MoBBED (Mobile Brain-Body-Environment Decision-making) Data Infrastructure (White Paper)

By

Arif Hossain, Jeremy Cockfield and Kay Robbins

University of Texas at San Antonio

March 19, 2012

UTSA CS Technical Report TR-2012-006

Updated: 4/15/2012, 5/3/2012

An updated version of the MoBBED data infrastructure is now available from the following two technical reports:

> *MoBBED (Mobile Brain-Body-Environment Decision-making) Part II: Database Design is available as TR-2013-005.*
>
> *MoBBED (Mobile Brain-Body-Environment Decision-making) Part II: User Guide is available as TR-2013-006.*

Contents

1. Introduction

The US Army CAN-CTA project focuses on understanding the brain and body in action using a data-driven approach with a goal of developing assistive and monitoring technologies for soldiers in real world environments. Project scientists collect EEG, eye-tracking, motion capture, muscle activity, heart rate, force-plate responses, as well as sensory and galvanic skin responses from subjects performing a variety of tasks in realistic environments. Video and audio recordings, both of subjects and of the environment are also available. Some datasets, acquired in controlled simulation environments, have detailed simulation-specific information that describes and controls the context. Some subjects will undergo additional imaging modalities (such as diffusion tensor imaging or simultaneous fMRI and EEG recording) in fixed environments to obtain detailed structural brain information.

Brain-body imaging in realistic environments has only recently become feasible with the advent of portable dry-electrode technology. Various partners within the CAN-CTA are beginning to acquire datasets that combine various modalities. However, the equipment, data format, processing, and storage techniques vary considerably across groups. The CAN-CTA data corpus will be, for the most part, annotated streaming data characterized by diverse formats, high data rates, and complex interrelationships. Analysis and discovery in such a data corpus requires a data handling structure that is robust, extensible, and capable of extracting complex combinations of features for analysis and visualization.

Two fundamental assumptions have been widely articulated among CAN-CTA members. The first assumption is that effective data-driven approaches require a data corpus. Without training and test data that covers the range of possibilities, machine learning and other computational approaches fail. The second assumption is that the corpus of data acquired under the initial CAN-CTA can provide a lasting legacy with far-reaching potential beyond the individual experiments and particular research questions that initiated these experiments.

2. What is MoBBED and why does it need a data infrastructure?

The ultimate MoBBED (Mobile Brain-Body-Environment Decision-making) system is the human being --- compact, mobile, adaptive. Environmental observation occurs through the senses, while processing, command, and control functions occur in the brain. Human systems have limits in terms of bandwidth for sensory acquisition, processing capabilities, and decision-making acuity. Furthermore, humans are not machines --- they are subject to disease, injury, fatigue, and stress --- all of which can degrade their performance. In addition, humans have evolved under vastly different conditions than are typically present in a technological, information-rich environment. Assistive and monitoring technologies are needed to improve and stabilize human performance in these situations.

Fig. 1 shows a schematic of a typical real-time MoBBED system for assistance and monitoring. Examples of environmental acquisition include GPS positioning, information from cameras and microphones mounted inside or outside vehicles or robots, body mounted cameras. Feedback and instructions may come in the form of audio or visual cues or possibly cues from other sensory devices. The human instrumentation may include EEG, eye-tracking, voice tracks, motion capture, and physiological measures such as heart rate.

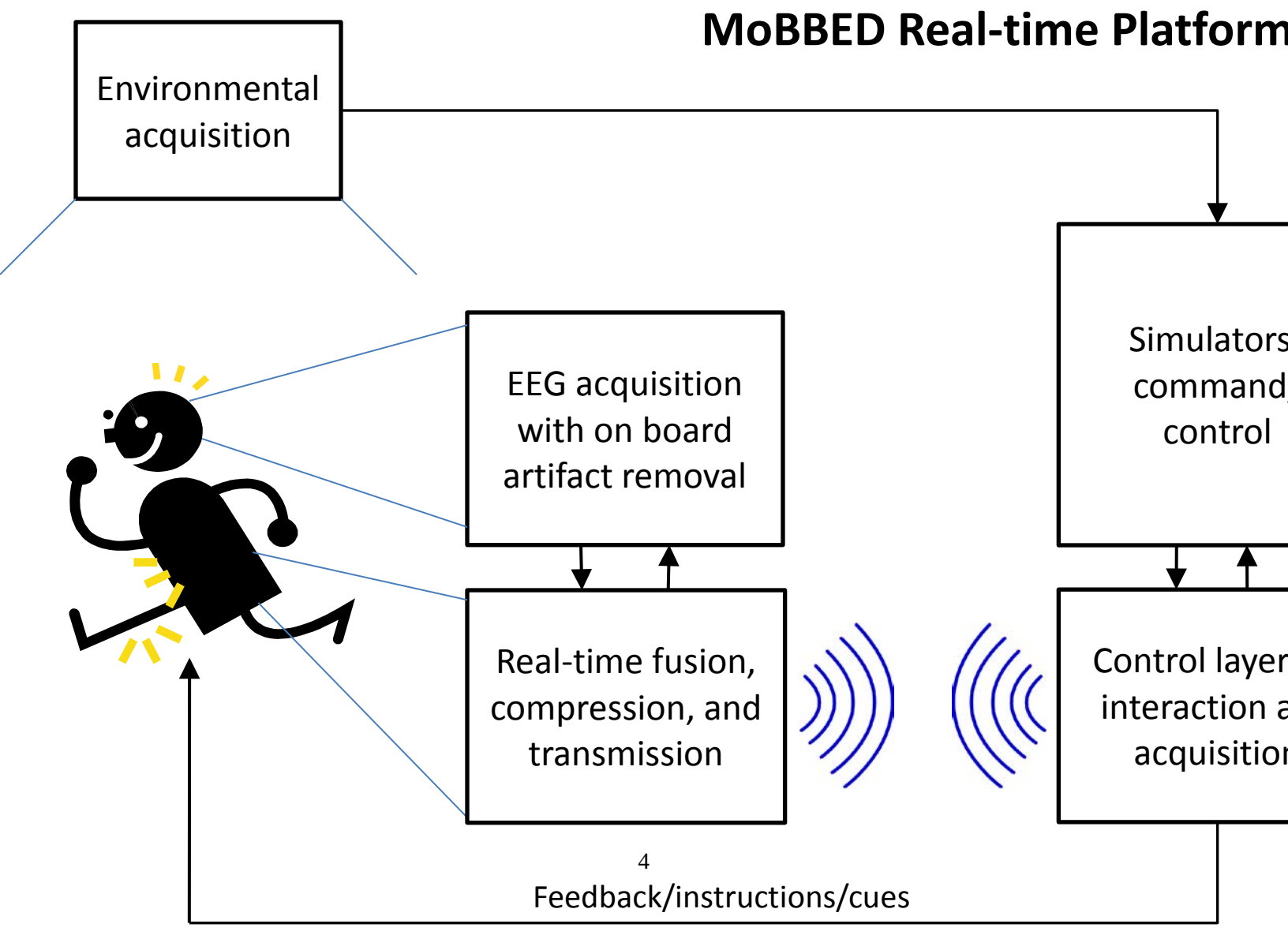

Fig. 1: MoBBED real time platform.

.

Command and control may come from human commanders, automated assistants, or from simulators such as the DCS Crewstation Simulator. These entities require data from other sources to provide a context for decision-making. In training and testing, some of this data can come from preplanned scenarios. Automated assistants may need extensive data for making decisions. Human commanders need their own information assistants to integrate and summarize complex situations.

Fig. 2 shows a proposed CAN-CTA analysis platform dataflow. Data from laboratory instrumentation is stored in archive files that come in a variety of formats including proprietary instrumentation formats, the Data River XML[1] format, and laboratory-specific formats.

In the current incarnation of the typical data flow for CAN-CTA members, the analysis software consists of EEGLAB, BCILAB, and other software written in MATLAB. The access layer for EEG data is `EEG`, an instrumentation-independent MATLAB data structure used in most EEGLAB analysis. Original archival data is stored in various formats and in various states of processing by individual experimenters. There is currently no standard for integrating EEG data and other types of data for analysis. The current workflow does not contain either archival or computational databases.

To address the problem of fusing different types of data for analysis, members of SCCN at UCSD have proposed development of standardized MATLAB structures similar to `EEG` for other modalities [2]. The MoBBED analysis suite will need standardized structures for eye tracking, motion capture, and physiological indicators to permit the development of general purpose processing and machine learning algorithms.

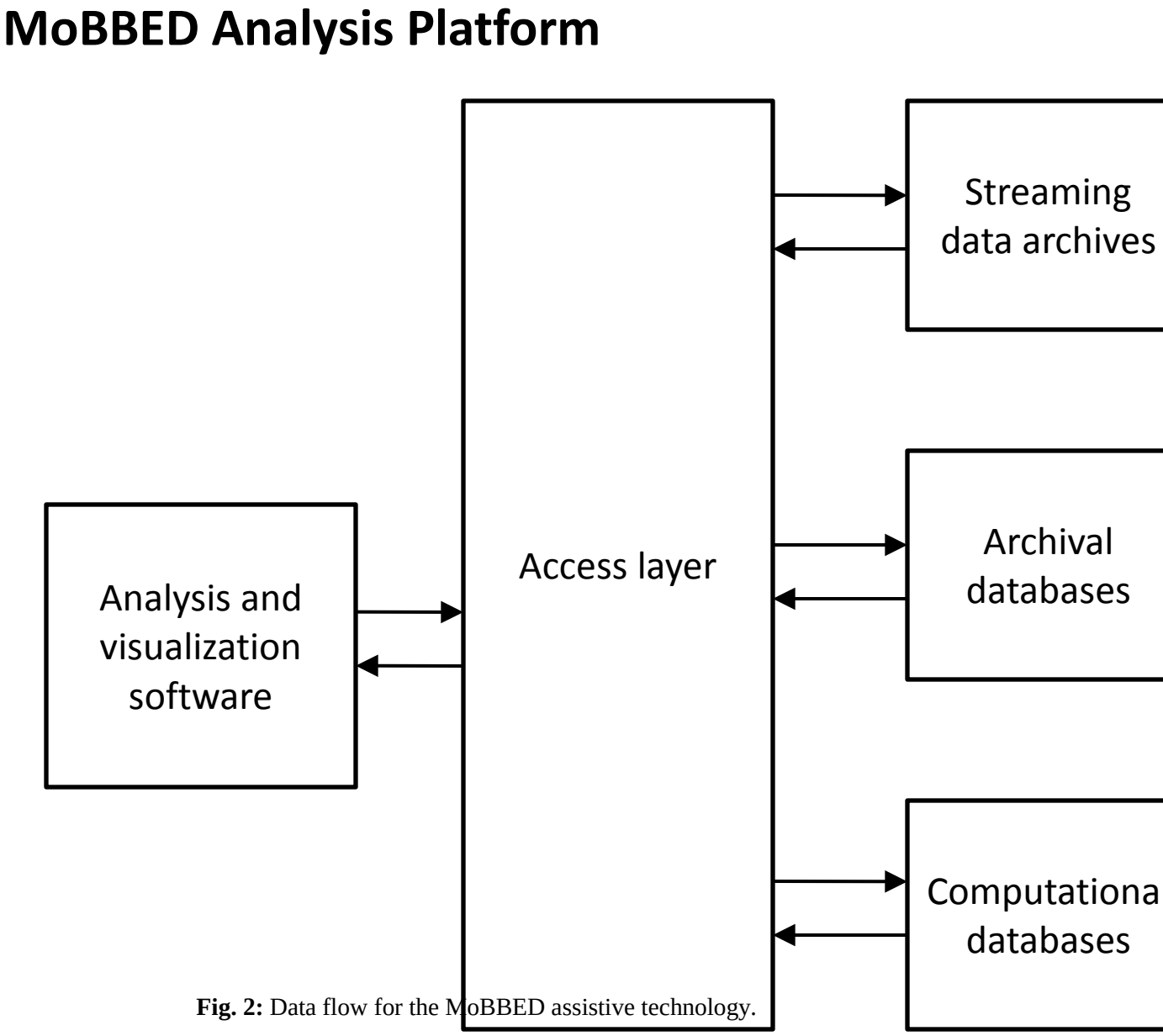

Fig. 2: Data flow for the MoBBED assistive technology.

The current approach of storing data in MATLAB structures provides an efficient and highly effective access layer that allows researchers to develop algorithms in MATLAB that are independent of the acquisition instrumentation. However, the approach also has some limitations in terms of scalability and access. Researchers must keep the entire dataset in memory during computation, which will become prohibitive as the datasets become larger and data mining requires combinations of multiple datasets. Furthermore, stored MATLAB structures are not searchable, making it hard to extract and combine data across multiple datasets without reading all of the datasets into memory and scanning them. Finally, because no central data repository exists, the EEG structures must be redundant, storing a copy of the original dataset along with the transformation, each time researchers perform an ICA or apply a different

version of artifact removal. EEGLAB has standardized the MATLAB EEG structure for ICA transformations, so analysts must include their own non-standardized fields if they wish to perform a different type of transformation.

To address the memory issues, Nima Bigdely Shamlo, Christian Kothe, and Alejandro Ojeda of SCCN are developing an access layer infrastructure called Mobi based on memory-mapped files [3]. This development is an important step in improving the scalability of the MoBBED data flow. However, by its nature, memory-mapping is a local strategy. The data structures map to the local file system, and each machine must keep its own copy. One might envision an implementation that uses proxies to distribute pieces across the network on other machines, but it is not clear what the performance of such a strategy would be.

Memory mapped files will play a central role in the local caching structure needed to support future large-scale MoBBED computation. However, this strategy does not address some of the other limitations mentioned above. In an effort complementary to the UCSD work, UTSA is implementing an experimental database back end (MobbedDB) to address the following issues:

- Accessibility of large sets of MoBBED data across research groups for searching, extraction, and mining.
- Separation of the original data from various transformations such as channel and epoch rejection or ICA while maintaining an association so that researchers know where the data came from and what transformations it has undergone (provenance).
- Storage of the data in a platform-independent way so that tool developers and researchers are not locked into a particular platform such as MATLAB, but still have the option of taking advantage of these platforms if they choose.
- Provision of the basic infrastructure needed to apply large scale distributed processing techniques using tools such as Apache Hadoop [4] to perform data mining.
- Data abstraction to facilitate retrieval and classification.

As shown in Fig. 2, the envisioned workflow has two different types of databases. Archival databases provide permanent storage and a high-level of security. Computational databases provide more support for data mining and large-scale computation. The HeadIT project at UCSD [5] and the IEEG.org project [6, 7] are examples of archival databases. Data coming from instrumentation would initially be stored in instrumentation-specific formats or in the Data River XML format. UCSD's MobiLab [8] is an access layer tool that provides drivers to store streaming data into structured local files. The access layer software would also provide drivers for reading and writing this data to the archival database(s) in an offline batch processing operation.

The MoBBED data infrastructure includes a computational database that provides an alternative to flat files for facilitating large-scale computation. We have implemented a prototype MoBBED database called MobbedDB using PostgreSQL [9]. The analysis software is MATLAB and the access layer has a Java implementation. We have also implemented a graphical user interface for the database in MATLAB. The remainder of this white paper gives an overview of the prototype design and implementation. Section 3 gives an overview of the design strategy. Section 4 gives a more detailed description of the MobbedDB design, and Section 5 describes the basic access-layer user interface. Section 6 provides some performance measurements for the prototype implementation, and Section 7 discusses a future design and implementation path. Appendix A shows some screenshots of MobbedDB GUIs.

3. Implementation overview

MobbedDB has a relational schema currently implemented as a PostgreSQL database, a well-established open source database that is widely used for information retrieval research. PostgreSQL has a number of data handling and query extensions not available in mySQL. The open source pgAdmin tools [10] provide a nice suite of database management and browsing capabilities outside of the MoBBED framework. MobbedDB follows several design principles:

Principle 1: Disk storage is infinite

The design consequence of this principle is that large tables and database redundancy can pay for efficiency in extraction and in the capability of extension. The rapid decline in disk costs and the availability of federated servers affirms that this principle is practical for long-term design horizons. Every row and hence every entity in MobbedDB is uniquely identified by a 128-bit UUID.

Principle 2: Extensibility is primary and trumps efficiency

One design consequence of this principle is that MobbedDB tables are narrow and general rather wide and specific. As a result, more queries may be required to retrieve the required records than would ordinarily be required in a focused design. Additional assembly of information from diverse records may be required in such a design. A mitigating factor in this design choice is that analysts will access MobbedDB databases through a caching middleware layer of assembled information specific to individual applications. Often MobbedDB performs the assembly outside of the SQL query mechanism by copying records to a stream.

Principle 3: Structures should be controlled but extensible

Users work with a variety of structures to analyze their data and specification of these structures is necessary for storage and retrieval as well as searching. However, because of the complex and ever-evolving nature of MobbedDB, designers cannot anticipate all future structure types. MobbedDB treats structure layout as data and promotes definition of hierarchies of structure types.

Principle 4: Primary data acquired from experiments is distinct from user transformations

The EEG MATLAB structure in EEGLAB inter-mixes user transformations such as ICA, trial rejection, and event transformations with primary data. This approach makes the representation very compact and promotes efficiency of analysis for a single dataset. However, the approach has some drawbacks for large-scale analysis tasks. A researcher wishing to run an analysis with different epoching strategies or with different rejected channels must create two complete copies of the dataset. The current infrastructure does not provide an organized method of keeping track of the provenance of such operations. MobbedDB stores primary datasets separately from user transformations, but links them to allow users to perform limited provenance tracing of the transformation sequence.

Initially, MobbedDB will only store original data. However, as the CANCTA comes to an agreement on a standardized and automated preprocessing procedure (dereferencing, sampling, high pass filtering, artifact removal, saccade and blink separation, ICA transformation, etc.), the MobbedDB will represent and store "standardized" versions of the data. MobbedDB provides users with the capability of creating additional tables of transformed data and additional data mining capabilities. The next section outlines the basic structure of our initial MobbedDB PostgreSQL design.

4. MobbedDB entities

The MobbedDB design includes eleven (11) different primary entities (e.g., attribute, comment, contact, data_def, dataset, device, element, event, subject, tag, and transform). Each entity has a universally unique 128-bit identifier (UUID) that serves as the unique key for the corresponding database table. The unique identification of primary entities, allows users to merge databases without conflict and to back track entity types based on their UUIDs. Section 6 provides more implementation details.

4.1 Datasets and their structure

A dataset represents a group of related data. A dataset may represent data collected in a single modality such as EEG or eye-tracking that is identified as a unit and represents a single experiment. A dataset may also represent a collection of other datasets. Datasets are the fundamental organizing entities in MobbedDB. A dataset may have elements that represent streams of time-stamped measurements. They also may have events, attributes, metadata such as tags, and other types of data. Fig. 3 shows a schematic of the organization around a dataset.

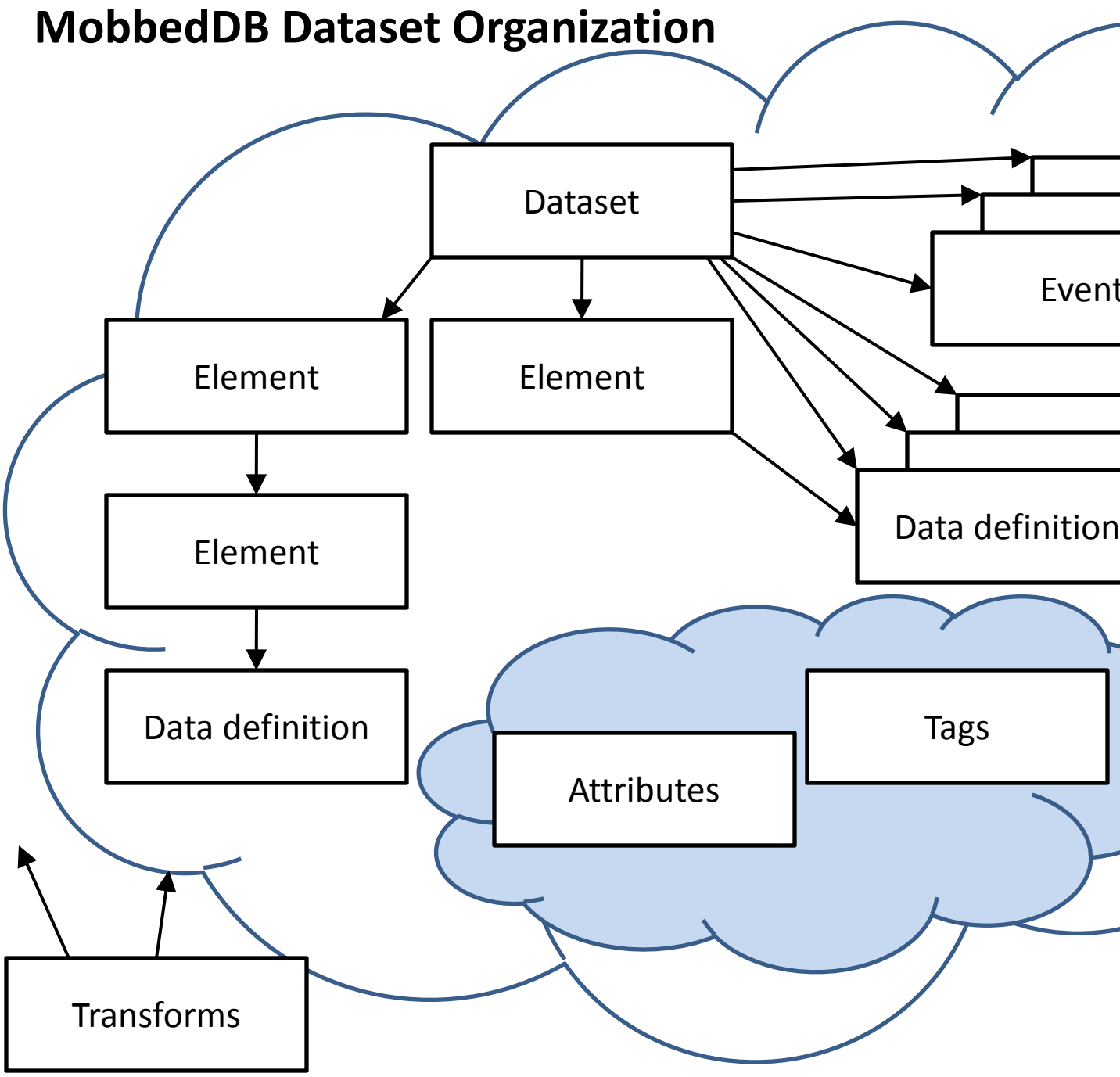

Fig. 3: Schematic organization of a dataset in MobbedDB.

Elements represent streams of time-stamped data. These elements may be organized in a hierarchy. For example, an element may represent an entire EEG cap or just a single channel in that cap. Elements or datasets may have data associated with them. The data may be stored in one of five ways: time-stamped numeric array, time-stamped XML blob, numeric-array, XML blob, or external file. Information and properties that do not fit into the event/data format are stored as attributes. Users may associate tags with any of the other entities and use the tags for searching and data mining.

Consider for example, the `EEG` struct that represents EEG data in EEGLAB. The `EEG.chanlocs` field is a structure array holding channel locations and corresponds naturally to the dataset elements. The current mapping of the `EEG` struct to the database creates a single element corresponding to the EEG cap and identifies the EEG data with this cap. If the `EEG.chanlocs` is not empty, MoBBED creates additional elements corresponding to each channel. The value of a channel property for the i^{th} channel, such as `EEG.chanlocs(i).labels`, is mapped to an attribute. The attribute has a structure ID that encapsulates information about structure layout for retrieval purposes. MobbedDB handles events in a similar manner, but treats events as primary entities rather than attributes because of their central role in searching and retrieval.

Events are entities with a type, a starting time, an ending time, and a unique entity owner. MobbedDB stores additional event properties as attributes. Events provide descriptive overlays for other MobbedDB entities. Researches can code traditional EEG events such as the appearance of a target as event entities. However, the event infrastructure also supports the coding of more general features, e.g., a strong correlation of a given channel with another or the appearance of energy of a certain level in a specified frequency band. Researchers must first extract these implicit events and then code them in MobbedDB by associating them with appropriate event types. Once coded, researchers can then retrieve a variety of signals associated with the occurrence of such events. Channel and epoch rejection also fit into the event infrastructure.

4.1.1 The DATASETS Table

The DATASETS table identifies the datasets in the database. We assume that each dataset consists of a single modality (such as EEG data). MobbedDB specifies the structures for extraction in type tables described in the next section. The DATASET_UUID identifies this dataset in operations and serves as the unique key for the DATASETS table. Primary datasets use their own DATASET_UUID as the DATASET_PARENT_UUID, while derived datasets use the DATASET_UUID of the dataset from which they were transformed. The description may contain information about the transformation. Child datasets inherit the events and attributes of their parents.

DATASETS: [key: DATASET_UUID**]**

Table column	Meaning
DATASET_UUID	UUID identifying the dataset
DATASET_NAMESPACE	string identifying the namespace (e.g., lab URL) for this dataset
DATASET_NAME	string name identifying this dataset
DATASET_VERSION	integer version number of this dataset
DATASET_CONTACT_UUID	UUID of the contact person for this dataset
DATASET_CREATION_DATE	date that dataset was entered into the database
DATASET_DESCRIPTION	description of the dataset
DATASET_PARENT_UUID	UUID of parent dataset (own UUID for primary datasets)
DATASET_MODALITY_UUID	UUID of the modality of dataset

Some systems that interact with MobbedDB require that datasets have a unique name. EEGLAB does not enforce uniqueness. MobbedDB provides unique naming through the combination of the namespace, the name, and the version. The DATASET_NAMESPACE is typically the URL or other string uniquely identifying the laboratory or the investigator. Once they have specified a namespace, researchers only have to be concerned about unique naming within their own workspace. Depending on the type of insertion, MobbedDB interface routines use a flag to indicate whether a duplicate insertion (without version number) should fail and return with an error or should complete correctly after incrementing the version number.

4.1.2 The ELEMENTS table

Element entities identify time stamped data streams. Elements may or may not have a fixed sampling rate and may be associated with multiple datasets. An element may correspond to a single entity or may contain sub elements. For example, researchers may choose to store the data corresponding to one EEG run using a cap with 36 channels as a single element with each time snapshot of the data stored in a single row of the numeric data table. However, because the individual channels of the cap have properties such as name and position, researchers could also create additional child elements for each channel and associate attributes with these elements. MobbedDB can index element data for individual channels through the parent element data using the ELEMENT_POSITION value.

ELEMENTS [key: ELEMENT_UUID**]**

Table column	Meaning
ELEMENT_UUID	UUID identifying this element
ELEMENT_LABEL	Name of the element (channel in EEG)
ELEMENT_PARENT_UUID	parent element if this element should be grouped
ELEMENT_POSITION	position of this element in parent's group (-1 if all)
ELEMENT_DESCRIPTION	description of what this element represents

4.1.3 The EVENT_NAMES and EVENTS table

Event entities have a name, a starting time, and an ending time. MobbedDB associates each event with a particular target entity. For example, EEG events from the EEGLAB `EEG.urevent` structure have entries in this table associated with a target dataset. This design decision allows fast extraction of events for an entity at the expense of space for duplication of similar events.

In order to facilitate tagging and association of events across multiple datasets, MobbedDB keeps a separate EVENT_TYPES table and uses the EVENT_TYPE_UUID in the EVENTS table rather than the string representing the event directly. All event types are strings. This representation emphasizes the discrete nature of events and also promotes identification of comparable events across multiple datasets and studies. An event type such as 101 is not suitable for data mining when taken beyond the context of a single experiment.

EVENT_TYPES [Key: EVENT_TYPE_UUID**]**

Table column	Meaning
EVENT_TYPE_UUID	UUID of the event type
EVENT_TYPE	String event type (e.g., 'ButtonPress')
EVENT_TYPE_DESCRIPTION	Description of this event type

EVENTS [Key: EVENT_UUID**]**

Table column	Meaning
EVENT_UUID	UUID of the event
EVENT_ENTITY_UUID	UUID of the entity associated with this event (usually the dataset)
EVENT_PARENT_UUID	UUID of the parent event or -1 if this was an original event
EVENT_TYPE_UUID	UUID of the event type
EVENT_POSITION	64-bit index of the event
EVENT_START_TIME	64-bit time stamp for the event start time
EVENT_END_TIME	64-bit time stamp for the event end time

4.1.4 The MODALITIES table

MobbedDB organizes information into datasets representing a single modality such as EEG. MobbedDB associates each modality with one or more standardized access layer structures. Currently, MobbedDB only supports a single modality: EEG data and a single standardized structure, the MATLAB EEG structure supported by EEGLAB. The MobbedDB infrastructure supports future extensions to different modalities and different standardized structures for given modalities. MobbedDB tries not to duplicate data storage, but may duplicate metadata to allow multiple forms of access layer extraction.

MODALITIES: [key: MODALITY_ UUID**]**

Table column	Meaning
MODALITY_ UUID	UUID identifying the modality
MODALITY_ NAME	name of the modality (e.g., EEG)
MODALITY_PLATFORM	access layer platform for extraction (e.g., EEGLAB)
MODALITY_DESCRIPTION	description of the modality and access layer

4.1.5 The TRANSFORMS table

Each MobbedDB dataset corresponds to data collected from a single experiment and single modality. MobbedDB can organize multiple datasets from the same experiment into a collection. During analysis these raw datasets may be transformed to produce new datasets. MobbedDB uses transform entities to store and reuse transformed data. MobbedDB transforms usually represent strings that the access layer can evaluate. For a MATLAB access layer, MobbedDB would apply the MATLAB `eval` function to evaluate such a transform. In BCILAB [11], for example, this string is a fully parenthesized expression containing the full transformation history starting with the original data. The TRANSFORM_MD5_HASH is the MD5 hash of the transform string. The evaluation of the transformation string should result in a single output dataset whose UUID is the TRANSFORM _UUID. A user can fetch the dataset resulting from this transformation by querying with a specific MD5 hash value rather than recalculating.

TRANSFORMS: [key: TRANSFORM _UUID**]**

Table column	Meaning
TRANSFORM _UUID	UUID identifying the dataset resulting from the transform
TRANSFORM_MD5_HASH	MD5 hash of the transform string
TRANSFORM_STRING	string specifying the transform (often executable command string)
TRANSFORM_DESCRIPTION	description of the transform

4.2 Data

MobbedDB allows a dataset to store its data in multiple ways. The simplest way is as a flat file retrievable from the server through an OID (object ID). This method of storage is efficient for retrieval and storage. MobbedDB also can store data directly in the database in one of four formats: a time-stamped array of double values, a time-stamped self-defining XML blob, an array of double values, or an XML blob. The representation choice involves a trade-off between search, extraction, and space efficiency. The researcher may choose not to store the data in the database at all but only use the flat file representation.

The DATA_DEFS table identifies a piece of data (an array, a blob, a stream, or a file). The description may include information about data provenance. Because a piece of data may be associated with many different entities and an entity may be associated with many different pieces of data, a separate DATA_MAPS table provides the many-to-many associations. Data streams associate multiple pieces of data with a single data definition, identifying the pieces within the stream by the respective RECORD_POSITION, the position or point number in the dataset. The actual time in seconds of the record is given by the respective RECORD_TIME.

DATA_DEFS [key: DATA_DEF_UUID**]**

Table column	Meaning
DATA_DEF_UUID	UUID identifying the data
DATA_DEF_FORMAT	currently NUMERIC_DATA, NUMERIC_STREAM, XML_DATA, XML_STREAM, or EXTERNAL
DATA_DEF_SAMPLING_RATE	sampling rate in Hz (null if not fixed sampling rate or not time series)
DATA_DEF_TIMESTAMPS	vector of doubles giving times in seconds from start (or null if fixed sampling rate or not a time series)
DATA_DEF_OID	OID of the data file stored in the system table if external
DATA_DEF_DESCRIPTION	description of what this element represents

DATA_MAPS

Table column	Meaning
DATA_DEF_ UUID	UUID of the data definition
DATA_MAP_ENTITY_UUID	UUID of the entity associated with data by this map entry
DATA_MAP_STRUCTURE_UUID	UUID of the structure for insertion and extraction (or null)

NUMERIC_DATA [key: DATA_DEF_UUID**]**

Table column	Meaning
DATA_DEF _UUID	UUID of data definition associated with this value
NUMERIC_DATA_VALUE	array of doubles or XML blob depending on table

NUMERIC_STREAMS [key: DATA_DEF_UUID, NUMERIC_STREAM_RECORD_POSITION**]**

Table column	Meaning
DATA_DEF_UUID	UUID of data definition associated with this value
NUMERIC_STREAM_RECORD_POSITION	64-bit long indicating position in stream (starting from 1)
NUMERIC_STREAM_RECORD_TIME	double value with time in seconds from start
NUMERIC_STREAM_DATA_VALUE	array of doubles or XML blob depending on table

XML_DATA [key: DATA_DEF_UUID**]**

Table column	Meaning
DATA_DEF _UUID	UUID of data definition associated with this value
XML_DATA_VALUE	array of doubles or XML blob depending on table

XML_STREAMS [key: DATA_DEF_UUID, XML_STREAM_RECORD_POSITION**]**

Table column	Meaning
DATA_DEF_UUID	UUID of data definition associated with this value
XML_STREAM_RECORD_POSITION	64-bit long indicating position in stream (starting from 1)
XML_STREAM_RECORD_TIME	Double value giving time in seconds of this record from start
XML_STREAM_DATA_VALUE	array of doubles or XML blob depending on table

4.3 Attributes and structure

Attributes, which do not have timestamps, may encapsulate entity metadata or other characteristics as well as time independent data. The semi-structured nature of MoBBED data makes mapping of attribute data into a relational database more challenging, since MobbedDB must recover the structures on retrieval. For example, EEG events (as encapsulated in the `EEG.urevent` and `EEG.event` structures) always have a type and a start time but may also have other properties. Researchers often use these additional properties to characterize the event or the experimental conditions under which the event was observed. MobbedDB stores each extra property as an attribute of the event and associates a structure with the attribute.

The ATTRIBUTES table associates string values with entities. An attribute entity has a target ATTRIBUTE_ENTITY_UUID indicating the specific entity associated with this property. The ATTRIBUTE_ORGANIZATIONAL_UUID should reflect the higher-level organization associated with this attribute value. For example, the ATTRIBUTE_ORGANIZATIONAL_UUID for attributes associated with events is the dataset UUID to allow more efficient search and retrieval. Other types of attributes may have a higher-level organizational UUID such as a collection. Attributes for entities such as person or equipment may have the same ATTRIBUTE_ENTITY_UUID and ATTRIBUTE_ORGANIZATIONAL_UUID.

ATTRIBUTES [Key: ATTRIBUTE_UUID**]**

Table column	Meaning
ATTRIBUTE_UUID	UUID of the attribute
ATTRIBUTE_ENTITY_UUID	UUID of the entity that has this attribute
ATTRIBUTE_ORGANIZATIONAL_UUID	UUID of dataset or other organizational entity
ATTRIBUTE_STRUCTURE_UUID	UUID of the structure for insertion and extraction
ATTRIBUTE_POSITION	position within the structure for insertion and extraction
ATTRIBUTE_VALUE	value of the attribute (a string that may be an XML blob)

The STRUCTURES table encapsulates the organization of the dataset for insertion into or extraction from a particular access layer. MobbedDB structures form a forest for categorization and reconstruction. Each row in the STRUCTURES table corresponds to a unique node in this forest identified by its STRUCTURE_UUID. The STRUCTURE_PARENT_UUID encapsulates the tree representation. For example, suppose that an EEG dataset had an event field `targetType` (e.g., the event structure has a field `EEG.event.targetType`). The STRUCTURES table would contain entries for both `event` and `targetType`, with the `event` entry having a null STRUCTURE_PARENT_UUID. The STRUCTURE_PARENT_UUID of the `targetType` entry would be the STRUCTURE_UUID of the `event` entry. Section 6 shows a more complete example.

STRUCTURES: [key: STRUCTURE_UUID**]**

Table column	Meaning
STRUCTURE_UUID	UUID identifying the structure field
STRUCTURE_NAME	name of the structure field (e.g., EEG or setname)
STRUCTURE_HANDLER	type of handler (REQUIRED, OPTIONAL, or MANUAL)
STRUCTURE_PARENT_UUID	UUID of parent field (-1 indicates the top level structure name)

MobbedDB creates all REQUIRED fields even if they have no associated attribute values, but it only creates OPTIONAL fields for existing attributes. MANUAL fields have specific handlers, but MobbedDB can process REQUIRED and OPTIONAL fields automatically.

4.4 Collections

A single experimental run may acquire EEG, eye tracking, and motion capture as well as environmental information. Each of the measurement modalities (e.g., EEG, eye tracking, and motion capture) would appear as an individual dataset, but the three datasets form a group representing the experimental run. MobbedDB organizes this group as a collection. Collections may contain any number of datasets and other entities**.** The COLLECTIONS table defines the associations between a collection and the entities they contain. Users should take care not to insert cycles into the map.

COLLECTIONS [Key: COLLECTION_UUID, COLLECTION_ENTITY_UUID**]**

Table column	Meaning
COLLECTION_UUID	UUID of the dataset representing the collection
COLLECTION_ENTITY_UUID	UUID of an entity in the collection

4.5 Tags

A tag is a word or flag that users can assign to any entity to facilitate searching and association. Effective searching requires users to reuse tags where possible, so rather than allowing arbitrary tag names, MobbedDB requires users to explicitly create tags prior to using them. The TAGS table lists the available tags.

TAGS [Key: TAG_ UUID]

Table column	Meaning
TAG_ UUID	UUID of the type
TAG_ NAME	descriptive name of the type

The TAG_MAP table encapsulates the many-to-many relationships between tags and entities.

TAG_MAP [Key: TAG_ UUID, TAG_ENTITY _ UUID**]**

Table column	Meaning
TAG_ UUID	UUID of the tag
TAG_ENTITY _ UUID	UUID of entity in this mapping
TAG_ENTITY_CLASS	class of entity (‘Dataset’, ‘Event’ etc.)

4.6 Miscellaneous entities

4.6.1 Comments

Comments are informational strings that users can associate with any entity. Comments serve to document the data and may contain references to more complex documents such as images, papers, code, and bibliographic references. Users may search comments.

Comment entities have a comment string for a specific entity along with the time and the UUID of the person who wrote it. This design allows users to add multiple comments for any entity.

COMMENTS [Key: COMMENT_UUID**]**

Table column	Meaning
COMMENT_UUID	UUID of the comment entity
COMMENT_ENTITY_UUID	UUID of the entity associated with this condition
COMMENT_ENTITY_CLASS	class of the entity (e.g., 'dataset', 'event')
COMMENT_CONTACT_UUID	UUID of the person entering the comments entity
COMMENT_TIME	64-bit time stamp for the time of entry
COMMENT_VALUE	string value of the comments entry

4.6.2 Contacts

A contact entity represents the contact information for individuals associated with the data including the owner of the data or the creator of database entries such as comments. The CONTACTS table promotes central updating of contact information without modifying the rest of the database. The table keeps the basic required contact information. Other information such as institutional affiliations, web sites, and alternate phone numbers are stored as attributes.

CONTACTS [Key: CONTACT_UUID**]**

Table column	Meaning
CONTACT_UUID	UUID of the person represented by this contact
CONTACT_FIRST_NAME	first name of this contact
CONTACT_LAST_NAME	last name of this contact
CONTACT_MIDDLE_INITIAL	middle initial
CONTACT_ADDRESS_LINE1	first line of postal address
CONTACT_ADDRESS_LINE2	second line of postal address
CONTACT_CITY	city
CONTACT_STATE	state
CONTACT_COUNTRY	country
CONTACT_POSTAL_CODE	postal code
CONTACT_TELEPHONE	contact phone number
CONTACT_EMAIL	primary email address

4.6.3 Subjects

The SUBJECTS table holds subject entities, which represent data sources such as instrumented humans, simulators, or robots. MobbedDB represents additional information about each subject through its attributes, comments, and tags.

SUBJECTS [Key: SUBJECT_UUID**]**

Table column	Meaning
SUBJECT_UUID	UUID of the subject
SUBJECT_DESCRIPTION	description of the subject

4.6.4 Devices

The DEVICES table holds entities that represent measuring devices such as a particular EEG cap. Researchers can identify the particular piece of equipment used to acquire a dataset and document its history and properties by assigning appropriate attributes and types.

DEVICES [Key: DEVICE_UUID**]**

Table column	Meaning
DEVICE_UUID	UUID of the piece of equipment or other device
DEVICE_CONTACT_UUID	UUID of the contact person for this device
DEVICE_DESCRIPTION	description of the equipment and its history

5. Use cases and MobbedDB /MATLAB interface

This section presents several use cases that describe basic access layer functionality for MobbedDB databases. Features supporting basic exploration and extraction are candidates for implementation in early releases. Functionality that is more sophisticated is possible as data mining approaches evolve. We describe the use cases in terms of an access layer for MATLAB.

Each case has both a programmatic implementation for MATLAB. Appendix A shows example MATLAB graphical user interfaces for some of these operations. This white paper focuses on the programmatic implementation. To support another analysis platform, such as R, an equivalent programmatic interface would be required. These use cases do not support transformations of data or the creation, representation and extraction of feature sets for data mining.

5.1 Browsing databases using external tools

The simplest use case is for tools that allow users to understand the structure and content of the database. PostgreSQL has an excellent tree view browser in the pgAdmin tool suite [10] for simple exploration. This tool displays the table structure and allows users to perform simple SQL queries. PostgreSQL also has a number of open source web interface tools such as phpPgAdmin [12] for remote exploration of the data base contents. Exploration is a useful prelude for users who want to insert their own data or extract data for analysis.

5.2 Creating and delete databases from MATLAB

To create a database from MATLAB, you must have first installed PostgreSQL on the machine that will host the database. Use the MATLAB `createdb` function to create a new database:

```
Mobbed.createdb(name, hostname, user, password, script)
```

The `name` argument specifies the name of the database. The `hostname` is a string specifying the location of the database, `user` is the database user name, and `password` is that user's password. The `script` is the name of an XML file containing statements to create the database. The following example creates a database on the local machine:

```
Mobbed.createdb('mobbed', 'localhost', 'postgres', 'admin', 'mobbed.xml')
```

The `mobbed.xml` script comes with the MobbedDB distribution. If unsuccessful, `createdb` throws an exception.

The `deletedb` command is similar to `createdb`, but used to drop a database.

```
Mobbed.deletedb(name, hostname, user, password)
```

5.3 Connecting to a database from MATLAB

To access databases from MATLAB, create a database connector by creating a `Mobbed` object:

```
DB = Mobbed(name, host, user, password)
```

This call uses parameters similar to `createdb` and assumes that the database and user already exist. If the connection attempt fails, the constructor throws an exception.

The DB return value is an open database descriptor or connector object to be used in future calls to the database. Users may simultaneously access different databases by creating separate connectors to each database. The database connector supports close and commit operations. Once closed, additional attempts to access the database using this connector fail unless a reopen operation occurs.

The following table lists the most important public methods of the Mobbed class:

Method name	Description
getdb	retrieve rows from a single table
putdb	create or update rows from a single table
db2mat	retrieve a dataset (e.g., eeg, eeg study, or mobi) from the database
mat2db	create a dataset in the database
commit	commit the current transaction, if uncommitted
close	close the database descriptor (further method calls cause an exception)
rollback	rollback the current transaction if any is uncommitted
setAutocommit	determines whether the database automatically commits each transaction

By default, MobbedDB database connections do not automatically commit each transaction. Closing the connection automatically commits the uncommitted transactions. If the commit operation fails, close attempts to rollback the operation prior to closing the database connection.

The mat2db method writes data structures such as the EEG structure from EEGLAB to the database. This write must modify several tables in a consistent manner. The mat2db method commits any uncommitted transactions. If the commit operation fails, mat2db calls the close method before throwing an exception. After writing the data structures by calling the libraries specified by type, mat2db commits the transaction, using the same procedure as above for error handling.

Ordinary users usually do not call putdb except to create independent entities in the database such as contacts, subjects, and equipment. Several GUIs use putdb to update single rows in the database.

5.4 Mapping tables to MATLAB structures for retrieval

MobbedDB has a standard mapping of database tables to MATLAB structures: each table column name has a corresponding structure field name. By convention, the database column names are upper case and the structure fields are lower case.

The getdb method returns MATLAB structures with entries from a table in the database specified by the open database connector DB:

```
mStructure = getdb(DB, table, limit, varargin)
```

The table argument specifies the name of the database table as a string (e.g., 'contacts'or 'contacts') and is case insensitive. The return mStructure is a structure or structure array whose fields correspond to the columns of the specified table. The limit argument specifies the maximum number of records to return. A limit value of inf specifies that all records should be returned, while a value of 0 specifies that an empty structure should be returned. The following call:

```
mStructure = getdb(DB, 'contacts', 0)
```

returns a structure with the following fields (all empty):

Field name	Description
contact_uuid	uuid (ignored on input and filled in on return)
contact_first_name	first name of the contact
contact_last_name	last name of the contact
contact_middle_initial	middle initial of the contact
contact_address_line1	first line of postal address
contact_address_line2	second line of postal address
contact_city	city
contact_state	state
contact_country	country
contact_postal_code	postal code
contact_telephone	telephone number
contact_email	email address

The following call returns a structure array with fields as in the table above:

```
mStructure = getdb(DB, 'contacts', inf)
```

Each field of the `mStructure` structure array corresponds to a column of the specified table in the database.

If `inMStructure` is a structure, `getdb` matches the fields of the structure with the corresponding column names of the table specified by `type`. The `getdb` method uses matched fields to qualify the retrieval of entries from the table. For example, suppose `mStructure` is a structure that only has a `last_name` field which contains a string `'Sm*'`. The following statement returns a structure array containing all of the contacts whose last name starts with the letter combination Sm:

```
mStructure = getdb(DB, 'contacts', inf, inMStructure)
```

The `varargin` arguments are name/value pairs specifying additional qualifications for the search. The following table shows the allowed qualifications:

Name	Value
'Tag'	string or cell array of string tags to search on
'Attribute'	string or cell array specifying attribute values to match

For example, the following call retrieves all datasets that have an attribute value `'Biosim'` and have a tag of `'VisualTarget'` or `'AudioTarget'` or tag starting with `'Odd'`. In addition to at least one of these three tags, the dataset must have a tag of `'EyeTracked'`.

```
mStructure = getdb(DB, 'dataset', 10, [], 'Attribute', 'Biosim', ...
                   'Tag', {'VisualTarget', 'AudioTarget', 'Odd*'}, ...
                   'Tag', {'EyeTracked'});
```

Statements including an `inMStructure` structure, tag specifications, and attribute specifications require that the returned items meet each specification separately. If an error occurs, `getdb` throws an exception. Note, that `getdb` does not treat an empty return value as an error.

5.5 Mapping tables to MATLAB structures for creation and update

The `putdb` method updates existing rows of a table in the database specified by the open database connector `DB`:

```
mStructure = putdb(DB, type, inMStructure)
```

The `type` argument specifies the name of the database table as a string (e.g., `'contacts'`) and is case insensitive. The `inMStructure` is a structure or array of structures whose field names correspond to column names of the corresponding database table. If the fields corresponding to the UUID table keys are not filled in or do not match an existing UUID key, `putdb` creates a new row and generates the appropriate UUID keys. If the fields corresponding to the UUID table keys are not empty and match an existing key, MobbedDB uses the other non-empty fields to update the row. The return `mStructure` is a structure contains an element for each updated row in the table. If an error occurs, `putdb` throws an exception.

5.6 Inserting a new dataset in a database

The `getdb` and `putdb` methods map MATLAB structures into MobbedDB tables for insertion, updates, and retrieval. Unfortunately datasets usually have information spread over multiple tables and would require multiple calls to these methods. MobbedDB provides two methods (`mat2db` and `db2mat`) specifically for handling datasets. The `mat2db` method inserts a dataset in the database specified by the open database descriptor `DB`:

```
dataset = mat2db(DB, dataset, fileOnly, isUnique, tags, eventUUIDs)
```

The `dataset` argument is a structure with fields shown in the following table. All of the fields except the field correspond to column names in the `DATASET` table. The `data` field contains a structure with the data or another type of reference to the data. For `EEG` modality, the `data` field is an EEGLAB `EEG` structure or a reference to a local file. MobbedDB ignores the `version` and `creation_date` fields during insertion.

Field name	Description
`dataset_uuid`	UUID of dataset (ignored on input/filled in on return)
`dataset_namespace`	name space for this dataset (e.g., a URL specifying the lab)
`dataset_name`	name of this dataset (often a filename-like string)
`dataset_version`	version number of this dataset
`dataset_contact_uuid`	UUID of the contact person for this dataset
`dataset_creation_date`	date string of insertion time (ignored on input/ filled in on return)
`dataset_description`	description of the dataset
`dataset_parent_uuid`	UUID of parent for derived datasets (for inheriting events)
`dataset_modality_uuid`	UUID of modality for this dataset
`data`	structure containing data or reference to the data

The last four arguments of `mat2db` are optional. The `fileOnly` argument is a logical value specifying how to save the dataset data values. If `fileOnly` is true (the default), then MobbedDB saves the data as a file but does not explode the data into database tables. If `fileOnly` is false, then MobbedDB saves the data both in the database and as a separate file.

The `isUnique` argument is a logical value specifying how to handle duplicate names. If `isUnique` is true (the default), only one version with the same name and namespace appears in the database. An attempt to write a dataset with a duplicate (`nameSpace, name`) results in an immediate return with error. If `isUnique` is false, MobbedDB creates an entry with a higher version number than those that exist in the database. Some applications require exactly one dataset associated with a given name, while others do not.

The `tags` argument is a cell array of strings specifying the tags to be associated with this dataset. The `eventUUIDs` argument is a cell array of strings containing the event value UUIDs of events associated with this dataset. If omitted, MobbedDB creates new entries in the `EVENT_TYPES` table for the unique events associated with this dataset. If present, MobbedDB reuses the event types corresponding to the listed UUIDs and creates new entries in `EVENT_TYPES` for those event types not accounted for. This mechanism allows users to associate event types across multiple datasets and studies. Users can then apply tags to the event types only once for all of the datasets and be guaranteed of consistent tagging for later data mining.

If successful, `mat2db` returns the input `dataset` argument with the `dataset_uuid`, `dataset_version`, and `creation_date` filled in and the transaction committed. If an error occurs, `mat2db` throws an exception after rolling back the transaction.

5.7 Retrieving a dataset from a database

The `db2mat` method extracts a dataset from the database specified by the open database connector `DB`:

```
dataset = db2mat(DB, UUID, reconstruct)
```

The `UUID` is a string or a cell array of strings specifying the UUIDs of the data units to be retrieved. The `reconstruct` argument is a logical value specifying how to retrieve the data. If `reconstruct` is true, MobbedDB attempts to reconstruct the data by extracting information from the database tables rather than just returning the externally stored data. The default value of `reconstruct` is false.

If successful, `db2mat` returns a structure array with the fields specified by the table of Section 5.5. If an error occurs, `db2mat` throws an exception. Section 5.5 also describes the commit strategy of `db2mat` in more detail.

5.8 Extracting by regular slice

The user should be able to extract a time slice of the dataset. For EEG data, this functionality would allow users to extract a subset of channels for a single time point or for a fixed interval of time. MobbedDB does not implement this feature directly at this time. Users would need to read in the data and apply extraction filters from there.

5.9 Extracting by time-locking to events

The user should be able to specify events of a particular type and a collection of datasets and extract a time slice of the data time-locked to these events. MobbedDB does not implement this feature directly at this time. Users can extract the desired events, but under the current implementation need to read in the datafiles and apply the extraction filter to the data.

6. Prototype implementation and performance

We implemented the MoBBED data infrastructure, MobbedDB, using MATLAB, Java, and PostgreSQL database as shown schematically in Fig. 4. MATLAB provides the access layer to user code, while the Java libraries handle the transactions between the access layer and MobbedDB. MobbedDB uses PostgreSQL because of its support of arrays and other advanced features. The figure below provides an overview of the infrastructure with all the major components.

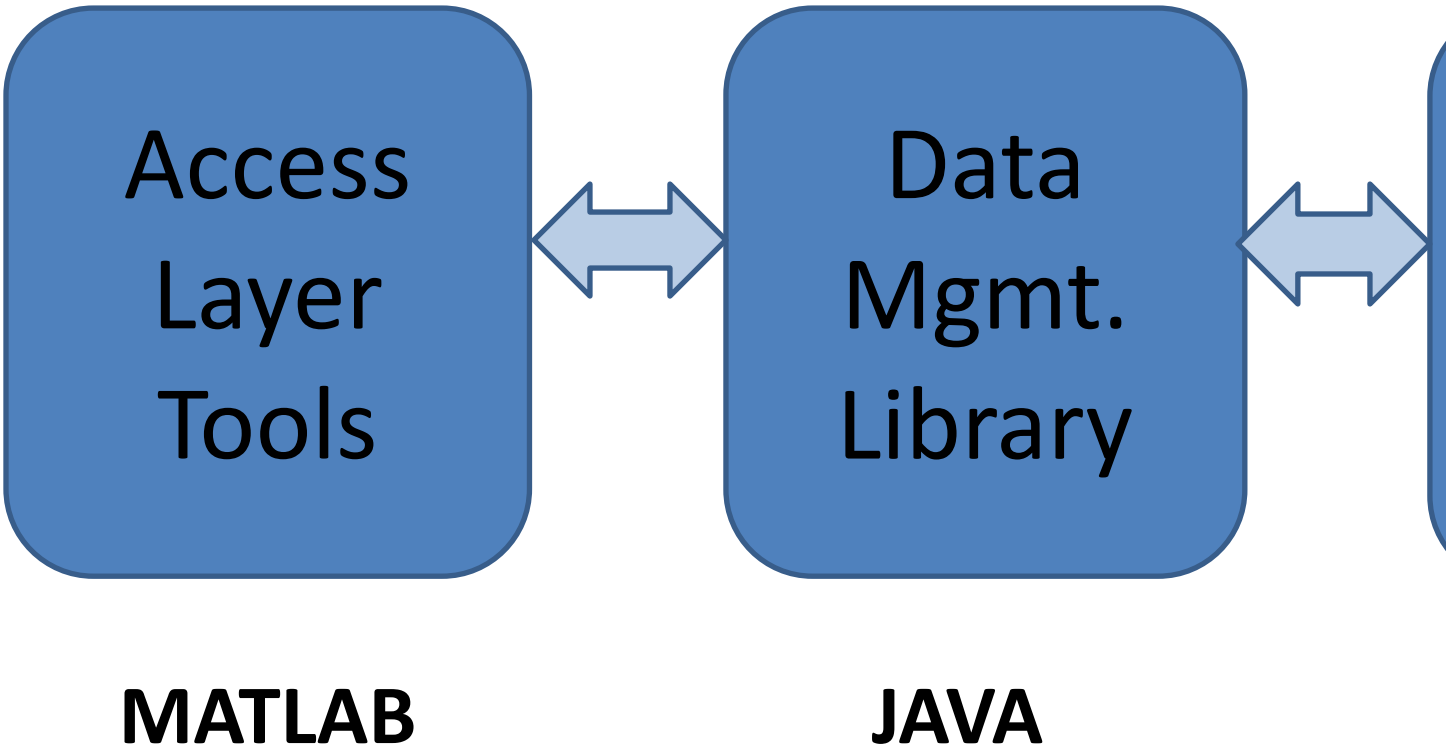

Fig. 4. Overview of MobbedDB structure for storage and retrieval from MATLAB.

Using the MATLAB scripts or the GUI described in Appendix A, researchers can store, retrieve, or query the database. Researchers can pick any number of datasets or experiments according to their requirements and load them into MATLAB environment from the database.

The steps for storing a typical EEGLAB `EEG` structure are:

1. Create the `PERSON` table entry for dataset contact person if that person's entry does not exist.
2. Create the `DATASET` table entry.
3. Create the `ELEMENT` table entries.
4. Create the `EVENT` entries and their associated `ATTRIBUTE` table entries.
5. Create additional structure types as needed for the other fields of the `EEG` structure.
6. Create the `ATTRIBUTE` table entries for the other required, optional, and manual fields of the EEG structure.
7. Write the data.

The tables in MobbedDB do not directly map to the EEGLAB `EEG` structure. For example, `EEG.event` is a structure array that must have `type` and `latency` fields, but may also have any number of other fields:

```
EEG.event(1).type          'rt'
EEG.event(1).latency       256
EEG.event(1).condition     1
EEG.event(1).urevent       145
```

The `latency` field represents the position of the event with respect to the data points in the file, and value of `type` is a string. In this example, the `condition` field represents additional metadata about the event. MobbedDB converts the value of associated with `condition` to a string and stores the value as an attribute. The `STRUCTURE_UIDD` corresponds to a node in the structure forest representing `event.condition` so that MobbedDB can recover the value's place in the structure on retrieval. The `urevent` field indicates that this is a derived event. MobbedDB uses the `PARENT_UUID` column of the `EVENT` table to encapsulate information about the origin of such a derived event. The following steps describe how these events are stored:

1. Create a `mobbed.Event` object, storing the latencies and types of all events in the structure.

2. Check whether the `STRUCTURE` table already contains nodes for the names of other fields of the event array. If yes, MobbedDB retrieves the unique identifiers of corresponding to these nodes. Otherwise, MobbedDB creates the nodes and returns the identifiers.

3. Add the events:

    3.1. For each event, add a record for the event in the `EVENT` table.

    3.2 For each additional field add a record for field in the `ATTRIBUTE` table

We use prepared queries and batching to improve the efficiency of these additions. The actual database operations are usually part of a larger transaction, which MobbedDB commits when all pieces complete successfully. Otherwise, MobbedDB rolls back the transaction and returns an error.

After MobbedDB saves the example event, the related tables appear as follows:

Structure		
UUID	**Name**	**Parent**
STR_1	EEG	
STR_2	event	STR_1
STR_3	type	STR_2
STR_4	latency	STR_2
STR_5	condition	STR_2
STR_6	urevent	STR_2

Event					
UUID	**Type**	**Start**	**End**	**Pos**	**Parent**
EV_1	'rt'	256	256	1	EV_1
EV_2	'rt	256	256	145	EV_2

Attribute			
UUID	**Structure**	**Value**	**Entity**
ATTR_1	STR_5	1	EV_1
ATTR_2	STR_6	1458	EV_1

The data samples are typically large arrays of double values. Simple insertion of data using batch queries is time-consuming, making the approach unusable for large datasets. We read and write the data in binary using a multi-threaded approach. A master thread opens a connection to the database and creates a thread to write the binary data as shown in Fig. 5.

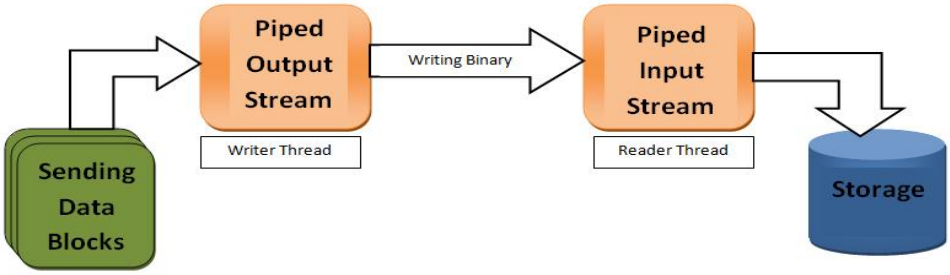

Fig. 5: Use of piped streams to send data blocks as binary.

MobbedDB also allows users to store large data objects directly as files. The file can be of any format (`.mat`, `.dat` etc.). PostgreSQL stores data files in a system file and returns an OID, which is stored in a `DATADEF` table along with other details.

Storing and retrieving files is faster than storing and retrieving binary data. However, binary insertion allows users to make queries on the data where storing files will not. Researchers may choose the combination of data storage methods depending on the application.

7. Performance of the prototype

We used a number of EEG datasets each with 36 channels, approximately 6K events, and 700K recorded data samples from each channel. The database and the application resided on the same machine with two cores and 4 GB of memory. The table below shows the storing and retrieval times in seconds. Since the application allows researchers to save signal data in two formats (i.e., numeric in a table or as file), we have shown execution times for both cases. Saving and retrieving using a data file is an order of magnitude faster than retrieving individual samples and assembling them.

	Store		Retrieve	
	FILE_DATA	**NUMERIC_DATA**	**FILE_DATA**	**NUMERIC_DATA**
channels	0.339	0.314	1.231	1.29
events	5.895	7.335	0.679	0.867
data	20.859	47.48	1.513	87.087
Total	27.478	55.137	3.434	89.264

8. Future implementation path

The phase I implementation focuses on the EEGLAB `EEG` structure for raw or unprocessed data. MobbedDB can read and write these structures to promote basic searching and analysis. Phase II will focus on encapsulating transformations, particularly retrieval of overlaid epochs, and retrieval by conditions to promote data mining.

Acknowledgments

We wish to acknowledge useful discussions with Scott Makeig, Nima Bigdely Shamlo, Christian Kothe, Alejandro Ojeda, Kaleb McDowell, Scott Kerick, and Andre Vankov. This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-10-2-0022. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] A. Vankov, DataRiver XML Technical Specification Draft.

[2] S. Makeig, SCCN, personal communication.

[3] N. Bigdely-Shamlo, C. Kothe, and A. Ojeda, SCCN, personal communication.

[4] Hadoop description: http://en.wikipedia.org/wiki/Apache_Hadoop.

[5] HeadIT project: http://sccn.ucsd.edu/wiki/HeadIT.

[6] The IEEG portal: http://braintrust.seas.upenn.edu/.

[7] Z. Ives, T. Green, G. Karvounarakis, N. Taylor, V. Tannen, P. Talukdar, M. Jacob, and F. Pereira. 2008. The ORCHESTRA Collaborative Data Sharing System. *SIGMOD Rec.* 37, 3 (September 2008), 26-32. DOI=10.1145/1462571.1462577 http://doi.acm.org/10.1145/1462571.1462577.

[8] MoBI lab homepage: http://sccn.ucsd.edu/wiki/MoBI_Lab.

[9] PostgreSQL homepage: http://www.postgresql.org/.

[10] pgAdmin PostgreSQL administration and management tool homepage: http://www.pgadmin.org/development/.

[11] A. Delorme, T. Mullen, C. Kothe, Z. Akalin, N. Bigdely-Shamlo, A. Vankov and S. Makeig (2011). EEGLAB, SIFT, NFT, BCILAB, and ERICA: New Tools for Advanced EEG Processing, Computational Intelligence and Neuroscience, Article ID 130714.

[12] phpPgAdmin homepage: http://phppgadmin.sourceforge.net/doku.php?id=start.

Update notes:
4/15/2012: Database table column names were changed to be unique and a few other minor renaming occurred.
5/3/2012: Additional table column names were changed to fix additional inconsistencies.
Added the `createdb` and `deletedb` methods to Mobbed, cleaned up various inconsistencies, modified the column names in the `TRANSFORMS` table, and fixed naming inconsistency between MoBBED and MobbedDB. Added the `EVENT_TYPES` table to handle consistent tagging of events.

Appendix A: A MATLAB MoBBED GUI

MoBBED provides GUI implementations for basic browsing, storage, and retrieval in MATLAB. Fig. A.1 shows a screenshot of `MoBBEDBrowse`, a basic interface for accessing and browsing a MoBBED database. Users may search for particular datasets or other entities by name or by tag after entering a database URL and password. Following a successful database connection, the user will not need to re-enter in this information. Users have the option of downloading or uploading files in addition to searching.

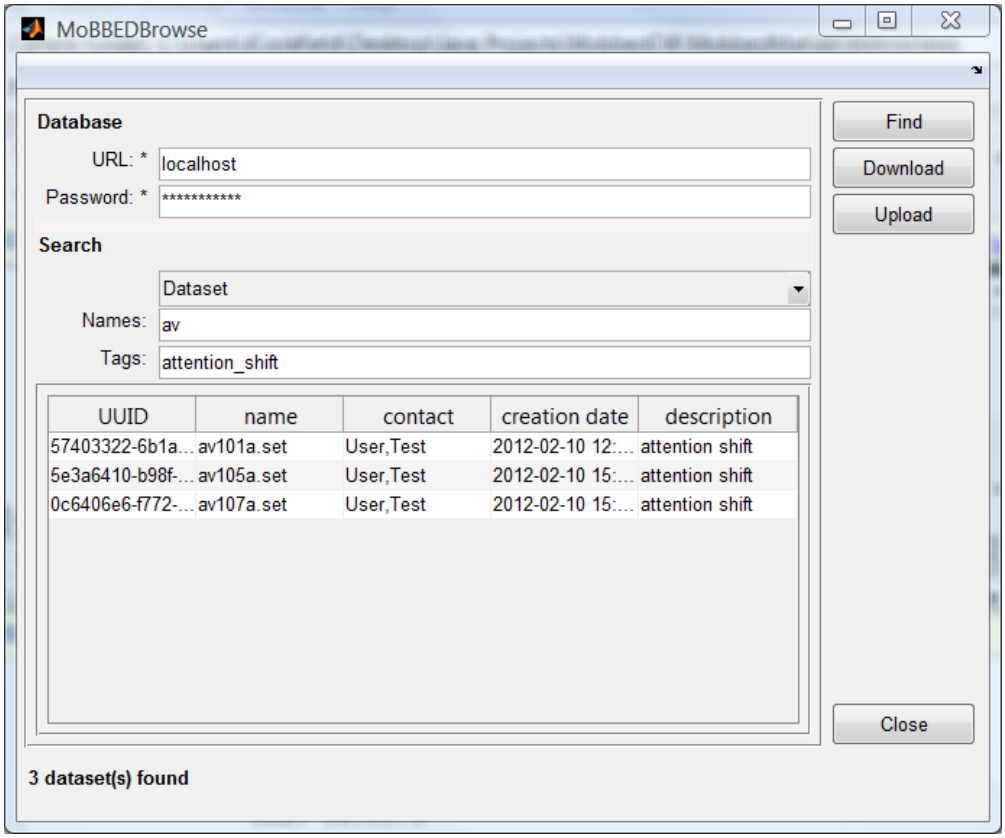

Fig. A.1: GUI form for accessing MoBBED databases in MATLAB.

When the user chooses to upload a dataset by pressing the `Upload` button, `MoBBEDBrowse` displays the form shown in Fig. A.2. The `Browse` button shows a file chooser for selecting the `.set` file containing the `EEG` structure to upload. The name for the dataset does not have to match the file name. The tags are comma separated words or short phrases used for searching. Currently the GUI does not have a history or tag suggestion mechanism, so slightly different tag wordings result in different entries in the `TAG_TYPE` table.

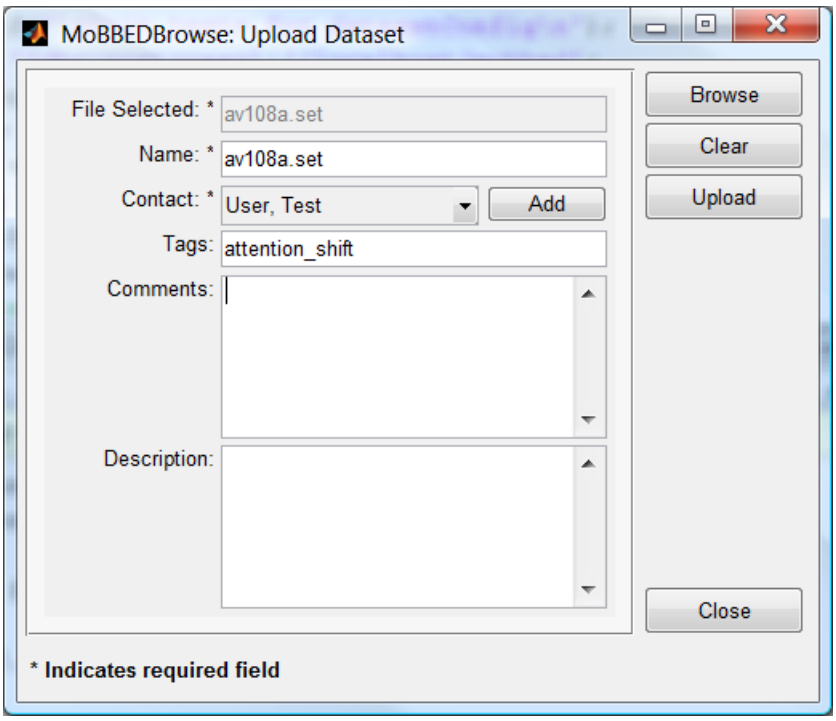

Fig. A.2: GUI form for uploading files to a MoBBED database in MATLAB.

A prerequisite for uploading a primary dataset to the database is that a contact must exist and be associated with the dataset. `MoBBEDBrowse` makes it convenient to add a new contact by popping up the form shown in Fig. A.3 when it needs a contact. Users can add one or more contacts without leaving this window. The form displays an asterisk in front of required fields. After completing the form, the user hits the Apply button to save the entered contact information. MoBBED does not add duplicate contacts to the database. The users may modify existing contacts or create new ones. The `Reset` button clears the form.

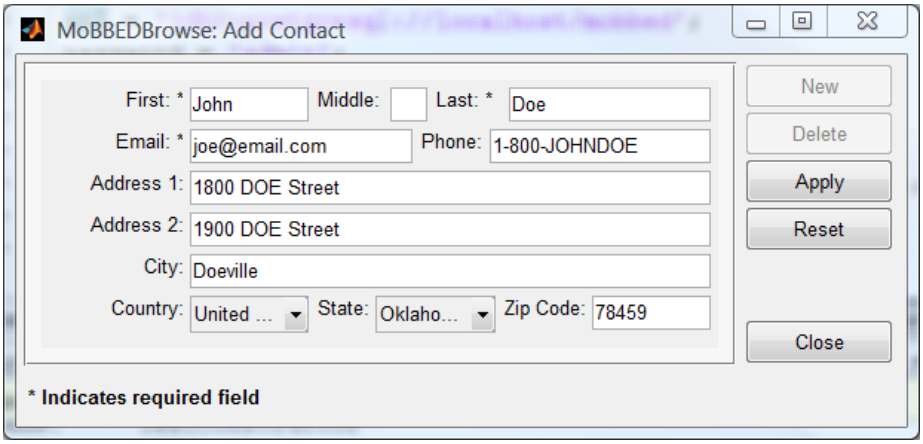

Fig. A.3: GUI form for adding a contact to a MoBBED database in MATLAB.