Motion Data Editor software architecture oriented on efficient and general purpose data analysis

Marek Kulbacki, Mateusz Janiak, Wojciech Knieć

Polish-Japanese Institute of Information Technology,
Koszykowa 86, 02-008 Warszawa, Poland
{kulbacki,mjaniak,wkniec}@pjwstk.edu.pl
http://www.pjwstk.edu.pl

Abstract. We present the architecture of the Motion Data Editor (MDE) real-time development framework for multi-modal motion data management, visualization and analysis. There is an emerging need for such tools due to the capability of recording and computing large number of motion modalities with high precision, synchronized in time domain. Such tools should be efficient, easy to use and flexible to apply to various data types and algorithms. Proposed MDE is dedicated to general data processing, and supports most of the common functionalities during data analysis. We discuss the most important functional requirements and present selected elements of system architecture: core data types, functionality and processing logic elements. The MDE with Human Motion Laboratory (HML) and cloud based Human Motion Database (BDR) [3] constitute collaborative environment for acquisition and analysis of multi-modal synchronized motion data for medical research and entertainment.

Keywords: software architecture, c++ , generic programming, variant type, data flow, continues integration, plug-ins, data analysis, data processing

Introduction

There are many solutions dedicated to general data processing, but most of them are either too specialised and limited for particular applications (biomechanics, medical imaging), or general enough, but offering poor efficiency. This forces the development of dedicated applications that fit particular research projects needs, where very often similar functionality is shared among different applications and only data types with algorithms are different. This leads to lack of compatibility between many tools, error propagation during data processing and other disadvantages. We propose an universal solution, offering flexibility in managed data types with efficiency in data processing, standardizing most common operations and functionalities present during data analysis.

Data processing procedure

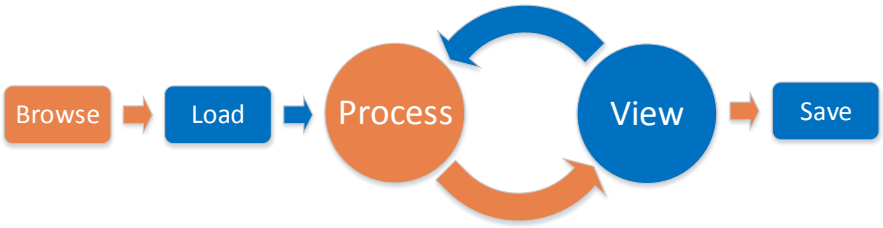

Fig. 1. Generalized data processing pipeline

Figure 1 presents generalized data processing pipeline. We distinguished five independent steps:

1. **Browse** – search for data available for analysis and processing;
2. **Load** – data extracting by allocating required resources and normalize data;
3. **Process** – operating on data with various algorithms;
4. **View** – data observing at different perspectives, making decisions about further analysis and processing steps;
5. **Save** – storing results for further analysis, potentially share them with other users.

Such pipeline, or its parts, can be found in almost any kind of data processing oriented software, although depending on the application purpose and internal realisation, those steps can differ significantly. Most of underlying functionalities are shared across applications, encapsulated with different user interface (UI). Providing general architecture, dedicated to such processing pipeline should allow to apply it for any kind of data and algorithms, reusing those common functionalities. This allows to limit costs and time, focusing on developing and testing essential analysis tools instead of creating once again similar software or looking for new tools for particular application and learn them from scratch for this single case.

Requirements for multi-modal data processing software

Before we present the architecture and functionalities of MDE software, we want to introduce basic requirements, that tools oriented on data processing should provide. Table 1 presents the most important features of such applications, guaranteeing standardization for performed experiments and developed analysis algorithms. This should lead to easier knowledge exchange between team members, allow to reuse already developed algorithms in other approaches increasing overall productivity.

Table 1. Data processing software functional requirements

Requirement	Description
Support for any data type	Various applications require using specific data types, users should be free in defining and using their custom data types
Unified time management for time indexed data	Shifting, scaling, splitting or merging in time domain should be provided ensuring their efficiency
Standardized and efficient data management	Loading large data sets requires careful memory management and fast data access
Generalized data loading and normalization	Data from various sources (containers) must be extracted and converted before analysis
Support data viewing	Data can be viewed at different perspectives nearby other data types and instances
Expandable with user solutions	Software must be flexible for new, user specific functionalities
Utilization of available computing resources	Efficient data processing should use all available computing resources to limit time required for various tests and experiments
Simple data processing pipelines composition	Tool should provide easy mechanism for creating complex processing pipelines and allow to reuse them in the future or store as an independent processing components

Motion Data Editor

Motion Data Editor (MDE) is a software developed at Polish-Japanese Institute of Information Technology (PJWSTK) in Bytom (Poland). Originally it was designed to support clinicans in viewing medical data for diagnosis of various human movement disorders. It has been re-organised and re-factored to become a general purpose data processing software.

Architecture Architecture of MDE is designed to support data processing pipeline presented in Figure 1. Additionally, proposed functionality and logic realise most common operations during data analysis. Figure 2 presents an overview of MDE architecture.

Introduced data types were proposed to create an abstraction layer for uniform data management for strongly typed C++ programming language [5], standardize time index data and system specific operations.

Data types Among various data types especially two should be described in details:

- ObjectWrapper,
- DataChannel.

ObjectWrapper type is a generalised approach to variant type in C++.
DataChannel is a generic approach to data indexed with time.

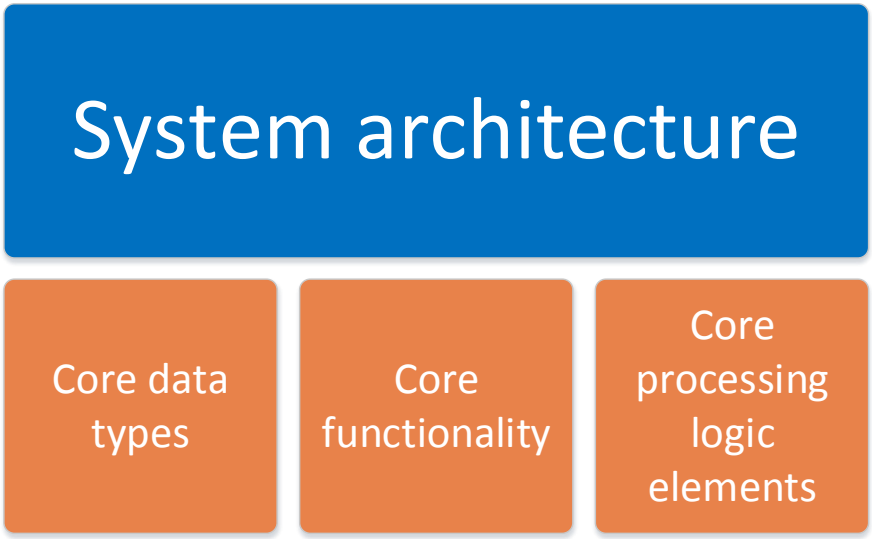

Fig. 2. MDE architecture

ObjectWrapper To provide uniform data management in strongly typed C++ programming language a dedicated *ObjectWrapper* type has been proposed. It is based on generic programming [18] and *RTTI* mechanism (*typeid* and *type_info*) offering runtime type information about encapsulate data with its hierarchy. To make *ObjectWrapper* applicable for any valid data type in C++ it is based on policies [1,11,10], allowing to customize pointer type storing the data, cloning functionality and for the inherited types their hierarchy information. It simplifies querying data about particular types. Additionally *ObjectWrapper* offers the functionality of lazy initialization, acquiring resources for encapsulated data on data extraction and meta-data information. The whole application architecture is based on *ObjectWrapper* functionality.

DataChannel Dedicated type was designed for uniform time indexed data management. *DataChannel* provides functionalities for treating discrete time data as continuous in time domain by introduction of specialized interpolation methods. It also allows to extend data with time property, saving memory when data is used in different time contexts. Data access according to time is optimized for channels with equally spread time samples. Moreover, when access to data for time values outside of the *DataChannel* is required, it is possible to use various built-in extrapolation techniques:

– **exception**. An exception is thrown on querying time index values outside of the *DataChannel* time range;

– **border copy**. First or last values are returned, depending on queried time index value;
– **periodic**. Queried time index value is truncated to mimic data periodic behaviour.

Functionalities

MDE provides many built-in functionalities supporting data processing and analysis procedure. They offer out-of-the box solutions for common operations, usually implemented from scratch in each dedicated software fit for particular need. Among many features, several are worth to be mentioned briefly:

– threads management with thread pool concept [9];
– optimal computing resources utilization with job manager approach [19];
– system status logging through dedicated hierarchical log mechanism;
– standardized file system operations;
– dedicated plug-in system [6];
– standardized UI [2].

All those components introduce an universal abstraction level for operating specific operations, making MDE software a cross-platform tool.

Processing logic elements

Based on presented data types and functionalities the architecture of MDE was decomposed to independent elements supporting particular stages of discussed data processing pipeline. The main goal was to provide a flexible system, easily expandable with users‘ custom functionalities and data types, handling any kind of data source in an uniform manner. Figure 3 presents five processing logic elements: Parser, Visualizer, DataSource, Service and Plug-in.

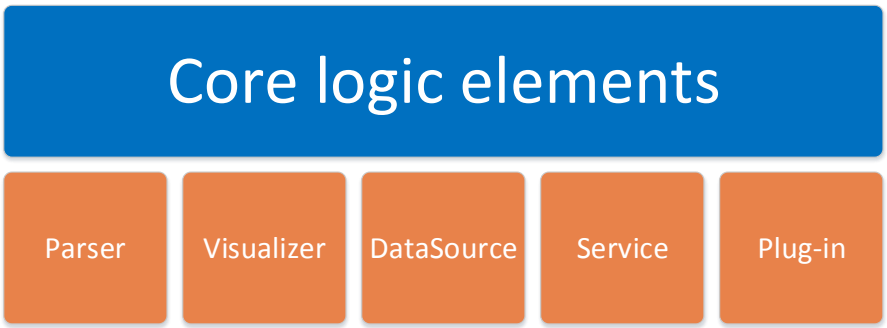

Fig. 3. Processing logic elements

DataSource To allow users to load data from various sources a dedicated element *DataSource* was introduced. It is responsible for connecting to the source and browsing the content of the source with its specific perspective to present user data for potential analysis. Once user chooses the data of interest, *DataSource* delivers the data in its specific container, unpack the container, finally load the converted and normalised data to application. As most of the data are stored in form of files or provided with some kind of streams, MDE offers a dedicated mechanisms for automated handling such types of containers. They are based on functionalities of *Parsers*.

Parsers *Parser* objects are propose to extract particular data types from a given file format or stream. Sometimes only a portion of data stored in a container is required, therefore it might be more efficient to skip other data knowing the container format structure and properties. As an example one can point out a video file, where audio and image can be used independently, considering speech analysis and image processing. Therefore proposing two *Parsers* for video file format seems to be reasonable, giving user greater flexibility in choosing data of interest from containers. With *Parsers* user can extend MDE to handle new data containers.

Visualizers Various data types can be observed from many perspectives. To simplify procedure of viewing the data and standardize it, *Visualizer* objects are proposed. Their main task is to manage presented data in form of data series, where user can add and remove more data to scene of the *Visualizer* to compare them visually, depending on *Visualizer* capabilities. The same data can be viewed in many *Visualizers* showing its various perspectives (i.e. 2D plot and 3D scene). With help of *Visualizers* user can introduce new data perspectives for any kind of handled data types in MDE.

Service Although MDE provides many useful functionalities for the most common operations in general data processing pipeline, it may happen that in particular specific cases some additional and specialised tasks are repeated in this process. To allow users application extension with completely new functionalities, *Service* object service was proposed. This is a very general concept of an element capable to handle any kind of new functions in MDE, having access to almost all application resources and having highest privileges among already presented logic elements.

Plug-in *Plug-in* system simplifies and standardize extending application with new data types and processing logic elements. It is responsible to initialize properly the environment for each loaded component, embed it to application logic and control its general behaviour. Additionally, *Plug-in* system verifies compatibility of the *Plug-in* itself with application - if their interfaces match and they were built with the same external libraries versions.

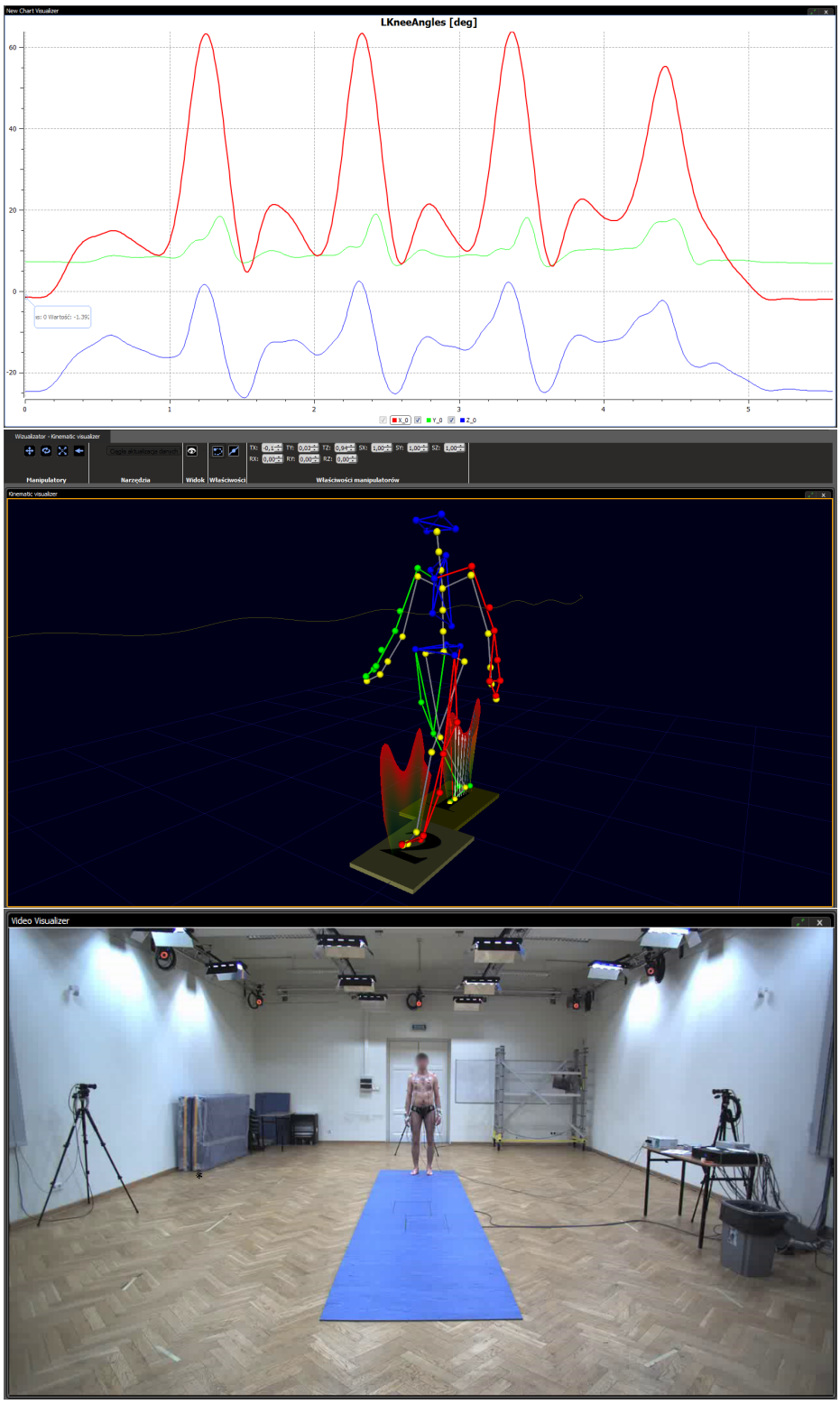

Fig. 4. Types of visualizers from MDE (from top to bottom): 2D Visualizer with motion trajectory, 3D Visualizer with kinematic and GRF data, Video Visualizer with front camera view

Centralised file data storage

Considering team work and data sharing among team members a dedicated service was developed. It is based on a dedicated data base, file transfer protocol (FTP) and web services, introducing a centralised, efficient, well organised, secure and flexible storage for various data stored in different files formats. Data can be organised in more abstract structures, extended with some additional meta data and properties. Access to data can be configured individually for each user or all groups of users. Technical details of assumptions, used data structures and database have been outlined in [3]. For MDE exists a dedicated plug-in with a data source supporting this technology. It allows browsing the data, downloading it, unpacking and loading to application for analysis.

Visual programming - data flow processing framework

Visual programming concept was developed to simplify process of software creation by manipulating graphically logic blocks, instead of writing their equivalent code.

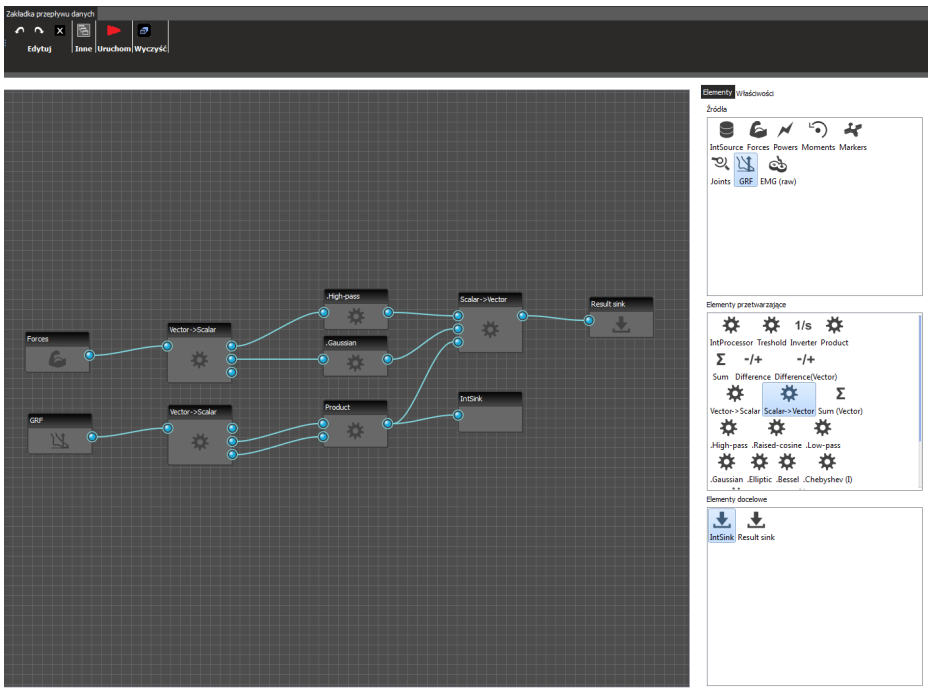

Fig. 5. An example of Visual Data Flow enviroment for MDE

As the final element of our system architecture we decided to provide users with similar visual programming environment, despite processing model and logic, to simplify and speed up process of data flow [4,7,8] creation. It allows users to dynamically create new data flows without need of writing new fragments of code and compilation procedure, based on delivered nodes functionality (see in fig.5). They can now define and launch data flows without any knowledge about programming. Visual data flow environment supports user in creating data flow by presenting graphically available nodes. It guides users, how particular nodes can be connected according to basic model rules and data types compatibility. Required and dependent pins are marked graphically with different styles to point out user places in the model, where connections are still required to make model complete. In the end any model verification failures are also presented to the user, with detailed description of elements and actions leading to fix those problems.

Conclusion

We see a great potential in presented modular system especially visual programming for multi-modal spatio-temporal data processing, therefore many new ideas were introduced for its possible applications. MDE is used with success for such research areas like presented in [15,16,17]. Additionally, we see a great potential in applying MDE for work presented in [12,13,14]. Two major solutions cover topics of utilizing GPU computational power and scheduling and distributing work in clusters environment. Because of modular structure, and good separation from external libraries, we are going to fast migrate our system into cloud based scalable environment for multi-modal big data processing, with clients based on mobile solutions.

Acknowledgments. Thanks to Piotr Gwiazdowski, Mariusz Szynalik, Marek Daniluk, Michał Szafarski, Michał Sachajko, Norbert Bastek, Krzysztof Kaczmarski, Piotr Habela, Wiktor Filipowicz and many others for help on this work. This work was supported by projects NN 518289240 and NN 516475740 from the Polish National Science Centre.

References

1. Alexandrescu, A.: Modern C++ Design: Generic Programming and Design Patterns Applied. Addison-Wesley Professional, 1 edn. (2 2001)
2. Blanchette, J., Summerfield, M.: C++ GUI Programming with Qt 4 (2nd Edition) (Prentice Hall Open Source Software Development Series). Prentice Hall, 2 edn. (2 2008)
3. Filipowicz, W., Habela, P., Kaczmarski, K., Kulbacki, M.: A generic approach to design and querying of multi-purpose human motion database. In: ICCVG (1). pp. 105–113 (2010)

4. Hennessy, J.L., Patterson, D.A.: Computer architecture: a quantitative approach. Elsevier (2012)
5. Lippman, S., Lajoie, J., Moo, B.: C++ Primer. Addison-Wesley Publishing Company (2012)
6. Reddy, M.: API Design for C++. Morgan Kaufmann, 1 edn. (2 2011)
7. Shen, J.P., Lipasti, M.H.: Modern processor design: fundamentals of superscalar processors, vol. 2. McGraw-Hill Higher Education (2005)
8. Smith, J.E., Pleszkun, A.R.: Implementing precise interrupts in pipelined processors. Computers, IEEE Transactions on 37(5), 562–573 (1988)
9. Smith, J.M.: Elemental Design Patterns. Addison-Wesley Professional, 1 edn. (4 2012)
10. Sutter, H.: Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions. Addison-Wesley Professional (11 1999)
11. Sutter, H., Alexandrescu, A.: C++ Coding Standards: 101 Rules, Guidelines, and Best Practices. Addison-Wesley Professional, 1 edn. (11 2004)
12. Szczesna, A.: The multiresolution analysis of triangle surface meshes with lifting scheme. Computer Vision/Computer Graphics Collaboration Techniques pp. 274–282 (2007)
13. Szczesna, A.: Designing lifting scheme for second generation wavelet-based multiresolution processing of irregular surface meshes. Proceedings of Computer Graphics and Visualization (2008)
14. Szczesna, A.: The lifting scheme for multiresolution wavelet-based transformation of surface meshes with additional attributes. Computer Vision and Graphics pp. 487–495 (2009)
15. Szczesna, A., Słupik, J., Janiak, M.: Motion data denoising based on the quaternion lifting scheme multiresolution transform. Machine graphics & vision 20(3), 238–249 (2011)
16. Szczesna, A., Słupik, J., Janiak, M.: Quaternion lifting scheme for multi-resolution wavelet-based motion analysis. In: ICONS 2012, The Seventh International Conference on Systems. pp. 223–228 (2012)
17. Szczesna, A., Słupik, J., Janiak, M.: The smooth quaternion lifting scheme transform for multi-resolution motion analysis. In: Proceedings of the 2012 international conference on Computer Vision and Graphics. pp. 657–668. ICCVG'12, Springer-Verlag, Berlin, Heidelberg (2012)
18. Vandevoorde, D., Josuttis, N.M.: C++ Templates: The Complete Guide. Addison-Wesley Professional, 1 edn. (11 2002)
19. Williams, A.: C++ Concurrency in Action: Practical Multithreading. Manning Publications, 1 edn. (2 2012)