DOI: 10.20378/irb-59285

ISSN 0937-3349
ISBN

Herausgeber

Prof. Dr. Walter Augsburger
Lehrstuhl für Wirtschaftsinformatik,
insbes. Büro- und Verwaltungsautomation

Prof. Dr. Elmar J. Sinz
Lehrstuhl für Wirtschaftsinformatik,
insbes. Systementwicklung und Datenbankanwendung

Prof. Dr. N. N.
Lehrstuhl für Wirtschaftsinformatik,
insbes. Industrielle Anwendungssysteme

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG, Postfach 1549, 8600 Bamberg

BAMBERGER
BEITRÄGE ZUR WIRTSCHAFTSINFORMATIK

ISSN 0937-3349

Nr. 5/1991

Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU

erscheint in: Workshop Information Retrieval,
Darmstadt 24.–25. Juni 1991

Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten durch den Endanwender

erscheint in: 36. Internationales Wissenschaftliches Kolloquium,
Ilmenau 21.–24.Okt. 1991

Walter Augsburger, Helge Rieder, Jürgen Schwab

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG

Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU

W. Augsburger, H. K. Rieder, J. Schwab
Universität Bamberg, Wirtschaftsinformatik

Zusammenfassung

Derzeit verfügbare statistische Retrievalsysteme haben eine Architektur nach dem Paradigma der traditionellen Programmierung: der Trennung von Programmen und Daten. Diese Systemarchitektur stößt bei den Anforderungen an ein intelligentes Informationssystem an ihre systemimmanenten Grenzen. Um diese zu überwinden, wird eine neue Systemarchitektur, basierend auf einem objektorientierten Datenmodell, vorgeschlagen. Diese hebt die klassische Trennung zwischen numerischen Daten und Programmen auf. Die Darstellung des Modellansatzes erfolgt dabei am Beispiel der Zeitreihenanalyse.

1 Motivation

Ausgangspunkt der nachfolgenden Überlegungen war die Konzeption eines Informationssystems für Führungskräfte in Wirtschaft und Verwaltung auf der Grundlage numerischer Daten[1]. Dabei zeigte sich, daß die nach dem Paradigma traditioneller Programmierung aufgebauten numerischen Informations- und Retrievalsysteme bei Bildung komplexer Modelle, bei der Klassifikation numerischer Daten nach inhaltlichen Gesichtspunkten und bei kontinuierlicher Überwachung komplexer Datenbestände ihre Grenzen finden. Diese Grenzen lassen sich nicht durch Kosmetik an bestehenden Systemen überwinden, sondern fordern einen neuen Modellansatz. Nach einer kurzen Betrachtung der systemimmanenten Schwächen traditioneller Systemarchitekturen wird als Lösungsmöglichkeit ein objektorientierter Ansatz vorgeschlagen. Durch Ausnutzung der Möglichkeiten eines objektorientierten Konzepts können neue Objekte ohne eine Änderung von Programmen generiert werden. Die Vererbung von Attributen und Werten auf Klassen- wie auf Objektebene eignet sich besonders zur Darstellung einer Halbordnung über einer Objektmenge. Diese Struktur findet sich oft in einem betriebswirtschaftlichen Kennzahlensystem über Einzelkennzahlen.

[1]Das Projekt EISREVU (**Entscheidungsunterstützendes Informationssystem für Energieversorgungsunternehmen**) wird finanziell und durch fachliche Zusammenarbeit von den Unternehmen Mainkraftwerke AG Frankfurt Höchst (MKW) und Energieversorgung Oberfranken AG (EVO) gefördert. Es wurde eine in [Augs 90] beschriebene Pilotversion implementiert.

2 Statistische Analyse- und Retrievalsysteme

... sind in ihrer Grundkonzeption noch dem Batch-Zeitalter verhaftet. Wie in Abb. 1 dargestellt, unterstützen Statistikprogramme eine einmalige Analyse des zuvor eingegebenen Datenbestands. Sie unterstützen verschiedene statistische Verfahren und eine graphische Aufbereitung. Die Ausgabe von Texten

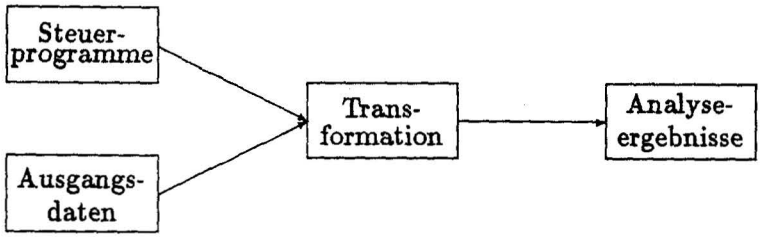

Abb. 1: Arbeitsweise eines Statistikpaketes

ist (wie beispielsweise bei SPSS) möglich, doch muß der Text auf Programmiersprachenlevel codiert werden. Der interaktive Modus beschränkt sich auf das schrittweise Absetzen von Kommandos[2]. Statistische Retrievalsysteme[3] ergänzen das Statistikpaket durch eine integrierte Datenbank. Diese enthält jedoch nur die nackten numerischen Daten und ist somit lediglich ein Ersatz für den Batch-File.

Statistikpakete und Auswertungen auf statistischen Datenbanken sind darauf angewiesen, die benötigten Daten beim Programmaufruf in einem Batch-File[4] oder in der systemeigenen Datenbank[5] vorzufinden. Kopplungen zwischen Statistiksystemen und Wissensbasierten Systemen haben i. d. R. die Zielsetzung, Statistiksysteme für den Anwender besser zugänglich und begreifbar zu machen[6]. Die Aufgabenstellung eines intelligenten Zeitreihenverwaltungssystems erfordert jedoch genau das Gegenteil; nämlich die Nutzbarmachung statistischer Teilkomponenten in einem Wissensbasierten System.

Zeitreihenverarbeitende Statistikpakete und Zeitreihendatenbanken unterstützen eine strikte Trennung von Daten und Programmen. Auch Systeme mit einer integrierten Datenbank sind aufgrund ihrer prinzipiellen Systemarchitektur nur für Schnappschußanalysen einsetzbar. Für die Überwachung sich laufend ändernder Datenbestände und für ein Retrieval nach inhaltlichen Gesichtspunkten sind sie eher ungeeignet. Eine derartige Aufgabenstellung erfordert vielmehr einen Ansatz, der die numerischen Daten zusammen mit ihren Attributen und Methoden in einer Objektstruktur integriert.

---

[2] Vgl. [SPSS 88, S. 155ff.].
[3] Vgl. hierzu [Stau 85].
[4] Das war früher der Kartenstapel mit den Kommandos
[5] Das war früher der Kartenstapel mit den Daten.
[6] Vgl. hierzu [Gale 85].

3 Die Idee der Zeitreihenobjekte

Kern des vorgestellten Modellansatzes zur Lösung der o.g. Problemstellung sind sogenannte Zeitreihenobjekte. Diese stellen einen neuen abstrakten Modellansatz zur Darstellung von zeitabhängigen numerischen Daten dar. Seine Grundbausteine sind sogenannte Zeitreihenobjekte. Die Idee der Zeitreihenobjekte besteht darin, als Zeitreihen dargestellte Kennzahlen zusammen mit allen zu einer Zeitreihe gehörenden Informationen, d.h. auch Methoden, die auf die Zeitreihe anwendbar sind, in einem Zeitreihenobjekt zu kapseln und als strukturiertes Zeitreihenobjekt abzulegen. Damit ist die Trennung von Daten und Programmen aufgehoben.

Die Verwendung eines objektorientierten Ansatzes als Grundlage eines Systems zur Zeitreihenanalyse war bzgl. der EISREVU-Projekthistorie nicht etwa der Ausgangspunkt des wissenschaftlichen Ansatzes, sondern wurde im Laufe des Projekts als geeignetes Instrument zur Modellierung der vorliegenden Aufgabenstellung "entdeckt". Die Definition der Objekte wurde (auch) deshalb in keiner eingeführten Objektbeschreibungssprache durchgeführt. Sie leitet sich vielmehr aus den Anforderungen an die Ausdrucksfähigkeit einer derartigen Beschreibung ab und ist bewußt idealtypisch, d.h. es wird keine Rücksicht auf eine eventuelle Implementierbarkeit genommen.

Es erscheint berechtigt, zu fragen, inwieweit das im folgenden dargestellte Modell wirklich dem Paradigma objektorientierter Systeme genügt, oder ob es nur des besseren Images wegen "objektorientiert" getauft wurde. Hier stellt sich jedoch das Problem, daß in der Literatur der Begriff "objektorientiert" nicht immer ganz einheitlich verwendet wird[7]. Erschwerend kommt hinzu, daß es für den Autor nicht nachzuvollziehen war, von wem und in welchem Zusammenhang der Begriff "objektorientiert" geprägt wurde. Da aufgrund der hier angedeuteten Begriffsverwirrung eine Überprüfung des eigenen Ansatzes in bezug auf die Erfüllung der Kriterien für "objektorientiert" mangels einer sauberen, allgemeinverbindlichen Definition leider unmöglich ist, wird "objektorientiert" in Anlehnung an Kempel/Pfander[8], Kreutzer[9] und Wegner[10] folgendermaßen definiert:

Objektorientierung = gekapselte Objekte + Klassen + Vererbungsmechanismen

Der gewählte Ansatz von Zeitreihenobjekten wird als objektorientiert betrachtet, weil er die folgenden Eigenschaften besitzt:

- Realisierung eines Klassen[11]/Instanzen-Konzepts.

[7] Vgl. in diesem Zusammenhang Stoyans Vergleich zwischen den Charakteristiken objektorientierter Programmiersprachen nach den Kriterien von Wegner [Wegn 87] und eigenen Kriterien in [Stoy 91, S. 192ff.].

[8] Vgl. [Kemp 90, S. 12].

[9] Vgl. hierzu [Kreu 90, S. 213].

[10] Vgl. [Wegn 87] und [Wegn 89].

[11] Im folgenden wird der Begriff Klasse verwendet, wenn es sich um einen leeren Frame handelt, der Begriff Typ bleibt für Datentypen reserviert.

- Kapselung: Zeitreihen bilden zusammen mit allen zu einer Zeitreihe gehörigen Informationen[12] eine Einheit, die in einem Objekt gekapselt ist[13]. Die einzelnen Objekte können nur mit Operationen bearbeitet werden, die in dem Objekt selbst definiert sind.

- Attribute (bzw. Sloteinträge) sind keine untrennbaren Einheiten. Dadurch ergibt sich definitionsgemäß ein NF^2-Modell[14].

- Vererbung: Es existiert eine Vererbung auf Klassenebene und ein (im Vergleich mit dem originären objektorientierten Ansatz) stark modifiziertes Konzept zur Vererbung von Sloteinträgen auf Instanzebene.

Ein rudimentäres Nachrichtenkonzept[15] ist, z.B. zur Kommunikation mit der Außenwelt, ebenfalls vorhanden, wird im folgenden aber nicht weiter dargestellt.

4 Der Aufbau von Zeitreihenobjekten

Im folgenden wird zwischen Klassen und Instanzen von Zeitreihenobjekten unterschieden. Ist von einem Zeitreihenobjekt die Rede, so ist die Instanz gemeint. Klassen werden als ZRO-Klassen bezeichnet. Eine Instanzierung ist von solchen ZRO-Klassen möglich, die *OPERATION create_instance()* besitzen. Dabei wird eine Instanz mit allen in der Objektdefinition definierten Slots, allen ererbten Slots[16] und allen Subobjekten[17] erzeugt. Andere ZRO-Klassen sind entweder Basistypen, die an andere instanzierbare ZRO-Klassen vererbt werden oder Teilobjekte von instanziierbaren ZRO-Klassen, die bei einer Instanzierung ihre Teilobjekte mitgenerieren. Die folgende Definition von Zeitreihenobjekten ist idealtypisch. Es wird keine Rücksicht auf die Ausdrucksfähigkeit von gängigen Implementierungs- und/oder Spezifikationssprachen[18] genommen.

Die Modellierung erfolgt auf der Grundlage eines hybriden Ansatzes mit Elementen aus frameorientierten—objektorientierten Ansätzen[19] und deklarativ-regelorientierten Ansätzen[20]. Jede Objektbeschreibung besteht aus einem OBJ_Header und der Slotliste. Der

[12] D.h. Attributen und Methoden.

[13] Vgl. hierzu die Betrachtung von Objekten als Datenkapseln in [Flei 91, S. 26 (1-6)].

[14] Zur Definition von NF^2-Modellen s. z.B. [Pare 89, S. 177ff.].

[15] Vgl. hierzu als kurze Einführung in ein derartiges Konzept [Rich 89, S. 139ff.].

[16] Da es sich hierbei um eine Instanzierung von einer Klassendefinition handelt, sind auch diese slots nach der Erzeugung der Instanz zunächst leer.

[17] Um ein Sub-Objekt als Instanz erzeugen zu können, ist in der Klassendefinition des Subobjekts keine *OPERATION create_instance()* erforderlich.

[18] Einen Überblick über derartige Sprachen bieten Stoyan ([Stoy 88], [Stoy 91]) und Hawley ([Hawl 87]).

[19] Verwendung eines Klassen-Instanzen-Konzepts, einer Modelldefinition mit Klassen, von Slots und von Demons.

[20] Zur Definition der Berechnungsvorschrift von berechnteten Zeitreihenobjekten und von Demons.

OBJ_Header besteht aus dem Schlüsselwort *"OBJECT"* und dem Objektnamen in Form eines Strings. Eine Slotliste ist folgendermaßen aufgebaut:

slot
: *slottyp, slotname, slotbeschreibung*
| *slot, slot*

slotname ist ein innerhalb des Objekts eindeutiger Bezeichner in String-Konventionen[21]. Slots werden über ihren Slotnamen angesprochen. *slottyp* ist genau ein Bezeichner aus einer definierten Menge möglicher Bezeichner für Slottypen. In jedem Objekt muß genau ein Slot des *slottyps FATHER_OBJ* vorhanden sein. Alle anderen Slottypen können beliebig oft auftreten. Die Syntax der *slotbeschreibung* steht in Abhängigkeit vom jeweiligen Slotnamen. In Abhängigkeit vom *slotnamen* hat die *slotbeschreibung* folgende Funktion:

slottyp	*slotbeschreibung*
ARGUMENT	Argument (String, Integer oder Real) in 1. Normalform oder eine Menge derartiger Argumente
L_ARGUMENT	Wohlgeformter Ausdruck einer formalen Sprache – z.B. der Sprache zur Beschreibung von Zeitreihenobjekten oder der Sprache zur Beschreibung von Demons.
V_ARGUMENT	Definition eines Arguments in 1. Normalform, dessen Inhalt dynamisch durch eine Funktion erzeugt wird. Die Ausprägung folgt dabei dem Schema: *datentyp* "←" *funktion*
P_OBJ	Definition eines Teilobjekts, das bei einer Instanzierung miterzeugt wird. Als Ausprägungen sind die Namen von Objektklassen zugelassen.
FATHER_OBJ	Definition Objektklasse, deren Slots geerbt werden sollen. Als Ausprägungen sind die Namen von Objektklassen zugelassen.

[21] Vgl. hierzu die Definition von Strings in Programmiersprachen.

CONSTRAINT Formulierung einer semantischen Integritätsbedingung über die Ausprägungen der Slotbeschreibungen des jeweiligen Objekts. Sie enthält eine prädikatenlogische Formel zur Überprüfung der semantischen Integrität von Objekten dieser Objektklasse, die bei allen Kombinationen zulässiger Ausprägungen aller Variablen stets den Wert *true* zu haben hat.

OPERATION Definition einer zulässigen Operation auf den Ausprägungen der Slotbeschreibung eines Objekts. Die Ausprägung enthält einen Funktionenamen, ggf. mit Parameter(n). Operationen können auch durch eine Bedingung konditioniert sein. Dabei bedeutet:
$B \Rightarrow op(V) \quad :\Leftrightarrow \quad op$ kann auf V nur ausgeführt werden, wenn die Bedingung B erfüllt ist.

Zur Symbolik in den Slotbeschreibungen:

Im folgenden werden nur solche Sprachkonstrukte erklärt, deren Semantik nicht selbstverständlich ist.

a) Sprachkonstrukte zur Definition von *ARGUMENT*-Slotbeschreibungen:

$< zro - klasse >$	Der Name einer ZRO-Klasse (erzeugt bei der Instanzierung ein Sub-Zeitreihenobjekt)
$\{el_1, \ldots, el_n\}$	Auswahl: ein Element aus der definierten Menge von Elementen gleichen Datentyps
SET OF <zro-klasse>	Eine Instanz oder mehrere Instanzen der *zro-klasse*
SET OF $\{el_1, \ldots, el_n\}$	Ein Element oder mehrere Elemente der definierten Menge von Elementen gleichen Datentyps
ARRAY OF <datentyp>	Eine Reihung aus Elementen von *datentyp*

b) Sprachkonstrukte zur Definition von *V_ARGUMENT*-Slotbeschreibungen:

$< datentyp > \leftarrow func()$	Die Funktion *func()* errechnet einen Sloteintrag vom Typ *datentyp*. Die Ausprägung des Slotinhalts errechnet sich dabei durch die Funktion *func*()

c) Sprachelemente zur Konstruktion von *CONSTRAINT*-Slotbeschreibungen:

<*zro*> *INSTANCE_OF* <*zro-klasse*>	Logischer Ausdruck: liefert den Wahrheitswert true, wenn <*zro*> eine Instanz von $< zro - klasse >$ ist
$< date > AFTER < date >$	Logischer Ausdruck: liefert den Wahrheitswert true, wenn das erste Zeitreihenobjekt vom Typ $< date >$ das neuere ist
$\Rightarrow$	Bezeichnet eine Implikation
XOR	Logischer Junktor zur Darstellung des exklusiven OR
<*slotname*> $\bowtie$ <*zro-klasse*>, <*datentyp*>	Dieses Konstrukt ist anwendbar, wenn im Slot <*slotname*> verschiedene <*zro-klasse*>-en und/oder $< datentyp >$-en als Einträge zulässig sind. Der logische Ausdruck nimmt den Wahrheitswert *true* an, wenn die aktuelle Ausprägung von $< slotname >$ vom Typ $< zro - klasse >$ oder $< datentyp >$ ist

5 Elementare Zeitreihenobjekte

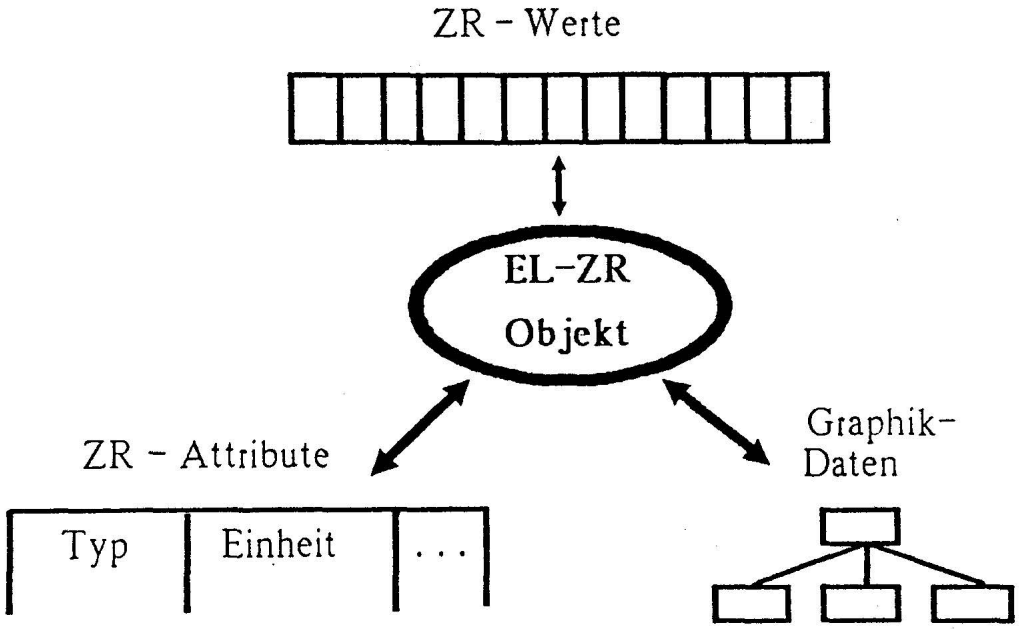

Abb. 2.: Aufbau eines elementaren Zeitreihenobjekts

Zeitreihenobjekte gliedern sich in elementare und höhere Zeitreihenobjekte. Elementare Zeitreihenobjekte wurden zur Repräsentation von Basiswerten, die von außen eingespeist werden, konzipiert. Die Klassenbeschreibung zur Definition von elementaren Zeitreihenobjekten besitzt, wie in Abb. 2. dargestellt, eine innere Struktur mit identifizierenden Namen und einer Zeitreihe mit Zusatzangaben (Einheit, Periodizität etc.). Bei der Konstruktion eines konkreten Modells wird dabei jeder von außen in das System eingeführten Zeitreihe einer betriebswirtschaftlichen Größe ein elementares Zeitreihenobjekt zugeordnet.

Das Objekt *zeitreihen_object* ist die gemeinsame Grundlage für elementare und höhere ZRO-Klassen. Seine Slots werden an beide vererbt. Die Ausprägung im Slot *name* ist der modellweit eindeutige Identifikator für das Zeitreihenobjekt. *level* ist eine natürliche Zahl. Für elementare Zeitreihenobjekte ist *level* stets 0[22]. *graphic_data* beinhaltet ein Teilobjekt. *update* merkt sich das Datum des letzten Updates der *values* bei jedem schreibenden Zugriff auf das Objekt mit der Funktion *systemdate*. Die Operationen *op1 und op2* definieren lesbare und beschreibbare Bereiche. *read()/write()*-Operationen auf

[22]Vgl. hierzu den *CONSTRAINT* in der Definition von *el_zeitreihen_object*.

nachgeordnete Objekte sind so definiert, daß diese Operationen auf allen Slots mit einer *ARGUMENT*-Definition möglich sind, ohne daß bei den nachgeordneten Objekten eine explizite *OPERATION*-Definition erforderlich ist.

OBJECT all_zr_objects:

FATHER_OBJ:		*OBJECT all_objects*

OBJECT zeitreihen_object:

FATHER_OBJ:		*OBJECT all_zr_objects*
ARGUMENT	*name*	*STRING*
ARGUMENT	*level*	*INTEGER*
P_OBJ	*graphic_data*	*OBJECT graphik-daten*
P_OBJ	*update*	*OBJECT date* ← *systemdate()*
OPERATION:	*op1*	*read({name, level, graphic_data})*
OPERATION:	*op2*	*write({name, graphic_data})*

OBJECT el_zeitreihen_object:

FATHER_OBJ:		*OBJECT zeitreihen_object*
P_OBJ	*attributes*	*OBJECT zr_attribute*
P_OBJ	*values*	*OBJECT zr_werte*
CONSTRAINT:	*c1*	*el_zeitreihen_object.level* == 0
OPERATION:	*o1*	*create_instance()*
OPERATION:	*o2*	*remove_instance()*
OPERATION:	*o3*	*read({attributes, values})*
OPERATION:	*o4*	*write({attributes, values})*

Das Objekt *el_zeitreihen_object* erbt alle Slots von *zeitreihen_object*. Des weiteren hat es die im folgenden noch erläuterten Teilattribute *attributes* und *values*. *el_zeireihen_object* enthält die *OPERATION o1* "create_instance()", da eine Instanz dieser ZRO-Klasse erzeugt werden kann[23]. Weitere Operationen erlauben das Lesen und Schreiben auf *attributes* und *values*.

[23]Im Gegensatz dazu besitzt beispielsweise das Objekt *zeitreihen_object* keine Operation *create_instance()*, da es nicht sinnvoll ist, Objekte dieser Klasse zu erzeugen. Es können jedoch Instanzen von Objekten erzeugt werden, deren Vater-Klasse *zeitreihen_object* ist. Ebenso besitzt das Objekt *zr_werte* keine Operation *create_instance()*, da keine eigenständigen Objekte dieser Klasse erzeugt werden können. Ein Objekt *zr_werte* wird z.B. dann erzeugt, wenn ein Objekt *el_zeitreihen_object* erzeugt wird, als dessen Teilobjekt *zr_werte* definiert ist.

OBJECT zr_attribute:		
FATHER_OBJ:		*OBJECT all_zr_objects*
ARGUMENT	*type:*	*{value, relation, index}*
ARGUMENT	*d_unit:*	*OBJECT date*
P_OBJ	*index_date:*	*OBJECT date*
CONSTRAINT:	*c1*	*d_unit == NULL XOR index_date == NULL*
CONSTRAINT:	*c2*	*type == index ⇒ d_unit == NULL*
CONSTRAINT:	*c3*	*type <> index ⇒ str_unit == NULL*

Das Objekt *zr_attribute* enthält den Typ der Zeitreihe (d.h Werte, Index oder Verhältnis) und die Einheit der Zeitreihe. Ist die Zeitreihe vom Typ *index*, so hat diese keine Einheit. In diesem – und nur diesem – Fall wird unter *index_date* das Datum, auf das sich der Index bezieht, festgelegt (vgl. hierzu die *CONSTRAINT*s c1, c2, c3).

Das Objekt *zr_werte* beinhaltet die Werte des Zeitreihenobjekts als *ARRAY* von Zahlen. Bei Nichtvorliegen des Wertes ist auch der *NULL*-Wert zugelassen. Weiterhin enthält das Objekt *zr_werte* das erste und das letzte Datum, für das ein Wert existiert. Das *COSTRAINT c1* überprüft, ob die notwendige Anzahl von Werten zwischen dem ersten und dem letzten Datum mit der tatsächlich vorhandenen Anzahl an Werten übereinstimmt.

OBJECT zr_werte:		
FATHER_OBJ:		OBJECT all_zr_objects
ARGUMENT	values_list:	ARRAY OF {INTEGER,REAL,NULL}
P_OBJ	first_date:	OBJECT per_date
P_OBJ	last_date:	OBJECT per_date
CONSTRAINT:	c1	distance(first_date, last_date) == number_of(values)

Das Objekt *per_date* dient (einschl. seiner von *date* geerbten Slots) zur Darstellung von Datumsangaben. Eine Datumsangabe besteht aus 1-3 Stellen und kann nur in Abhängigkeit von der Periodizität semantisch interpretiert werden. Hat diese die Ausprägung *year*, so ist sie einstellig. Die andereren Stellen sind in diesem Fall mit NULL-Werten zu belegen[24] . Bei den Periodizitäten *halfyear*, *quarteryear* und *month* sind zwei Stellen mit Werten belegt. Bei der Periodizität *day* sind alle Stellen mit Werten belegt. Je nach Periodizität hat der Wert des Attributs *n_idf* somit verschiedene Grundmengen als Wertebereiche. Diese werden in den *CONSTRAINT*s *c1, c2, c3* definiert. Die *CONSTRAINT*s *c4, c5, c6, c7, c8* definieren den Wertebereich von *day* in Abhängigkeit von *n_idf* (als Feld für den Monatswert) und *year*[25]. Die *CONSTRAINT*s von per_date sind somit Integritätsbedingungen für wohlgeformte Datumsangaben.

[24] Siehe erstes *CONSTRAINT* von *per_date*.

[25] Anzahl der Tage einschließlich Berücksichtigung der Schaltjahrprobleme.

OBJECT date:

FATHER_OBJ:		*OBJECT all_zr_objects*
ARGUMENT	*year:*	*1500 .. 2200*
ARGUMENT	*n_idf:*	*{1 ... 12, NULL}*
ARGUMENT	*day:*	*{1 ... 31, NULL}*

Im OBJECT *graphik_daten* können die Koordinatenwerte zur graphischen Visualisierung von Systemen aus Zeitreihenobjekten abgelegt werden. Es wäre auch denkbar, das Objekt durch zusätzliche Slots (wie z.B. "Hintergrundfarbe") über die Koordinatenangaben hinaus zu erweitern.

OBJECT graphik_daten

FATHER_OBJ:		*OBJECT all_zr_objects*
ARGUMENT	*x*	*INTEGER*
ARGUMENT	*y*	*INTEGER*

OBJECT per_date:

FATHER_OBJ:		*OBJECT date*
ARGUMENT	*period:*	$\{year, quarteryear, halfyear, month, day\}$
CONSTRAINT:	*c1*	$per\_date.period == year \Rightarrow$ $per\_date.n\_idf == NULL \wedge$ $per\_date.day == NULL$
CONSTRAINT:	*c2*	$per\_date.period == quarteryear \Rightarrow$ $per\_date.n\_idf == \{1,2,3,4\} \wedge$ $per\_date.day == NULL$
CONSTRAINT:	*c3*	$per\_date.period == halfyear \Rightarrow$ $per\_date.n\_idf == \{1,2\} \wedge$ $per\_date.day == NULL$
CONSTRAINT:	*c4*	$per\_date.period == month \Rightarrow$ $per\_date.n\_idf == \{1,2, ..,12\} \wedge$ $per\_date.day == NULL$
CONSTRAINT:	*c5*	$per\_date.period == day \Rightarrow$ $per\_date.n\_idf == \{1,2, ..,12\} \wedge$ $per\_date.day == \{1,2, .. ,31\}$
CONSTRAINT:	*c6*	$per\_date.period == day \wedge$ $per\_date.n\_idf == \{4,6,9,11\} \Rightarrow$ $per\_date.day == \{1,2, .. ,30\}$
CONSTRAINT:	*c7*	$per\_date.period == day \wedge$ $per\_date.n\_idf == \{2\} \Rightarrow$ $per\_date.day == \{1,2, .. ,29\}$
CONSTRAINT:	*c8*	$per\_date.period == day \wedge$ $per\_date.n\_idf == \{2\} \wedge$ $(per\_date.year \bmod 1000 <> 0 \wedge$ $(per\_date.year \bmod 4 <> 0 \vee$ $per\_date.year \bmod 100 == 0$ $)) \Rightarrow per\_date.day == \{1,2, .. ,28\}$

6 Höhere Zeitreihenobjekte

Inhaltliche Aufgabe des Konzepts höherer Zeitreihenobjekte ist die Verdichtung der Vielzahl von in elementaren Zeitreihenobjekten repräsentierten Basisdaten auf Maßzahlen.

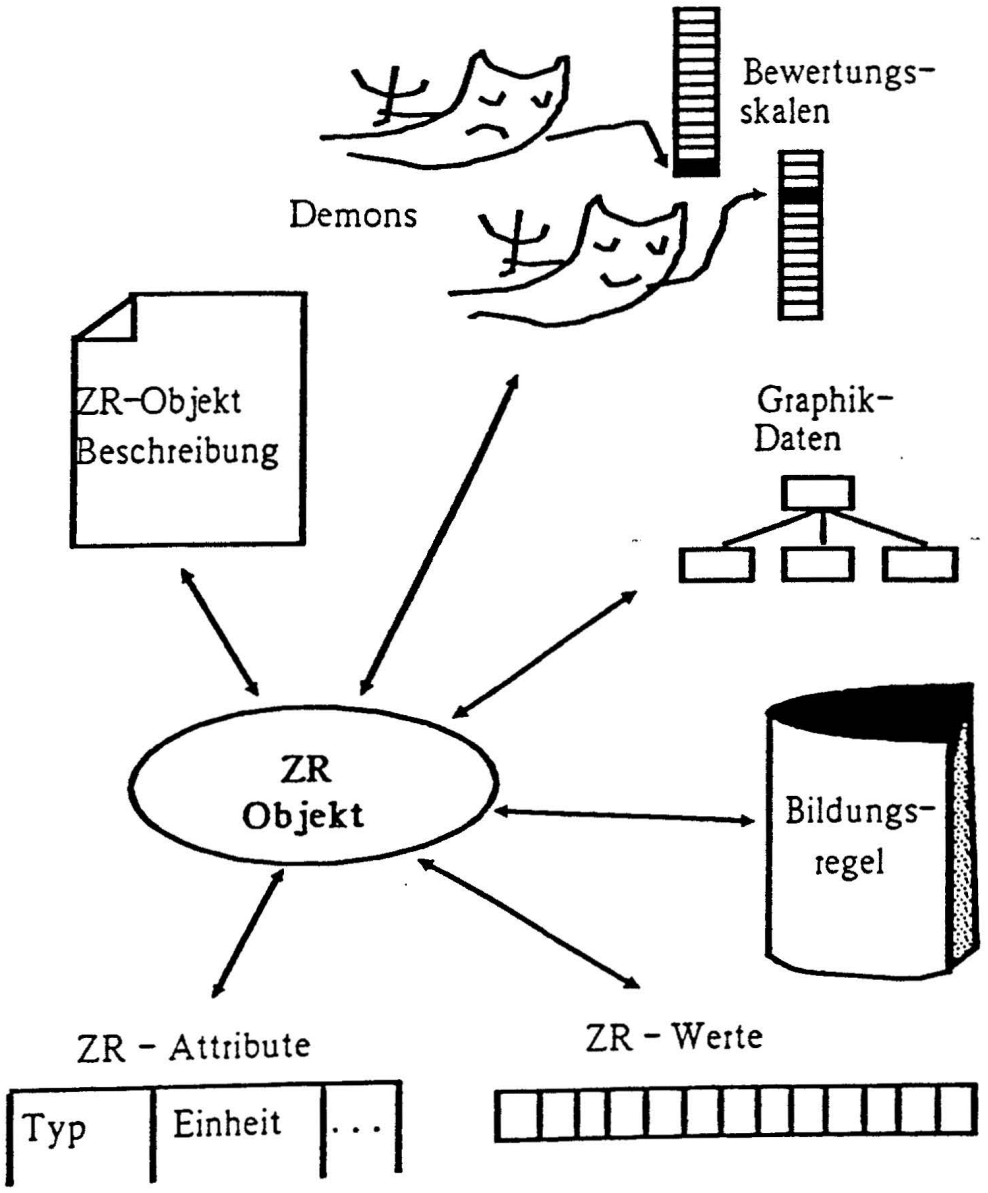

Abb. 3.: Aufbau eines höheren Zeitreihenobjekts

Der in Abb. 3 veranschaulichte Aufbau eines höheren Zeitreihenobjekts ist wie folgt definiert: *hzr* ist der Name der ZRO-Klasse "höheres Zeitreihenobjekt". *zr_obj_beschr* bietet die Möglichkeit der Generierung einer natürlichsprachlichen Beschreibung der dem Zeitreihenobjekt zugrunde liegenden Konstruktionsvorschrift. ***attributes*** und ***values*** werden

nicht – wie bei elementaren Zeitreihenobjekten – mit einer write-Operation gefüllt, sondern mit einer Funktion dynamisch errechnet.

Der Slot *construction* beinhaltet ein Objekt mit einer Vorschrift zur Berechnung der Werte und der Attribute eines Zeitreihenobjekts. Wird diese Regel mit den Funktionen *com_attr()* bzw. *comp_val()* aktiviert, so kann der Inhalt der Subobjekte *zr_attributes* bzw. *zr_values* und damit der Sloteintrag in die Slots *hzr.attributes* bzw. hzr.values berechnet werden. Analog dazu kann mit der Funktion *obj_constr()* die Menge der Zeitreihenobjekte, von denen die Werte und Attribute des (aktuellen) Zeitreihenobjekts funktionell abhängen, bestimmt werden.

Das *CONSTRAINT c1* prüft, ob der Level des *hzr*'s höher ist als der Level aller Zeitreihenobjekte, aus denen es errechnet wird. Operationen ermöglichen die Erzeugung einer Instanz der *hzr*-Klasse und können diese wieder löschen. Die Konstruktionsvorschrift *construction* und ihre natürlichsprachliche Beschreibung *zr_obj_beschr* kann gelesen und neu beschrieben werden. Demons können neu angelegt werden (*o5*) und vorhandene Demons gelesen, aktualisiert und gelöscht werden (*o6* bis *o8*).

OBJECT hzr:		
FATHER_OBJ:		*OBJECT zeitreihen_object*
ARGUMENT	*zr_obj_beschr:*	*STRING*
P_OBJ	*attributes:*	*OBJECT zr_attribute* $\leftarrow$ *comp_attr(hzr.construction)*
P_OBJ	*values:*	*OBJECT zr_werte* $\leftarrow$ *comp_val(hzr.construction)*
V_ARGUMENT	*depends_on:*	*SET OF all_object* $\leftarrow$ *obj_constr(hzr.bildungsobject)*
L_ARGUMENT	*construction:*	*CONSTRUCT_OF descr_lan*
P_OBJ	*demons:*	*SET OF OBJECT demon*
CONSTRAINT:	*c1*	$\forall x\ (x \in hzr.depends\_on \Rightarrow x.level < hzr.level)$
OPERATION:	*o1*	*create_instance()*
OPERATION:	*o2*	*remove_instance()*
OPERATION:	*o3*	*read(zr_obj_beschr, construction)*
OPERATION:	*o4*	*write(zr_obj_beschr, construction)*
OPERATION	*o5*	*write(demon)*
OPERATION	*o6*	$(demon \in hzr.demons) \Rightarrow$ *read(demon)*
OPERATION	*o7*	$(demon \in hzr.demons) \Rightarrow$ *update(demon)*
OPERATION	*o8*	$(demon \in hzr.demons) \Rightarrow$ *remove(demon)*

Das Object *all_object* dient dazu, daß an Stellen, an denen ein elementares Zeitreihenobjekt oder ein *hzr* stehen kann, nicht immer diese Alternative ausgeschrieben werden muß:

OBJECT all_object:
FATHER_OBJ: *f1* *OBJECT el_zeitreihen_object*
FATHER OBJ: *f2* *OBJECT hzr*
CONSTRAINT: *c1* $f1 == NULL \ XOR \ f2 == NULL$

Aufgabe des Slots *hzr.construction.bildungsregel* eines höheren Zeitreihenojekts ist die Definition des Berechnungsalgorithmus für den Sloteintrag *hzr.values* und den Slot-Eintrag *zr_attributes*[26]. Bei der Definition bzw. der Auswahl eines hierfür geeigneten Formalismus war es zunächst naheliegend, auch hier einen objektorientierten Ansatz in Erwägung zu ziehen. Allerdings sollte das berühmt-berüchtigte Beispiel der Realisierung von 1 + 1 in Smalltalk[27] zur Vorsicht mahnen und den Designer veranlassen, objektorientierte Ansätze nur in für diesen Formalismus geeigneten Aufgabenstellungen einzusetzen. Da im vorliegenden Fall arithmetische Verknüpfungen und statistische Umformungen von Zeitreihenobjekten unterstützt werden sollen und dabei ein Mechanismus angeboten werden sollte, mehrere Verfahren hintereinander zu schalten, um auf die Speicherung von Zwischenergebnissen - wie etwa von temporären Hilfs-Zeitreihenobjekten - verzichten zu können, erscheint ein mehr funktionsorientierter Ansatz als Grundlage zur Definition der Beschreibungssprache naheliegender[28].

Aus diesem Grund besitzt der Slot *bildungsregel* den Eintrag *CONSTRUCT OF descr_lan*, der für Instanzen der Klasse *hzr.construction* einen wohlgeformten Ausdruck der Beschreibungssprache *descr_lan* erfordert. Diese ist modellierungstechnisch auf der gleichen Ebene wie die Regelsprachen von Expertensystemen angesiedelt.

Die Regelsprache für die Bildungsregel[29] eröffnet dem modellierenden Benutzer die Möglichkeit, höhere Zeitreihenobjekte einschl. der Berechnungsvorschrift für Werte und Attribute zu definieren. Unbenommen vom funktionalen Ansatz der Beschreibungssprache ist es vom Modellansatz her prinzipiell möglich, statistische[30] und arithmetische Operationen als Objekte zu definieren, deren Berechnungsverfahren von der Beschreibungssprache aus aufgerufen werden können.

Die Beschreibungssprache ermöglicht somit eine Art *Vererbung* von Werten und Attributen der Quell-Zeitreihenobjekte auf das Zeitreihenobjekt, in dessen Slot sie sich befindet. Im Gegensatz zur normalen Vererbung werden die Ausprägungen in den Slots *values* und *attributes* nicht (nur) kopiert, sondern aufgrund der in der Beschreibungssprache codifizierten Vorschriften modifiziert.

---

[26] Die selbst wiederum eine komplexe Struktur haben.
[27] Vgl. hierzu [Smal 88,S. 49].
[28] Vgl. hierzu auch andere Ansätze mit hybriden Wissensrepräsentationen, wie z.B. [Chri 89].
[29] Eine detaillierte Bechreibung dieser findet sich in [Augs 90].
[30] Wie in [Oldf 86] vorgeschlagen.

Über sein Vater-Objekt erbt eine höhere ZRO-Klasse (durch Vererbung auf Klassenebene) auch den *values*-slot. Dieser ist in der ZRO-Klasse *hzr* eigentlich redundant, da mit der Bildungsregel ein Algorithmus zur Berechnung existiert. Denkt man an eine Implementierung des dargestellten Modells, so müßten bei häufigen Zugriffen auf die *values*[31] eines höheren Zeitreihenobjekts diese bei jeder Anfrage, ggf. über Kaskaden von höheren Zeitreihenobjekten hinweg, stets neu berechnet werden, was eine permanente Residenz der Werte in höheren Zeitreihenobjekten - auch im Modellentwurf - ratsam erscheinen läßt.

7 Die Datenüberwachung durch Demons

Operationelle Mittel für die Datenbewertung sind die Demons. Diese untersuchen Zeitreihenwerte[32] auf vorgegebene Muster hin und erstellen, falls diese Muster vorliegen, Meldungen. Nach der Klassifikation von [Gene 89] handelt es sich bei den Demons um tropistische Agenten. Muster innerhalb einer Zeitreihe können beispielsweise periodische Schwankungen, Trends oder das Über- oder Unterschreiten vorgegebener Werte sein.

Bedingt durch den Sloteintrag *SET of demon* des slot *demon* der ZRO-Klasse *hzr* kann jedes höhere Zeitreihenobjekt keinen, einen odere mehrere Demons besitzen.

OBJECT demon:		
FATHER_OBJ:		*OBJECT all_zr_objects*
L_CONSTRAINT	*trigger:*	*CONSTRUCT OF demon_lan*
L_CONSTRAINT	*action:*	*CONSTRUCT OF demon_lan*
OPERATION:	*op*	*trigger*() $\Rightarrow$ *action*()

Jeder Demon besteht aus einem System von Produktionsregeln[33]. *trigger* und *action* sind Konstrukte in der Demon-Beschreibungssprache, die durch eine *OPERATION* aktiviert werden können. *trigger* liefert einen Wahrheitswert zurück. Im Gegensatz zu manchen Produktionsregeldefinitionen ist es mit derartigen Aktionen nicht möglich, die Regeln des Regelsystems selbst zu verändern. Ebenfalls ist es nicht möglich, damit die Definition höherer Zeitreihenobjekte zu verändern. Gilt

$$trigger() == true,$$

so wird *action* ausgeführt und liefert selbst bei erfolgreicher Ausführung den Wahrheitswert true. *trigger* selbst ist im Prinzip eine prädikatenlogische Formel und wird in der Syntax der Beschreibungssprache für Demons formuliert. Aus dieser heraus können statistische Verfahren aufgerufen werden. Diese können eine Zeitreihe auf einen Wert oder

[31] Z.B. von Demons im Bewertungsprozeß.
[32] D.h. dessen numerische Werte.
[33] Zur Begriffsdefinition s. z.B. [Wins 82,S. 201ff.].

einen Wahrheitswert abbilden. Im ersten Fall ist dann (innerhalb des Demons) noch eine Abbildung des Wertes auf einen Wahrheitswert notwendig. Bei den implementierten Verfahren wurde eine Auswahl von gut interpretierbaren, wenig Voraussetzungen benötigenden statistischen Verfahren getroffen, deren Anwendungsmöglichkeit gegeben schien. Dabei handelt es sich unter anderem um:

- statistische Tests wie Run_Test und Tests auf Basis des einfachen linearen Modells,
- Verfahren, die Istwerte der Zeitreihen oder Maßzahlen, deren Wachstum etc., mit Sollwerten vergleichen.

Bei den Aktionen handelt es sich um Verfahren zur

- Ausgabe von Zeitreihenwerten,
- Berechnung von Maßzahlen etc.,
- Ausgabe von Meldungen.

Letzteres entspricht damit dem "Versenden einer Meldung" in der objektorientierten Terminologie. Die Resultate von Aktionen können sich auch an das Betriebssystem des Rechners wenden. Sei *VORSTAND* eine Adresse des E-Mail-Systems, welches von der Objektwelt der Zeitreihenobjekte aus erreichbar ist, so kann an diese eine Meldung gesendet werden:

IF	*"Vgl._Lagerkosten_Vorjahr"*	*> +20%*
AND	*"Vgl._Lagerbewegungen_Vorjahr"*	*< +10%*
THEN	*SEND MAIL TO VORSTAND*	

Die Meldungen können neben Texten auch statistische Informationen usw. umfassen, die vom einem Demon berechnet werden. Zur Beschreibung der Demons wurde eine formale Sprache entwickelt, die aber zu umfangreich ist, um sie im Rahmen dieses Artikels vorzustellen.

Ein Beispiel für einen einfachen Demon ist die Überwachung einer Kostenstelle in Relation zum Vorjahreswert. Wird überdurchschnittliches Wachstum oder überdurchschnittliche Abnahme der Kostengrößen festgestellt, so löst dies eine Demonmeldung aus. Im allgemeinsten Fall besagt die Demonmeldung, daß hier eine auffällige Entwicklung eingetreten ist, die von zuständiger Stelle näher betrachtet werden sollte.

8 Implementierung

Auf Grundlage des vorgestellten Modellansatzes wurde ein Prototyp in C und Prolog implementiert[34]. Durch die Beschränkung der Ausdrucksfähigkeit der verwendeten Implementierungssprachen kann diese Implementierung jedoch nur eine Annäherung an den vorgestellten Modellansatz darstellen.

Literaturverzeichnis

[Augs 90] Augsburger W., Rieder H., Schwab J., 1990
Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen; in: Herget J., Kuhlen R.; Pragmatische Aspekte beim Entwurf und Betrieb von Informationssystemen; Proc. 1. Int. Symp. für Informationswissenschaft; Konstanz

[Chri 89] Christaller T., di Primo F., Voss A., 1989
Die KI-Werkbank Babylon; Bonn

[Flei 91] Fleischer P., Behdjati A., Bagdon S., Schlüter P., 1991
Der objektorientierte Software-Entwicklungsprozeß und seine Unterstützung durch Werkzeuge; in: Softwaretechnik-Trends, 11 (1991) 1, S. 24-53

[Gale 85] Gale W., 1985
Artificial Intelligence Research in Statistis; in: The AI Magazine 4 (1985), S. 72-75

[Gene 89] Genesereth M. R., Nilson N. J., 1989
Logische Grundlagen der Künstlichen Intelligenz; Braunschweig / Wiesbaden

[Hawl 87] Hawley R., 1987
Artificial Intelligence Programming Environments; Chichester

[Henn 90] Hennerkes, W. A., 1990
MAXDATA; Berlin u.a.

[Kemp 89] Kempel H.-J., Pfander G., 1990
Praxis der objektorientierten Programmierung; München, Wien

[34] Für Details siehe [Augs 90].

[Kreu 90] Kreutzer W., 1990
Grundkonzepte und Werkzeuge objektorientierter Systementwicklung - Stand der Forschung und Anwendung; in: Wirtschaftsinformatik 32 (1990) 3, S. 211-227

[Oldf 86] Oldford R., Waterloo, Peters S, 1986
Object-Oriented Data Representations for Statistical Data Analysis; in: Compstat 1986, S. 301-306, Heidelberg

[Pare 89] Paredaens J., De Bra P., Gyssens M., Van Gucht D.; 1989
The Structure of the Relational Database Model; Berlin et. al.

[Rich 89] Richter M. M., 1989
Prinzipien der Künstlichen Intelligenz; Stuttgart

[Smal 88] Smalltalk; 1988
Smalltalk/V286, Object-Oriented Programming System (OOPS); Los Angeles

[SPSS 88] SPSS, 1988
SPSS-X User's Guide, 3rd edition; Chicago

[Stau 85] Staud Josef, 1985
Online Retrieval in Numeric Data Bases; in: IATUL Proceedings 17; pp. 164-228

[Stoy 88] Stoyan H., 1988
Programmiermethoden der Künstlichen Intelligenz; Band 1; Berlin u.a.

[Stoy 91] Stoyan H., 1991
Programmiermethoden der Künstlichen Intelligenz; Band 2; Berlin u.a.

[Wegn 87] Wegner P., 1987
Dimensions of Object-Based Language Design; in: OOPSL 1987 Proceedings, New York ACM Sigplan Notices 22 (Octr. 1987), pp. 168-182

[Wegn 89] Wegner P., 1989
Learning the Language; in: BYTE March 1989, pp. 245-253

[Wins 82] Winston, P. H., 1984
Artificial Intelligence; Addison-Wesley Deutschland, Reading Mass.