Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 1 (23)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

UDC 004.43; 004.051; 811.93 DOI: https://doi.org/10.30837/ITSSI.2023.23.045

R. GAMZAYEV, M. TKACHUK

DEVELOPMENT OF PROBLEM-SPECIFIC MODELING LANGUAGE TO SUPPORT SOFTWARE VARIABILITY IN "SMART HOME" SYSTEMS

Building conceptual models for software design, in particular for high-tech applications such as smart home systems, is a complex task that significantly affects the efficiency of their development processes. One of the innovative methods of solving this problem is the use of domain-specific modeling languages (DSMLs), which can reduce the time and other project resources required to create such systems. **The subject** of research in this paper is approaches to the development of DSML for Smart Home systems as a separate class of *Internet of Things* systems. **The purpose** of this work is to propose an approach to the development of DSMLs based on a model of variability of the properties of such a system. The following **tasks** are being solved: analysis of some existing approaches to the creation of DSMLs; construction of a multifaceted classification of requirements for them, application of these requirements to the design of the syntax of a specific DSML-V for the creation of variable software in smart home systems; development of a technological scheme and quantitative metrics for experimental evaluation of the effectiveness of the proposed approach. The following **methods** are used: variability modeling based on the property model, formal notations for describing the syntax of the DSML-V language, and the use of the open CASE tool metaDepth. **Results:** a multifaceted classification of requirements for a broad class of DSML languages is built; the basic syntactic constructions of the DSML-V language are developed to support the properties of software variability of "Smart Home" systems; a formal description of such syntax in the Backus-Naur notation is given; a technological scheme for compiling DSML-V specifications into the syntax of the language of the open CASE tool metaDepth is created; the effectiveness of the proposed approach using quantitative metrics is experimentally investigated. **Conclusions:** the proposed method of developing a specialized problem-oriented language for smart home systems allows for multilevel modeling of the variability properties of its software components and provides an increase in the efficiency of programming such models by about 14% compared to existing approaches.

Keywords: domain-specific language; software; modeling; variability; Smart-home.

Introduction

Research actuality and motivation

In high-tech areas of IT development, such as Smart-home systems, mobile applications [1], and distributed information systems based on cloud platforms [2], innovative approaches to software development are required. This, in turn, involves the use of the latest design methods and technologies: domain-driven development, model-driven architecture, and the use of knowledge-based software tools and technologies [3]. The main practical goal of applying such approaches to software development is to reduce the time and other project resources required to meet the requirements of different groups of future users. To achieve this goal, methods and technologies for building domain-specific programming languages (DSLs) [4] and domain-specific modeling languages (DSMLs) [5] for a particular domain of software development have been successfully used recently, which makes it possible to ensure more efficient software development and maintenance processes. Another significant advantage of using DSL and DSML is the ability to support the variability and adaptability of relevant software solutions consistently and effectively, which in modern software engineering has acquired a common definition – "variability" of software [6]. It is the provision of software variability properties that is a critical factor in automating the process of creating software product lines (SPL), which are complex software applications that have a common and manageable set of functions to meet the requirements of users of a particular segment of the IT market and are built from a common set of project assets (components) in a predetermined manner [7]. Thus, the above confirms the scientific relevance and practical focus of this study.

Brief review of some related publications

A considerable number of recent publications are devoted to the problems of developing approaches to the creation and application of DSL and DSML. In the course of the study, a brief review of works is made and special attention is paid to the issues of ensuring software variability in SPL.

Thus, work [8] analyzes and summarizes the features of such modern software applications as Internet

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 1 (23)

of Things systems, Smart Home systems, Embedded Control Software and some others at a fairly high methodological level. The authors of this paper propose to designate them as smart service systems (SSS). It is also shown that it is to automate the development and subsequent software implementation of SSS systems that it is advisable to use appropriate subject-oriented modeling languages (DSML), which also ensures the achievement of a certain level of software variability in such systems. The authors propose an approach to creating a syntax, grammatical rules and a compilation mechanism for such a language. The effectiveness of this approach has been tested on the example of an intelligent system for controlling thermostats in a Smart Home.

Paper [9] considers an original approach to creating sets of semantically related DSML languages for a particular domain, which are considered as a special type of software product lines (SPL) and for the development of which well-known domain engineering methods and technologies can be applied. In particular, the authors propose an appropriate metamodel for describing the entire DSML family of languages, which also provides support for the properties of variability in design decisions in the process of developing the target software system, and also offers an appropriate CASE tool for automating the proposed approach.

Study [10] focuses on how the use of DSML can increase the level of variability of SPLs at the stage of generating the source code of their individual components. The authors propose to apply the Model Driven Engineering (MDE) software development methodology consistently. To do this, the paper introduces a methodology for building a DSML using the concept of a metamodel, which is then transformed into an equivalent set of feature diagrams, which then allows the use of a well-known and effective method of analyzing and modeling various domain models FODA (Feature-oriented Domain Analysis) (for example, in [11]). This approach is illustrated by the development of software for a learning management system, where it is the support of a sufficient level of variability that allows you to customize flexibly its functionality to the needs of different groups of students and teachers. The study [12] also shows positive results of creating and applying SPL construction methods and MDE principles to develop a multi-level information system for the purpose of communication among university students.

And finally, a new publication [13] (published in 2022), which addresses an important methodological problem of ensuring the possibility of transforming a domain model built using a particular DSML into a set of corresponding property models, i.e., FODA models, in the process of developing any SPL, attracts special attention. The authors propose to use a set of specific tags to annotate the text of the description of the origin model in DSML and a system of rules for its transformation to obtain the corresponding property models that can be represented in the form of XML specifications.

It should be noted that, despite a considerable number of publications on the development and application of DSML, the following issues remain insufficiently developed: analysis and structuring of the set of requirements for creating DSML, use of formalized notations to describe DSML syntax, selection and methodology of applying certain quantitative metrics to assess the effectiveness of using a particular DSML in a particular subject area.

The purpose of this research is to study some of the existing approaches to creating promising DSMLs; to build one of the possible generalized classifications of requirements for them; to apply the identified requirements to design the syntax of a specific DSML to support the processes of creating variable software in Smart Home systems; to develop a technological scheme and quantitative metrics for experimentally evaluating the effectiveness of the proposed approach.

Materials and methods

To achieve the research objective, it was first necessary to formulate the basic requirements for creating an effective DSML in any problem domain. Based on the analysis of the well-known work [14], as well as our own positive experience [15, 16], such requirements are presented in a structured form in Figure 1.

The Formal requirements define the basic rules for choosing the notation and forms of describing the syntax and grammar of the target DSML, namely (see Figure 1):

requirement F1 – it is necessary that the notation for a modeling language have the means for accurate and complete specification of its syntax and corresponding formal grammar (e.g., Backus-Naur notation, etc.);

requirement F2 – the concepts used to define DSML should be consistent with the general concepts used in modern software development, such as object-oriented approach, domain-driven design, model-driven architectures, etc.;

requirement F3 – the modeling language should allow building models at different levels of abstraction,

Сучасний стан наукових досліджень та технологій в промисловості. 2023. №1 (23)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

i.e., support hierarchical sets of such models with the possibility of their step-by-step detailing, i.e., support mechanisms for building metamodels and meta-metamodels;

requirement F4 – the modeling language should contain concepts of the system's domain of application to maintain the integrity of models created with the help of DSML and ensure their reusability.

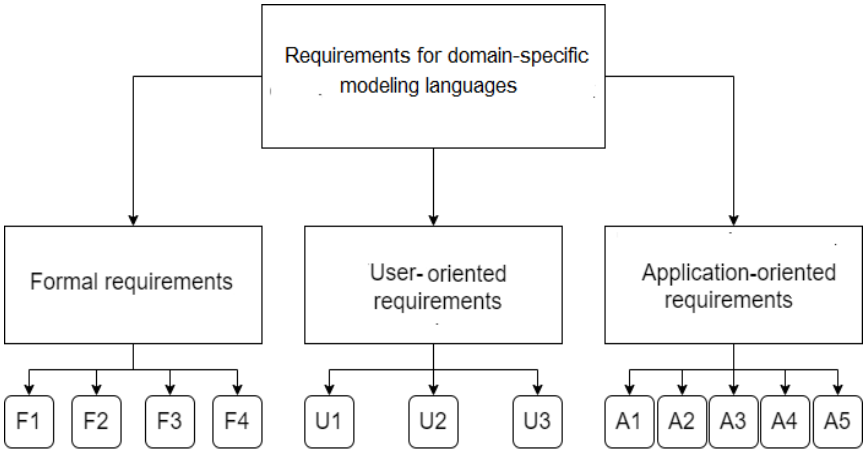

Fig. 1. Classification of requirements for domain-specific modeling languages DSML

The next group of requirements in Figure 1 are User-oriented requirements: these are domain experts, system analysts, and software developers. These requirements include:

requirement U1 – the syntax and grammatical constructions of the target DSML should correspond to the semantic concepts and business processes that its future users already work with to the maximum extent possible;

requirement U2 – the modeling language should provide a manageable set of basic functionalities sufficient to create domain models with regard to the variability of their properties, for example, in the form of models in the FODA notation;

requirement U3 – grammatical rules for defining the semantics of the target DSML must meet the functional requirements of its future end users (see above).

Finally, the last group of requirements is focused on supporting the ability to configure and adapt the target DSML to the specifics of the relevant subject area (Application-oriented requirements), and they can be formulated as follows (see figure 1):

requirement A1 – the modeling language should provide opportunities for describing additional objects and bindings that are specific to certain user groups, i.e., support certain mechanisms for extending language structures and rules for their construction;

requirement A2 – based on the correct specification of the problem task performed in DSML, it should be possible to generate source code in one of the common programming languages: Java, C#, C++, etc.;

requirement A3 – to support the effective work of end users for the practical application of DSML, it is necessary to have appropriate CASE tools with a friendly user interface;

requirement A4 – for further programmatic implementation of components in the target software based on the obtained domain model (see above), it is necessary to be able to use DSML compatibly with the functionality of the corresponding DSL compiler;

requirement A5 – appropriate quantitative and/or qualitative metrics should be developed and applied to evaluate the effectiveness of the DSML in a particular domain.

In creating the syntax (grammar) of the proposed DSML, the main requirements of the above were met: **F1–F4, U1–U3, A1–A5**.

The next important concept in the context of this study is the concept of software variability [6]. In the design of SPL, an appropriate variability model of its properties is developed [7], which can reflect the ability to change them flexibly at different phases of the life cycle: domain analysis, architectural design, coding, and maintenance; for different project assets: variability of software requirements, variability of design options, variability of software components, and variability of test artifacts. Figure 2 shows a generic UML entity diagram, which is a metamodel of project asset variability in some SPL development process [17].

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 1 (23)

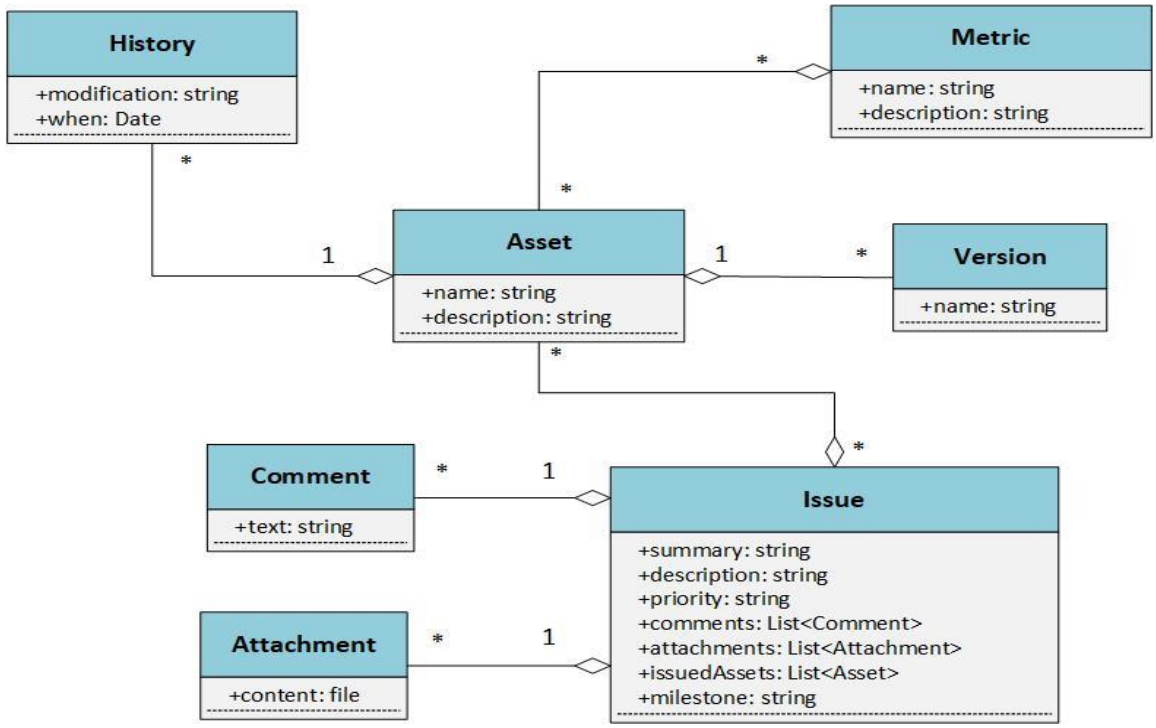

Fig. 2. Metamodel of project asset variability in SPL development

Figure 2 shows the main Asset entity, which summarizes all possible artifacts in the development of the LSP, and it is linked to the History entity, which is responsible for tracking changes made to some Asset objects, and the Issue entity, which contains information about what specific changes were made, in which object, and by whom. The "Asset" entity is linked to the "Metric" entity, which contains metrics for describing and defining the parameters of individual assets, as well as to the "Version" entity, which stores the current versions for each modification of the "Asset" objects. The information contained in the above-mentioned basic entities of the variability metamodel can be expanded by defining records in additional entities, such as Comment and Attachment.

For a better understanding of the principles of variability, let us consider more specifically the task of modeling variability in the development of a hardware and software system such as Smart Home Systems (SHS). Paper [16] states that the variability of Smart Home Systems can manifest itself in the following features (in general):

1) variability of the user interface; a typical set of options may, for example, include a graphical interface via a control panel, via a web interface on a personal computer or via an interface on a smartphone, and in some cases, voice control (Amazon Alexa, Apple HomePod, etc.)

2) variability of hardware and software components; this feature implies the presence of different groups of connected smart devices: lighting, water and heat supply, control of electronic devices (e.g., TV or audio speakers), various motion sensors, sensors responsible for opening/closing objects (doors/windows), etc.;

3) variability of the level of controllability and reliability of operation; the home automation system can support different levels of these indicators, for example, the basic level can provide self-diagnostics and error reports. Higher levels can contain redundant system components for the most important functions (light, water, gas, opening and closing doors), perform tasks for backing up and updating components through cloud services (Google Cloud, Microsoft Azure, etc.).

Figure 3 shows a FODA model that represents the variable components of a Smart Home system [15] and can be used to comply with the above-mentioned variability features in the development of a corresponding DSML (see requirements **A1–A5** above).

Based on this FODA model, a syntax and grammar for the DSML language with variability support (hereinafter referred to as DSML-V) can be created in the process of designing a LSP for Smart Home tasks. Due to the complexity of this task, we decided to apply

Сучасний стан наукових досліджень та технологій в промисловості. 2023. №1 (23)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

a bottom-up rather than top-down approach to this development [16], i.e., first, it is necessary to develop the most important individual language constructs at the lower logical levels of modeling, which then need to be generalized at the upper level of description of the corresponding formal grammar.

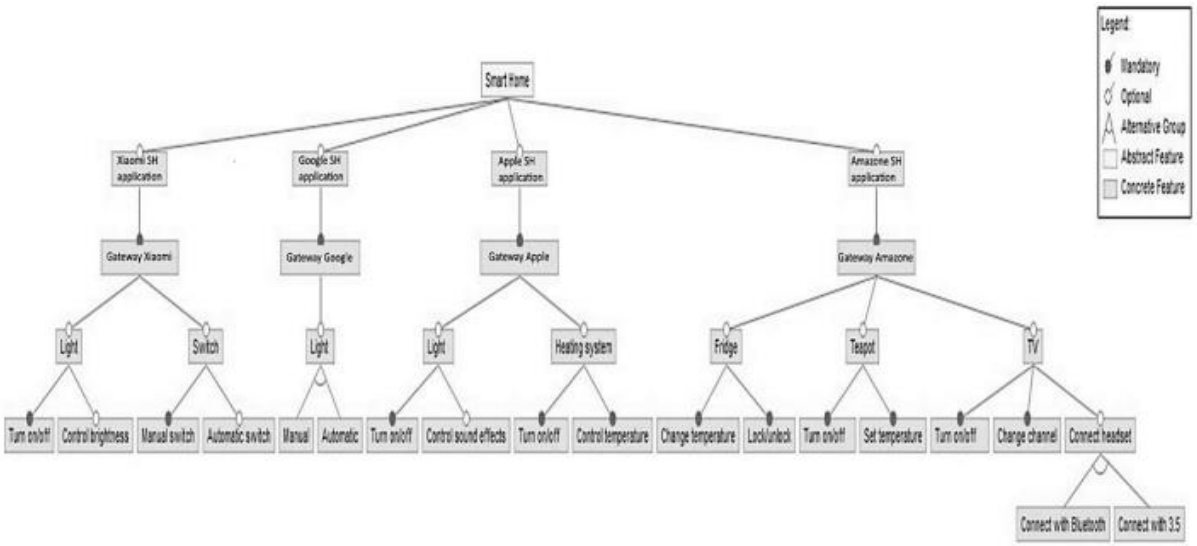

Fig. 3. Variable FODA-model of Smart Home system features

The first task is to determine how the proposed DSML-V language will be used and extended by its users, taking into account the list of requirements **F1–F4**. For this purpose, special tags were used to identify language elements that should be extensible and to introduce restrictions on a certain degree of extensibility. The list of possible tags was taken from the syntax of the open source metaDepth toolkit [18]. Figure 4 shows the definition of the textual syntax of the DSML-V language for the Smart Home system:

```
01) strict Model SmartHome@1 {
02)     strict Node Device {
03)             variability@1: double = 0;
04)             isTurnedOn: boolean = false;
05)             positiveVar: $self.variability > 0$
06)     }
07)}
```

Fig. 4. Code snippet to describe the basic syntax of DSML-V

The *SmartHome* model does not have an ontological type (that is, it is not a instance of another model), so it is declared using the *Model* keyword (code line 1). This model defines the *Device* class using the *Node* keyword (line 2), which has certain attributes (lines 3, 4). It is in *Device* that the key attribute *variability* is introduced, which will help solve the problem of supporting and ensuring variability.

The level of model variability is indicated by the "@" symbol. The "$" symbol should be used to define certain constraints on model attributes. For example, the *positiveVar* constraint requires that the *$variability* attribute be more than 0 (code line 5). Figure 5 (code line numbering continued) shows an instance of creating the *XiaomiShApplication* model and a *LightSwitcher* instance of this model:

```
08)SmartHome XiaomiShApplication {
09)  Device GatewayXiaomi {
10)             variability = 0.4;
11)     }
12)}
13)14)XiaomiShApplication LightSwitcher {
14)  GatewayXiaomi light {
15)     isTurnedOn = true;
16)  }
17)}
```

Fig. 5. Example of extending the basic DSML-V syntax

The *XiaomiShApplication* model created in lines 8–12 has the *SmartHome* ontological type used instead of the *Model* keyword. Line 10 sets the initial variability for this model. If the value of the attribute is negative, this will lead to a model verification error, because the corresponding constraint was set in Figure 4 (line 5). Lines 13–17 create an instance

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 1 (23)

of *LightSwitcher* of the *XiaomiShApplication* model, as well as an instance of light of the *GatewayXiaomi* model, and set the value of its *isTurnedOn* attribute.

Figure 6 shows an instance of creating a *XiaomiShApplication* model (lines 01–03) and a *LightSwitcher* instance (lines 04–06) using the developed text syntax:

```
01)XiaomiShApplication {
02)GatewayXiaomi(0.4)
03)}
04)XiaomiShApplication LightSwitcher {
05)GatewayXiaomi light(true)
06)}
```

Fig. 6. An instance of creating a *XiaomiShApplication* model

Now, summarizing the above particular examples of the proposed syntax of the DSML-V language, we can describe the corresponding formal grammar for this language in the Backus-Naur notation [19], which is shown in Figure 7.

```
01)<metamodel> ::= <model lvl 1> <EOL> <model lvl 0>
02)<model lvl 1> ::= <identifier> "{" <identifier> "("
    <variability> ")" "}"
03)<identifier> ::= <letter> | <identifier> <letter> |
    <identifier> <decimal digit>
04)<letter> ::= [A-Za-z]
05)<decimal digit> ::= [0-9]
06)<variability> ::= <decimal digit> | <decimal digit> "."
    <decimal digit>
07)<model lvl 0> ::= <identifier> <identifier> "{"
    <identifier> <identifier> "(" <boolean> ")" "}"
08)<boolean> ::= "true" | "false"
```

Fig. 7. Syntax of the DSML-V language in the Backus-Naur notation

As you can see from Figure 7, the metamodel (line 1) consists of two models: the level 1 model (line 2) and the level 0 model (line 7), whose syntax pattern is defined in Figure 4. The models of each level contain identifiers (line 3) and other auxiliary syntactic constructs. It is important to note that the variability (line 6) in this case corresponds to a fractional number.

We propose to extend the templates to customize the allowed extensions for a specific syntax. To do this, let's introduce new keywords: *flingext* – to allow the declaration of new attributes without an ontological type; *lingext* – to allow the addition of new object classes without an ontological type; *constraints* – to declare constraints; *super* – to define new inheritance relationships for object classes. In addition, two additional keywords allow you to define how these extensions should be created at meta-level 0: *flinginst* for field instances and *linginst* for object class instances.

Figure 8 shows the definition of a specific extensible text syntax with the addition of suggested keywords:

```
01)Syntax for SmartHome[".smarthome"] {
02)    template@1 TSmartHome for Model SmartHome:
03)            "id '{' (&TDevice | lingext)* '}'"
04)    template@1 TDevice for Node Device:05)    "id
('(' #variability ')')? ('extends' supers)?
05)      (';' | '{' (flingext ';' | constraint)* '}')"
06)    template@2 DeepDevs for Model SmartHome:
07)            "typename id '{' &DeepDev* '}'"
08)    template@2 DeepDev for Node Device:
09)            "typename id '(' isTurnedOn flinginst* ')'"
```

Fig. 8. Code fragment for defining extensible syntax

The use of *lingext* (line 03) allows you to define new object classes at meta-level 1. The expression on line 04 enables inheritance between *Device* instances, and the expression on line 07 allows you to define new fields and constraints in *Device* instances. In addition, line 09 allows you to create instances of these additional fields in indirect *Device* instances.

Figure 9 shows an example of creating the *GatewayXiaomi* model using the inheritance mechanism:

```
01)GatewayXiaomi {
02)    BaseController(0.25) {
03)            isTurnedOn: boolean = false;
04)    }
05)    PressureController extends BaseController;
06)    LightController extends BaseController {
07)            light: Light;
08)    }
09)    Node Light {
10)
10)            brightness: double;
11)    }
12)}
```

Fig. 9. An example of creating a *GatewayXiaomi* model

For the possible software implementation of the proposed DSML-V language, an appropriate tool environment is required, a fairly complete

Сучасний стан наукових досліджень та технологій в промисловості. 2023. №1 (23)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

overview of which can be found, for example, in [18]. We have analyzed the functionality of several tools for creating DSML.

- *Clooca* – is a development environment that allows you to create DSML and their code generators (JavaScript, Python, PHP, Ruby, etc.), is used according to the Software as a Service (SaaS) architectural model, allows you to create a description of the syntax of the corresponding DSML in JSON text format using a visual graphical interface where the user works with tabular forms and diagrams to describe objects, relationships and properties in a particular domain. Figure 10 shows an example of such an interface, which is quite typical for other similar CASE tools (see below).

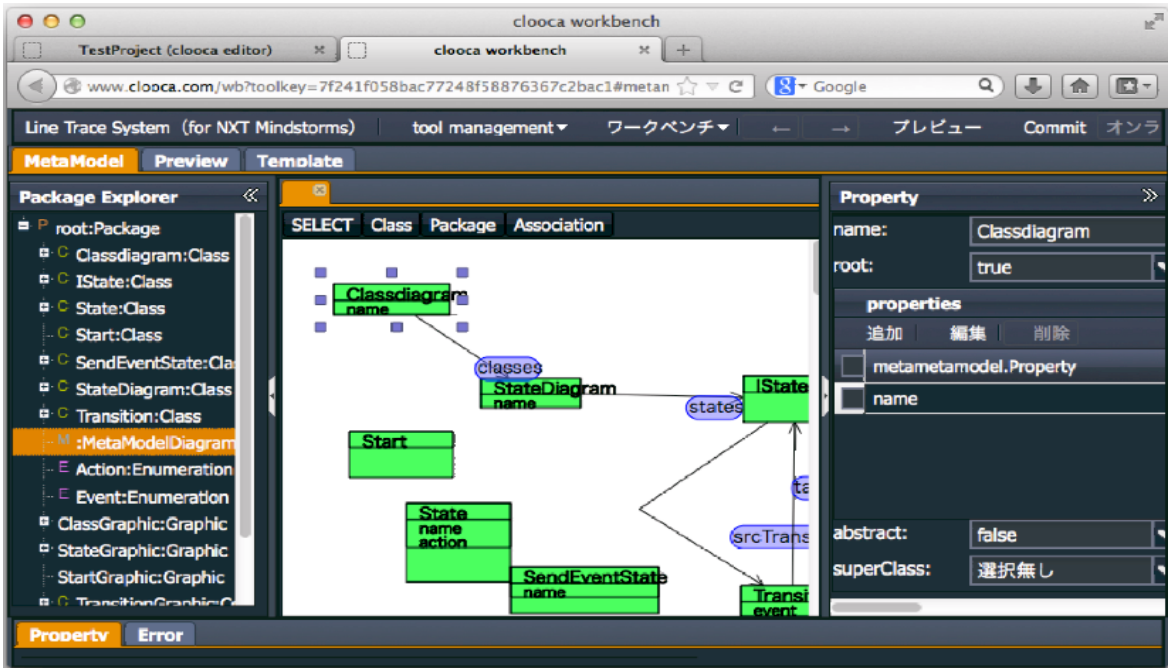

Fig. 10. An example of a developer interface in the Clooca CASE tool

- *Generic Modeling Environment* (GME) is a software tool in the form of a web-based application for creating DSMLs for various purposes, which can be further extended by adding new properties that are specific to the selected domain. GME has a well-developed visual user editor in which you can create appropriate metamodels, which can be further exported in XML format or stored in external distributed noSQL databases [20] for reuse.
- *Eclipse Modeling Framework* (EMF) – is an open-source, extensible, Java-based domain modeling platform that offers advanced functionality in the form of its own IDE. EMF allows you to create or import a suitable domain data model, as well as tables and XMI schema for such a model. The Java source code generated from this model can be edited by the user to develop the target application.

Along with the above tools for creating DSML, we have considered the MetaDepth toolkit [21], which uses ontological specifications to describe syntax and grammatical rules, has functionality that is largely similar to the already analyzed Clooca, GME, and EMF environments, but, unlike them, allows building systems with an arbitrary number of meta-levels of modeling a particular domain that meets the requirements of **F1–F4** (see the above diagram at Figure 1).

That is why this tool was chosen to develop a technological scheme for the implementation of the proposed DSML-V language, shown in IDEF0 notation at Figure 11.

In this diagram, the first functional block (FB), designated as A0, compiles a program created by experts in a particular subject area using the proposed DSML-V syntax, as well as taking into account the variable FODA model of the Smart Home system (see Figure 3). The output of this FB is the corresponding metamodel in the .mdepth format [21]. The second FB A1 processes the resulting metamodel for a subject-oriented metamodeling language using the metaDepth API and Multi-Level Language tools. The output of FB A1 is a specific specification of a subject-oriented metamodeling language.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 1 (23)

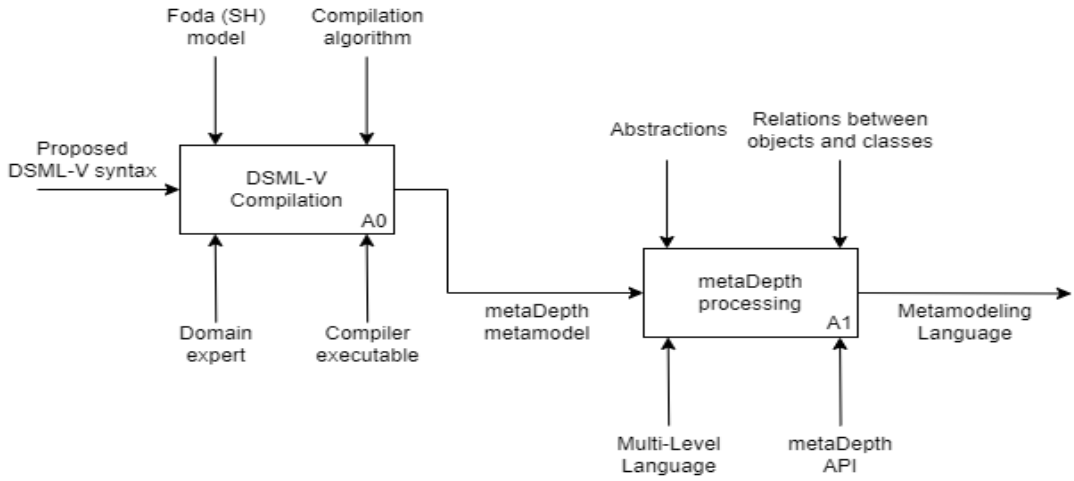

Fig. 11. Compilation flowchart for DSML-V language

Experimental results and their discussions

To investigate the effectiveness of using the DSML-V language, it is proposed to compare its syntax with the built-in syntax of the metaDepth tool [21] to build a conditional model in the Smart Home domain based on the variable FODA model of such a system shown in Figure 3.

As an example, let's consider the task of creating a model of variable software for controlling the equipment of one room, which is part of the Smart Home system. A variety of sensors can be installed in such a room: light, pressure, air, motion, etc. They can be controlled by a common control mechanism – a specific gateway. Each such gateway has its own variability index, as well as special attributes. A light sensor, for example, should turn on the night light and turn off the daylight depending on the time of day. Accordingly, there can be several devices using such a gateway. To illustrate the user interface for managing the configurations of hardware and software resources in such a system, you can use the functionality of the open CASE tool LabView with the built-in visual modeling environment Home I/O [22]. The example of a possible interface is shown in Figure 12.

Fig. 12. An example of a fragment of the user interface of the Smart Home system

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2023. №1 (23) ISSN 2524-2296 (online)

Figure 13 shows the code snippet required to create the described model using the built-in metaDepth syntax:

```
01) strict Model SmartHome@2 {
02)     strict Node Device {
03)     variability@1: double;
04)     isTurnedOn: boolean = false;
05)     positiveVar: $self.variability > 0$
06)     extid@1: $self.newFields().exists(f | f.isId())$
07)     }
08) }
09) SmartHome XiaomiShApplication {
10)  Device GatewayXiaomi {
11)             variability = 0.4;
12)     }
13) }
14) XiaomiShApplication LightSwitcher {
15)  GatewayXiaomi light {
16)     isTurnedOn = false;
17)  }
18)  GatewayXiaomi switch {
19)     isTurnedOn = true;
20)  }
21) }
```

Fig. 13. Code fragment for creating a model in the metaDepth language

Figure 14 shows the code fragment necessary to create the described model using the proposed syntax of the DSML-V language, which is presented in Figure 7 and Figure 8.

```
01)Syntax for SmartHome[".smarthome"] {
02)     template@1 TSmartHome for Model SmartHome:
03)             "id '{' (&TDevice | lingext)* '}'"
04)     template@1 TDevice for Node Device:05)     "id
('(' #Variability ')')? ('extends' supers)?
06)                     (';' | '{' (flingext ';' | constraint)* '}')"
07)     template@2 DeepDevs for Model SmartHome:
08)             "typename id '{' &DeepDev* '}'"
09)     template@2 DeepDev for Node Device:
10)             "typename id '(' action flinginst* ')'"
11)
12)XiaomiShApplication {
13)     GatewayXiaomi(0.4)
14)}
15)XiaomiShApplication LightSwitcher {
16)     GatewayXiaomi light(false)
17)  GatewayXiaomi switch(true)
18)}
```

Fig. 14. Code snippet for creating a model in DSML-V

To compare the code snippets in Figure 13 and Figure 14, you can use such an efficiency indicator as the number of lines of code (LOC). Then the corresponding efficiency factor for the proposed DSML-V syntax, $K_{ef}(1)$, can be calculated using the following formula:

$$K_{ef}(1) = \frac{LOC_{MD} - LOC_{DSML}}{LOC_{MD}} \times 100\%, \qquad (1)$$

where LOC_{MD} – number of code in the metaDepth; LOC_{DSML} – number of code lines in the DSML-V language.

The indicator $K_{ef}(1)$ determines the advantage of using the appropriate DSLM-V syntax to reduce the clinesost of describing the model of the Smart Home system. The calculation of this indicator by formula (1) is

$$K_{ef}(1) = \frac{21-18}{22} \times 100\% = 14.28\%.$$

As you increase the number of model instances created, the efficiency of using the proposed syntax also increases, since the number of lines required to create each subsequent model instance decreases. For example, in Figure 13, the 8 lines of code are required to create instances of the *GatewayXiaomi* light, switch model (lines 14–21), and to create the same instances in Figure 14 using the proposed syntax, the 7 lines are required (lines 12–18).

Then the second possible performance indicator, $K_{ef}(2)$, can be calculated using the following formula:

$$K_{ef}(2) = \frac{LOC_{E\_MD} - LOC_{E\_DSML}}{LOC_{E\_MD}} \times 100\%, \qquad (2)$$

where LOC_{E_MD} is the number of code lines required to create model instances (lines 14–21 in Figure 13) using the built-in metaDepth syntax; LOC_{E_DSML} is the number of code lines required to create model instances (lines 12–18 in Figure 14) using the proposed syntax.

For the described case, this indicator, calculated by formula (2), is:

$$K_{ef}(2) = \frac{8-7}{8} \times 100\% = 12.5\%.$$

In order to obtain a weighted average estimate of the final effectiveness of the proposed DSML-V syntax K_{avg}, the assessment described by the following formula can be applied:

$$K_{avg} = \frac{K_{ef}(1) + K_{ef}(2)}{2}, \qquad (3)$$

where $K_{ef}(1)$ is the efficiency indicator from formula (1); $K_{ef}(2)$ is the efficiency indicator from formula (2).

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 1 (23)

The value of the weighted average estimate of the final efficiency, calculated using formula (3), is

$$K_{avg} = \frac{14.28 + 12.5}{2} = 13.39\%.$$

Thus, the approximate weighted average efficiency estimate (13.39%) of the use of the developed problem-oriented variability modeling language DSML-V obtained in this study basing on the formulas (1)–(3) proves its superiority over such a common open tool for general DSML language development as metaDepth [21].

Conclusions and further work

The proposed work substantiates the relevance of solving the scientific and technical problem of building conceptual models for the design of high-tech applications such as Smart Home systems, in particular, using domain-specific modeling languages (DSML), which can significantly reduce the time and other project resources required to create these systems. To this end, the authors have identified and accomplished the following tasks Analyzing some existing approaches to the development of DSML, proposing a multifaceted classification of requirements for them, based on which the basic syntactic structures of the target DSML-V language were created, which provides support for the variability properties of Smart Home systems; formally describing such a syntax in the Backus-Naur notation; developing a technological scheme for compiling metamodels in DSML-V into the syntax of the language of the open CASE-tool metaDepth. A study of the effectiveness of this approach using selected quantitative metrics was conducted, which proved that the proposed method of developing a specialized problem-oriented language for Smart Home systems allows for multilevel modeling of the variability properties of its software components and provides an increase in the efficiency of programming such models by about 14% compared to some existing approaches.

To continue this research, it is planned to extend the proposed DSML-V syntax with the addition of more complex, comprehensive, and subject-oriented constructs, as well as to implement fully programmatically a prototype compiler for this language for modeling Smart Home systems.

References

1. Joanna, F., DeFranco, a, Mohamad, Kassab. (2021), "Smart Home Research Themes: An Analysis and Taxonomy", *Procedia Computer Science*, Vol. 185. P. 91–100. DOI: https://doi.org/10.1016/j.procs.2021.05.010
2. Davydov, V., & Hrebeniuk, D. (2020), "Development of the methods for resource reallocation in cloud computing systems", *Innovative Technologies and Scientific Solutions for Industries*, 3 (13), P. 25–33. DOI: https://doi.org/10.30837/ITSSI.2020.13.025
3. Gamzayev R.O., Tkachuk M.V., Shevkoplias D.O. (2020), "Knowledge-oriented Information Technology to Variability Management on Domain Analysis Stage in Software Development", *Advanced Information Systems*, Vol. 4, No. 4, P. 39–47. DOI: https://doi.org/10.20998/2522-9052.2020.4.06
4. D. Karagiannis, H.C. Mayr, J. Mylopoulos. (2016), "Domain-Specific Conceptual Modeling: Concepts, Methods and Tools", *Springer, Berlin*, 606 p.
5. Tomaž Kos, Marjan Mernik and Tomaž Kosar. (2022), "Evolution of Domain-Specific Modeling Language: An Example of an Industrial Case Study on an RT-Sequencer", *Appl. Sci*., 12 (23), 12286. https://doi.org/10.3390/app122312286
6. Berger, Th., Chechik, M., Kehrer, T. (2019), "Software Evolution in Time and Space: Unifying Version and Variability Management", *Dagstuhl Seminar Reports*, Vol. 9, Issue 5, P. 1–31.
7. Jaffari, A., Lee, J., Kim, E. (2021), "Variability Modeling in Software Product Line: A Systematic Literature Review", *Studies in Computational Intelligence*, vol 930. Springer, Cham. https://doi.org/10.1007/978-3-030-64773-5_1
8. Huber, R., Pueschel, L., Roeglinger, M. (2019), "Capturing smart service systems: Development of a domain-specific modelling language", *Inf. Systems Journal*, Volume 29, Issue 6, P. 1207–1255.
9. Leila Samimi-Dehkordi, Bahman Zamani, Shekoufeh Kolahdouz-Rahimi. (2019), "Leveraging product line engineering for the development of domain-specific metamodeling languages", *Journal of Computer Language,s* Volume 51, P. 193–213. DOI: https://doi.org/10.1016/j.cola.2019.02.006
10. Maouaheb Belarbi (2018), "A methodological framework to enable the generation of code from DSML in SPL", *Proceedings of the 22nd International Systems and Software Product Line Conference (SPLC 2018)* – Vol. 2, P. 64–71. DOI: https://doi.org/10.1145/3236405.3236426

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2023. №1 (23) *ISSN 2524-2296 (online)*

11. Eko K. Budiardjo, Elviawaty M. Zamzami. (2014), "Feature Modeling and Variability Modeling Syntactic Notation Comparison and Mapping", *Journal of Computer and Communications*, Vol. 2, No. 2, P. 102–108. DOI: 10.4236/jcc.2014.22018
12. Vale, A., Fernandes, S., Magalhães, A. P. (2019), "Towards a customizable Student Information System integrating MDD and SPL (S)", *Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering (SEKE 2019)*, Lisbon, Portugal, July 10–12 2019, P. 98–106. DOI: https://doi.org/10.18293/SEKE2019-089
13. Cunha, A., Fernandes, S. and Magalhães, A. (2019), "Integrating SPL and MDD to Improve the Development of Student Information Systems", *Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, P. 197–204. DOI: https://doi.org/10.5220/0007711201970204
14. Maouaheb Belarbi and Vincent Englebert (2022), "Transforming Domain Specific Modeling Languages into Feature Models", *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, P. 137–146. DOI: https://doi.org/10.5220/0010772000003119
15. Frank, U., (2010), "Outline of a method for designing domain-specific modelling languages", *Research Reports, Institut für Informatik und Wirtschaftsinformatik (ICB)*, Universität Duisburg-Essen, Germany, P. 1–76.
16. Rustam Gamzayev, Mykola Tkachuk and Oleksandr Nelipa. (2021), "Domain-Specific Language for Adaptive Development of "Smart-Home" Applications", *Proceedings of the 1st International Workshop on Information Technologies: Theoretical and Applied Problems 2021 (ITTAP-2021)* Ternopil, Ukraine, November 16-18, 2021, CEUR-WS.org/Vol-3039, P. 154–165.
17. Rustam Gamzayev, (2023), "A Methodology for Development and Usage of Problem-oriented Modeling Languages in "Internet Of Things" Systems", *Proceedings of the V International Scientific and Practical Conference Stockholm, Sweden (February 07–10, 2023)*, P. 603–608. DOI: https://doi.org/10.46299/ISG.2023.1.5
18. Cavalcanti Y.C., Machado I.C., Lobato L.L. et al. (2011), "Towards Metamodel Support for Variability and Traceability in Software Product Lines", *Proceedings of the 5th International Workshop on Variability Modelling of Software-Intensive Systems, Namur, Belgium (January 27–29, 2011)*, P. 1–10. DOI: https://doi.org/10.1145/1944892.1944898.
19. Bashroush R., Garba M., Rabiser E. et al. (2017), "CASE Tool Support for Variability Management in Software Product Lines", *ACM Computing Surveys*, 50 (1), P. 1–45. DOI: https://doi.org/10.1145/3034827
20. Quinlan, D, Wells, JB & Kamareddine, F., (2019), "BNF-Style Notation as It Is Actually Used", *Proceedings of the 12th Conference on Intelligent Computer Mathematics* 2019, Prague, Czech Republic, P. 187–204. DOI: https://doi.org/10.1007/978-3-030-23250-413
21. Mazurova, O., Naboka, A., Shirokopetleva, M. (2021), "Research of ACID transaction implementation methods for distributed databases using replication technology", *Innovative Technologies and Scientific Solutions for Sndustries*, № 2 (16), P. 19–31. DOI: https://doi.org/10.30837/ITSSI.2021.16.019
22. Juan de Lara, Esther Guerra, Jesús Sánchez Cuadrado. (2015), "Model-driven engineering with domain-specific meta-modelling languages", *Software and Systems Modeling (Springer)*, Vol 14(1). P. 429–459.
23. A. Philippot, B. Riera, M. Koza, et al. (2017). "HOME I/O and FACTORY I/O: 2 Pieces of innovative PO simulation software for automation education", *European Association for Education in Electrical and Information Engineering Annual Conference (EAEEIE)*, Grenoble, France, P. 1–6. DOI: https://doi.org/10.1109/EAEEIE.2017.8768639

Received 22.02.2023

Відомості про авторів / About the Authors

Гамзаєв Рустам Олександрович – кандидат технічних наук, доцент, Харківський національний університет імені В. Н. Каразіна, доцент кафедри моделювання систем і технологій, Харків, Україна; e-mail: rustam.gamzayev@karazin.ua; ORCID: https://orcid.org/0000-0002-2713-5664

Ткачук Микола В'ячеславович – доктор технічних наук, професор, Харківський національний університет імені В. Н. Каразіна, завідувач кафедри моделювання систем і технологій, Харків, Україна; e-mail: mykola.tkachuk@karazin.ua; ORCID: https://orcid.org/0000-0003-0852-1081

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 1 (23)*

Gamzayev Rustam – PhD, associate professor, Kharkiv National University named after V. N. Karazina, associate professor of the Department of Systems and Technologies Modeling Kharkiv, Ukraine.

Tkachuk Mykola – doctor of technical sciences, professor, Kharkiv National University named after V. N. Karazina, head of the Department of Systems and Technologies Modeling, Kharkiv, Ukraine.

РОЗРОБКА ПРОБЛЕМНО-ОРІЄНТОВАНОЇ МОВИ МОДЕЛЮВАННЯ ДЛЯ ПІДТРИМКИ ВАРІАБЕЛЬНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В СИСТЕМАХ "РОЗУМНИЙ БУДИНОК"

Побудова концептуальних моделей для проєктування програмного забезпечення (ПЗ), зокрема для таких високотехнологічних застосунків, як системи "Розумний будинок", є складним завданням, від результатів виконання якого суттєво залежить ефективність процесів їхнього розроблення. Одним з інноваційних методів вирішення цієї проблеми є використання предметно орієнтованих мов моделювання (DSML), що дають змогу скоротити витрати часу та інших проєктних ресурсів, потрібних для створення таких систем. **Предметом** дослідження в роботі є підходи з метою розроблення DSML для систем "Розумний будинок" як окремого класу систем *Internet of Things*. **Мета роботи** – запропонувати підхід до розроблення DSML на основі моделі варіабельності властивостей такої системи. Вирішуються такі **завдання:** аналіз деяких уже наявних підходів до створення DSML; побудова багатоаспектної класифікацій вимог до них, застосування цих вимог щодо проєктування синтаксису конкретного DSML-V для створення варіабельного ПЗ у системах "Розумний будинок"; розроблення технологічної схеми й кількісних метрик для експериментального оцінювання ефективності запропонованого підходу. Використовуються такі **методи:** моделювання варіабельності, основане на моделі властивостей, формальні нотації для опису синтаксису мови DSML-V, застосування відкритого інструментального CASE-засобу *metaDepth*. Здобуті **результати:** побудовано багатоаспектну класифікацію вимог до широко класу мов DSML; розроблено основні синтаксичні конструкції мови DSML-V для підтримки властивостей варіабельності ПЗ систем "Розумний будинок"; надано формальний опис такого синтаксису в нотації Бекуса – Наура; створено технологічну схему компіляції специфікацій мовою DSML-V у синтаксис мови відкритого інструментального CASE-засобу *metaDepth*; експериментально досліджено ефективність застосування запропонованого підходу з використанням кількісних метрик. **Висновки:** запропонований метод розроблення спеціалізованої проблемно орієнтованої мови для систем "Розумний будинок" дає змогу проводити багаторівневе моделювання властивостей варіабельності її програмних компонентів і забезпечує зростання ефективності програмування таких моделей приблизно на 14% порівняно з наявними підходами.

Ключові слова: проблемно-орієнтована мова; програмне забезпечення; моделювання; варіабельність; "Розумний будинок".

Бібліографічні описи / Bibliographic descriptions

Гамзаєв О. Р., Ткачук М. В. Розробка проблемно-орієнтованої мови моделювання для підтримки варіабельності програмного забезпечення в системах "Розумний будинок". *Сучасний стан наукових досліджень та технологій в промисловості*. 2023. № 1 (23). С. 45–56. DOI: https://doi.org/10.30837/ITSSI.2023.23.045

Gamzayev, O., Tkachyk, V. (2023), "Development of problem-specific modeling language to support software variability in "Smart Home" systems", *Innovative Technologies and Scientific Solutions for Industries*, No. 1 (23), P. 45–56. DOI: https://doi.org/10.30837/ITSSI.2023.23.045