Scientific Visualization, 2024, volume 16, number 4, pages 11 - 24, DOI: 10.26583/sv.16.4.02

A High-Level Adaptation Toolkit for Unified Integration of Software Systems with Neural Interfaces

S.I. Chuprina[1,A], I.A. Labutin[2,B]

[A] Perm State Humanitarian Pedagogical University, Perm, Russia
[B] Perm State University, Perm, Russia

[1] ORCID: 0000-0002-2103-3771, chuprinas@inbox.ru
[2] ORCID: 0000-0001-6858-1479, labutin.ia@psu.ru

Abstract

The paper is devoted to the urgent problem of automating the process of integrating different types of neurointerfaces into the infrastructure of the Internet of Things. Due to the low-level nature of these devices and related tools, integrating neurointerfaces with a wide range of IoT devices is a complex task requiring specialized knowledge and skills in the fields of neuroscience and signal processing. To tackle the significant challenge of automating such kind integration we continue to improve our previously proposed ontology-driven methods and tools for seamless integration of software systems with neural interfaces in a unified way. The presented notable improvements include the introduction of a new data processing pipeline that utilizes a portable device to validate the effectiveness of the system and the development of an intuitive graphical user interface enabling real-time data visualization facilities that provides an easy and understandable feedback.

Keywords: neurointerface, brain-computer interface, Internet of Things, ontology engineering, high-level integration, real-time data visualization.

1. Introduction

An exponentially growing interest in developing tools and methods for controlling software systems via neurocomputer interfaces have been fueled in recent times due to the rapid advancement of digitization and the exponential growth of applications in the Internet of Things (IoT) as well as augmented and virtual reality. However, a major challenge remains: there is currently no universal standard for integrating different IoT ecosystems with neurocomputer interfaces. The diversity of industrial standards and communication protocols governing device interactions within IoT, combined with frequent instances of incompatibility, complicates the integration of various IoT devices into a cohesive system. This highlights an urgent need for new unified concepts that can improve device interoperability, particularly when incorporating neurocomputer interfaces into existing software ecosystems.

A review of the literature [1, 2, 3, 4] reveals a consensus regarding the insufficient investigation of the challenges associated with creating unified methods and tools for the automatic integration of neurocomputer interfaces into IoT environments, especially concerning the control mechanisms for their components (target systems and subsystems). Generally, BCI devices are tailored to operate effectively only within a limited range of scenarios dictated by proprietary software from their manufacturers. As a result, applying BCIs in contexts or workflows not supported by the original software presents considerable hurdles, which require substantial expertise in both computer science and neuroscience.

The primary aim of this work is to build on our previous studies [5, 6, 7, 8, 9, 10, 11] and to advance our framework's adaptable facilities expanding them to IoT ecosystems which include the mobile devices and new types of neurointerfaces. To do this in a unified way the

central concept involves creating a "smart mediator" is used. Smart mediator serves as an embedding new node within the existing IoT infrastructure, providing the connection between different IoT frameworks and neurocomputer interfaces adaptably. The firmware for this smart mediator is generated based on a collection of ontological descriptions pertaining to the IoT ecosystem, BCI, and data transformation pipeline. A formal model and implementation details were outlined in our earlier work [11], and this paper aims to highlight recent advancements that offer comprehensive improvements, new extensions and facilities.

It's important to note that this paper does not attempt to provide a complete overview of the current state of brain-computer interfaces; interested readers are encouraged to refer to a well-written summary paper [12] for more detailed information.

2. Formal model of the system

Following [11] let's recite the following formal model of the system described in set-theoretic notation:

$$\Xi = \langle \Omega, \Delta, O_b, O_F, O_i, \Gamma, M, E, \Lambda, O_L, S \rangle, \quad (1)$$

where

Ω is the operator for generating source code for mediator, $\Omega : O_D \rightarrow S$;

Λ is the operator for building an ontology of module connections, $\Lambda : O_b \times O_F \times O_i \times D \rightarrow O_L$;

O_b is the ontological description of a specific neurointerface;

O_D is the ontological description of the domain area;

O_L is an ontological description of the links between modules included in the mediator's firmware;

$O_F = i = 1kO_{fi}$ – the set of k ontological descriptions of modules that implement data transformation;

O_i is an ontological description of the infrastructure, management of which is assumed to be implemented (module for controlling this infrastructure);

Δ is the operator for building an ontology from parts, $\Delta : O_b \times O_F \times O_i \times O_L \rightarrow O_D$;

Γ is an interaction operator, $\Gamma : E \times M \rightarrow D$;

E is a set of supported user actions;

M is a set of supported control elements;

D is a user-created formal description of the interconnection of modules in the form of DFD diagrams in SciVi toolbox;

S is the source code for the mediator's software.

We shall mention that operators Γ, M and E were created by our colleagues within the framework of the SciVi platform [13, 14, 15, 16] and are out of the scope of this work.

3. Modules as SciVi operators

System prototype designed by us earlier [11] was implemented as a command-line application consisting of few Python modules tied together with a Shell script. Although it was more that suitable for demonstrating a proof of concept, this implementation had several drawbacks:

1. **Lack of User-Friendliness**: While a command-line interface might be very powerful and efficient when used by a seasoned IT professional, it poses a significant wall for users that aren't very well-versed in computer science. For such users, a graphical user interface might be much simpler and more intuitive to operate.

2. **Semantic gap**: A few operations (selecting required nodes, connecting and configuring them) were performed in SciVi interface, but then the user had to switch into a terminal or a console emulator to perform a rest of the work. This disrupts a user experience and requires the user to master both parts of the interaction.

3. **Limited Scope for Improvement and Iteration**: This solution wasn't very good for further improvement. Due to the structure and design of the prototype, it posed challenges in

terms of enhancing its features, addressing issues, and incorporating feedback for iterative development.

4. **Tight Coupling**: Modules of the system was combined in a single way and was unable to be reused in other situations without writing a corresponding code for the task.

To address these challenges effectively, we decided on a strategic approach: designing a comprehensive set of SciVi operators that encapsulate the system's functionality. By implementing all operations within the SciVi framework, users will be presented with the ability to interact consistently through a unified interface, streamlining the creation and management of data processing pipelines. This approach not only enhances user experience by maintaining continuity in interaction methods but also promotes modular reuse of system components. Modules developed under this paradigm can be easily adapted and integrated into various contexts without requiring extensive reconfiguration or redevelopment, thus facilitating smoother iteration and improvement cycles for the system as a whole.

To cover all functionality of the system, we made a few new SciVi operators:

1. *Pipeline Loader* – this is a utility operator that fetches ontological descriptions from the SciVi storage.
2. *Ontology Merger* – this is an implementation of the Δ operator from (1).
3. *DFD2Link* – this is an implementation of the Λ operator from (1).
4. *Generator* – this is an implementation of the Ω operator from (1).

„Pipeline Loader" operator (fig. 1) is responsible for analyzing the firmware's data pipeline created in SciVi by the user and fetching all required ontological resources from the SciVi repository for their latter use in the generator's pipeline. All of presented in the paper ontologies are created in ONTOLIS environment – visual ontology editor (developed in PSU) [10, 15, 16].

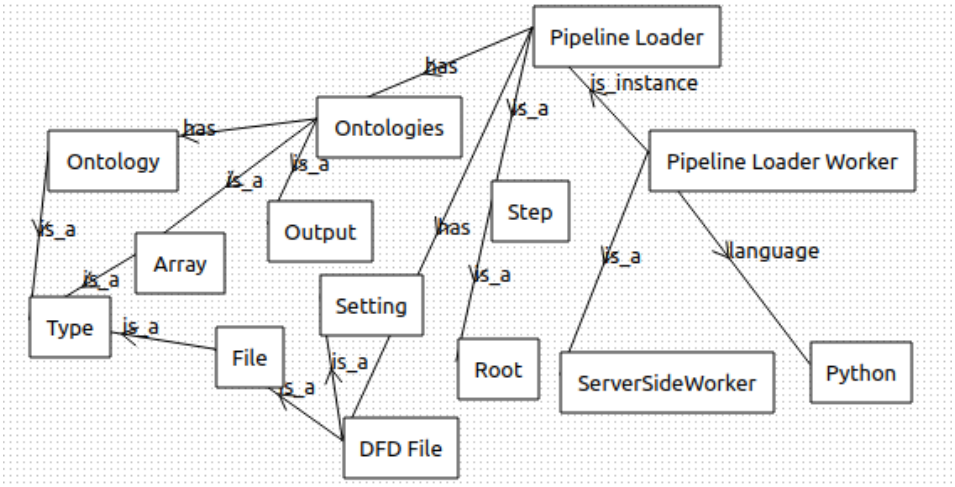

Fig. 1. „Pipeline Loader" operator

„Ontology Merger" operator (fig. 2) accepts a set of ontologies as an input and performs a fusing (merging [14, 15]) of them together. It respects namespaces, but doesn't perform alignment procedure on them, thus not solving the problem of semantic equivalency – ontologies are supposed to be aligned beforehand. This is largely thanks to them being stored in SciVi repository and therefore using the same set of concepts and relations.

„DFD2Link" operator (fig. 3) has two inputs. One of them is a DFD diagram created in a SciVi editor that describes a data pipeline (module interconnection graph). Another one is a merged domain ontology that has all required module descriptions. The operator then analyses the domain ontology and DFD to produce an ontological description of module's connections.

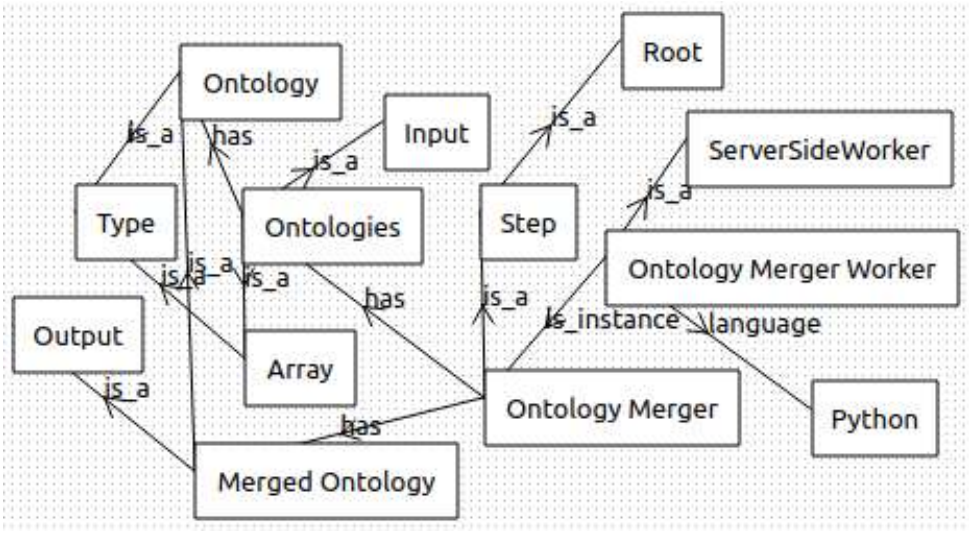

Fig. 2. An implementation of Δ operator

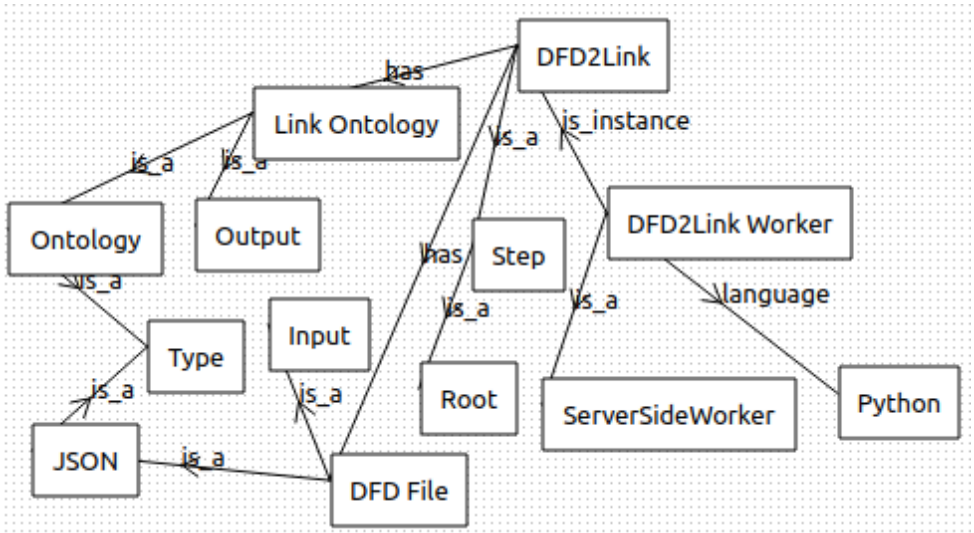

Fig. 3. An implementation of Λ operator

„Generator" operator (fig. 4) is the one responsible for producing the firmware's source code for the smart mediator. This operator generates a directed graph that represents the modules and their connections for data transfer by utilizing knowledge about the modules and their interdependencies as outlined in the ontology. In the subsequent step, the operator performs a topological sort to establish a partial order among the vertices. This partial order is defined such that node V_i which corresponds to a module M_i, precedes node V_j, representing module M_j, if and only if there is no path leading from V_j to V_i. If it proves impossible to establish this partial order for a subset of modules, the generator will halt its processing and return an error message indicating a potential cyclic data dependency between the modules. Conversely, if the transformation occurs without any errors, the generator continues to produce source code that specifies all necessary data structures, initializes them, and subsequently calls the entry points of the modules following the sequence established by the dependency graph. This process takes into account the types of parameters and the direction of data flow—whether it is being sent to or received from the module.

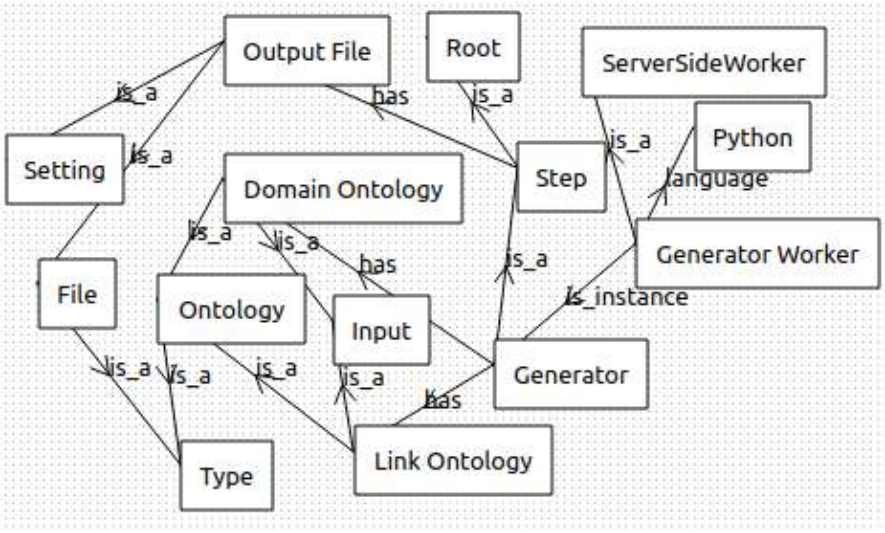

Fig. 4. An implementation of Ω operator

A complete pipeline with all interconnected operators is presented in fig. 5.

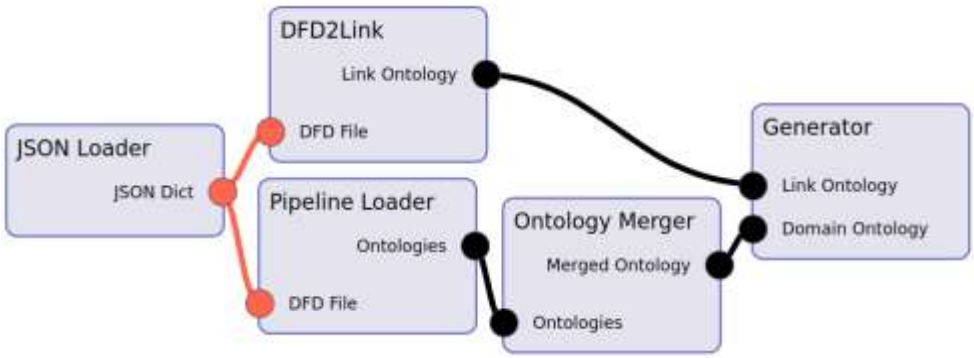

Fig. 5. A pipeline overview

4. Controlling Media Playback via BCI

As a test for the generation pipeline created anew, we have performed a BCI-powered control over media playback on a smartphone. As in our previous work [11], the control signal will be derived from the imaginary movement detection classifier we already have as a module in our system, but this time the signal will be used to issue a play/pause command on the smartphone playing a media file. For this task we'll need to create a new module that will allow us to interface with a smartphone's media playback capabilities. We decided to target an Aurora OS-powered device.

4.1 Aurora OS

Aurora OS[1] is a mobile operating system that is based on the Linux kernel and a set of GNU utilities, thus being POSIX-compliant, incorporating both open-source projects and components with closed source code, developed by Russian company "Open Mobile Platform". The project is a proprietary fork of Sailfish OS[2] and is designed to build a trusted mobile infrastructure and protect sensitive information in government organizations, educational and healthcare institutions, as well as large and medium-scale commercial companies. It contributes to the technological sovereignty of Russia through a secure mobile environment and

1 https://auroraos.ru/
2 https://sailfishos.org/

complies with all requirements of regulators in the field of national security and information protection.

The OS has been in active development since 2016 by the Russian company „Open Mobile Platform". On November 7, 2023, devices running Aurora OS version 4.0.2, produced by FPlus, were released for a retail sale.

The primary development tools for applications on the Aurora OS are the C++ programming language, Qt framework, and Silica module. The Silica module provides essential components for building the user interface of Aurora OS applications, as well as tools for styling Aurora OS applications and managing their behavior. The Silica module not only facilitates the construction of application interfaces but also ensures stylistic consistency between many Aurora OS applications.

The main reason of choosing Aurora OS for our demonstration purposes is the fact that applications for it are basically a regular GNU/Linux programs (with some quirks) written in C++, which makes interfacing with a C code generated by our system transparent and allows very efficient debugging. Other mobile operating systems, like Android, require much more sophisticated setup for that.

4.2 Media Playback Operator

Aurora OS, being a descendant of Sailfish OS, behaves very much like any standalone GNU/Linux distribution, and provides same APIs for programmers to employ. Specifically, it follows a set of FreeDesktop.Org[3] specifications that were created to ensure interoperability of a desktop software in the Unix-like environments.

One of such specifications that interest us is a Media Player Remote Interfacing Specification (MPRIS)[4] that is used to control multimedia applications. It's built on top of a D-Bus message bus system which is supported by Qt framework. We can use it to issue a command to a running mediaplayer instance for running or pausing playback.

For this purpose, a simple operator named "Media Playback" based around QDBusConnection and QDBusInterface classes was created and added to the system's repository. Fig 6 displays an ontology for this operator.

4.3 Adaptation to Signal Acquisition Equipment

To demonstrate the adaptability of our solution in different scenarios we employ two distinct signal acquisition devices during the testing. Both of them rely on an electroencephalography (measuring an electrical activity of the brain) but that's where the similarity ends.

The first device employed in our test is the OpenBCI headset[5], a hobbyist-level, open-source hardware EEG designed for recording EEG data. The headset features a non-invasive dry-electrode design with 16 EEG channels, which allows for multi-channel brainwave data acquisition without the need for conductive gels or wet electrodes. This makes it more user-friendly for experimenters and reduces setup time, though dry electrodes may introduce higher impedance compared to wet electrodes. The device operates with a sampling rate of 125 Hz, which is suitable for capturing low-frequency EEG signals but may limit the resolution of higher-frequency signals like gamma waves. This sampling rate, however, is often sufficient for research applications involving cognitive state monitoring, neurofeedback, and some other ones. For more demanding applications that require higher temporal resolution, such as high-gamma research or real-time BCI applications, a higher sampling rate may be necessary.

Interaction with the OpenBCI headset is performed wirelessly via a USB dongle that communicates with the headset using a proprietary radio frequency protocol based on Bluetooth. The wireless communication ensures flexibility in movement and reduces the need for cum-

3 https://freedesktop.org/

4 https://specifications.freedesktop.org/mpris-spec/latest/

5 https://docs.openbci.com

bersome cables during experiments, which is particularly advantageous in settings where user mobility is required. However, this introduces measurable data transmission latency, which can be disadvantageous in some scenarios.

Experimental setup featuring this device is shown in fig. 6. Here you can see a user wearing an OpenBCI headset and holding a smartphone running an instance of the app taking a role of the smart mediator.

Fig. 6. An OpenBCI setup (demonstrated by I.A. Labutin)

Another device we used in our testing process is a medical-grade EEG acquisition amplifier, the EBNeuro BePlus LTM[6], which is designed for long-term monitoring (LTM) applications in clinical settings such as epilepsy monitoring, sleep studies, and advanced neurological research. We employed the single-module version of the amplifier, coupled with a 21-channel Ag/AgCl (silver/silver chloride) non-invasive wet electrode headcap. This type of electrode is the gold standard in EEG recording, providing high signal fidelity due to its excellent conductive properties and low impedance. The 21-channel configuration follows the 10-20 electrode placement system, which covers key scalp locations to capture brain activity from different regions for a comprehensive neurophysiological assessment.

The BePlus LTM amplifier is capable of high-resolution signal acquisition at sampling rates of up to 2048 Hz, which ensures precise temporal resolution necessary for capturing fast cortical dynamics, including high-frequency oscillations and epileptic spikes. However, in our test, we limited the acquisition rate to 512 Hz, which is a commonly used rate for general EEG studies. This rate provides a good balance between capturing the fine details of brain activity while reducing the volume of data and processing load, particularly for real-time applications. At 512 Hz, the system can still effectively capture signals up to 256 Hz in frequency, covering all major EEG frequency bands, including delta, theta, alpha, beta, and gamma bands, making it suitable for our test.

Communication between the BePlus LTM amplifier and the external infrastructure is handled via a wired Ethernet connection with a static IP address assignment, thus we employed a Wi-Fi-enabled router to establish a local area network (LAN). This LAN allowed seamless communication between the EEG amplifier and the test application running on a smartphone. The router effectively bridged the wired Ethernet connection from the amplifier to the wireless interface used by the smartphone, ensuring low-latency and robust data transfer during the test.

This device and the corresponding setup are illustrated in fig. 7. Here you can see a user wearing a wet-electrode headcap connected to EBNeuro BePlus LTM amplifier and holding a smartphone which runs an instance of the app taking a role of the smart mediator.

6 http://www.ebneuro.com/en/long-term-monitoring-ltm/be-plus-ltm-ltm

Fig. 7. A BePlus LTM setup (demonstrated by I.A. Labutin)

4.4 Data Processing Pipeline

A general data processing pipeline is presented in Fig. 8.

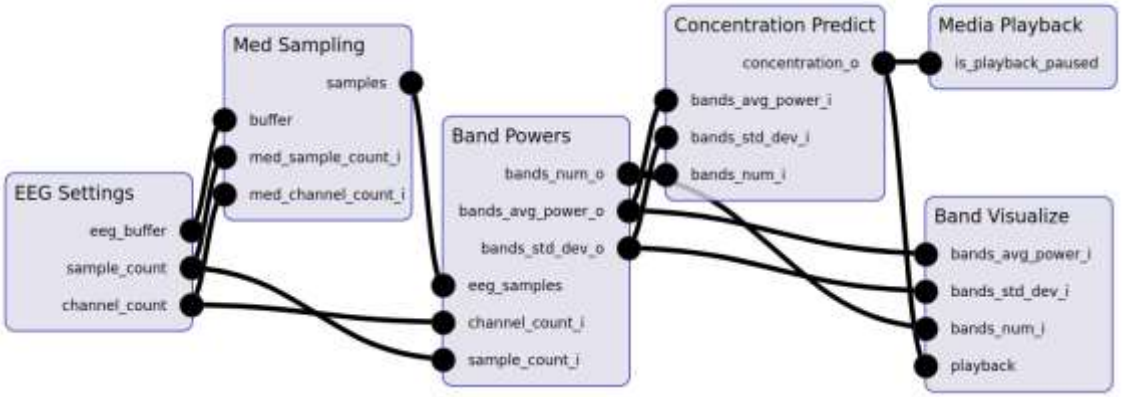

Fig. 8. Media playback pipeline

It consists of several modules:

• „EEG Settings" is a simple operator that allows for the user to set a few constants for later use in the data processing pipeline.

• „Med Sampling" abstracts away EEG connection and data acquisition. It communicates with EEG device and retrieves a raw data from it in a form of two-dimensional array of floats each tick pipeline is being executed.

• „Band Powers" is an operator built upon BrainFlow's library implementation of power spectrum analysis module that was adapted by us to meet the limitations of an embedded and portable systems. It accepts a window of raw EEG data and outputs a number of power spectrum's density values.

• „Concentration Predict" is the conventional name of the threshold-based classifier that deduces the presence or absence of a certain pattern in the EEG signal (in our case, imaginary movement).

• „Band Visualize" operator (fig. 9) is used to display band powers as well as a current playback status on the smartphone's screen while the app is running.

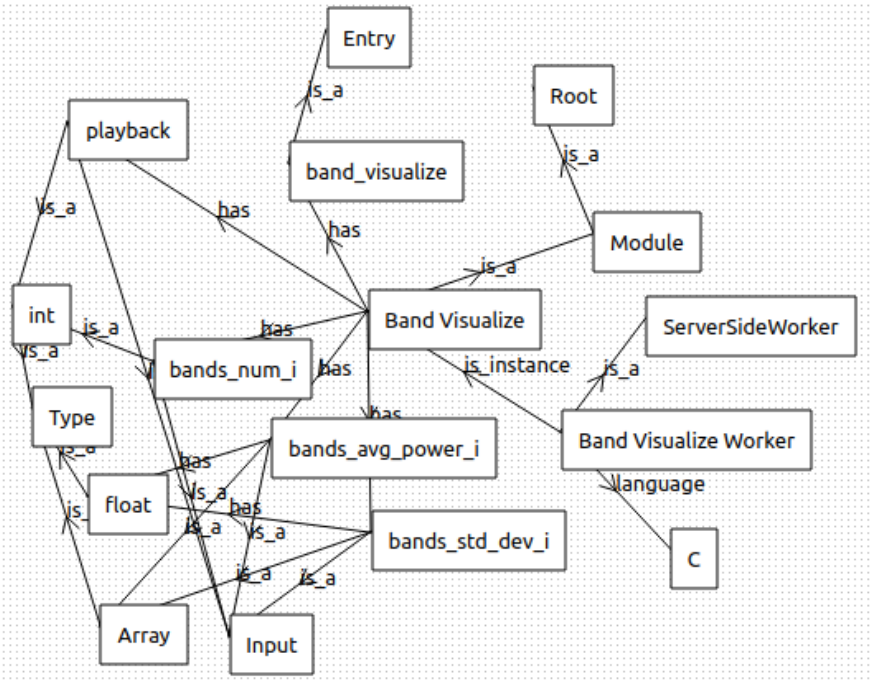

Fig. 9. Ontology of the „Band Visualize” operator

It was introduced as a simple way to get an easy and understandable feedback on the mediator's current internal state. This operator can be useful both to a system's operator (the one who created pipeline) as a debug tool and to the end user of the smart mediator, who can be a casual user (for example, a hospital's patient) to get a general idea of what's happening in dynamic during experiment by means of visualization tools.

• At the final stage of the data processing pipeline, the new „Media Playback” operator (fig. 10) is introduced to send a command to the running media player instance based on the output from the classifier.

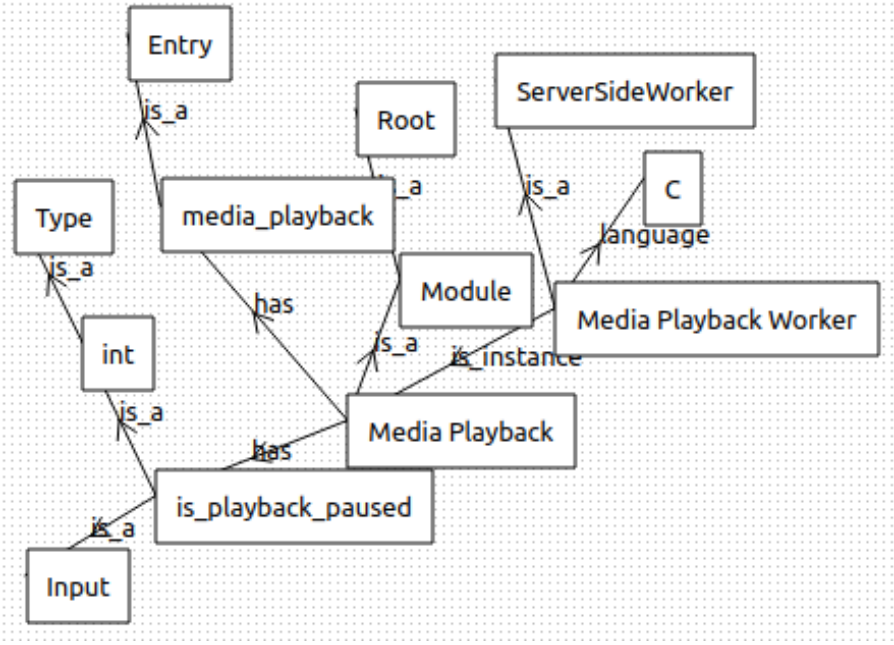

Fig. 10. Ontology of the „Media Playback” operator

After this pipeline was created, it was saved in a form of JSON file and then reused as an input to the firmware generation (see fig.5). The resulting domain ontology (fig. 11) and the generated source code (fig. 12) are presented below.

The generated source code was then used in an application build process as a glue that calls each module entry point in the specified order and thus processes data from the EEG to

achieve the intended result – a media playback control. In fig. 13 screenshot of the mobile device screen is presented that displays the information about current relative band powers and playback status.

The pipeline presented in fig. 5 can be expanded with additional modules to increase both functionality and usability due to a flexibility of proposed framework. By incorporating more control signals and actions, this pipeline can evolve to handle a broader range of tasks, enabling users to interact with the environment in a more sophisticated manner.

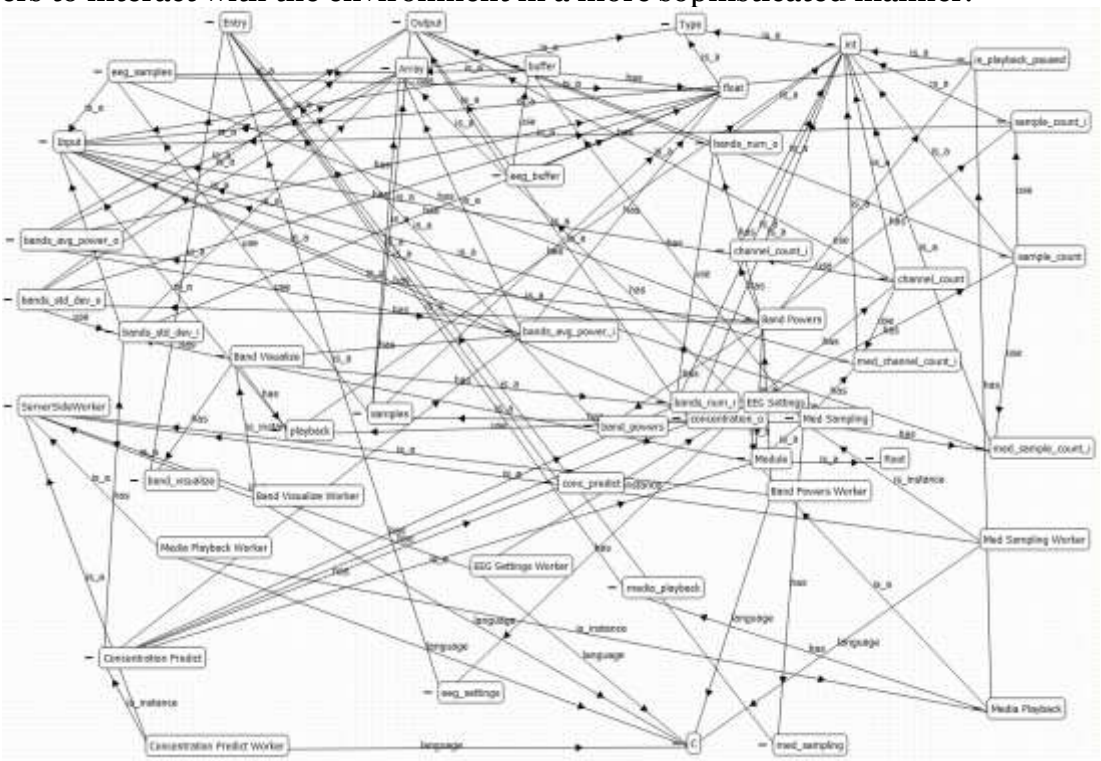

Fig. 11. Resulting domain ontology

```
// Variable declarations
int bands_num_o;
float* bands_avg_power_o;
float* bands_std_dev_o;
int concentration_o;
float* eeg_buffer;
int channel_count;
int sample_count;
float* samples;

// Module entrypoint declarations
void eeg_settings(float** eeg_buffer, int* channel_count, int* sample_count);
void med_sampling(float* buffer, int med_sample_count_i, int med_channel_count_i,
                  float** samples);
void band_powers(float* eeg_samples, int channel_count_i, int sample_count_i,
                  int* bands_num_o, float** bands_avg_power_o, float** bands_std_dev_o);
void conc_predict(int bands_num_i, float* bands_std_dev_i, float* bands_avg_power_i,
                  int* concentration_o);
void media_playback(int is_playback_paused);
void band_visualize(int bands_num_i, float* bands_std_dev_i,
                  float* bands_avg_power_i, int playback);

// Processing code
eeg_settings(&eeg_buffer, &channel_count, &sample_count);
med_sampling(eeg_buffer, sample_count, channel_count, &samples);
band_powers(samples, channel_count, sample_count, &bands_num_o, &bands_avg_power_o,
                  &bands_std_dev_o);
conc_predict(bands_num_o, bands_std_dev_o, bands_avg_power_o, &concentration_o);
media_playback(concentration_o);
band_visualize(bands_num_o, bands_std_dev_o, bands_avg_power_o, concentration_o);
```

Fig. 12. An example of generated code

Fig. 13. Application screenshot displaying bands power and playback status

For instance, in a healthcare setting, the pipeline could be adapted to allow patients to call for assistance from healthcare professionals, such as a nurse or doctor, directly with their BCI. This would enhance patient care by providing a seamless and efficient way to communicate urgent needs, ultimately improving response times and outcomes in critical situations.

Moreover, the visualization aspect of the pipeline can be significantly expanded to display a wider array of signal characteristics. This feature could be particularly beneficial in specialized fields like neurology research, where researchers need real-time insights into complex data streams, such as EEG signals. By leveraging the adaptability of the pipeline in a unified way, neuroresearchers have opportunity to visualize and analyze neural data on their smartphones or tablets as an experiment unfolds, giving them the ability to monitor brain activity or other physiological metrics in a real-time. This dynamic, real-time visual analysis enhances the research process, offering a more deeply understanding of identified cognitive or neural patterns and enabling quicker adjustments during experiments.

A key adaptive strength of this integration approach lies in its ontology-driven nature. By using ontologies, the system can easily incorporate new data processing modules and BCIs into the existing pipeline without the necessity of previous developed source code modifying. This allows for a unified integration of various components without requiring significant reengineering efforts. The adaptability of an ontology-driven approach enables the system to be tailored to different domains such as healthcare, neuroresearch, entertainment and other ones. In this way, the ontology acts as a backbone for managing complexity, ensuring that the system remains scalable and future-proof as new needs and technologies emerge.

5. Conclusion and Future Work

In this paper, an adaptable approach is introduced, utilizing the SciVi platform as a comprehensive user interface for constructing a data processing and firmware generation pipeline. Successfully validation the pipeline has been demonstrated to tackle the task of controlling media playback with two different neurocomputer interfaces, which designed to control media playback on a portable device. All that proves is the adaptability power of our ontology-driven high-level toolkit for unified integration of software systems with neural interfaces including real-time experimental data visualization.

The effectiveness of this proposed method has been validated through practical testing at the Educational and Scientific Laboratory of Sociocognitive and Computational Linguistics at PSU. Recently, a new method has been developed to eliminate the existing cyclic dependencies between the data of various software modules of the embedded software of the smart mediator. The possible next directions of our research include aiming towards improving both the functional characteristics of the system as well as usability of the developed tools.

6. Acknowledgments

The authors wish to express their sincere gratitude to the Educational and Scientific Laboratory of Sociocognitive and Computational Linguistics at Perm State University's Faculty of Philology. Their appreciation extends especially for the invaluable equipment and steadfast support that were critical to the successful execution of this research.

References

1. Huang S., Tognoli E. Brainware: Synergizing software systems and neural inputs // Companion Proceedings of the 36th International Conference on Software Engineering, ICSE Companion 2014, Association for Computing Machinery, New York, NY, USA, 2014, p. 444–447. (doi:10.1145/2591062.2591131)

2. McCullagh P., Ware M., McRoberts A., Lightbody G., Mulvenna M., McAllister G., González J. L., Cruz Medina V. Towards standardized user and application interfaces for the brain computer interface // Stephanidis C. (Ed.), Universal Access in Human-Computer Interaction. Users Diversity, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 573–582.

3. Huggins J., Guger C., Aarnoutse E., Allison B., Anderson C., Bedrick S., Besio W., Chavarriaga R., Collinger J., Do A., Herff C., Hohmann M., Kinsella M., Lee K., Lotte F., Müller-Putz G., Nijholt A., Pels E., Peters B., Zander T. Workshops of the seventh international brain-computer interface meeting: not getting lost in translation // Brain-Computer Interfaces (2019) 1–31. (doi:10.1080/2326263X.2019.1697163)

4. Allison B. The I of BCIs: Next generation interfaces for brain–computer interface systems that adapt to individual users // Jacko J. A. (Ed.), Human-Computer Interaction. Novel Interaction Methods and Techniques, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 558–568.

5. Labutin I. A., Chuprina S. I. Kontseptsiya postroeniya ontologicheski upravlyaemykh neirointerfeisov [Conception of building ontology-driven neurointerfaces] // Intellektual'nye sistemy v nauke i tekhnike, 2020, pp. 106–112. [in Russian]

6. Ryabinin K. V., Chuprina S. I., Labutin I. A. Ontology-driven toolset for audio-visual stimuli representation in EEG-based BCI research // Proceedings of the International Conference on Computer Graphics and Vision "Graphicon", CEUR, volume 31, Keldysh Institute of Applied Mathematics, 2021, pp. 223–234. (doi:10.20948/graphicon-2021-3027-223-234) (https://keldysh.ru/papers/2021/prep_vw.asp?pid=9273)

7. Ryabinin K. V., Chuprina S. I., Labutin I. A. Ontology-driven tools for EEG-based neurophysiological research automation // Scientific Visualization,volume 13 (2021), pp. 93–110. (doi:10.26583/sv.13.4.08)

8. Chuprina S., Labutin I. Scientific visualization tools to improve utilizing neural interface // Proceedings of the International Conference on Computer Graphics and Vision "Graphicon", volume 32, Keldysh Institute of Applied Mathematics, 2022, pp. 391–402. (doi:10.20948/graphicon-2022-391-402)
(https://keldysh.ru/papers/2022/prep_vw.asp?pid=9592)

9. Ryabinin K., Chuprina S., Labutin I. Tackling IoT interoperability problems with ontology-driven smart approach // Rocha A., Isaeva E. (Eds.), Science and Global Challenges of the 21st Century - Science and Technology, Springer International Publishing, Cham, 2022, pp. 77–91.

10. Chuprina S. I., Labutin I. A. How to improve utilizing neural interface by means of ontology-driven scientific visualization tools // Scientific Visualization, volume 14, 2022, 83–96. doi:10.26583/sv.13.4.08.

11. Chuprina S., Labutin I. Ontology-based neurointerface IoT integration approach // Trudy Instituta sistemnogo programmirovaniya RAN, volume 36, Keldysh Institute of Applied Mathematics, 2024, pp. 91-108. (doi:10.15514/ISPRAS-2024-36(2)-8)

12. Lunev D., Poletykin S., Kudryavtsev D. O. Brain-computer interfaces: technology overview and modern solutions // Modern Innovations, Systems and Technologies, volume 2, 2022, pp. 117–126. (doi:10.47813/2782-2818-2022-2-3-0117-0126)

13. Ryabinin K. V. Metody i sredstva razrabotki adaptivnykh mul'tiplatformennykh sistem vizualizatsii nauchnykh eksperimentov [Methods and tools for developing multiplatform scientific research visualization systems], Moscow: Thesis for the degree of Candidate of Physical and Mathematical Sciences, 2015. (https://library.keldysh.ru/diss.asp?id=2015-ryabinin) [in Russian]

14. Chuprina S. I., Ryabinin K. V., Koznov D. V., Matkin K. A. Ontology-Driven Visual Analytics Software Development // Programming and Computer Software, volume 48, 2022, pp. 70–77. (doi.:10.1134/S0361768822030033)

15. Chuprina S. I. Using data fabric architecture to create personalized visual analytics systems in the field of digital medicine // Scientific Visualization, volume 15, №5, 2023, pp. 50–63. (doi:10.26583/sv.15.5.05) (https://api.semanticscholar.org/CorpusID:266356696)

16. Chuprina S. I., Adaptatsiya tekhnologii fabrik dannykh k razrabotke sistem vizual'noi analitiki v oblasti tsifrovoi meditsiny [To Adapt Data Fabric Technology to Visual Analytics Systems Development in the Field of Digital Medicine] // Trudy 33 Mezhdunarodnoi konferentsii po komp'yuternoi grafike i mashinnomu zreniyu "GrafiKon-2023", Institut problem upravleniya im. V.A. Trapeznikova RAN, 2023, pp. 405–416. (doi:10.20948/graphicon-2023-405-416) [in Russian]