiert das Package an ADC\_ADMIN. Daher stellen die Methoden des Packages lediglich Wrapper um die Methoden des Packages ADC\_ADMIN dar, zum Teil mit unterschiedlicher Parameterausstattung.