Kaeding, Anne-Kathrin; Heyn, A.; Detschew, Vesselin; Funkat, Gert; Specht, Martin:

Objektorientierte Modellierung für mehrkanalige EEG-Signale

Zuerst erschienen in:	Biomedizinische Technik = Biomedical Engineering. - Berlin [u.a.] : de Gruyter. - 43 (1998), s3, S. 57-59.
Erstveröffentlichung:	1998
Datum Digitalisierung:	17.07.2009
ISSN (online):	1862-278X
ISSN (print):	0013-5585
DOI:	10.1515/bmte.1998.43.s3.57
[*Zuletzt gesehen:*	31.07.2019]

Objektorientierte Modellierung für mehrkanalige EEG-Signale

Kaeding A.-K.[1], Heyn A.[1], Detschew V.[1], Funkat G.[1] und Specht M.[2]

[1] Institut für Biomedizinische Technik und Informatik, Technische Universität Ilmenau
PF 10 05 65, 98684 Ilmenau

[2] Klinik für Anästhesiologie und Intensivtherapie, Friedrich-Schiller-Universität Jena
Bachstraße 18, 07740 Jena

ABSTRAKT

Der Erfolg moderner Softwareentwicklung ist maßgeblich abhängig von einer sorgfältigen Analyse und der anschließenden Modellierung der betrachteten Problemdomäne. Der Beitrag stellt die formale Beschreibung von EEG-Signalen mit Hilfe der Unified Modelling Language vor und berichtet über die Implementierung des resultierenden Datenmodells.

EINLEITUNG

Im klinischen Bereich und in der Forschung wird für die Analyse von mehrkanaligen bioelektrischen Signalen eine Vielzahl unterschiedlichster Dateiformate verwendet. Sie sind häufig hardware-spezifisch oder für ganz spezielle Anwendungen entwickelt. Die Aufzeichnung bioelektrischer Signale gestaltet sich durch eine Reihe von Zusatzinformationen u.U. sehr komplex. Dadurch bedingt ist immer wieder ein großer Zeitaufwand für die programmtechnische Umsetzung simpler Aufgaben (z.B. Lese- und Schreibroutinen). Ebenso inakzeptabel ist es, daß uneinheitliche Datenformate wieder und wieder Neuimplementierungen von Standardroutinen der Signalverarbeitung erzwingen.

Durch die Entwicklung eines einheitlichen Datenmodells zur Kapselung und Verwaltung von mehrkanaligen bioelektrischen Signale kann ein gemeinsamer Ausgangspunkt für weitere Applikationen geschaffen werden.

Folgende Anforderungen werden als wichtig für ein solches Modell erachtet:

- Stabilität gegenüber Änderungen und Erweiterungen. Neue Dateiformate und Ergebnisse von Standardisierungsbemühungen sollen ohne großen Aufwand eingeführt werden können. Gleiches gilt für die Integration von neuen oder verfeinerten Signalverarbeitungsroutinen.
- Wahrung eines hohen Grades an Flexibilität und Transparenz der zugrundeliegenden Klassenhierarchie. Dadurch ist eine effektive Wartung des Modells möglich.
- Kapselung der Datenstrukturen und Bereitstellung einer Schnittstelle nach außen. Daten in unterschiedlichen Dateiformaten werden über diese Schnittstelle in einem einheitlichen Format verfügbar gemacht.
- Einsatz von objektorientierten formalen Beschreibungsmitteln für das Modell. Dadurch wird gleichzeitig eine sehr gute Dokumentation der entwickelten Klassen erreicht und die Wiederverwendbarkeit unterstützt.
- Realisierung des Modells mit Hilfe einer objektorientierten Programmiersprache.
- Wiederverwendbarkeit in Applikationen.

Für ein Forschungsprojekt gemeinsam mit der Klinik Anästhesiologie und Intensivtherapie der FSU Jena zum Entwurf von Mustererkennungsobjekten und gekapselten Methoden zu deren Verarbeitung war es erforderlich, für verschiedene EEG-Formate ein einheitliches Datenmodell zu entwickeln. Die dazu erforderlichen Schritte werden im Artikel beschrieben.

VERWENDETE TECHNOLOGIEN: OBJEKTORIENTIERUNG UND UNIFIED MODELING LANGUAGE

Für die Entwicklung des Datenmodells ist eine methodische Vorgehensweise und ein formales Beschreibungsmittel notwendig. Beides kann das objektorientierte Paradigma leisten. Dieses in seinen Grundlagen einfache Paradigma verwendet Konzepte, die von Nutzern, Analysten, Designern und Programmierern gleichermaßen verstanden werden. Objektorientierte Modelle sind in der Lage, die Realität in hohem Maße zu reflektieren und können eine für den Anwendungsbereich hinreichend genaue Beschreibung der zusammenhängenden Daten und Prozesse geben. Die objektorientierte Analyse basiert auf der Zerlegung natürlicher Komponenten. Die Verwendung von objektorientierten Prinzipien bei der Software-Entwicklung ermöglicht die Wiederverwendung von Architektur und Programm-Code. Weiterhin nehmen objektorientierte Systeme für sich in Anspruch, daß sie einfach zu verstehen und zu warten sind.
Die genannten Vorteile des objektorientierten Paradigmas treten nur in Erscheinung, wenn es in allen Phasen der Softwareentwicklung konsequent angewendet wird. Das betrifft vor allen Dingen die Eigenschaft der Stabilität gegenüber Änderungen und die Wiederverwendbarkeit.

Die Durchsetzung der Objekttechnologie wird durch die Verwendung eines formalen Beschreibungsmittels unterstützt, das in allen Phasen der Softwareentwicklung verwendet wird. Ein solches Beschreibungsmittel ist die

Unified Modeling Language (UML) [2]. Die UML ist das Ergebnis eines Standardisierungsprozesse zur Beschreibung von objektorientierten Modellen in der Analyse-/Designphase des Software-Engineerings. Sie ist von der Object Management Group (OMG) als Standard für die Beschreibung von Objektmodellen akzeptiert. Durch den Einsatz der UML können Ausschnitte einer Problemdomäne mit Hilfe verschiedener Modellsichten repräsentiert werden. Dazu wurde eine ausdrucksstarke und konsistente Notation definiert. Die UML ermöglicht dadurch einen einfachen Übergang von der Analyse über das Design zur Implementierung. Damit wird die Kommunikation zwischen Entwicklern und Nutzern erleichtert. Mißverständnisse und Inkonsistenzen sind bereits im Modell auffindbar und eventuell fehlerhafte Implementierungen können vermieden werden. Die UML unterstützt die Analyse und das Design sowohl von kleinen als auch von großen Systemen. Eine Auswahl der am häufigsten benötigten UML-Elemente ist in den Abbildungen 1, 2 und 3 zu sehen.

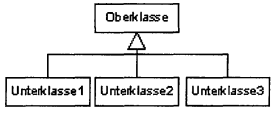

Abb. 1: UML-Notation für die Vererbung [2]

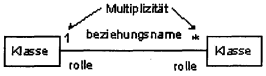

Abb. 2: UML-Notation für Assoziationen

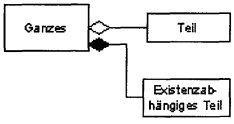

Abb. 3: UML-Notation für Aggregationsbeziehungen

Vorgehensweise bei der Modellierung

Die in der Klinik für Anästhesiologie und Intensivtherapie der FSU Jena verwendeten EEG-Dateiformate Walter-Graphtek, verschiedene Dateiformate der Analyse-Software ATISA für Windows, EBS [1], [3], sowie ein ASCII-Format sind in ihrer Struktur analysiert und verglichen worden. Die Ergebnisse der Analyse führten zur Entwicklung einer Klassenstruktur zur Kapselung mehrkanaliger bioelektrischer Signale.

Der Anspruch eines stabilen und vor allem problemlos änder- und erweiterbaren Datenmodelles ist an Hand des konkreten Problemfeldes geprüft worden. Bedingt durch die Projektaufgabe ist diese Klassenstruktur um spezielle Klassen und Methoden zur Repräsentation von EEG-Daten erweitert worden. Trotz der hohen Komplexität und der Vielzahl der notwendigen Attribute ist die Erweiterung des Datenmodells für mehrkanalige Biosignale weitgehend problemlos möglich gewesen.

Das entstandene Datenmodell wurde formal mit der UML beschrieben. Die entstandene Klassenstruktur ermöglicht die Repräsentation der EEG-Daten aus unterschiedlichen Dateiformaten in einem einheitlichen Datenmodell.

Die Implementierung des Datenmodells ist in Java erfolgt. Da sowohl UML als auch Java dem puren objektorientierten Paradigma entsprechen, birgt der Schritt der Implementierung auch nicht das Risiko von Informationsverlusten oder –verfälschungen. Wird für die Speicherung der Datenobjekte eine objektorientierte Datenbank eingesetzt, so wird auch dort ein Paradigmenbruch vermieden.

Beschreibung der Klassenhierarchie

Für die Realisierung der Klassenhierarchie standen mehrere Möglichkeiten der Implementierung zur Auswahl. Komplexität und Anzahl der benötigten Attribute erforderte eine Aufteilung der benötigten Funktionalität auf mehrere Klassen. Im Modell ist die Abbildung der bei bioelektrischen Signalen üblichen Strukturierung in einzelne Kanäle auf die Datenstruktur erhalten geblieben. Im Folgenden werden kurz Aufbau und Zusammenspiel der einzelnen Datenstrukturen anhand ihrer Schnittstellen erläutert.

Die Interface-Klasse `DataSource` steht an der Wurzel der Hierarchie. Sie stellt verschiedene Lesemethoden bereit. Ein beliebiges Objekt, das dieses Interface implementiert, kann als Datenquelle für entsprechende Views und Datenverarbeitungsklassen dienen. Die Daten sind in Zeilen und Spalten (bzw. indizierten Kanälen) angeordnet. `DataSource` unterstützt Kanäle mit unterschiedlicher Länge und Abtastrate. Es werden keine Datenmanipulationen unterstützt.

Die Interface-Klasse `EditableDataSource` repräsentiert eine editierbare Datenquelle und implementiert zusätzliche Schreibmethoden.

Die Interface-Klasse `Markable` definiert alle Methoden, die zur Verwaltung eines Sets von Markern benötigt werden (kopieren, löschen, hinzufügen, sortieren...).

Die Interface-Klasse `Observer`. Eine Klasse (View, Verarbeitungsobjekt) kann dieses Interface implementieren, wenn es über Änderungen von "Observable"-Objekten (Model) informiert werden will. Dazu muß eine Instanz sich selbst bei einem "Observable"-Objekt registrieren.

Die Interface-Klasse `Observable`. Dieses Interface definiert diejenigen Methoden, die ein "Observable"-Objekt implementieren muß. Eine Klasse sollte dieses Interface implementieren, wenn sie nicht direkt von der Klasse `Model` abgeleitet werden kann. Ein "Observable"-Objekt kann eine beliebige Anzahl von "Observern" verwalten.

Die Klasse `Model` repräsentiert ein "Observable"-Objekt. Sie implementiert das "Observable" Interface und

stellt eine Minimal-Implementierung von diesem Interface zur Verfügung.

Die Klasse `GlobalInfo` sammelt und verwaltet einige globale Eigenschaften zur Beschreibung von bioelektrischen Signalen.

Die Klasse `Channel` bildet die Superklasse für alle spezialisierten Kanalklassen und repräsentierte einen Kanal eines mehrkanaligen Signals. Sie implementiert verschiedene Eigenschaften zur Beschreibung eines Datenkanals, es werden jedoch keine wirklichen Daten in dieser Klasse gespeichert. Die Klasse `DataFile` nutzt diese Klasse zur Beschreibung der in einer Datei gespeicherten Daten.

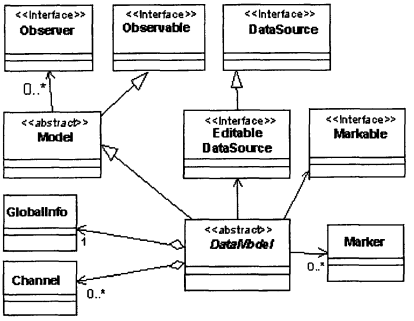

Abb. 4: EEG-unspezifische Klassen des Datenmodells

Die Klasse `DataChannel` ist direkt von der Klasse `Channel` abgeleitet und ermöglicht die Speicherung von Gleitkomma-Werten im Speicher. Sie dient der Klasse `DataObject` zur Verwaltung der einzelnen Kanäle.

Die abstrakte Klasse `DataModel` stellt die abstrakte Basisklasse für alle weiteren Datenklassen dar. Sie ist direkt von der Klasse `Model` abgeleitet. Registrierte Beobachter werden durch geeignete Ereignisobjekte über Änderungen des Zustandes des Datenobjekts informiert. `DataModel` implementiert das Interface `Editable-DataSource` und ermöglicht so das definierte Lesen und Schreiben der verwalteten Daten. Über eine bool'sche Eigenschaft läßt sich der Schreibzugriff steuern. Sie implementiert das Interface `Markable` und kann so eine beliebige Anzahl von Markern verwalten. Die gespeicherten Marker werden automatisch nach ihrem Start-Sample sortiert. Außerdem verwaltet sie eine Instanz der Klasse `GlobalInfo` zur Speicherung der allgemeinen Daten-Eigenschaften. Die eigentlichen Daten werden in einer Liste von Instanzen der Klasse `Channel` gespeichert. Unterklassen von `DataModel` arbeiten teilweise mit speziellen Unterklassen der Klasse `Channel`.

Die Klasse `DataFile` bildet die Basisklasse für alle speziellen Daten-Dateiklassen und implementiert zusätzliche Routinen zum gepufferten Lesen und Schreiben.

Die Klasse `DataObject` stellt das eigentliche gewünschte Datenobjekt dar. Hier wird die Speicherung von Gleitkomma-Werten im Speicher ermöglicht.

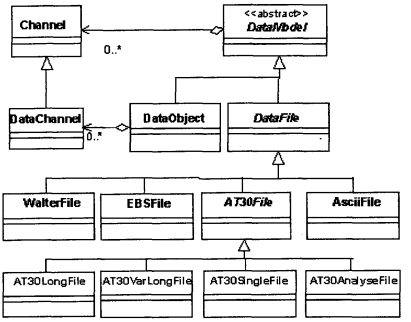

Abb. 5: EEG-spezifisches Klassen des Datenmodells

Ergebnisse

Die beschriebene Klassenhierarchie ermöglicht die Darstellung und die Manipulation von ein- und mehrkanaligen EEG-Daten aus dem Speicher und aus unterschiedlichen Dateiformaten in einem einheitlichem Datenmodell. Es unterstützt unterschiedliche Kanallänge und variable Frequenzen Das beschriebene Datenmodell ist die Grundlage für die Implementierung in einer prototypischen Stationssoftware für die Intensivstation.

Der entscheidende Vorteil der vorgestellten Lösung besteht in der sicheren und weitgehend problemlosen Änder- und Erweiterbarkeit des Datenmodells sowie der anschließenden Überführung in die Implementation. Weitere Datenformate können hinzugefügt werden, ohne daß im Modell grundsätzliche Änderungen vorgenommen werden müssen. Denkbar ist z.B. eine Erweiterung für des Feld der Polygraphie. Damit ist zu jedem Zeitpunkt der Entwicklung die Kontrolle über das System möglich. Es werden Inkonsistenzen, Redundanzen und Unvollständigkeiten weitgehend vermieden.

Darüber hinaus ist ein nicht zu unterschätzende Nebeneffekt die entstandene Dokumentation der Entwicklung, die z.B. einen Vergleich mit dem Pflichtenheft des Anwenders ebenso wie eine effektive Einarbeitung weiterer Entwickler ermöglicht.

Literatur

[1] "Specification of the EBS File Format for Biosignals" http://www.ipb.uni-erlangen.de/biokybernetik/ebsspec.html

[2] Oestereich, B.: "Objektorientierte Softwareentwicklung mit der Unified Modeling Language" 3. Aufl., München : Oldenburg, 1997

[3] Hellmann, G. ; Kuhn, M. ; Prosch, M. ; Spreng, M.: Extensible biosignal (EBS) file format simple method for EEG data exchange. EEG Clin. Neurophysiol. 99 (1996) 426-431