CAD by XML.
Was XML in Planungssystemen leisten kann

Dr.-Ing. Michael Huhn
Michael.T.Huhn@gmx.de
huhn@uni-wuppertal.de

Abstrakt

Das Markup-Format XML hat den ersten Hype hinter sich gelassen. Jeder ‚kennt' XML, jedes System ‚schreibt und liest' XML. In der Praxis der Bauplanungs-Systeme wird die Bandbreite der XML-Technologie jedoch wenig genutzt.
Auf der Grundlage eigener Untersuchungen und mit Blick auf andere Bereiche der Angewandten Informatik präsentiert dieses Papier eine Anzahl von Möglichkeiten der Anwendung. Nutzen und Grenzen werden aufgezeigt, die Idee eines komplett XML-basierten Planungssystems wird entwickelt.

1. Einführung

1.1. Situation XML

XML-Technologien haben in allen denkbaren -und vormals undenkbaren- Bereichen der Informationstechnologie Einzug gehalten. Der Nutzen liegt dabei meist nicht in der Lösung einer isoliert betrachteten Aufgabe, sondern im Zusammenwirken einer Vielzahl von Faktoren.

- XML wird von allen großen Wettbewerbern am Softwaremarkt unterstützt.
- XML ist allgemein als Schnittstellenformat anerkannt. Existierende (Datei-)Schnittstellen werden umgeschrieben /1/, um XML-Features erweitert (*property sets* in den IFC), neue Schnittstellen von vornherein in XML formuliert (/3/, /4/).
- XML erreicht Endgeräte aller Formfaktoren. Palm, Psion, PocketPC, Handies von Motorola etc. bieten Basissoftware, um XML zu verarbeiten. Es gibt leichtgewichtige Toolkits, welche XML auch in einer Umgebung mit beschränkten Ressourcen parsen und formatieren können. Große Anbieter von CAD-Lösungen wollen Designinhalte per XML auf mobile Endgeräte bringen /5/.
- Existierende, native Datenformate werden auf XML umgestellt. Prominentes Beispiel ist das neue Office /6/,/7/,/8/ von Microsoft, die Open Office Initiative /9/ bietet diese Möglichkeit schon seit längerem.
- In vielen Projekten, die sich mit der Verteilung bzw. leichteren Verteilbarkeit von Aufgaben im Netz befassen, wird XML mindestens für die Kommunikation verwendet.
- Die neue „Wunderwaffe" Web Services zur plattformübergreifenden Zusammenarbeit von Applikationen /10/ basiert auf XML.

1.2 Situation Planungssysteme

Im Bauwesen resultieren neue Anforderungen aus einer zunehmend offenen Planungsumgebung. Es wird erwartet, dass beliebige IT-Systeme verknüpft werden können. Der bloße Austausch von Daten über Dateischnittstellen reicht oft nicht mehr aus. Der Einsatz mobiler Kommunikation und neuer Endgeräte wird erschwinglich, es können weitere Personenkreise (Baustelle, Montage) am Datenaustausch teilnehmen. Die Beteiligten kommunizieren nicht nur administrative (Meta-)Daten, sondern Planungsinhalte.

Bildeten in der Vergangenheit monolithische CAD-Systeme das (technische) Rückgrat der Planung, so sind es in Zukunft virtuelle Plattformen. *Early adopters* dieser Technologie sind

Bauportale, die als Projekt-Hoster Planungsdokumente verwalten /11/. Dies wird nicht ausreichen, es muss auch die Kommunikation zwischen den Systemen (AVA, Statik, CAD, AV etc.) realisiert werden. Letztendlich müssen sich die Systeme ‚öffnen' im Sinne von Software-Komponenten, die temporär so „zusammengesteckt" werden können /26/, wie auch Planungsteams gerade zusammenarbeiten.

1.3 Motivation

Die erfolgreiche Anwendung von XML-Technologien für einzelne Bereiche ermutigen, Teilsysteme zu verknüpfen. Insbesondere die Web Services liefern die Basis dazu.
Weiterhin haben XML-Technologien eine Stufe erreicht, auf der nicht mehr nur Baumstrukturen, sondern vielfältige Datenmodelle repräsentiert werden können.

Nachfolgend wird zunächst auf die Bildung dieser Modelle eingegangen, anschließend werden Beispiel-Anwendungen in Planungssystemen diskutiert. Strategisches Ziel ist eine Applikationslandschaft, bei der im Web veröffentlichte Komponenten zu Planungssystemen zusammengeschaltet werden können.

2. XML

2.1 Begrifflichkeiten XML

XML ist ein Meta-Standard. Er beschreibt zunächst abstrakt die Bestandteile von XML Dokumenten im XML *Information Set*, sodann eine Basissyntax (*Extensible Markup Language*, *namespaces* und *xmlbase*), Datentypen und Strukturen. Weiterhin beschrieben werden das Konzept des Dokuments und die Möglichkeiten des Zugriffs darauf (*Document Object Model* DOM, XML *Path Language*, *Simple API for XML* SAX). Auf diesem Level wird im nachfolgenden Text vom XML Modell gesprochen.

Der Begriff XML Sprache meint hingegen nachfolgend ein konkretes Datenmodell, dessen Bestandteile syntaktisch dem XML Information Set folgen, dessen Datentypen in einem XML Schema definiert sind etc.

XML Dokumente für ein XML Schema bzw. für Daten werden nachfolgend als Schemadokumente bzw. Instanzdokumente bezeichnet.

2.2 Objektmodelle und XML-Technologien

Ein Objektmodell repräsentiert Struktur und Verhalten. Objekte besitzen Eigenschaften und enthalten Methoden bzw. können *messages* behandeln, die wiederum Objekte erzeugen und löschen, Objekteigenschaften lesen und setzen, sowie die Struktur der Objekte modifizieren können.

Nun besteht bei den üblichen Programmiersprachen und Datenbanken eine Diskrepanz zu diesem allgemeinen Anspruch hinsichtlich der Serialisierung und Speicherung von Objekten. Es wird üblicherweise nur die Struktur übertragen und aufbewahrt, die Methoden verbleiben im lokalen, ausführbaren Code.
So reicht etwa in AutoCAD die Übertragung eines Modells per DWG (lediglich Daten) nicht aus, wenn anwendungsnahe Erweiterungen per ObjectARX-Aufsatzprogrammierung verwendet werden. Ein entsprechendes .dbx Modul (Methoden) muss ebenfalls geladen werden. Auch wenn zur Laufzeit über Internet quasi „unsichtbar" nachgeladen werden kann, ist hier eine Trennung von Daten und Methoden existent.
Objektorientierte Datenbanken /13/, /14/ versprechen Objektpersistenz, realisieren aber nur die transparente Speicherung der Objektstruktur, nicht der Methoden.

XML per se kann Daten strukturiert darstellen.
Mit Hilfe des XML Schema /15/ können komplexe Datenstrukturen (eigentlich die Struktur eines

XML documents) definiert werden. Ein Schemadokument beschreibt ein Instanzdokument ähnlich wie eine Klasse ein Objekt – mit Ausnahme der Methoden.

Tatsächlich kann ein in einem UML Klassendiagramm beschriebenes System von Klassen mit Hilfe der XMI Spezifikation /16/ direkt in ein XML Schema gewandelt werden /17/ und umgekehrt. Dabei ist zu beachten, dass dieses *mapping* nur eine von mehreren Varianten darstellt, es sind auch andere Produktionsregeln möglich. XMI wurde mit dem Ziel entwickelt, Daten zwischen CASE-Tools auszutauschen und beschreibt eigentlich eine Abbildung von Metamodellen aus Modellrepositories (nach MOF /18/) auf XML. Das vorrangige Ziel bestand also nicht darin, Klassen bzw. Objekte in XML zu serialisieren.

Man kann zeigen, dass übliche Konstruktionen der objektorientierten Modellierung (Klassenbildung, Zuordnung von Eigenschaften, Spezialisierung, Verallgemeinerung, einfache Vererbung, Bildung abstrakter Typen, Bildung von Relationen verschiedener Art) auf XML übertragen werden können. Der Autor hat für verschiedene Zwecke entsprechende Produktionsregeln entwickelt, Anwendungsbeispiele werden im nächsten Kapitel erläutert. Dies betrifft jedoch nur den strukturellen Aspekt, die Abbildung des Verhaltens erfordert mehr:

Eine der auf XML beruhenden Technologien sind die *Web Services* /10/. Hier wiederum beschreibt die WSDL /19/ Kommunikationsendpunkte, insbesondere die Signatur der *messages*, welche an diesen Endpunkten behandelt werden. In einer entsprechenden Architektur kann man dies interpretieren als die Definition von Interfaces und die Veröffentlichung von „Adressen“ statischer Objekte, welche diese Interfaces implementieren. Für die Abbildung der Parameter und Returnwerte der Methoden stehen einfache Datentypen zur Verfügung. Gemäß SOAP ist die Abbildung auf Datentypen der entsprechenden Sprachen definiert.

Wie in /20/ beschrieben, kann man jedoch beliebig komplexe Objekte als Parameter übertragen. Dabei handelt es sich um in XML serialisierte Objekte, die auch in einem Web Service manipuliert werden können, siehe Abb. 1.

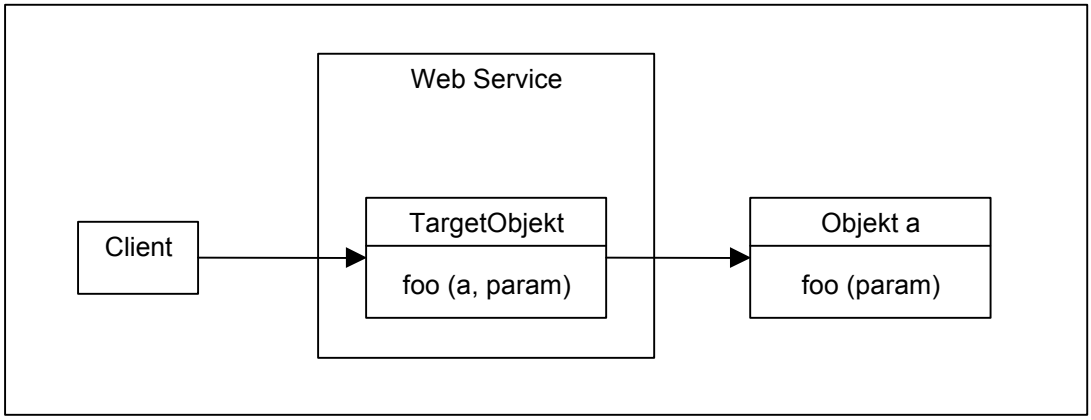

Abb. 1 Ansprechen eines Anwendungsobjekts per Web Service

Der Service bietet eine Methode foo, welche das Anwendungsobjekt (und die eigentlichen Parameter) als Parameter verwendet. Der Service (re-)instanziiert das Objekt und wendet die Methode an. Bei diesem Verfahren ist jedoch der Lebenszyklus derartiger Anwendungsobjekte auf Meta-Ebene zu beachten, in der objektorientierten Implementierung existieren hier zwei Objekte – auf Client-Seite und auf Service-Seite.
Durch konsequente Kapselung ist es möglich, Objekte auch logisch in der Implementierung des jeweiligen Web Service zu belassen. Bei Einhaltung bestimmter Regeln treten die oben erwähnten Lebenszyklus-Probleme nicht auf.
Bei entsprechender Implementierung können Objekte „hinter“ einem Service auch als persistent betrachtet werden. Eine adäquate Infrastruktur wird von großen Implementierungen bereitgestellt, bei Apache AXIS /20/ etwa für JavaBeans.

Damit beinhalten die XML-Technologien alle Werkzeuge, um ein Äquivalent zur heutigen Verwendung von Objektmodellen aufzubauen, Abb. 2.

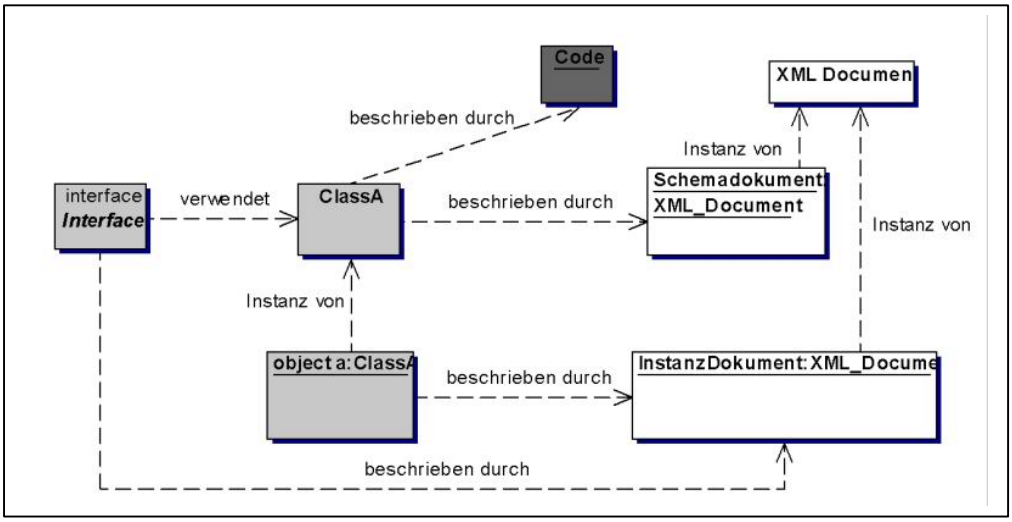

Abb. 2 Verhältnis von Objektmodell und XML-Dokumenten

Es ist - im übertragenen Sinne - möglich, Klassen zu bilden, Objekte zu instanziieren, Objekte (genauer: Objektdaten) zu speichern, Klassenmethoden zu definieren und diese Methoden zu verwenden.

3. Anwendungen

3.1 Customizing von Anwendungen

Jedes Planungssystem verwendet – wie jede Software generell – ein System von *defaults* für Voreinstellungen. Die *defaults* werden vorab oder zur Planungszeit bearbeitet, abgespeichert, ausgetauscht und natürlich bei Projektwechsel geladen. Dies betrifft

- die Steuerung der Software selbst in Form von Fenstergrößen, Farben usw.,
- administrative Daten des aktuellen Projekts wie Projektleiter, Standorte und ähnliches,
- Einstellungen des Planungsinhalts wie Materialfächer, Zeichnungsnorm usw.

Projektunabhängige und projektspezifische Voreinstellungen sollten getrennt werden, oft sind die Projekteinstellungen mit den Projektdaten selbst vermischt.

An *defaults* werden meist nur geringe Anforderungen gestellt, Abb. 3.

- Typischerweise werden *key-value*-Paare gebildet:
  Standard= S235JRG2
- Die Werte sind hierarchisch organisiert nach Bereichen, die Hierarchien sind verschieden tief:
  Material -- Winkelprofile = S235JRG2
  -- Rundmaterial = C45E
- Einigen *keys* wird eine variable Anzahl von Werten (Aufzählung) zugewiesen:
  Materialfächer = S235JRG2
  C45E
  S355J0
  S355JR

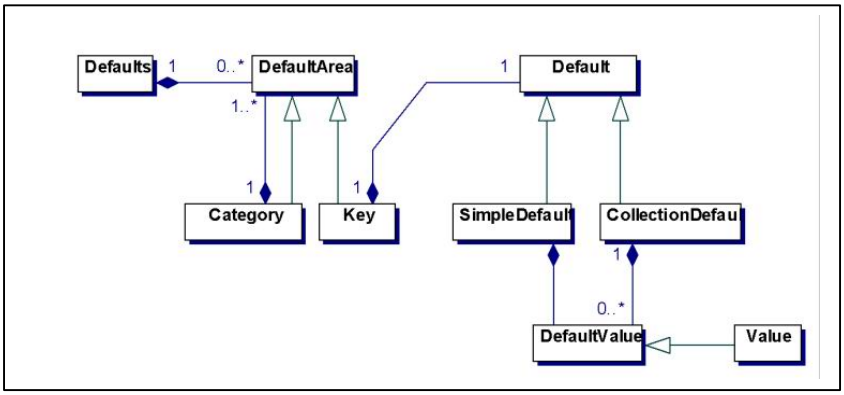

Abb. 3 Ein einfaches *defaults* Modell

Das Modell lässt sich als Schemadokument darstellen, Abb. 4. Ein Instanzdokument zeigt Abb. 5.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsd:schema xmlns:xsd=
      "http://www.w3.org/2001/XMLSchema"
      xmlns:ik1="urn:ikm:def"
      targetNamespace="urn:ikm:def">
  <xsd:element name="Defaults"
  type="ik1:defaultsType" />
  <xsd:complexType name="defaultsType">
    <xsd:sequence>
      <xsd:choice minOccurs="0"
  maxOccurs="unbounded">
          <xsd:element ref="ik1:Category" />
          <xsd:element ref="ik1:Key" />
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
    ...
  <xsd:element name="Value">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string" />
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="SimpleDefault">
    <xsd:complexType>
      <xsd:sequence>
          <xsd:element ref="ik1:Value" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="CollectionDefault">
    <xsd:complexType>
      <xsd:sequence>
          <xsd:element ref="ik1:Value"
  minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Abb.4 Ausschnitt aus dem Schemadokument des *defaults* Modells

```
:?xml version="1.0" encoding="iso-8859-1" ?>
<Defaults xmlns="urn:ikm:def"
xmlns:ik1="urn:ikm:def"
xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance" xsi:schemaLocation="urn:ikm:def
defaults.xsd">
  <Category name="Vorschlag">
      <Category name="Material">
          <Key name="Standard">
              <SimpleDefault>
                  <Value>S235JRG2</Value>
              </SimpleDefault>
          </Key>
          <Key name="Winkelprofile">
              <SimpleDefault>
                  <Value>S235JRG2</Value>
              </SimpleDefault>
          </Key>
          <Key name="Rundmaterial">
              <SimpleDefault>
                  <Value>C45E</Value>
              </SimpleDefault>
          </Key>
      </Category>
  </Category>
  <Key name="Materialfächer">
      <CollectionDefault>
          <Value>S235JRG2</Value>
          <Value>S355J0</Value>
          <Value>S355JR</Value>
          <Value>C45E</Value>
      </CollectionDefault>
  </Key>
</Defaults>
```

Abb.5 Ein Instanzdokument des *defaults* Modells

Das in Abb. 4 gezeigte Schema besitzt den Nachteil, dass die jeweils zugehörigen Elemente (im Objektmodell : *data member*) bereits explizit beschrieben sind. Bei einer weiteren Spezialisierung (Objektmodell : dem Ableiten weiterer Klassen) muss die Definition des jeweiligen Elternelements (Objektmodell : der Klasse) geändert werden.

Die in Abb. 6 gezeigte Konstruktion gestattet hingegen Erweiterungen ohne wesentliche Änderungen der Schema-Elemente. Es können neue Wertetypen von „DefaultValue“ abgeleitet werden, etwa ein „DoubleValue“ für numerische Defaults. Die neuen Wertetypen werden definiert, ohne die Elemente „SimpleDefault“ und CollectionDefault“ zu ändern. Das Element „Value“ wird nach wie vor verwendet, die Instanzdokumente sind nach wie vor gültig.

```
...   <xsd:element name="DefaultValue" type="xsd:anySimpleType" abstract="true" />
      <xsd:element name="Value" substitutionGroup="ik1:DefaultValue">
          <xsd:simpleType>
              <xsd:restriction base="xsd:string" />
          </xsd:simpleType>
      </xsd:element>
      <xsd:element name="SimpleDefault" substitutionGroup="ik1:Default">
          <xsd:complexType>
              <xsd:complexContent>
                  <xsd:extension base="ik1:defaultType">
                    <xsd:sequence>
                        <xsd:element ref="ik1:DefaultValue" />
                    </xsd:sequence>
                  </xsd:extension>
              </xsd:complexContent>
          </xsd:complexType>
      </xsd:element>    ...
```

Abb.6 Ausschnitt aus einem erweiterten Schemadokument des *defaults* Modells

Dies zeigt, wie durch geeignete Produktionsregeln die Qualität der Abbildung eines Objektmodells auf ein XML Schema verbessert werden kann.

Instanzdokumente wie aus Abb. 5 können direkt als Datei geschrieben und gelesen werden. In der einfachsten Form wird im Programm direkt auf das *Document Object Model* (DOM) der Struktur zugegriffen.

3.2 Massendaten

Planungssysteme sind umso anpassungsfähiger, je mehr Logik „ausgelagert“ wird. CAD-Systeme wie etwa Bocad /21/ oder HyperSteel /22/ realisieren mehr Konstruktionslogik über Makros als über fest kodierte Abschnitte. Es gibt generische Beschreibungen von Zeichnungsteilen wie Symbolen oder Schraffuren, von Bemassungs- und Beschriftungsstrategien u.v.a.
Diese externen Datenbestände können ebenfalls in XML modelliert werden. Das Beispiel in Abb. 7 zeigt in der Domäne Profile eine Reihe mit Kranschienen, hier mit einem einzigen Eintrag.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<root xmlns="urn:ikm:cat" xmlns:cat="urn:ikm:cat" xmlns:cat_sections="urn:ikm:cat:sections"  ...
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" urn:ikm:cat CatalogTypes.xsd urn:ikm:cat:sections   ...  >
  <catalog cat_class="catalog" name="IMPORT">
    <subcatalog cat_class="subcatalog" name="IMPORT">
      <itemgroup cat_class="itemgroup" name="KSA" xsi:type="cat_sections_KSA:itemgroupType">
                ...
          <source cat_class="itemgroupattr" standard="DSTV 1998" type="shouldConformTo" />
          <desc cat_class="itemgroupattr" local="de" val="Kranschienen, Form A (mit Fussflansch)" />
          <cat_sections_KSA:item cat_class="item" object="KSA45" A="2830" h="55" b="125" s="24"
t="11" r1="0.5" r2="0.4" G="22.2" M="0.38" Ix="91" Wx="27.5" Iy="169" Wy="27" Sx="0" Sy="-0.56">
            <cat_sections:outerContour cat_class="subitem">
                <cat_sections:point cat_class="subitem" x="-62.5" y="-27.5" />
                <cat_sections:point cat_class="subitem" x="62.5" y="-27.5" r="0" />
                ...
                <cat_sections:point cat_class="subitem" x="-27" y="-16.5" />
                <cat_sections:point cat_class="subitem" x="-62.5" y="-19.5" />
            </cat_sections:outerContour>
          </cat_sections_KSA:item>
      </itemgroup>
    </subcatalog>
  </catalog>
</root>
```

Abb.7 Instanzdokument mit Daten einer Kranschiene, Ausschnitt

Das Schemadokument, Abb. 8, zeigt zwei mögliche Prinzipien der Ableitung einer speziellen Klasse aus einer allgemeinen Klasse: Ableitung per Restriktion oder per Erweiterung. Dies kann beliebig tief und nach bestimmten Regeln auch gemischt erfolgen.
Die im Schema definierten Typen verweisen auf Basistypen, die wiederum auf weitere Ebenen. Analog existiert in diesem Beispiel eine Hierarchie von vier Schemadokumenten. Dies illustriert: Es ist auch per XML möglich, arbeitsteilig abzuleiten. Der Designer der Applikation entwickelt im vorliegenden Fall Basisklassen, die Grundstruktur, Persistenz, Editierbarkeit usw. sichern. Der Anwendungsentwickler erarbeitet Anwendungsklassen der Domäne, etwa allg. Walzprofile, darauf Kranschienen, darauf Profile der Reihe KSA.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:cat="urn:ikm:cat"
xmlns:cat_sections_KSA="urn:ikm:cat:sections:KSA" xmlns:cat_sections_rail="urn:ikm:cat:sections:rail"
xmlns:cat_sections="urn:ikm:cat:sections" targetNamespace="urn:ikm:cat:sections:KSA"
elementFormDefault="qualified">
    <xsd:import namespace="urn:ikm:cat" schemaLocation="CatalogTypes.xsd" />
    <xsd:import namespace="urn:ikm:cat:sections" schemaLocation="CatalogTypes_sections.xsd" />
    <xsd:import namespace="urn:ikm:cat:sections:rail"
schemaLocation="CatalogTypes_sections_rail.xsd" />
    <xsd:complexType name="restrictedGroupType">
      <xsd:complexContent>
        <xsd:restriction base="cat:itemgroupType">
          <xsd:sequence>
            <xsd:element ref="cat:name" minOccurs="0" maxOccurs="1" />
            <xsd:element ref="cat:creator" minOccurs="1" maxOccurs="unbounded" />
            <xsd:element ref="cat:source" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element ref="cat:implementation" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element ref="cat:desc" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element ref="cat:drop" minOccurs="0" maxOccurs="1" />
            <xsd:element ref="cat:edit" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element ref="cat_sections_KSA:item" minOccurs="1" />
          </xsd:sequence>
          <xsd:attribute name="cat_class" type="xsd:NCName" use="required" fixed="itemgroup" />
          <xsd:attribute name="name" type="cat:structureName" use="required" />
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="itemgroupType">
      <xsd:complexContent>
        <xsd:extension base="cat_sections_KSA:restrictedGroupType">
          <xsd:sequence>
            <xsd:element ref="cat_sections_KSA:item" minOccurs="0" maxOccurs="unbounded" />
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="item" type="cat_sections_KSA:itemType" substitutionGroup="cat:item" />
    <xsd:complexType name="itemType">
      <xsd:complexContent>
        <xsd:extension base="cat_sections_rail:itemType" />
      </xsd:complexContent>
    </xsd:complexType>
</xsd:schema>
```

Abb.8 (Oberstes) Schemadokument der Kranschienen der Reihe KSA

Generell kann für Massendaten der Datenumfang sehr groß werden (>100 MB), andererseits erfolgt der Zugriff auch nur sporadisch und nur auf Teile des Datenbestandes. In diesem Fall hat sich gezeigt, dass die Daten besser außerhalb des Planungssystems bleiben und nicht in ein Gesamt-DOM eingelesen werden sollten.

In einem Evaluationsprojekt /23/ wurden diese Dokumente als Textdaten in einer objektorientierten Datenbank abgelegt. Dazu erfolgte eine Zerlegung in logisch getrennte Domänen, denen dann jeweils eine Anzahl von Instanzdokumenten zugeordnet ist. Der Zugriff erfolgt nach der Bereitstellung aller Dokumente einer Domain durch die Datenbank auf XML-Ebene.

Für das Suchen einer unbekannten Anzahl von Objekten mit bestimmten Eigenschaften wurde die SAX API verwendet, für den gezielten Zugriff auf ein namentlich bekanntes Objekt wurde im DOM

des anderweitig ermittelten Dokuments gesucht. Mit Hilfe einiger Caching-Mechanismen wurden bessere Ergebnisse als der Zugriff per ID/XPath/XSLT erzielt.

Vor der eigentlichen Verwendung der Massendaten müssen diese zunächst erarbeitet und katalogisiert werden. Im dazu notwendigen Editor kann die XML-Struktur gut gehandhabt werden, siehe Abb. 9. Alle notwendigen Struktureigenschaften wurden in den Basis-Schemata verankert. Damit können für jede von Anwendungsentwicklern erstellte neue „Klasse“ sofort Objekte mit dem Editor kreiert werden. Auf der speziellen Anwendungsebene kann der Entwickler in seinem Subschema die Attributeigenschaften aller Objekte einer Klasse definieren: Datentyp, Wertebereich, Erläuterungstexte u.a. Abb. 9 zeigt ein Bild des Editors aus /23/.

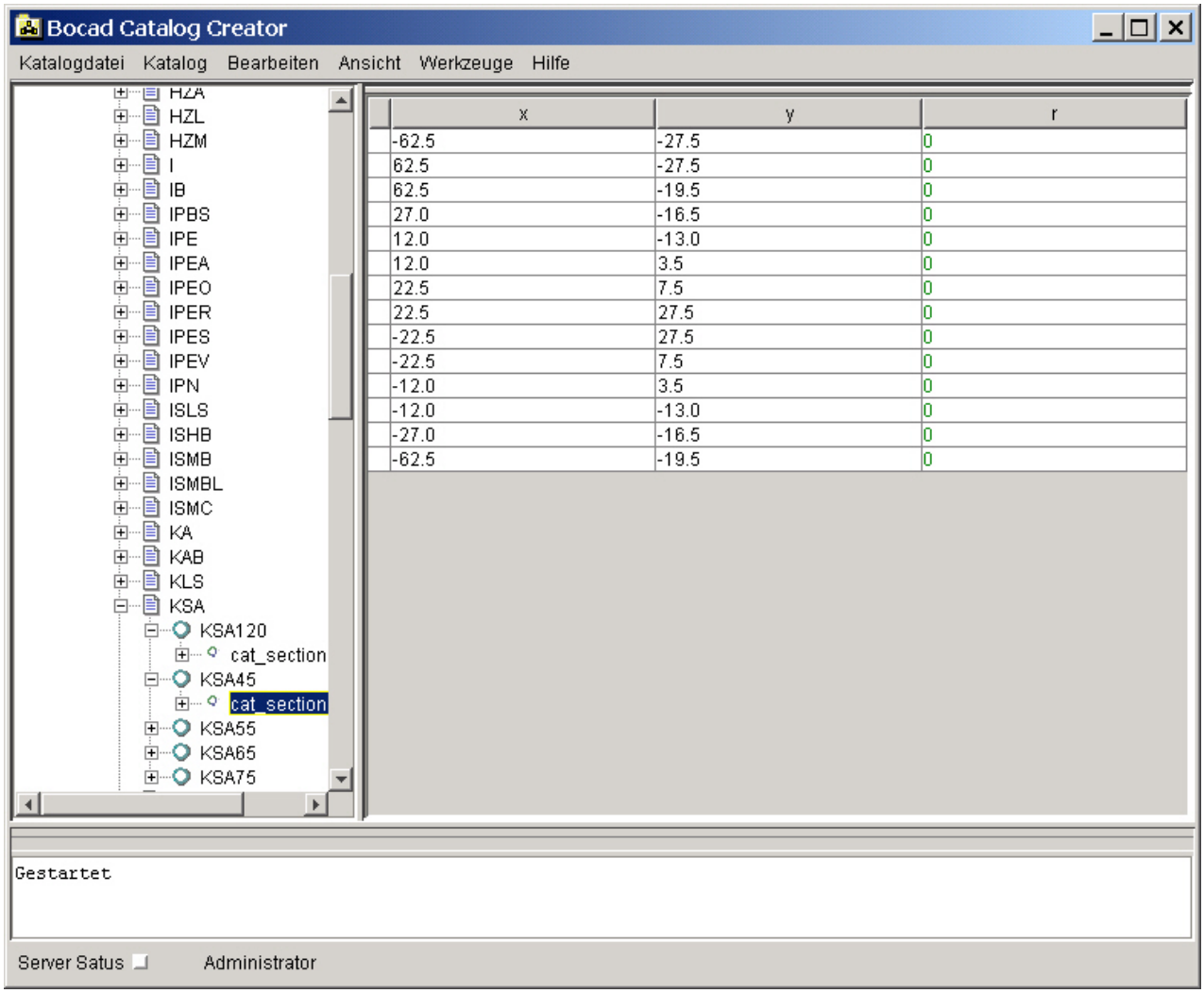

Abb.9 XML-basierter Editor für Massendaten

Der Explorer im linken Fenster zeigt einen Ausschnitt aus der hierarchischen Struktur aller Daten. Der Tabelleneditor zeigt die XML Attribute auf der entsprechenden Ebene. Die Koordinaten x und y wurden als double definiert, eine Kennung r besitzt einen gewissen Wertevorrat und die Voreinstellung 0.

3.3 Graphische Benutzungsoberfläche (GUI)

Insbesondere in portablen Toolkits wird seit jeher versucht,
das GUI neutral zu definieren und dann zur Übersetzungszeit oder zur Laufzeit auf ein konkretes Fenstersystem abzubilden.
Man kann versuchen, dies rein funktional zu tun, etwa Eingaben nur auf der Ebene *choice* (Auswahl eines Wertes aus einem definierten Wertevorrat), *evaluator* (freie Eingabe eines Wertes

bei Vorgabe des Datentyps), *locator* (Eingabe Raumpunkt) und *identificator* (Auswahl Objekt) zu betrachten. Dieser Ansatz ist sehr alt und schien angesichts der Vereinheitlichung der Fenstersysteme schon obsolet. Er erlangt heute erneut an Bedeutung unter dem Aspekt der Einbeziehung einer Flut portabler Geräte verschiedener Formfaktoren.
Dies wird hier nicht weiter diskutiert.
Ein konkreterer Ansatz ordnet bereits allgemein anerkannte Dialogelemente zu (die in unterschiedlichen Betriebs- und Fenstersystemen dann lediglich anders aussehen).
In diesem Fall erfolgt einmal die Beschreibung der Struktur eines Fensters, weiterhin der Mechanismus des Beschickens mit statischen oder dynamischen Daten, schließlich die Anbindung von Aktionen an die Ereignisse der Dialogelemente.

Im Java-Bereich gibt es viele Toolkits, welche mit Hilfe von XML Oberflächen beschreiben. Zur Laufzeit wird das GUI in Swing aufgebaut, ein Laufzeitsystem fängt die GUI *events* ab, um sie in *callbacks* zur Verfügung zu stellen. Ein gutes Beispiel stellt etwa JEasy /25/ dar.

Auch Qt /24/ hat eine XML Sprache entwickelt, in der die Komponenten eines GUI und ihre graphischen Eigenschaften beschrieben werden, siehe Abb. 10. Die entsprechende Datei wird zur Laufzeit geladen und das UI wird aufgebaut, siehe Abb. 11. Anschließend werden dynamische Inhalte eingefüllt und den Controller Komponenten die Aktionen – ebenfalls Programmteile – zugewiesen. Dies geschieht per (C++) Programm, also ebenfalls erst zur Laufzeit.
Das Prinzip erlaubt z.B. den Transport der GUI-Beschreibung vom Server zum Client. Obwohl so nicht vorgesehen, wären sogar Clients auf Basis anderer Programmiersprachen möglich, die sich ebenfalls ein GUI aus der Beschreibung zusammenbauen.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<UI>
  <class>FormTeileVerlegen</class>
  <widget class="QDialog">
  + <property name="name">
  + <property name="geometry">
  + <property name="sizePolicy">
  + <property name="minimumSize">
    <property name="caption">
          <string>Teile/Standardprofile</string>
    </property>
  + <property name="toolTip" stdset="0">
  - <widget class="QLayoutWidget">
      + <property name="name">
      + <property name="geometry">
      - <vbox>
          + <property name="name">
          + <widget class="QSplitter">
          - <widget class="QCheckBox">
              <property name="name">
                  <cstring>checkBox1</cstring>
              </property>
              <property name="text">
        <string>Teil als Leitteil generieren</string>
              </property>
            </widget>
        </vbox>
    </widget>
  </widget>
  <pixmapinproject />
  <layoutdefaults spacing="6" margin="11" />
</UI>
```

Abb.10 Instanzdokument eines Dialogelements, Teilansicht

Abb.11 Dialogelement in Qt

Auch in Qt werden - mit Hilfe eines eigenen Konzepts - den Ereignissen eigene Behandlungsmethoden zugewiesen. Wiederum geschieht dies zur Laufzeit im Code.
Wenn die Instanzdokumente aber als Objekte angesehen werden sollen, die im Web funktionsfähig zur Verfügung gestellt werden sollen, ist dies unbefriedigend. Die Trennung der Form des GUI vom Inhalt und von den Aktionen widerspricht dem Objektgedanken. Ein „UI-Objekt“ sollte graphische Eigenschaften und Methoden zum Setzen und Lesen dieser Eigenschaften besitzen.

Es sollte auf Ereignisse reagieren und Botschaften emittieren, analog einem *OLE Custom Control*. Dies kann man auf XML-Basis beschreiben:

Eine direkte Möglichkeit wäre die Zuordnung von Java-Klassen in XML, siehe Elemente load, init, action in Abb. 12. Diese Lösung ist plattformunabhängig, lokal und sprachabhängig.

```
...
<widget class="QCheckBox">
    <property name="name">
    <cstring>checkBox1</cstring>
    </property>
    <property name="text">
        <string>Teil als Leitteil generieren</string>
    </property>
  <load class="com.ikm.load"/>
  <init class="com.ikm.init"/>
  <action id="check" class="com.ikm.checkbox.handle">
  <action id="uncheck" class="com.ikm.checkbox.handle">
  <unload class="com.ikm.unload"/>
</widget>
...
```

Die bezeichneten Klassen werden mit dem Instanzdokument lokal zur Verfügung gestellt. Die Interaktion erfolgt beim Client. Ein Laufzeitsystem lädt die Klassen lokal, ordnet sie dem entsprechenden GUI-Element zu und führt sie aus.

Abb.12 Instanzdokument eines Dialogelements mit Ereignisbehandlung

Die indirekte Lösung ist die Zuordnung von *Web Services*. Die GUI-Beschreibung enthält lediglich Verweise auf die entsprechenden WSDL-Instanzdokumente, Abb. 13.

```
...
<widget class="QCheckBox">
  <property name="name">
      <cstring>checkBox1</cstring>
  </property>
  <property name="text">
        <string>Teil als Leitteil generieren</string>
  </property>
  <load desc="http://www.ikm.com/wsdl/load.wsdl"/>
  <init desc=" http://www.ikm.com/wsdl/init.wsdl"/>
  <action id="check" desc="http://www.ikm.com/wsdl/
checkbox.handle.wsdl">
  <action id="uncheck" desc="http://www.ikm.com/wsdl/
checkbox.handle.wsdl">
  <unload
desc="http://www.ikm.com/wsdl/unload.wsdl"/>
</widget>
...
```

Diese Lösung ist plattform- und sprachunabhängig und ist transparent hinsichtlich einer Verteilung. GUI und GUI-Verhalten sind deskriptiv formuliert. Der Client wird mit dieser Beschreibung beschickt, baut ein GUI (abhängig von der lokalen Plattform) auf und vollführt ein *customizing* mit Hilfe der mitgeschickten *Web Services*. Er interagiert mit dem(n) Objekt(en) auf der Serverseite ebenfalls per *Web Service*.

Abb.13 Instanzdokument eines Dialogelements mit Ereignisbehandlung

Interessant wird dieser Ansatz in einem größeren Kontext. Entwurfssysteme im Bauwesen laufen traditionell der Verwendung neuer Bauteile, Befestigungsmittel, Konstruktionsregeln etc. hinterher. Viele Systeme (/21/, /22/) haben ihre eigenen proprietären Verfahren entwickelt, um Oberflächen zu definieren und zur Laufzeit mit dem System zu verknüpfen. Im Windowsumfeld wird oft die COM-Variante gewählt: Es wird eine *type library* bereitgestellt, Anwender können etwa in VB eine eigene Oberfläche bauen, als Beispiel sei RSTAB/RFEM /27/ genannt.

Hier wird vorgeschlagen, auf XML-Basis jeweils komplette Anwendungsobjekte zur Verfügung zu stellen. Die Editieroberfläche wird verschickt, die Benutzungsoberfläche lokal aufgebaut. Anwendungslogik (*business logic*) und Modell liegen transparent auf (nicht notwendig zentralen) Servern, siehe folgende Kapitel. Zwischen Client und Server erfolgt lediglich ein XML-Verkehr. Zur Verarbeitung ist lediglich ein völlig neutrales Rahmenprogramm nötig.

3.4 Modelldatenstruktur

Branchenübergreifend haben einige Softwarehäuser bereits sehr früh Modellinhalte in XML dargestellt und serialisiert. So wurde in einem Produkt zur Installation von Software /28/ schon im Jahr 2000 der gesamte Installationsvorgang in XML abgebildet. (Dieser Vorgang repräsentiert für die Software das „Modell“, der Modellbearbeiter entwirft die Installation). In diesem Fall sind Modell- und Persistenzschicht identisch.

In der Bausoftware repräsentieren Produkt- und Prozessmodelle den Kern der Entwurfssysteme. Die entsprechenden Daten werden persistent gehalten. Typischerweise besteht hier eine Trennung von konzeptuellem Modell und Persistenzmodell. Historisch kann man folgende Triebkräfte beobachten:

- Auf konzeptueller Ebene werden proprietäre Eigenschaften implementiert, um im Wettbewerb der Systeme zu bestehen.
- Auf Persistenzebene werden Standards eingesetzt, um Qualität und Entwicklungsgeschwindigkeit zu erhöhen, sowie Nutzen aus existierender Basissoftware zu ziehen.
- Zugriff und Austausch von Daten werden auf einer semantisch möglichst hohen Ebene abgewickelt bzw. gekapselt.

Firmenich /29/ etwa verwendet eine spezielle Prädikatenlogik als konzeptuelle Ebene, die Speicherung der Daten selbst erfolgt z.B. in einer relationalen Datenbank.

Die IFC /30/ können einer Anwendung das Datenmodell per SDAI /31/ zur Verfügung stellen, die Daten selbst liegen u.U. ebenfalls in einer relationalen Datenbank.

ObjectARX-Anwendungen wie Hypersteel /22/ beinhalten ein unmittelbar verwendbares Objektmodell des Anwendungsbereiches, welches auf eine anwendungsneutrale objektorientierte Speicherungsstruktur abgebildet wird.

Eine mögliche Rolle von XML kann man also auf zwei Ebenen diskutieren, in der Anwendungssicht und als Speicherungsformat.

3.4.1 XML als Persistenzschicht

Neben vielen anderen Implementierungsmöglichkeiten kann XML heute „direkt“ gespeichert werden durch den Einsatz von XML-Datenbanken wie Tamino /32/ oder X-Hive /33/. Ein DOM wird hier unmittelbar persistent gehalten, es kann direkt darauf zugegriffen werden. Teilweise wird sogar ein Zugriff über eine Objektsicht per SOAP-*mapping* geboten, siehe Abschnitt 2.2 oben, i.d.R. in Form von *Web Services*.

3.4.2 XML als Modellschicht

Auf der konzeptuellen Ebene eines Entwurfssystems geht es darum, ein möglichst anwendungsnahes Modell der Wirklichkeit (eigentlich: einer ingenieurtechnischen Abstraktion der Wirklichkeit) zu bilden. Es wird zur Laufzeit populiert mit Instanzen und ist unmittelbar verknüpft mit einer Persistenzschicht. Die Gültigkeit der Modellinstanz (Abhängigkeiten, Werte, Bedingungen) wird von der *model engine* intern sichergestellt, nach außen werden Methoden zur Modifikation bereitgestellt. Die Anwendung einer Methode der *model engine* führt stets wieder auf ein gültiges Modell.

Eine Umsetzung mit XML-Mitteln ist momentan Gegenstand der Forschung, Ergebnisse sollen im Rahmen von /34/ publiziert werden.

3.5 Graphikschicht

Moderne CAD-Systeme benutzen unmittelbar 3D Graphiktreiber in der Visualisierungspipeline vom 3D Modell zur Bildschirmdarstellung. Solange es allerdings Zeichnungen gibt (und die gibt es im Bauwesen noch lange), wird auch eine 2D-Pipeline verwendet.

Wesentliche Bestandteile graphischer Modelle sind Primitive, allgemeine graphische Eigenschaften und Transformationen. Einheitlich werden Baumstrukturen aufgebaut. Entlang der Äste werden Eigenschaften vererbt, insbesondere Transformationen. Beispiele aus Forschung und Praxis sind Geo++ /36/ oder Autodesk’ HEIDI API (ursprünglich HOOPS von Ithaca Software) /39/. Auch *ray tracing* und *game engines* benutzen als Zwischenstruktur Bäume, z.B. den *Binary Space Partitioning* (BSP) *Tree* /35/.

Die Grundstruktur des Baumes kann offensichtlich unmittelbar auf XML abgebildet werden. Primitiva können dann nach den Regeln des *object mapping* entwickelt und an den Ästen eingesetzt werden. Es gibt bereits entsprechende XML Sprachen:

Im 2D-Bereich gestattet die Scalable Vector Graphics SVG /37/, /38/ die Repräsentation expliziter 2D Graphik. Der Standard beschreibt eine Menge einfacher graphischer Primitive, aus denen Zeichnungen zusammengesetzt werden können. Darüber hinaus gehende Features für die Animation erlauben die zeit- und/oder interaktionsbasierte Veränderung jeder Eigenschaft der Graphik, z.B. Ort, Farbe, Stil. Dies ist für Webpräsentationen und Dokumentationen interessant. Auch für klassische Entwurfsdokumente können diese Möglichkeiten eingesetzt werden, etwa zur Unterstützung des *redlining* in einer Mehrbenutzerumgebung.

Gegenwärtig wird eine XML-Sprache XNC für die feature-basierte Beschreibung von Stahlbauteilen entwickelt /1/. Repräsentiert werden z.B. Stahlbauträger inkl. Profilbeschreibung, Ausklinkungen, Bohrungen, Signierung, Zuordnung zu Projekt und Zeichnung u.v.a.
Bei entsprechender Implementierung kann der Anwender - über ein Webportal - aus unterschiedlichen CAD-Systemen Einzelinformationen in ein XNC Dokument extrahieren, Abb. 14.

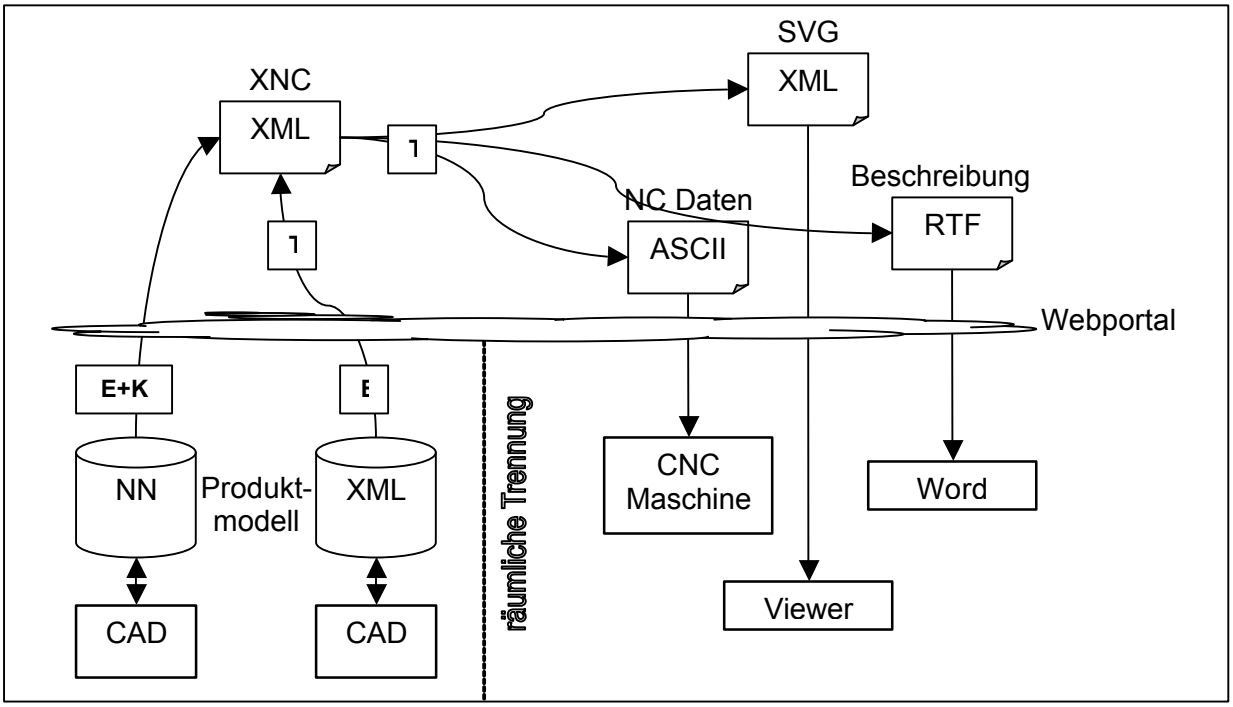

Abb.14 Bereitstellung von Fertigungsdaten aus CAD

Bei XML-basierten Modellen geschieht dies mittels Extraktion (E) und Transformation (T) auf reiner XSLT/XPath Basis. Für andere Modelle müssen Extraktion und Konvertierung (E+K) nativ implementiert werden.
Das Portal generiert per XSL Transformation (T) automatisch verschiedene, in der Arbeitsvorbereitung direkt nutzbare Formate. SVG-Dokumente enthalten Zeichnungsdaten und werden (direkt über Web/Intranet) von Viewern verwendet, Abb. 15. RTF Dateien werden als Word Dokumente gespeichert. NC-Daten werden von den Postprozessoren der CNC-Maschinen verarbeitet.

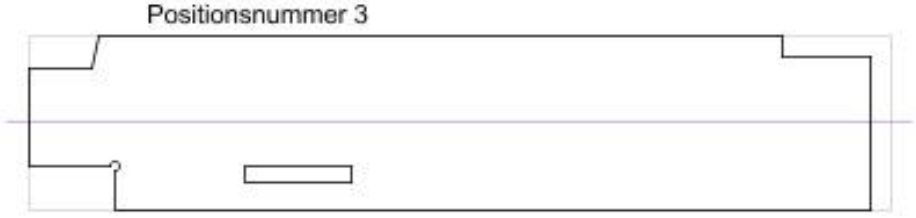

Abb.15 Graphik der Außenkontur, generiert

Die folgenden Abbildungen zeigen Ausschnitte aus einer XNC Datei, die entsprechenden Abschnitte in SVG bzw. NC (nativ).

```
...
<AI Ebene="v">
    <AKJob Bezug="o">
        <Vertex x="200" y="0" />
        <Vertex x="1952" y="0" />
        <FSegment>
            <Fase Seite="1" y="13.5" w="-18.43" />
        </FSegment>
        <Vertex x="1952" y="350" />
        <Vertex x="1750" y="350" />
        <Vertex x="1750" y="400" />
        <Vertex x="163.5" y="400" />
        <Vertex x="150" y="325" />
        <Vertex x="0" y="325" />
        <Vertex x="0" y="100" />
        <Vertex x="190" y="100" />
        <Segment>
            <Ausrundung r="-10" />
        </Segment>
        <Klinkung x="200" y="100" typ="w" />
        <Segment>
            <Ausrundung r="-10" />
        </Segment>
        <Vertex x="200" y="110" />
        <Segment>
            <Ausrundung r="-10" />
        </Segment>
        <Vertex x="200" y="90" />
        <Vertex x="200" y="0" />
    </AKJob>
    <IKJob Bezug="u">
        <Vertex x="500" y="60" />
        <Vertex x="500" y="100" />
        <Vertex x="750" y="100" />
        <Vertex x="750" y="60" />
        <Vertex x="500" y="60" />
    </IKJob>
</AI>
...
```

Abb.16 Ausschnitt XNC, generiert aus CAD

```
…
AK
  v   200.00o     0.00   0.00
  v  1952.00o     0.00   0.00 -
18.43  13.50
  v  1952.00o   350.00   0.00
  v  1750.00o   350.00   0.00
  v  1750.00o   400.00   0.00
  v   163.50o   400.00   0.00
  v   150.00o   325.00   0.00
  v     0.00o   325.00   0.00
  v     0.00o   100.00   0.00
  v   190.00o   100.00 -10.00
  v   200.00o   100.00w-10.00
  v   200.00o   110.00 -10.00
  v   200.00o    90.00   0.00
  v   200.00o     0.00   0.00
IK
  v   500.00u    60.00   0.00
  v   500.00u   100.00   0.00
  v   750.00u   100.00   0.00
  v   750.00u    60.00   0.00
  v   500.00u    60.00   0.00
…
```

Abb. 17 Ausschnitt NC, generiert aus XNC

```
<?xml version="1.0" encoding="UTF-8" ?>
<svg viewBox="0 0 2150 2150" width="100%" height="100%">
    <g title="this is a tooltip">
        <text y="70" x="345"
          style="font-size:60;font-family:TrebuchetMS-Bold;">Positionsnummer 3</text>
        <svg y="100" height="400" width="2150">
            <g stroke="blue">
                <line x1="20" y1="200" x2="2120" y2="200" stroke-width="1" />
            </g>
        </svg>
        <g transform="matrix(1,0,0,-1,70,500)" stroke="black">
            <polyline stroke="grey" stroke-width="1" fill="none"
                points="0,0 2000,0 2000,400 0,400 0,0" />
            <path stroke-width="4" fill="none" d="&#10; M200,0 &#10; L1952,0 &#10; L1952,350
&#10; L1750,350 &#10; L1750,400 &#10; L163.5,400 &#10; L150,325 &#10; L0,325 &#10;
L0,100 &#10; L190,100 &#10; &#10; &#10; A-10&#10; ,-10 0 0,0 200,110&#10; &#10; &#10; A-
10&#10; ,-10 0 0,0 200,90&#10; L200,0" />
            <path stroke-width="4" fill="none" d="&#10; M500,60 &#10; L500,100 &#10; L750,100
&#10; L750,60 &#10; L500,60" />
        </g>
    </g>
</svg>
```

Abb.18 Ausschnitt SVG, generiert aus XNC

3.6 Synergien

Die bisher erläuterten Möglichkeiten müssen im Zusammenhang betrachtet werden.

Für ein Softwarehaus ergeben sich große Synergie-Effekte. Auf der einen Seite kann eine ständig wachsende Zahl von Toolkits verwendet werden, um XML-Dokumente zu verarbeiten. Andererseits können Basisaufgaben aus verschiedenen Bereichen immer wieder ähnlich implementiert werden. Auch wenn Meta- und Meta-Meta-Ebenen eingeführt werden, kann XML zu Testzwecken immer noch gelesen werden, man kann sehr leicht automatische *logs* und *reports* implementieren. Viele Entscheidungen wurden in der Vergangenheit aufgrund vorhandener Hard- und Software gefällt. Die Portabilität von XML schafft hier neue Spielräume.

Die in diesem Papier genannten Beispiele beziehen sich vorrangig auf den technischen Bereich der Planung. Im administrativen Bereich ist XML in Form von aecXML bzw. GAEB2000/XML /40/ bereits längere Zeit auf dem „Vormarsch“. Eine gemeinsame „technische Basis“ kann hier ebenfalls nur Vorteile bringen.

Der Personenkreis, welcher Bauplanung koordiniert, wird seit Jahren mit immer neuen - und in den einzelnen Bereichen immer anderen - Technologien konfrontiert, die Planungssysteme verknüpfen sollen. XML bietet durch seine Verbreitung, seine Einfachheit und Offenheit hier eine besondere Chance zur Verbesserung der Situation.

4. Zusammenfassung und Ausblick

Es wurde gezeigt, warum und wie XML in verschiedenen Bereichen von Planungssystemen eingesetzt werden kann mit dem Ziel, diese Systeme modular und webfähig zu gestalten. Es wurde gezeigt, welche Anforderungen heute schon erfüllt werden können: Es ist möglich, komplexe Objekte aus dem Planungsalltag mittels XML darzustellen, zu speichern und zu verarbeiten. Es ist möglich, entsprechende Komponenten im Netz zu verteilen bzw. über Internet zu verbinden.

Die heute dominierende Sicht auf XML als Austauschmedium wird ergänzt um die Idee eines XML-basierten Systems: Entwurfsobjekte können als „XML-Objekte“ formuliert und im Sinne eines *late binding* verwendet werden. Die Arbeiten werden fortgeführt mit dem Ziel, ein derartiges Referenzsystem zu schaffen.

Quellen

/1/ adhoc Arbeitskreis „XML-Einsatz bei Schnittstellen“
Deutscher Stahlbauverband DSTV, Düsseldorf 2003

/2/ aecXML
http://www.iai-na.org/aecxml/mission.php

/3/ DesignXML
http://www.designxml.org

/4/ The Importance of DesignXML
Autodesk White Paper, 2003

/5/ The XML revolution
Autodesk White Paper, 2001

/6/ D. Brors “Totale Kommunikation”
c't 6, 2003

/7/ Co-Inventor of XML Says Office 11 is "A Huge Step Forward for Microsoft"
http://www.sys-con.com/xml/articlenews.cfm?id=513

/8/ J. Udell “XML for the rest of us”
http://www.infoworld.com, November 2002

/9/ http://www.openoffice.org

/10/ Web Services Activity
http://www.w3.org/2002/ws

/11/ W. Haas, D.Ilieva, K.Kessoudis
“Erfahrungen beim Einsatz Web-basierender Planmanagementsysteme im Planungsalltag“
VDI-Berichte 1668, Düsseldorf 2002

/12/ D. Box, A. Skonnard, J. Lam
“Essential XML. Beyond Markup“
Addison Wesley, 2000

/13/ http://www.objectivity.com

/14/ http://www.fastobjects.com/de

/15/ M. Hansch, S. Kuhlins, M. Schader „XML Schema“
Informatik-Spektrum, Oktober 2002

/16/ OMGXMLMetadata Interchange (XMI)Specification, Version 1.2
http://www.omg.org, January 2002

/17/ D. Carlson “Modeling XML Applications"
Software Development Magazine , May 2002

/18/ Meta Object Facility (MOF) Specification, Version 1.3
http://www.omg.org. March 2000

/19/ Web Services Description Language (WSDL) Version 1.2
http://www.w3.org/TR/wsdl12 , March 2003

/20/ Apache Project AXIS
http://ws.apache.org/axis

/21/ Produktlinie bocad-3D
http://www.bocad.de/home/de

/22/ Produktlinie HyperSteel
http://www.dsc.de

/23/ M.Huhn, H.-D.Koch, M. Steinrötter
“Projekt bocat: Bocad Catalogs“
Internes Papier, Bocad Software, 2003

/24/ Application framework Qt
http://www.trolltech.com

/25/ JEasy - a framework for JAVA applications using XML
http://www.jeasy.de

/26/ M. Huhn “Offene Systeme in Internet ?!
Ein Komponentenmodell für das Bauwesen”
VDI-Berichte 1668, Düsseldorf 2002

/27/ RFEM
http://www.dlubal.de

/28/ InstallShield MultiPlatform
http://www.installshield.com

/29/ B. Firmenich „CAD im Bauplanungsprozess:
Verteilte Bearbeitung einer strukturierten Menge von Objektversionen“
Dissertation, Bauhaus-Universität Weimar 2001

/30/ Industry Foundation Classes, IAI
http://www.iai-ev.de

/31/ Standard Data Access Interface SDAI
STEP ISO 10303, Part 22-26

/32/ Tamino XML Server
http://www.softwareag.com/tamino

/33/ X-Hive/DB
http://www.x-hive.com

/34/ Grundlagen vernetzt-kooperative Planungsprozesse im Komplettbau, 3. Stufe
Projekt im DFG-Schrwerpunktprogramm 1103
http://www.iib.bauing.tu-darmstadt.de/dfg-spp1103/de

/35/ Vlastimil Havran, Tomáš Kopal, Jirí Bittner, Jirí Žára
“Fast Robust BSP Tree Traversal Algorithm for Ray Tracing”
Journal of Graphics Tools, No. 4 / 1998

/36/ P. Wisskirchen
Object-Oriented Graphics. From GKS and PHIGS to Object-Oriented Systems
Springer-Verlag 1990

/37/ A. Neumann, A. M. Winter, P. Sykora
„Das Web auf neuen Pfaden. SVG und SMIL: Grafik und Animation fürs Web“
c't 20, 2002

/38/ Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation 2003
http://www.w3.org/TR/SVG11

/39/ HOOPS Graphics Engine
http://www.hoops3d.com/about/company/company.htm

/40/ GAEB-XML
http://www.gaeb-2000.de