Ontology-based neurointerface IoT integration approach

Ivan Labutin[1] and Svetlana Chuprina[2]
Faculty of Mechanics and Mathematics
Perm State University, 15 Bukireva Str., Perm, 614068, Russian Federation
ORCID: [1]0000-0001-6858-1479, [2]0000-0002-2103-3771

***Abstract*—Recently, there is a surge of interest in employing a neurocomputer interfaces for a control contours implementation, especially for different infrastructures of Internet of Things. However, due to a low-level nature of such devices and related tools, neurointerface integration with a large variety of IoT devices is quite a tedious task, and the one that requires a lot of knowledge in the neuroscience and signal processing to boot. In this work we propose an ontology-based solution for facing the upcoming challenges of unified integration of brain-computer interfaces (BCI) into IoT ecosystems. This ontology driven high-level approach enables researchers and engineers without strong background in BCI to automate the integration neurointerfaces with different infrastructures of Internet of Things.**

***Index Terms*—neurointerface, BCI, ontology engineering, IoT**

I. Introduction

Due to the widespread digitization and an active expansion of the application areas of Internet of Things (IoT), virtual reality and augmented reality, methods and tools for managing software systems based on neurointerfaces are developing rapidly. Of crucial concern here is that there is no universal standard for the integration of different IoT infrastructures nowadays. The diversity of existing protocols and standards for device interaction in the IoT, as well as their lack of compatibility, leads to communication problems between devices within a single infrastructure. Therefore, the call for development and implementation of a new unified concepts and refinement of existing ones to increase the level of interoperability of devices seems to be quite an essential one, especially when it comes to the task of embedding neurointerfaces into already existing software systems.

In the current state of the art, issues related to creating unified methods and tools to automate the process of integration of neurointerfaces into the IoT ecosystem with a goal of controlling its components (target systems and subsystems) are still insufficiently studied, although the literature recognizes the need to develop such methods and approaches [1]–[4].

In most cases, neurointerface equipment is able to function properly only with a narrow range of proprietary software provided by the manufacturer. Therefore, employment of neurointerfaces in the scenarios and pipelines not accounted by a manufacturer poses quite a challenge, if ever possible, and requires a deep knowledge in fields of both neuroscience and computer science.

In this paper we present the concept of ontology-driven system for integration of neurointerfaces into IoT ecosystems and the approach to its implementation. The general idea – automated generation of a firmware for a smart mediator connecting together an IoT infrastructure and a neurointerface – was introduced in our previous work[5]; in this paper, we focus on proposing a formal model and describe it's implementation.

We shall note here that a general overview of neurocomputer interfaces is out of scope of this work; interested readers can be advised to familiarize themselves with an excellent summary paper [6].

II. Use case scenarios

Before building the system model in question, it is necessary to understand the user's general portrait and possible scenarios of his interaction with the system.

It shall be noted that there are two categories of potential users of our system in relation to the fields of neurointerfaces, IoT and ontological engineering: *nonspecialists* and *specialists*.

Nonspecialists will use the system in the form provided to them by *specialists*; nonspecialists don't have the expertise to (and, ideally, should not) configure and extend the system as they may have no relevant competencies. In essence, nonspecialists will employ the system „as is" as a set of available tools for solving their own personal problems related to integrating neurointerfaces into specific IoT infrastructure.

Specialists, on the other hand, in addition to the aforementioned employment of the system, can also extend and reconfigure it by adding new modules and their ontological descriptions.

It is important to distinguish these two groups of users as the former ones may need a complete and convenient high-level interface; while the latter can use a lower-level means of interaction with the system to increase flexibility and efficiency of task solving.

The main nonspecialists' scenario for the platform's employment will be using a high-level visual user interface to create a formalized description of a „smart assistant"'s internal composition and employing some special means to automatically generate its source code based on this formal description. The generated code then becomes a part of the software of such assistant and unifies the integration of user selected modules. Based on our previous work[7] we propose to use the tools of SciVi platform to tackle this problem.

SciVi is Russian scientific visualization and analytics platform enabled describing the pipeline of analytical data processing by means of data flow diagrams and generate the source code in related programming language. This was made possible due to SciVi being an ontology-driven solution.

This scenario is presented in Fig. 1 as a use-case diagram, where two usage scenarios related to employing the platform by nonspecialists are illustrated. First scenario – composing DFD diagram in SciVi environment. Second scenario – generating code for the smart assistant based on previously created DFD diagram.

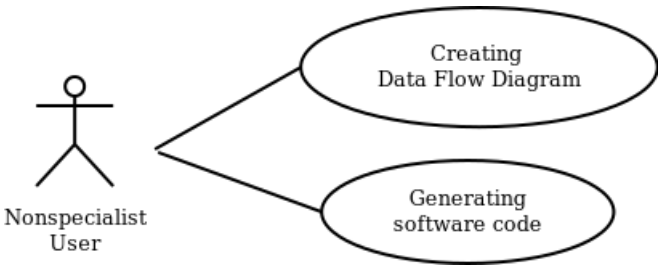

Fig. 1. Use cases for nonspecialists

For specialists, an additional scenario is added that involves extending the system with new components. Aside from components of interest they also need to add their ontological descriptions (manually crafted or generated by some tooling) into the system's repository. This is is illustrated on Fig. 2 with three use cases related to employing the platform by specialists. Compared with Fig. 1, there is an additional third scenario – filling the system repository with ontological descriptions of new modules available for potential inclusion in code for the smart mediator.

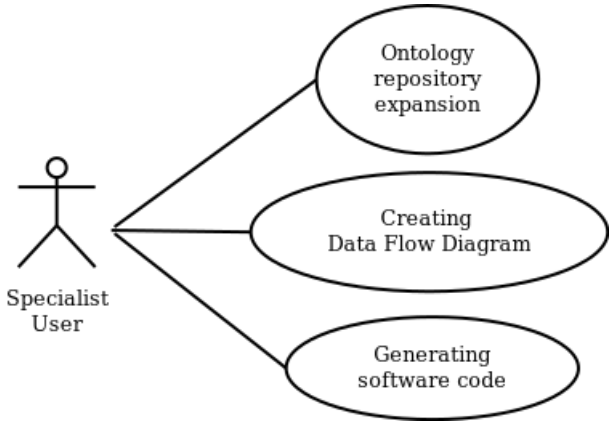

Fig. 2. Use cases for specialists

Note here that this is by no means is not an exhaustive list of potential ways to interact with the system; on practice, other interaction scenarios are possible, which are not covered in the paper. Their discovery and analysis is the topic of a further study. In this paper, we focus on providing users who belong to the category of „specialists" with the necessary tools for integrating neurointerfaces into arbitrary IoT infrastructure. To this end, we propose the ontology-driven solution in which the so-called „smart mediator" plays a key role.

III. Ontological approach towards integration

In the context of computer science, ontology is seen as a formal representation of knowledge about some domain in a form of sets of concepts and relations between them, as well as associated axiomatic. One of the founders of ontological engineering, Thomas Gruber, defined ontology as „explicit specification of conceptualization" [8].

Following [9] we will define a formal model of ontology O an ordered triplet of the form:

$$O = \langle X, R, \Phi \rangle, \tag{1}$$

where X is a finite non-empty set of concepts (notions, terms) in the domain area represented by ontology O;
R is a finite set of relationships between concepts;
Φ is a finite set of interpretation functions defined on the concepts and relations of ontology O (axiomatic).

Furthermore, [9] points out that in an ontology-based system model, there are three ontological components:

1) *Metaontology*, which contains general concepts and notions independent of the domain area.
2) *Domain ontology* (one or several), which describes a specific domain area.
3) *Task ontology* (one or several), which contains types of tasks to be solved and their decomposition into subtasks.

Regarding our goal, we will be mainly interested in domain ontologies. Metaontology for our purpose is not strictly required due to a clearly defined (at least on this stage of research) domain area. For similar reasons, we do not see a need for a task ontologies: as it was noted above, the system shall support only a narrow range of use cases, therefore instead of an ontological description of tasks, knowledge about them and ways of interpreting supported types of relationships should be built into the inference engine.

This seems to be the most effective solution in the context of current research, because in such a case it is possible to use so-called „lightweight" ontologies[10].

Here and further, if nothing else is specified explicitly, under the term „ontology" will be understood only „lightweight" ontologies, which do not contain axiomatic (that is, in 1 $\Phi = \varnothing$). Their interpretation falls entirely on the implementation of inference engine. We suggest to create ontologies using environment of our visual ontology editor named ONTOLIS[11]–[13]. ONTOLIS is aimed at ordinary users and at the same time allows developers uniformly storing a wide range of attributes in the nodes and arcs of the ontology graph in a unified form which is necessary to take into account the specifics of solving specific tasks during inference[13]. This allows the user to expand the possibilities of interpreting ontologies to some extent, providing an inference engine with additional information in simple machine-readable format, which gives the opportunity to modify its behavior (including referencing external tools).

IV. Requirements for system and „smart mediator"

Before describing the formal model of the system, it is necessary to define a set of requirements that both the system itself and its results should meet.

Functional requirements for the „smart mediator" logically follow from the purpose of his existence.

1) The smart mediator must receive a signal from the neurocomputer interface. Consequently, the smart mediator should include a module responsible for communication with the neurointerface.
2) The smart mediator must process the received signal. Processing signals is a nontrivial issue per se and is not the subject of this work. It is assumed that the mediator should include several modules responsible for this task. The question of suitability of these modules lies on the shoulders of the user of the system (as was said above, we are talking about a competent user). Within the framework of this work, a set of modules is implemented, sufficient for demonstrating the proof-of-concept of suggested approach to unified integration of the smart mediator into the IoT infrastructure.
3) The smart mediator must provide access to the result of signal processing to other devices in the IoT infrastructure. This can be done in at least two ways (see section V); the choice of a specific mechanism also depends on requirements imposed by the user of the system.

The main non-functional requirements for an smart mediator – flexibility, reliability, predictability – are described below.

1) *Flexibility* means that the algorithm of the „smart mediator" is not cut in stone once and for all: it can adapt according with changing user needs or surrounding infrastructure. Within the proposed solution this means the need for regenerating the code of the mediator based on a formalized description of the changed infrastructure.
2) *Reliability* implies that the mediator should have a certain degree of robustness to errors that inevitably occur during interaction between software systems. A detailed error typology is beyond the scope of this work, so from here we assume that the requirement for reliability is inherently implemented at the design stage of modules from which the „mediator" is composed of.[1]
3) *Predictability* guarantees that the mediator will perform only the task prescribed to it by the formal model. This requirement can be received „for free" to some degree if we strictly stick to the paradigm of open-source and free software[2].

[1]Strictly speaking, it was also worth to take into account problems arising during interaction between modules of the mediator with each other. The problem of organizing interaction between modules of the mediator among themselves is not a key issue in this paper and is left for the specialist who composes formal representation for further generation of code of the mediator.

[2]Of course, openness and freedom per se do not guarantee full absence of random or deliberate defects leading to unexpected behavior, but they at least minimize risks associated with them through the „thousand eyes effect" that declares a positive correlation between the quality of the software and the number of people who have access to its code

V. System development concept

Based on our experience in the field of ontology-driven human-machine interfaces [11]–[15], we propose a mechanism for integrating brain-computer interfaces into the Internet of Things infrastructure by introducing an intermediate layer – a smart mediator that will be responsible for communication between the environment (bci:Context in the terminology of [16]) and neurointerface (bci:Device in [16]). The mediator's software is generated automatically, and this process is driven by a managing ontology. In addition, to perform such a generation some extra resources will be required:

- Ontology of semantic filters[17] that describes the necessary transformations of data received from the neurointerface. Parts of this ontology were borrowed from SciVi and used in this work. In addition, it was extended with additional modules specific to neuroscience (for example, with a description of a module for inverse Fourier transform).
- Ontology of target platform that reflects the characteristics of the environment or device that will support the execution of the mediator. This ontology was created from the scratch but taking into account the quirks of SciVi platform.
- Component ontology[18] that provides a complete description of the characteristics of the neurointerface sufficient for establishing communication with it and exchanging data. This ontology was also created from the scratch.

It is important to emphasize in the proposed approach the key role of the mediator, which is considered and implemented as a software module running on one of the devices in the infrastructure of the Internet of Things. In essence, it represents a microservice that receives raw data of a biological nature from a neurointerface and provides access to the results of processing that data to other devices in the IoT infrastructure in one of two possible ways:

1) The smart mediator publishes results of data processing by some protocol, and other network nodes independently and periodically call in to receive information about some aspects of cognitive state of a human. For example, „smart" lighting bulb can automatically obtain fatigue level data and regulate the light intensity taking into account this information.
2) The smart mediator generates a direct control signal affecting one of IoT infrastructure devices. For example, mediator may define the state of concentration of a person and send it via MPRIS[3] protocol to media device playing multimedia (video / audio) command to pause playback.

The mediator itself may execute either on a specialized device (such as development board for ESP32 microcontroller), fully utilizing all its resources, or on an preexisting infrastructure node (for example, a router or even a personal

[3]https://specifications.freedesktop.org/mpris-spec/latest/

computer). In the latter case such an mediator can be considered „virtual”. The only significant requirement is a physical ability to connect the neurointerface to an equipment on which the mediator works.

As noted above, software code of mediator is being generated following the managing ontology, contents of which is vital for correctness of the mediator functioning. A very important features of ontologies are their transparency, documentability and readability both for human and program agent. This is a significant trait that allows reusing external ontological resources.

Despite of existence of automated tools for creation and validation of correctness of ontologies[19], [20], in reality most practical approaches to choice and quality assessment of ontologies is still rely on experts and knowledge engineers (ontology engineers). The other popular approach is to deduce a corectness of built ontology from the result of correctness evaluation of ontologically driven solution.

From our point of view ontological descriptions of various neurointerfaces are not obliged to be integrated in a framework of one domain ontology. It seems viable to develop an interface for so-called smart repository for conveniently search and reuse ontologies of other neurointerfaces or/and their parts. Work of such a repository shall be controlled by metaontology (highest-level ontology), i.e. ontology that is storing knowledge about all domain ontologies stored in the repository. Naturally, at same time it is required to ensure coherence and consistency of ontologies[21], [22], but with a moderately small sizes of ontologies this condition can be met with a relative ease.

Fig.3 demonstrates a general overview of proposed integration mechanism for neurointerfaces into IoT ecosystem. Here bci:Device is denoting the neurointerface, bci:Context describes environment to be integrated with, BCI-O – main managing ontology, mediator – mediator itself, adapter – a system that implements an integration mechanism integration for a specific neurointerface into an IoT ecosystem.

The ontological device description (bci:Device) is being automatically united with the ontology describing infrastructure (bci:Context) in one common domain ontology, using which a system („Generator”) generates software for smart mediator.

To perform an integration of a neurointerface into existing IoT infrastructure (for example, a „smart home” ecosystem) it's necessary to execute following steps (through suggested algorithm description every mention of creation of ontologies assumes that this can be performed employing ontology learning tools for automated creating of ontologies):

1) Either select an existing ontological neurointerface description from the repository of the SciVi platform[23] or create a new one in the visual ontology editor ONTOLIS saving it to the SciVi repository.
2) Choose a preexisting or create a new ontological descriptions of semantic filters for preprocessing input/intermediate/output signals based on the task requirements, similar to how this is implemented in our colleagues' work dedicated to the SciVi system for scientific visualization and visual analysis.
3) Either select an existing ontological description of the specific IoT ecosystem from the repository of the SciVi or create a new one consistent with BCI-O and IoT-O[24].
4) Use services of the proposed system for an automatic construction of adapters for neurodevices and an automatic generation of the smart mediator's firmware from obtained ontologies; this firmware will be responsible for integration of neurointerface in the infrastructure of IoT.

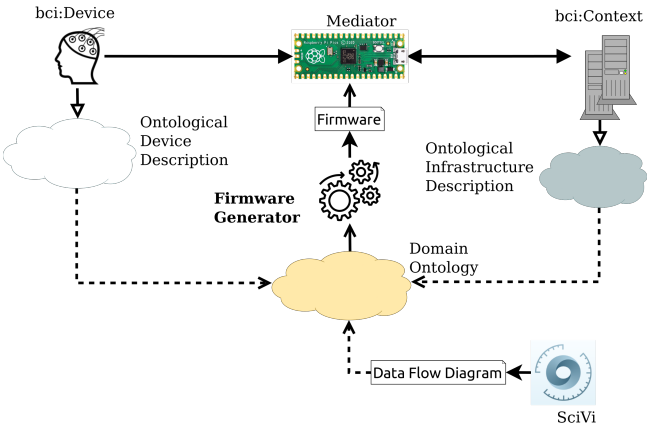

Fig. 3. Proposed integration mechanism

A mechanism for ontology-driven generation of firmware's source code for IoT ecosystem was previously developed by our colleagues as a part of a project aimed at creating and improving the scientific visualization and visual analysis platform SciVi[23]. We reuse similar concepts when solving tasks related to integrating neurointerfaces into an already existing, or created from scratch ecosystem of IoT.

The proposed solution allows to unify the process of integration of neurointerfaces in a specific IoT ecosystem. Building an ontology describing a particular neurointerface must be performed only once, after which it is added to the corresponding repository of ontologies and can be repeatedly reused for automatic regeneration of mediator's firmware with the goal of integrating it with different environments.

Thus, unification of integration tools is achieved by reusing ontologies and the ability to adapt an ontology-driven system to solve specific problems through replacing ontologies without the need to make changes in previously developed source code for other components of the system[12], [13], [15].

VI. Formal model of a system

In this paper we propose the following formal model of the system described in set-theoretic notation:

$$\Xi = \langle \Omega, \Delta, O_b, O_F, O_i, \Gamma, M, E, \Lambda, O_L, S \rangle, \qquad (2)$$

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

Λ is the operator for building an ontology of module connections, $\Lambda : O_b \times O_F \times O_i \times D \to O_L$;
O_D is the ontological description of the domain area;
O_b is the ontological description of a specific neurointerface;
$O_F = \cup_{i=1}^{k} O_{fi}$ – 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);
O_L is an ontological description of the links between modules included in the mediator's firmware;
Δ is the operator for building an ontology from parts, $\Delta : O_b \times O_F \times O_i \times O_L \to O_D$;
Γ is an interaction operator, $\Gamma : E \times M \to D$;
M is a set of supported control elements;
E is a set of supported user actions;
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.

Let's note that Γ, M and E are out of scope of this paper and are based on reusing results obtained by our colleagues within the framework of the SciVi platform[12], [13], [15], [23].

A Δ operator responsible for building an integrated domain ontology is described at Alg. 1.

Algorithm 1 Δ operator

procedure MERGE(O_m, O_i)
  for all $n \in$Nodes(O_i) **do**
    if $n \notin O_m$ **then**
      AddNode(O_m, n)
    end if
  end for
  for all $r \in$Relations(O_i) **do**
    if $r \notin O_m$ **then**
      AddRelation(O_m, r)
    end if
  end for
end procedure
$O_D \leftarrow \varnothing$
Merge(O_D, O_b)
for all $O_{fi} \in O_F$ **do**
  Merge(O_D, O_{fi})
end for
Merge(O_D, O_i)
Merge(O_D, O_L)

A Λ operator that builds an ontological representation of connections between modules is described at Alg. 2. It reuses a `Merge` function from Alg. 1.

Algorithm 2 Λ operator

Merge(O_D, O_b)
for all $O_{fi} \in O_F$ **do**
  Merge(O_D, O_{fi})
end for
Merge(O_D, O_i)
$O_L \leftarrow \varnothing$
$root\_out \leftarrow$ FindNode($O_T, 'Output'$)
for all $block \in D$ **do**
  $name \leftarrow$ Name($block$)
  $start \leftarrow$ FindNode($O_T, name$)
  $outputs \leftarrow$ GetAdjNodes($O_T, \{start, root\_out\}$)
  for all $dfd\_out \in$ Outputs($block$) **do**
    $ont\_out \leftarrow outputs$[Name($dfd\_out$)]
    if $ont\_out \notin L$ **then**
      AddNode(L, ont_out)
    end if
    for all $dfd\_conn \in$ Connections(dfd_out) **do**
      $t\_name \leftarrow$ Name(Target(dfd_conn))
      $ont\_in \leftarrow$ FindNode(O_T, t_name)
      if $ont\_in \notin L$ **then**
        AddNode(L, ont_in)
      end if
      $r \leftarrow$ Relation($ont\_in, ont\_out, 'use'$)
      AddRelation(L, r)
    end for
  end for
end for

VII. Basic concepts and relationships of ontological descriptions

In order to maintain compatibility with the SciVi system (in particular, for reusing the operator Γ, which is responsible for providing users with an easy-to-use interface for building a formal description of module interaction) in this work we largely reused a set of basic concepts and types of relationships that SciVi supports. Such modules and relationship types are highlighted in italics.

The basic (root) concepts of the ontological model of the system include:

- *Root* – root node, required by SciVi for the proper operation of Γ operator.
- Module – a concept reflecting a specific software module potentially available for inclusion in the firmware.
- *Type* – data type.
- *Entry* – entry point into the module.
- *Input* – input data of the module.
- *Output* – output data of the module.
- *Array* – subtype of „array" data introduced to simplify the implementation of an inference engine.

Relationship types supported by the system:

- *is_a* – inheritance relation.
- *has* – relation of ownership.

- use – relation of employment. In the context of the system, it means „consumes” in the sense „the input parameter A of procedure $P1$ consumes the output parameter B of procedure $P2$.”

The system allows for the existence of other basic concepts and types of relationships in ontologies, but does not try to interpret them. This architectural solution was also dictated by the desire to maintain compatibility between ontological descriptions with a representation understood by operator Γ. However, some relationship types, which are not strictly necessary, were excluded for maintaining readability of ontologies – for example, the „base_type” relation used by operator Γ to display array element type to user during building of formal description D.

Examples of an ontological description of the module and link between modules using aforementioned concepts and relationship types are shown on Fig. 4 and Fig. 5, respectively. The Test1 module illustrated on Fig. 4 has a „test1_process” entry point and three parameters: a1, a2 (input) and a3 (output). Parameters a1 and a3 have an integer type `int`, while parameter a2 – the integer type `short`. The fragment of ontological description of link between module parameters on Fig. 5 reflects the following dependencies: parameter a2 gets its value from a parameter k2, parameter a1 – from k1, and a3 – from q3.

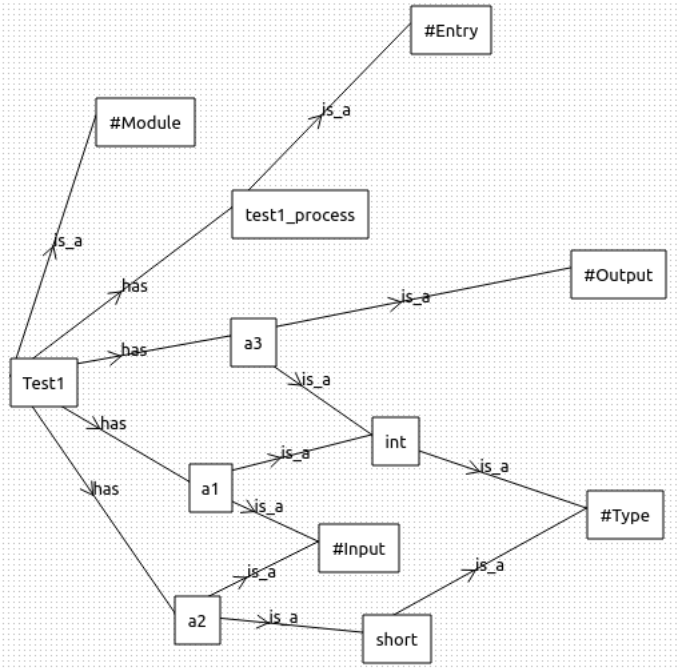

Fig. 4. Example of ontological description of module

VIII. System Development and Implementation

Within the framework of current research, a system prototype was developed and implemented based on the proposed approach. The system allows its users to:

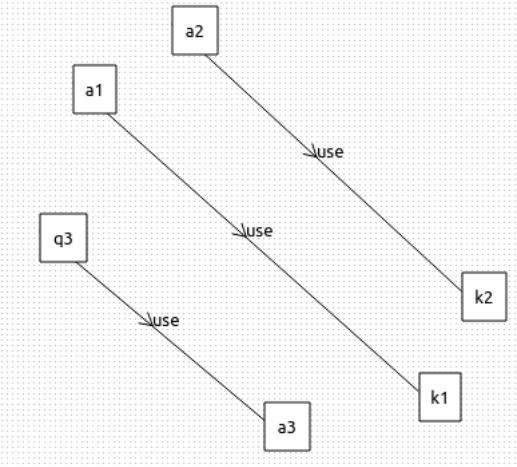

Fig. 5. Fragment of ontology describing link between modules

- create a formal description of the relationship between „smart mediator” modules employing the user-created ontological descriptions of these modules, then
- automatically build an overall domain ontology using merging strategy[12], [13] without performing the „alignment” procedure, describing the structure of a firmware for a smart mediator, and finally
- generate code that connects modules into a whole program component.

On Fig. 6, the architecture of the developed system is presented. Green color is used to highlight components implemented within the scope of this research. SciVi platform here acts as an ontology repository and is used by a user for creating a formal description of connection modules in the form of DFD diagram.

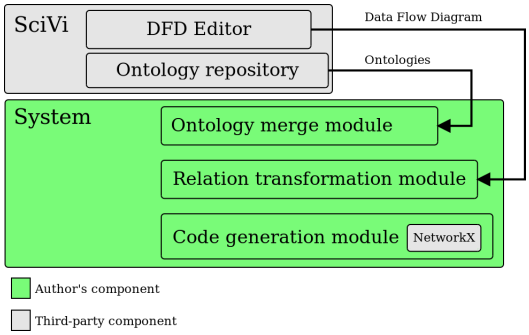

Fig. 6. Architecture of the developed system

General user-specialist workflow with system presented on Fig. 7. First (if necessary), all required program modules are created and their ontological descriptions are built. Then a DFD diagram describing connection these modules is created in SciVi environment. Based on this diagram and ontological descriptions of modules a general domain ontology is created, and after that a smart mediator code generator is executed.

System input data are a set of ontological descriptions of program modules available for inclusion in mediator's

Fig. 7. System usage

firmware and a formal description of connection these modules. As a result, the system generates a source code of a software module in C programming language, which after compilation and linkage with software modules chosen by user become a firmware for a „smart mediator" with goal to solve task of integrating a neurointerface into Internet of Things ecosystem.

Ontological descriptions of program modules are composed from a set of relations and basic concepts described in section VII. These ontological descriptions act as input data for the system, being combined into a whole general domain ontology that becomes the basis for further mediator's firmware code generation.

In addition, these ontological descriptions are used by SciVi platform to provide user with an interface of constructing formal description of program module connection.

Ontological descriptions aligned with program modules code are created (automatically or manually) and placed in SciVi repository from where they will be loaded during system's operation.

The system uses the JSON-based format of ontological descriptions – ONT. It's a proprietary format for representing ontologies supported by ONTOLIS[25], [26] and SciVi platform. Choosing it ensures interoperability between two systems.

Formal description of connection of modules is produced by a user in SciVi platform in form of a data flow diagram. Example of such description is presented on Fig. 8.

Pipeline presented above was designed for the task of brain activity analysis for processing recorded data and used in another work of the author[7].

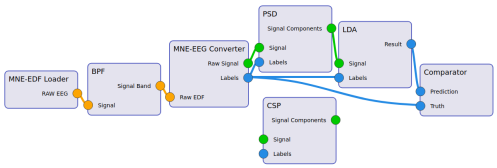

Fig. 8. DFD diagram example

After formal description is built it is exported from SciVi platform for further employment. This file uses the proprietary format based on JSON (JavaScript Object Notation) and should be integrated with ontological descriptions of modules mentioned above. This task is performed by a system's module that executes that transformation by accepting as input aforementioned file of proprietary format together with ontological descriptions of modules. It extracts information about parameters of modules and their connection, after which it creates ontology describing dependencies between data of modules which in turn is the result produced by this module.

The obtained ontology reflecting relations between parameters of modules gets combined with ontologies describing separate modules and general domain ontology is built describing internal structure of a smart mediator. This ontology represents different modules to be included into a firmware as well their data interconnections and in fact represents applied ontology for the task at hand.

On Fig. 9 example fragment of such ontology is presented.

Fig. 9. Example of domain ontology

This ontology gets transferred to the input of the code generator. Generator uses information about modules and their interdependencies extracted from ontology and builds a direct graph describing modules interconnected by data transfers. This graph is processed using topological analysis methods is then transformed such that a following partial order is established: vertex V_i describing module M_i precedes vertex V_j describing module M_j only when there is no path from vertex V_j to vertex V_i. If for some module pair establishing such an order is not possible (it means there is a data cycle dependency between modules) generator stops processing and displays an error message. In case of a successful transformation generator proceeds to generate a source code calling modules' entry points in the order defined by dependencies graph taking into account types of parameters and their direction (input / output).

As a final result the system produces a file with a source code in C language. This file contains instructions which ensure interaction between ontologically described modules according to given ontology. Example of such generated code is provided in Fig. 10. We'd like to point out that while such a code is by no means can be considered complex by a seasoned system programmer, for a user with little to no background in programming writing such a glue can pose a significant challenge. Automated generation allows us to provide the less acquaint users with a tools to solve their tasks in a more efficient way.

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

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 enable_led(int enable_led_i);

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);
enable_led(concentration_o);
```

Fig. 10. Example of generated code

Conclusion and future work

The paper is devoted to the urgent problem of automating the process of integrating different types of neurointerfaces into the IoT infrastructure and focuses on the development of methods and tools for unifying the ways of integrating neural interfaces with third-party systems on the principles of adaptability using ontology engineering methods.

The presented concept and formal model of the proposed solution are characterized by novelty, since a new ontology-driven method of firmware generation for the so-called "smart mediator" is proposed, which plays a major role in integrating a specific neural interface into arbitrary IoT infrastructure. The proposed approach makes it possible to reuse third-party ontological resources (BCI ontology) as a basis for building new ontologies of neurointerfaces and expands the existing functionality and ontological resources of the SciVi platform.

The proposed approach has been tested on several real-world task and has proven its viability, as evidenced by the existing act on the successful implementation of the developed tools in the practical activity of the Educational and Scientific Laboratory of Sociocognitive and Computational Linguistics of PSU. The certificate of ROSPATENT № 2023612016[27] on the state registration of the developed software has been received.

As a direction of further research we aim to concentrate on improving usability of implemented tools and expanding their abilities; one such point is introducing a recurring dependency between program modules of the smart mediator.

Acknowledgment

The authors express their deep gratitude to the Educational and Scientific Laboratory of Sociocognitive and Computer Linguistics, Faculty of Philology, Perm State University, for the provided equipment and support for the research.

References

[1] S. Huang and E. Tognoli, "Brainware: Synergizing software systems and neural inputs," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014, Hyderabad, India: Association for Computing Machinery, 2014, pp. 444–447, ISBN: 9781450327688. DOI: 10.1145/2591062.2591131. [Online]. Available: https://doi.org/10.1145/2591062.2591131.

[2] P. McCullagh, M. Ware, A. McRoberts, *et al.*, "Towards standardized user and application interfaces for the brain computer interface," in *Universal Access in Human-Computer Interaction. Users Diversity*, C. Stephanidis, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 573–582, ISBN: 978-3-642-21663-3.

[3] J. Huggins, C. Guger, E. Aarnoutse, *et al.*, "Workshops of the seventh international brain-computer interface meeting: Not getting lost in translation," *Brain-Computer Interfaces*, pp. 1–31, Dec. 2019. DOI: 10.1080/2326263X.2019.1697163.

[4] B. Allison, "The i of bcis: Next generation interfaces for brain–computer interface systems that adapt to individual users," in *Human-Computer Interaction. Novel Interaction Methods and Techniques*, J. A. Jacko, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 558–568, ISBN: 978-3-642-02577-8.

[5] I. A. Labutin and S. I. Chuprina, "Kontseptsiya postroeniya ontologicheski upravlyaemykh neirointerfeisov," Russian, *Intellektual'nye sistemy v nauke i tekhnike*, pp. 105–111, 2020.

[6] D. Lunev, S. Poletykin, and D. O. Kudryavtsev, "Brain-computer interfaces: Technology overview and modern solutions," *Sovremennye innovatsii, sistemy i tekhnologii - Modern Innovations, Systems and Technologies*, vol. 2, no. 3, pp. 0117–0126, Jul. 2022, ISSN: 2782-2818. DOI: 10.47813/2782-2818-2022-2-3-0117-0126. [Online]. Available: http://dx.doi.org/10.47813/2782-2818-2022-2-3-0117-0126.

[7] K. Ryabinin, S. Chuprina, and I. Labutin, "Ontology-driven toolset for audio-visual stimuli representation in eeg-based bci research," in *Proceedings of the International Conference on Computer Graphics and Vision "Graphicon", CEUR*, vol. 31, Keldysh Institute of Applied Mathematics, 2021, pp. 223–234. DOI: https://doi.org/10.20948/graphicon-2021-3027-223-234. eprint: http://ceur-ws.org/Vol-3027/paper21.pdf. [Online]. Available: https://keldysh.ru/papers/2021/prep_vw.asp?pid=9273&lg=e.

[8] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993, ISSN: 1042-8143. DOI: https://doi.org/10.1006/knac.1993.1008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1042814383710083.

[9] T. Gavrilova and V. Khoroshevskii, *Bazy znanii intellektual'nykh sistem: Uchebnik*. Piter, 2000, ISBN: 9785272000712. [Online]. Available: https://books.google.ru/books?id=biiTAAAACAAJ.

[10] S. I. Chuprina, "To adapt data fabric technology to visual analytics systems development in the field of digital medicine," in *Proceedings of the 33rd International Conference on Computer Graphics and Vision*, ser. GraphiCon-2023, Keldysh Institute of Applied Mathematics, 2023. DOI: 10.20948/graphicon-2023-405-416. [Online]. Available: http://dx.doi.org/10.20948/graphicon-2023-405-416.

[11] S. I. Chuprina, K. V. Ryabinin, D. V. Koznov, and K. A. Matkin, "Ontology-driven visual analytics software development," *Programming and Computer Software*, vol. 48, no. 3, pp. 208–214, Jun. 2022, ISSN: 1608-3261. [Online]. Available: https://doi.org/10.1134/S0361768822030033.

[12] S. I. Chuprina, K. V. Ryabinin, D. V. Koznov, and K. A. Matkin, "Ontologicheski upravlyaemye sredstva avtomatizatsii razrabotki prilozhenii vizual'noi analitiki," *Programmirovanie*, no. 3, pp. 70–77, 2022.

[13] S. I. Chuprina, "Using data fabric architecture to create personalized visual analytics systems in the field of digital medicine," *Scientific Visualization*, vol. 15(5), pp. 50–63, 2023. DOI: 10.26583/sv.15.5.05. [Online]. Available: https://api.semanticscholar.org/CorpusID:266356696.

[14] K. Ryabinin, S. Chuprina, and K. Belousov, "Ontology-driven automation of iot-based human-machine interfaces development," in *Computational Science – ICCS 2019*, J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro, *et al.*, Eds., Cham: Springer International Publishing, 2019, pp. 110–124, ISBN: 978-3-030-22750-0.

[15] S. I. Chuprina, "Adaptatsiya tekhnologii fabrik dannykh k razrabotke sistem vizual'noi analitiki v oblasti tsifrovoi meditsiny," in *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.

[16] S. J. R. Méndez and J. K. Zao, "Bci ontology: A context-based sense and actuation model for brain-computer interactions," in *SSN@ISWC*, 2018.

[17] K. Ryabinin and S. Chuprina, "High-level toolset for comprehensive visual data analysis and model validation," *Procedia Computer Science*, vol. 108, pp. 2090–2099, Dec. 2017. DOI: 10.1016/j.procs.2017.05.050.

[18] K. Ryabinin, S. Chuprina, and M. Kolesnik, "Calibration and monitoring of iot devices by means of embedded scientific visualization," Jan. 2018.

[19] C. Izumigawa, B. Taylor, and J. Sato, "Automated ontology generation," in *HCI International 2023 Posters*, C. Stephanidis, M. Antona, S. Ntoa, and G. Salvendy, Eds., Cham: Springer Nature Switzerland, 2023, pp. 433–438, ISBN: 978-3-031-36004-6.

[20] S. Elnagar, V. Y. Yoon, and M. A. Thomas, "An automatic ontology generation framework with an organizational perspective," *CoRR*, vol. abs/2201.05910, 2022. arXiv: 2201.05910. [Online]. Available: https://arxiv.org/abs/2201.05910.

[21] N. Sassi, W. Jaziri, and F. Gargouri, "How to evolve ontology and maintain its coherence - a corrective operations-based approach.," Jan. 2009, pp. 384–387.

[22] W. Jaziri, N. Sassi, and F. Gargouri, "Approach and tool to evolve ontology and maintain its coherence," *IJMSO*, vol. 5, pp. 151–166, May 2010. DOI: 10.1504/IJMSO.2010.033284.

[23] K. Ryabinin, "Metody i sredstva razrabotki adaptivnykh mul'tiplatformennykh sistem vizualizatsii nauchnykh eksperimentov," Dissertatsiya, IPM im. M.V.Keldysha, Moskva, 2015, pp. 1–207. eprint: https://keldysh.ru/council/1/2015-ryabinin/diss.pdf. [Online]. Available: https://library.keldysh.ru/diss.asp?id=2015-ryabinin.

[24] N. Seydoux, K. Drira, N. Hernandez, and T. Monteil, "Iot-o, a core-domain iot ontology to represent connected devices networks," in *Knowledge Engineering and Knowledge Management*, E. Blomqvist, P. Ciancarini, F. Poggi, and F. Vitali, Eds., Cham: Springer International Publishing, 2016, pp. 561–576, ISBN: 978-3-319-49004-5.

[25] S. Chuprina and D. V. Zinenko, "Ontolis: Adaptiruemyi vizual'nyi redaktor ontologii," Russian, in *Vestnik Permskogo universiteta. Seriya: Matematika. Mekhanika. Informatika.*, vol. 3 (22), 2013, pp. 106–110.

[26] S. Chuprina and O. Nasraoui, "Using ontology-based adaptable scientific visualization and cognitive graphics

tools to transform traditional information systems into intelligent systems," vol. 8, pp. 23–44, Jan. 2016.

[27] I. A. Labutin, "Sistema generatsii programmnogo obespecheniya dlya ustroistv interneta veshchei na baze ontologicheskogo opisaniya infrastruktury," Russian, Jan. 17, 2023.