Vom Rechteck zur Fabrikvisualisierung – Prozessvisualisierung in Apex mit HTML5 und SVG

Frank Schubert und Angela Wobar, AIS Automation Dresden GmbH

Für die Entwicklung von datenbasierten Web-Anwendungen stellt Apex ein ausgereiftes Entwicklungswerkzeug dar. Es bietet umfassende Möglichkeiten zur Präsentation von Daten in tabellarischer und grafischer Form. Mithilfe von HTML-Tags und CSS lassen sich vorgegebene Layouts einfach anpassen. HTML5 mit seinen erweiterten Sprach-Elementen bietet ganz neue Möglichkeiten, innovative und interaktive Web-Seiten zu gestalten.

Im Rahmen eines Projekts bestand die Aufgabe darin, die bestehende „C++"-Applikation FabEagle TFM auf eine mobile Plattform zu portieren. Augenmerk sollte dabei auf Plattform-Unabhängigkeit und ein flexibles Layout zur Umsetzung eines „Responsive Webdesign" gelegt werden. Die Visualisierung, die einen schnellen Überblick über den Zustand der Fertigung bietet, sollte dabei erhalten bleiben. Da die vorhandene Applikation auf einer Oracle-Datenbank basiert, stand der Verwendung von Apex als Framework nichts im Wege. Aber wie können die Fabrik- und Prozessvisualisierungen migriert werden? Dieser Artikel soll dabei keine Anleitung für eine Implementierung sein, sondern mit möglichst ein-

fachen Beispielen Anregungen für eine Umsetzung liefern.

SVG und seine Möglichkeiten

Mit der Einführung von HTML5 rendern die modernen Browser Scalable Vector Graphics (SVG) ohne Plug-in in HTML-Seiten. SVG stellt für das Zeichnen beliebiger Formen das Pfad-Element zur Verfügung. Zur Vereinfachung gibt es für die Grundformen Rechteck, Kreis, Ellipse, Linie und Polygon ebenfalls entsprechende Elemente. Darüber hinaus bietet SVG Sprachelemente für Text, Animation und zum Einfügen externer Grafiken. SVG basiert auf XML, lässt sich mit CSS formatieren und mit JavaScript dynamisch ändern. Eine vollständige Abhandlung aller Sprachelemente würde an dieser Stelle den Rahmen sprengen. *Listing 1* zeigt ein kleines Beispiel, *Abbildung 1* die entstehende Grafik.

Aufgrund der einfachen Umsetzbarkeit und der in der Datenbank bereits vorhandenen Layout- und Equipment-Koordinaten lag es nahe, das SVG selbst zu generieren. Beispielhaft werden hier zwei Objekte, eine Produktionslinie („line 1“) und ein Equipment („equipment 1“), in einem Fertigungsbereich („shop-floor“) platziert. *Abbildung 2* zeigt das zugrunde liegende Koordinatensystem. In *Listing 2* ist die beispielhafte Umsetzung in SVG dargestellt; Fertigungsbereich, Equipment und Produktionslinie werden vereinfacht durch Rechtecke symbolisiert *(siehe Abbildung 3)*.

Schöner geht immer …

Die grundsätzlichen Anforderungen an die Fabrikvisualisierung sind nun erfüllt. Anlagen lassen sich positionsgenau im Fertigungsbereich platzieren. Ein Farbschema dient zur Visualisierung der Anlagenzustände. Werden nun aber mehrere Anlagen dargestellt, wird es schnell unübersichtlich. Zusätzliche räumliche Merkmale fehlen.

Natürlich könnte man die Gruppe „shop-floor“ erweitern und zusätzliche Information einzeichnen. Die Autoren sind jedoch einen anderen Weg gegangen. Zu jeder Fertigung existieren CAD-Zeichnungen, die als SVG exportiert werden können. Werden diese SVG-Dateien unter den „Shared Components“ der Applikation abgespeichert, können sie über ein Image-Tag eingebunden und so als Hintergrund genutzt werden.

Ein Beispiel dafür zeigen *Listing 3* und die daraus resultierende *Abbildung 4*. Somit haben wir mit wenigen Zeilen Code eine sehr ansprechende Visualisierung implementiert. Wem die Equipments noch zu rechteckig sind, kann auch deren Darstellung durch SVG-Dateien ersetzen.

```
<svg width="640" height="480">
  <g stroke="#000000" stroke-width="2">
      <rect height="50" width="100" y="50" x="50"
          fill="#7fff00"/>
      <circle r="30" cy="100" cx="150" fill="#aad4ff"/>
  </g>
</svg>
```

Listing 1: Einfaches SVG-Beispiel

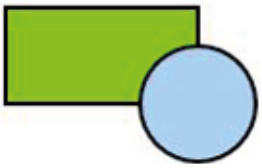

Abbildung 1: Das Ergebnis von Listing 1 – Rechteck und Kreis

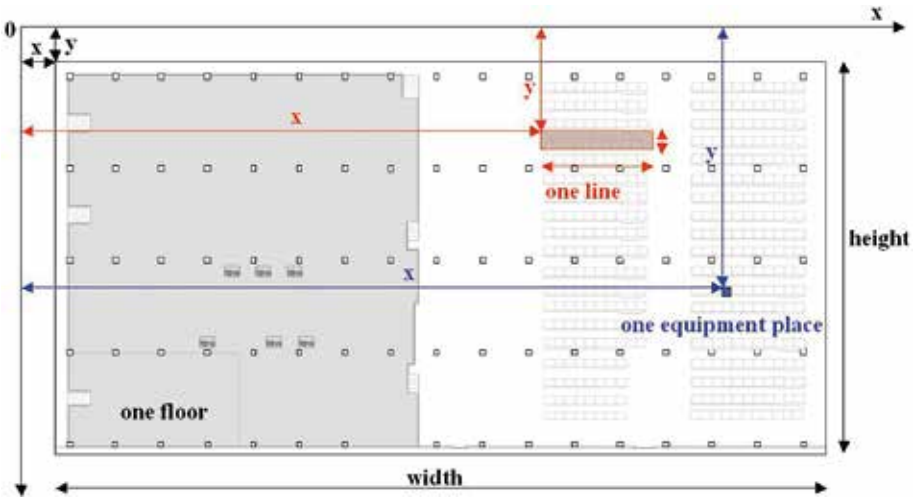

Abbildung 2: Fab-Koordinatensystem

```
<svg width="1000" height="400">
  <g stroke="#000000" stroke-width="1">
      <!—shop-floor -->
      <rect x="0" y="0" width="1000" height="400"
          fill="#ffffff" />
      <!-- line 1 -->
      <rect x="600" y="100" width="100" height="10"
          fill="orange" />
      <!-- equipment 2 -->
      <rect x="800" y="230" width="20" height="10"
          fill="blue" />
  </g>
</svg>
```

Listing 2: Implementierung als SVG anhand der existierenden Datenpunkte

Generieren von SVG in PL/SQL

Bei komplexeren Visualisierungen empfiehlt es sich, die Generierung des SVG nicht direkt in Apex durchzuführen, sondern in die Datenbank auszulagern und entsprechende Funktionen beziehungsweise Prozeduren zu erstellen. Das ermöglicht einfacheres Testen, Debuggen und Fehlerhandling. Außerdem sollten die in Oracle vorhandenen XML-Funktionen genutzt werden. Das garantiert Typsicherheit und eine bessere Performanz. Ein Anwendungsbeispiel ist in *Listing 4* zu sehen. Zudem bietet PL/SQL viele Funktionen zum Modifizieren von existierendem XML (wie „insertchildxml", „updateXML" etc.).

Abbildung 3: Resultierende Darstellung

```
<svg width="1010" height="710">
  <!—shop-floor -->
  <image x="0" y="0" height="710" width="1040"
     xlink:href="#APP_IMAGES#FAB.svg" />
  <!-- equipments -->
  <g stroke="#000000" stroke-width="1">
     <!-- equipment 1 -->
     <rect x="555" y="127" width="100" height="10"
        fill="orange" />
     <!-- equipment 2 -->
     <rect x="800" y="230" width="10" height="20"
        fill="blue" />
  </g>
</svg>
```

Listing 3: Integration von externen SVG-Dateien über ein Image-Tag

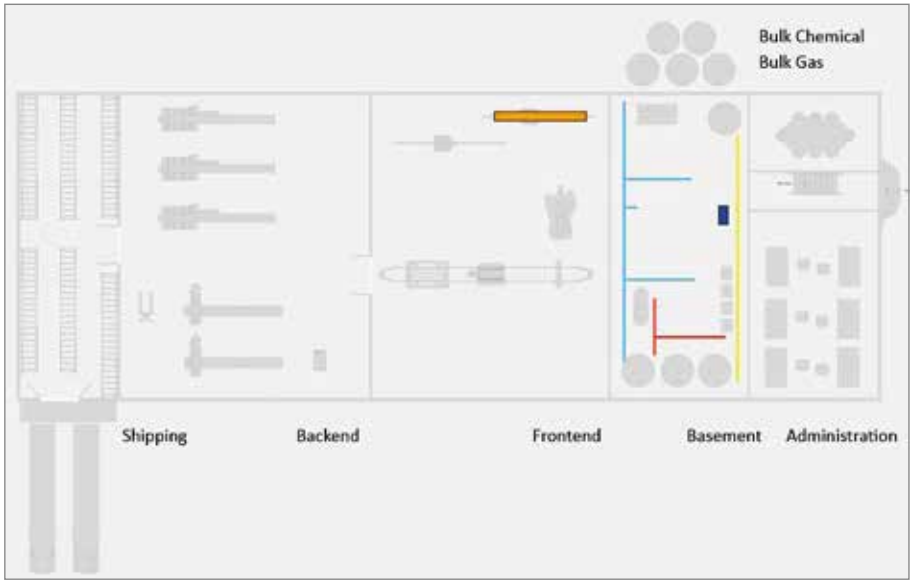

Abbildung 4: Resultierende Darstellung – externes SVG als Fabriklayout und 2 Equipments

Benutzer-Interaktionen

Von innovativen Benutzeroberflächen erwarten Anwender heutzutage dennoch mehr Funktionalität als nur eine statische Grafik. Tooltips, Links und Zoom gehören zu modernen Anwendungen einfach dazu. Die Kombination von JavaScript und SVG macht auch das möglich.

Den SVG-Elementen können Maus-Events („onmouseout", „onmousemove", „onclick") hinzugefügt werden. So lassen sich mit JavaScript-Funktionen Tooltips ein- beziehungsweise ausblenden und Links öffnen *(siehe Listing 5 und Abbildung 5)*. Im Internet lassen sich viele Beispiele für die Umsetzung von Tooltips in JavaScript finden. So lässt sich für jedes Equipment ein eigener Tooltip-div-Container erstellen, der mit Informationen über das Equipment gefüllt ist. Über CSS werden das Aussehen des Tooltip sowie dessen Position geregelt.

Neben dem im Browser vorhandenen Zoom möchte mancher Anwender eventuell auch die Visualisierung vergrößern, ohne die restliche Seite zu beeinflussen. Um diese Forderung umzusetzen, kann das ViewBox-Attribut des SVG-Elements genutzt werden. Die ViewBox definiert einen Anzeigebereich. Sie enthält vier Zahlenwerte, die den x- und y-Versatz sowie die Breite und Höhe des anzuzeigenden Koordinatensystems beschreiben. Alle Kind-Elemente des SVG werden in dem in der ViewBox beschriebenen Koordinatensystem dargestellt.

Wie groß die Grafik auf der Seite dargestellt ist, bestimmt sich aus den Attributen „height" und „width" des SVG-Elements *(siehe Listing 6 und Abbildung 6)*. Zum Zoomen wird dem SVG-Element beispielsweise ein Mousewheel-Event hinzugefügt, das eine Neuberechnung der ViewBox auslöst.

Heutzutage kann nicht davon ausgegangen werden, dass auf eine Website nur von einem Computer-Desktop aus zugegriffen wird. Der steigende Marktan-

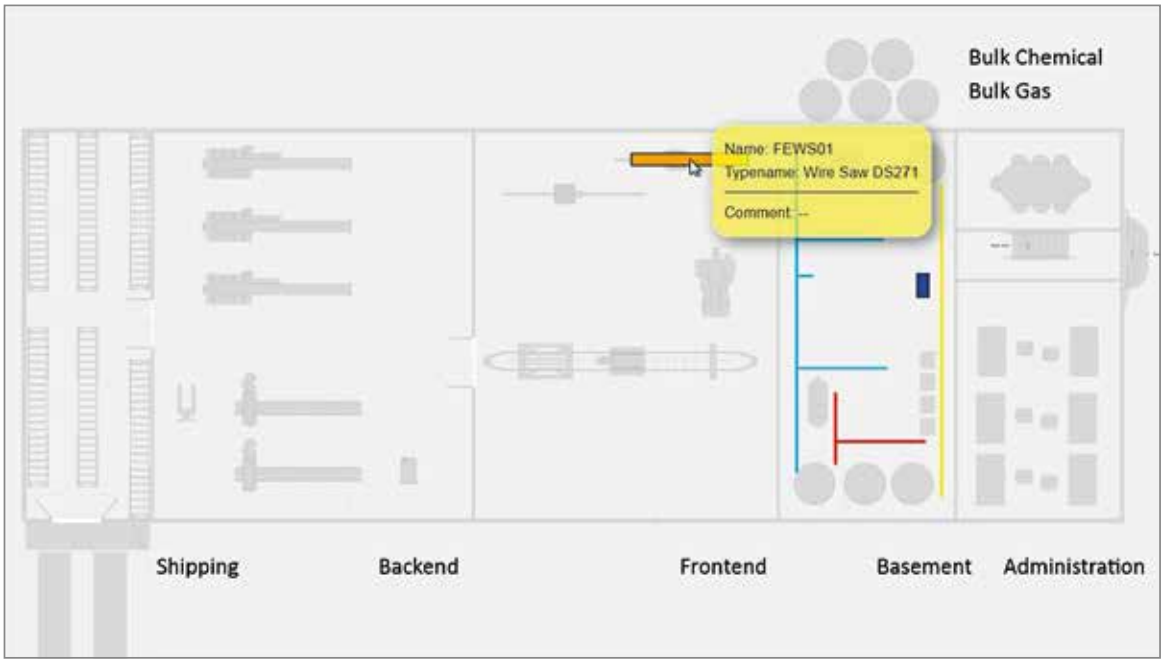

Abbildung 5: Resultierende Darstellung – Equipment mit Tooltip

teil von Smartphones und Tablets führt zu der Anforderung, die Website so flexibel zu gestalten, dass die Benutzerfreundlichkeit unabhängig vom verwendeten Endgerät erhalten bleibt. „Responsive Webdesign" lautet das Stichwort.

Für ein SVG ist diese Technik besonders einfach umzusetzen. Anstatt einen festen Wert für die Breite und Höhe des SVG anzugeben, kann eine flexible Breite in Prozent zugewiesen werden *(siehe Listing 7)*. So passt sich das SVG jeder Auflösung an.

Dynamische Aktualisierung

Nun bringt eine Prozessvisualisierung reichlich wenig, wenn man nur statische Daten anzeigt. Daher ist es notwendig, eine Aktualisierung umzusetzen. Der einfachste Weg wäre, eine Funktion zur Aktualisierung der gesamten Seite, auf der sich die Visualisierung befindet, zu implementieren und diese in einem bestimmten Zeitintervall aufzurufen. Dies führt jedoch zu Performance-Einbußen, da auch Daten nachgeladen werden, die sich nicht verändert haben. Eine performantere Lösung stellt das dynamische Aktualisieren der Seite dar, bei der nur die DOM-Elemente bearbeitet werden, deren zugrunde liegende Daten sich auch tatsächlich verändert haben.

```
<svg id="svg_zoom" width="100" height= „100" viewBox="0 0 100 100">
[…]
<svg id="svg_zoom2" width="100" height= „100" viewBox="-100 -100 90 90">
[…]
```

Listing 6: Zwei SVGs mit gleicher Höhe und Breite, aber unterschiedlichem ViewBox-Attribut

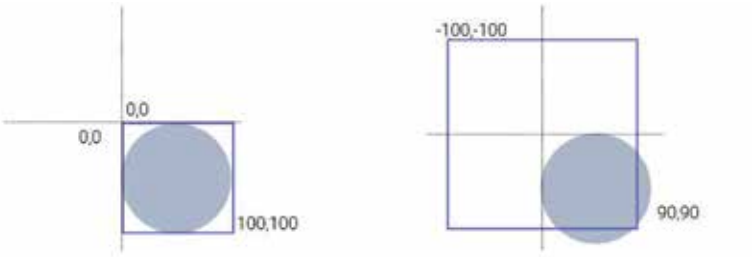

Abbildung 6: Resultierende Abbildung – Der blau umrandete Bereich würde jeweils mit einer Größe von 100x100 Pixeln dargestellt werden.

```
<svg style= ="width:100%" viewBox="0 0 100 100">

 […]
```

Listing 7: Dieses SVG passt sich flexibel an die Bildschirmgröße an.

Dazu kann beispielsweise ein Prozess erstellt werden, der bei einem AJAX-Aufruf gestartet wird und ein JSON-Objekt zurückliefert. *Abbildung 7* zeigt die Konfiguration eines solchen Prozesses. Der Source-Typ ist PL/SQL-Code, in dem das JSON-Objekt in den HTP-Buffer geschrieben wird. Die Daten kann der Apex-Entwickler manuell einfügen. Dazu können PL/SQL-Funktionen wie „htp.prn()" zum Einsatz kommen.

Um ein valides JSON-Format zu erhalten, ist jedoch genau darauf zu achten, an den richtigen Stellen entsprechend den Datentypen Anführungszeichen und alle erforder-

lichen Klammern und Kommata zu setzen. Um Formatfehler im JSON zu vermeiden, empfiehlt sich die Verwendung des mit Apex 5.0 eingeführten „Apex_JSON"-Package. *Abbildung 8* zeigt das JSON-Objekt, das unser Beispiel-Prozess zurückliefert.

Der AJAX-Aufruf wird clientseitig gestartet. In Apex lässt sich der „apex.server"-Namespace für eine solche Kommunikation mit der Datenbank nutzen. Durch die asynchrone Datenübertragung bleibt die Seite bedienbar, während auf die Server-Antwort gewartet wird. Außerdem ist es möglich, mehrere asynchrone Anfragen gleichzeitig zu stellen.

Die vom Server erhaltenen Daten lassen sich im JavaScript-Code weiterverarbeiten. Über die zugrunde liegende DOM-Struktur können SVG-Elemente direkt adressiert werden. Dabei wird das „id"-Attribut verwendet. *Listing 8* zeigt einen beispielhaften Serveraufruf zur Daten-Aktualisierung.

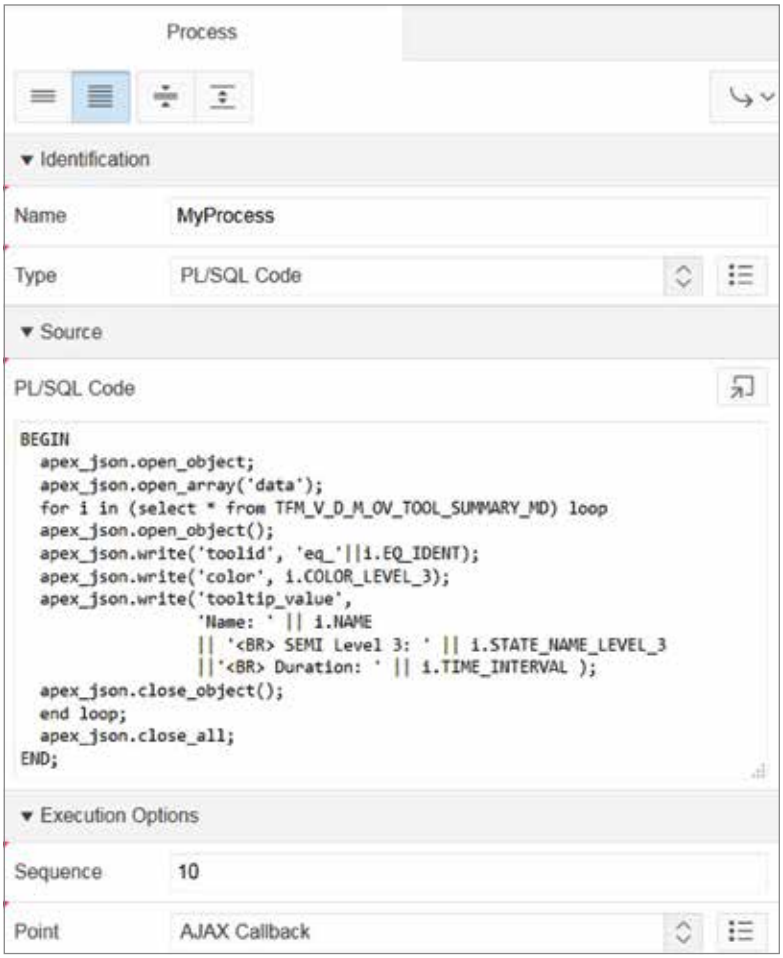

Abbildung 7: Beispiel-Konfiguration eines AJAX-Prozesses zur Aktualisierung

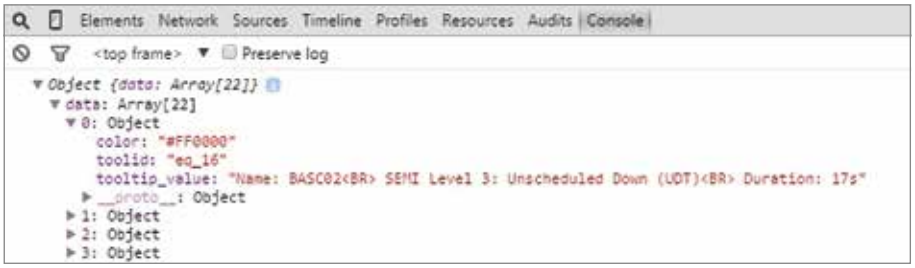

Abbildung 8: Das entstandene JSON-Objekt

Prozessvisualisierung

Nach dem gleichen Prinzip kann man auch Prozessvisualisierungen beziehungsweise Anlagen-Fließbilder *(siehe Abbildung 9)* in eine HTML-Seite integrieren. Das Fließbild wird dabei komplett als SVG-Grafik erstellt und die Aktualisierung der Prozesswerte und Zustände erfolgt über JavaScript. Die Objekte werden über das „id"-Attribut der SVG-Elemente angesprochen und aktualisiert.

Wiederwendbarkeit

Um die realisierte Fabrikvisualisierung auch in anderen Projekten verwenden zu können, ist ein Apex-Plug-in entstanden. Um dieses nutzen zu können, sind nur ein Select-Statement für das Fabrik-Layout und ein weiteres Statement für die anzuzeigenden Zustände und Detail-Informationen der einzelnen Anlagen zu übergeben.

Fazit

Mit HTML5 und SVG lassen sich komplexe Visualisierungen einfach in Apex integrieren. Die resultierenden Grafiken sind skalierbar und auch auf mobilen Endgeräten gut lesbar. Auch Benutzer-Interaktionen lassen sich einfach umsetzen. Trotz komplexer

```
/*Serveraufruf um aktuelle Daten zu erhalten*/
apex.server.process(„MyProcess", {/*Übergabeparameter*/}, {
      /*Callback-Funktionen*/
   success: function(pData) {
      //pData enthält die Serverantwort
      /* Funktion wird im Falle einer erfolgreichen Datenabfrage
          ausgeführt --> Weiterverwendung pData nach Wunsch, z.B.
          zur Aktualisierung der Farben und Tooltips*/
      },
   error: function(pData) {console.log(„error", pData);}
      } );

/*Funktion zur Aktualisierung der Farbe*/
function setToolColor(tool_id, color){
      $(,#'+tool_id).attr(„fill", color);
```

Listing 8: Verwendung des „apex.server"-Namespace und Beispielfunktion zur Farb-Aktualisierung

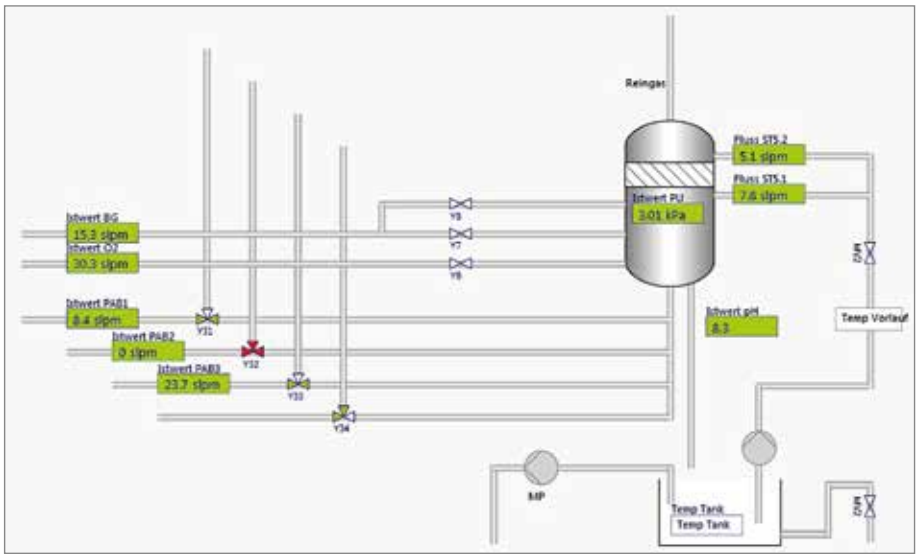

Abbildung 9: Prozessvisualisierung mit SVG

Grafiken waren keine Performance-Probleme bei der Darstellung in verschiedensten Browsern und Endgeräten feststellbar. Wer Interesse hat, findet über das Projekt unter *„https://tfmportable.ais-automation.com/apex/f?p=400:700"* eine Live-Demo.

Frank Schubert
frank.schubert@ais-automation.com

Angela Wobar
angela.wobar@ais-automation.com

Heatmap und ADO in der Praxis

Florian Feicht, Trivadis GmbH

Aufgrund der rasant wachsenden Datenmengen gewinnt der Lebenszyklus der Daten in der Datenbank immer mehr an Bedeutung. Müssen die Archivdaten auf teurem SAN liegen? Können weniger genutzte Daten komprimiert werden? Gibt es einfachere Möglichkeiten? Die Oracle-Antwort in der Version 12*c* ist ein in die Datenbank integriertes Information-Lifecycle-Management-System (ILM). Dazu wurden die Features Heat Map und Automatic Data Optimization (ADO) eingeführt.

Seit der Version 9*i* bietet die Oracle-Datenbank die Möglichkeit, Daten zu komprimieren. Dadurch benötigen diese im Idealfall weniger Speicherplatz und der Lesezugriff wird beschleunigt. Leider ist es aufgrund des Performance-Overheads nicht möglich, Komprimierung für die gesamte Datenbank auf Knopfdruck zu aktivieren. Die Entscheidung darüber, wo Komprimierung sinnvoll ist, erfordert gute Kenntnisse der Applikation. Dazu musste bisher eine aufwändige Auswertung erstellt werden. Außerdem ist es oft nicht sinnvoll, ganze Tabellen zu komprimieren, sondern nur die selten benutzten Einträge. Eine Methode, verschiedene Storage-Klassen am Datenbankserver automatisiert zu verwalten, stand nicht zur Verfügung.

12c bringt die Übersicht

Mit Einführung der Version 12c wurde ein Information Lifecycle Management implementiert. Eine automatisierte Verwaltung der Daten durch ihren „Lebenszyklus" ist dadurch sehr einfach möglich. Um den aktuellen Lifecycle-Stand zu überwachen, wird die Heat Map genutzt *(siehe Abbildung 1)*. Mit diesem Feature werden die Zugriffe auf die Datenbankobjekte zentral protokolliert. Dadurch entsteht eine feingranulare Übersicht auf folgende Ebenen:

- Block
- Extent