ONTOLOGICAL MODELLING AND REASONING FOR THE ABSTRACTHOME CLASS OF THE BAB FRAMEWORK FOR PAIS SYSTEMS

Borivoj Bogdanović[1]*,
[0009-0008-2025-176X]
Vidan Marković[1],
[0000-0002-5334-2237]
Đorđe Obradović[1],
[0000-0003-1988-8874]
Milan Segedinac[2],
[0000-0003-1743-9522]
Zora Konjović[1]
[0000-0001-9997-1285]

[1]Singidunum University,
Belgrade, Serbia

[2]Faculty of Technical Sciences,
University of Novi Sad,
Novi Sad, Serbia

Abstract:
Modern process-aware information systems (PAIS) demand a robust, flexible, and semantically consistent approach to managing the ever-changing business requirements––– of today's dynamic business environment. The BAB (Business Application Builder) framework responds to these challenges with an ontology-driven development methodology, which formalizes the design, validation, and automatic code generation processes. This paper presents a detailed analysis of the AbstractHome class, a central element in the BAB ontology, with a focus on its ontological modelling. Acting as a critical intermediary, the AbstractHome class serves as the backbone between domain entities (AbstractEntity) and their corresponding data transfer objects (AbstractDTO). It not only encapsulates the essential business logic needed to drive complex processes but also streamlines the transition from high-level design to executable code through automated generation. In our approach, abstract constructs, implemented through abstract classes, generic parameters, and well-defined interfaces, are rigorously formalized using OWL standards. This formalization supports a comprehensive framework that facilitates process simulation and validation before any code is generated. The ontological definitions enable developers to reason about business processes at a high level of abstraction, ensuring that every constraint and relationship within the AbstractHome class is explicitly captured.

Keywords:
Business Information Systems, PAIS, Ontological Modelling, Code Generation.

INTRODUCTION

1.1. RESEARCH MOTIVATION, HYPOTHESIS AND METHODOLOGY

Rapid changes in business processes, in a globalized and competitive environment, require information systems to be both agile and reliable. Process-aware information systems (PAIS) [1] enable dynamic workflows and rapid adaptation to new requirements. On the other hand, traditional PAIS development and operation approaches are not suitable for small teams with limited resources and often have trouble maintaining consistency when manually implementing and/or automating the translation of abstract models into source code [2]. These issues are the main motivation of the research shown in this paper.

Correspondence:
Borivoj Bogdanović

e-mail:
borivoj.bogdanovic.22@singimail.rs

DOI: 10.15308/Sinteza-2025-128-135

Accordingly, the hypothesis of the research presented in this paper is as follows.

Having an ontology-based conceptual model of a business information system, it is possible to build a (software) system that facilitates automated code generation and rigorous formal consistency checking.

The research methodology of this paper is as follows:

- We base our research on the BAB ontology [2, 3] which is a conceptual model of an information system with generalized common business processes, and the CodeOntology [4] which is an ontology of object-oriented programming languages both implemented using standardized Semantic Web technologies (RDF, OWL);
- For the automated code generation and consistency checking, we use templates that are aware of generalized common business processes implemented in BAB ontology, the CodeOntology, and ontological reasoning; and
- To validate the results, we use an example of applying the BAB framework to create an ontology of a simple business process and automatically check the consistency of the resulting model.

For that purpose, we have developed the AbstractHome class, which acts as a bridge between domain entities (AbstractEntity class) and their corresponding data transfer objects (AbstractDTO class) facilitating consistency checking and automated code generation through templates that are aware of generalized common business processes implemented in BAB framework. This paper provides a comprehensive study of the AbstractHome class, its formal definition, structural constraints, and its key role in the BAB system architecture.

The rest of the paper is structured as follows. Section 2 presents briefly related work. Section 3 introduces the BAB ontology emphasizing the role of the AbstractHome class. Section 4 describes the AbstractHome class and its ontology modelling and explains the business process modelling within BAB, while Section 5 brings the discussion of the results and future research.

2. RELATED WORK

Ontologies are traditionally used in the development and operation of information systems and can be classified into two groups [5]: (1) ontologies for information systems that represent knowledge about the domain of an information system, and (2) ontologies of information systems that represent knowledge about the domain of information system design and implementation. The first group dominates, so the number of publications is extremely large. We will mention here only a few sources just to illustrate the range of application domains: production [6], health care [7], education [8], e-Government and administration [9], agriculture [10], environmental protection [11], culture and arts [12]. Publications from the second group deal with the ontological modelling of the information system fundamentals [13, 14], software engineering [15] and software life cycle phases: requirement specification [16], design [17, 18], programming [19, 20, 21, 22, 23], testing [24], deployment and operation [25].

Due to limited space, we will shortly present only the sources [20, 23] that most directly influenced the development of the BAB framework. Both sources deal with the same research subject as the research whose part is presented in this paper. The main difference between these studies and BAB is that the BAB framework already contains a highly abstract ontological model of a business information system that is further adapted to the needs of the specific user, which is not the case in any of these studies. Source [20] defines five ontologies for generating Java code using tools from the Semantic Web stack. It has a significant similarity with the BAB framework (the use of ontology to represent the information system and the technology stack used). It differs from the BAB framework in the level of abstraction of the information system and in that it does not provide explicit support for other programming languages or deal with checking the consistency of the generated code. While BAB is focused on the code generation of business information systems, work [23] deals with the incorporation of the ontological paradigm into general-purpose programming languages with the aim of providing support for semantically rich domain-driven programming. As such, it also provides support for the business domain and various programming languages. In addition to the absence of a highly abstract ontological model of the business information system, the difference in relation to the BAB framework is also in the approach to the target language. Unlike the BAB framework, which integrates knowledge of the target programming language into the BAB ontology and enables automatic code generation in the target language, research [24] internalizes the ontological paradigm into the Clojure programming language, thus enabling ontological programming on two basic technological platforms, Java and .NET.

3. THE BAB FRAMEWORK AND THE ROLE OF THE ABSTRACTHOME CLASS

The BAB framework [2, 3] is aimed to facilitate the accelerated development and continuous delivery of information systems, with a special emphasis on the needs of smaller companies. The ontological approach was chosen to easily represent and record both static (structural) and dynamic (behavioural) aspects of the system.

3.1. BAB ONTOLOGY

The BAB ontology is conceived as a foundation framework for systematic representation of information systems that facilitates the seamless generation of programming code for web applications. It serves not only as a conceptual blueprint for defining and organizing parts of information systems, but also as a bridge to automated code generation with the capability to validate design in modelling phase. This is achieved by aligning it closely with the principles and structure provided by CodeOntology project (WoC) used as a base for BAB ontology.

A closer look at the composition of the classes reveals that they are defined as subclasses of owl:intersectionOf of collection of owl:Restriction based on properties like woc:hasModifier, woc:hasPackage, woc:implements, and woc:hasFormalTypeParameter if they are generic classes. Properties are listed in the same way, intersection of restrictions. Individual fields follow the same organization as classes. Following the same principle enables the use of the same queries to get the internal structure of the inspected construct.

The ontology is composed of several key components. AbstractEntity, AbstractDTO, and AbstractHome are the classes that represent the basis of an information system, atomic entities that are used in business processes. AbstractEntity represents the base for all subjects and is later mapped to database tables and entities in Object-Relational Mapping (ORM). AbstractDTO is a principal concept for all different “looks” that we can have at our data. It is used to define DTOs (Domain Transfer Objects) and build SQL queries for populating views. The class AbstractHome connects data from the classes AbstractEntity and AbstractDTO by enabling the transformation from entity to DTO and encapsulates business logic. All these classes can be extended by adding interface classes. There are also classes that represent a formal description of web application common elements, the abstractions of entity properties (including their type, name, and access modifier), and the restrictions on those properties (such as NotNull, NotEmpty). Object properties model intra-class relations, while inter-class relations are classes, modelling common four types of relationships (one-to-one, one-to-many, many-to-one and many-to-many). Finally, there is a special class Task which is used to model business process. It is modelled as an RDF triplet commonly annotated by duration. It can be combined with other tasks to form a complex processes, knows which business subject has privileges to start it, can require some external data, resources (other system components), and has status of completion. All classes are expressed using corresponding structures in ontology, including their type in programming language, information about form of use and location in the package which absolves a code generation tool of hard-coded information. All the names are the same as in the BAB framework, and there is a mapping one-to-one for all main concepts. This enables code generators that mimic human reasoning in writing code and the possibility to mix and match automated and manual modes of code generation. RDF and OWL standards make BAB highly interoperable and flexible, capable for dynamic code generation with minimal manual intervention. Side effects that we also get by using those techniques are the up to date documentation and information that can also be used for Artificial Intelligence (AI) as the means of contextual help, validation, or simulation.

4. THE ABSTRACTHOME CLASS AND ITS ONTOLOGY MODELLING

AbstractHome class represents the central data access point in the BAB ontology. As an abstract generic class, it provides specialized entity transformations and basic CRUD operations for entities in the system. It is designed to work with all entity types derived from the AbstractEntity class, allowing for code flexibility and reusability.

4.1. ONTOLOGICAL MODELLING OF THE ABSTRACTHOME CLASS

The AbstractHome class is modelled as an ontological entity, specifically as owl:Class with type woc:Class. In addition, it has modifiers represented by the woc:hasModifier property, while the interface implementation is provided by the woc:implementsInterface property. Furthermore, the class is associated with the package rs.co.bora5.programs.bab.session, which is

clearly visible through the woc:hasPackage property. This connection allows for organization and categorization within the broader structure of the software system. Regarding to generic parameters, types T and D are modelled with the help of restrictions (owl:Restriction). This uses properties such as woc:extends and woc:hasTypeArgument, which explicitly set the conditions and relations for these types. Methods within the class are represented as individuals of type woc:Method, and their parameters are modelled separately as individuals of type woc:Parameter with appropriate properties, which enables precise definition of class functionality. Finally, woc:ParameterizedType is used to represent parameterized types. This approach clearly defines generic types such as List<T>, Map< Key, Value >, Set<T>, thereby providing additional flexibility and precision in ontology modelling.

4.1.1. Core features of the AbstractHome class

The core features of a class include the key elements of a class that allow it to be accurately modelled. In contrast to Java programming language which uses strictly defined syntactic and semantic constructs, such as generic parameters, interfaces, and access modifiers, which are checked at a compile time, OWL allows modelling those concepts as ontological entities, properties, and restrictions that provide dynamic inference and more flexible interpretation of meaning. So, in our framework generic parameters, interfaces and modifiers are modelled as follows. We have two generic parameters, T and D. In the ontology, T is defined as owl:Class which is a subclass woc:TypeVariable, with the constraint that it extends (woc:extends) AbstractEntity. This is represented by owl:Restriction. Similarly, D is modelled as a variable type that extends AbstractDTO<T>, using parameterized types in the ontology. The constraint is represented via the woc:extends property with a value that is a parameterized type of AbstractDTO<T>. The interfaces that AbstractHome implements are modelled as classes in the ontology, with the woc:implementsInterface property connecting AbstractHome to the corresponding interfaces (e.g., AbstractHomeCRUDInterface<T>). Finally, the class is marked as public or abstract, which is represented by the woc:hasModifier properties in the ontology, with the values woc:Public and woc:Abstract.

4.1.2. Methods

The methods within the AbstractHome class can be divided into three main groups (CRUD operations, entity transformations and utility methods), each group including specific operations and functionalities. The continuation of the section depicts the three main methods groups (method definition and method description).

Table 1 shows methods that deal with basic operations on entities (creating, updating, deleting, and retrieving). They directly manipulate the data in the database, thus ensuring the integrity and consistency of the system.

Table 2 shows data transformation methods that are responsible for preparing and transforming data, generating SQL statements, as well as creating DTO objects based on defined meta-data. They enable flexible formation of queries and adaptation of results to the specific needs of the application.

Utility methods (Table 3) are methods that facilitate working with a session (synchronizing the state of an entity, clearing the session) and provide additional information about the class (getting metadata about the class).

Minimum details concerning the ontological modelling of methods are shown in Table 4 by three selected methods, each representing the corresponding method group.

Table 1. CRUD methods

Method definition	Description
public T save(T entity)	Saves or updates the given entity
public void remove(T entity)	Deletes the given entity from the database
public void remove(Long id)	Deletes an entity based on its id
public T find(Long id)	Finds and returns an entity based on its id
public T getFullObject(Long id)	Retrieves the complete entity with all associated data
public List<T> findAll()	Returns a list of all entities of type T

Table 2. Data transformation methods

public String getSelect()	**Generates a SQL SELECT statement with a DTO constructor call**
public String getSelect(Map<String, AbstractEntity > map)	Generates a SQL JOIN expression for the related entities
public String getJoin()	Generates a SQL JOIN expression for the related entities
public List<D> findAll(QueryMetaData<T, D> queryMetaData)	Returns a list of DTO objects based on the given query meta-data
public Stream<D> findAllLazy(int startIndex, int count, QueryMetaData<T, D> queryMetaData)	Returns a stream of DTO objects for lazy loading
public int findSizeLazy(QueryMetaData<T, D> queryMetaData)	Returns the total number of results based on the query
public List<D> findAllLazyDTO(int startIndex, int count)	Returns a list of DTO objects for lazy loading without additional meta-documentation

Table 3. Utility methods

Method definition	**Description**
public void flush()	Synchronizes the entity state with the database
public void clear()	Clears the current session of tracked entities
public void refresh()	Refreshes the entity state from the database
public String getName()	Returns the name of the class with the package
public String getShortName()	Returns the short name of the class
public Class<T> getEntityClass()	Returns a reference to the entity class that AbstractHome manages

Table 4. Method representatives

Method definition	**Ontological modelling**	**Access**	**Parameter**	**Return type**
public T save (T entity)	An individual of type woc:Method associated with the AbstractHome class via the property woc:isMethodOf	public; defined via the woc:hasModifier property	entity; modelled as an individual of type woc:Parameter, with properties woc:hasName (value "entity") and woc:hasType indicating type T	generic T; modelled by the property woc:hasReturnType that references the type of the T variable
public String getSelect()	An individual of type woc:Method	public; defined via the woc:hasModifier property	entity; modelled as an individual of type woc:Parameter, with properties woc:hasName (value "entity") and woc:hasType indicating type T	String; modelled by the property woc:hasReturnType with a reference to java.lang.String;
public void refresh()	An individual of type woc:Method, without input parameters and without return type(void)	public; defined via the woc:hasModifier property	None	None

4.2. MODELLING A BUSINESS PROCESS

Business processes are ontologically modelled as the Task class. Here, we present an example of a simple business process (Listing 1) that shows how the onSave method, which executes the save and sendMailNotification operations, is encapsulated as a Task.

The save method (persists a Shipment entity) and the sendMailNotification method (sends an email notification to a postman based on his ID) were defined in our BAB ontology. Next, the onSave method was created to execute these two methods in sequence. To capture the entire business process, we modelled an individual onSaveTask—based on the Task class—that executes the onSave method. This modelling approach leverages the inheritance of methods from the AbstractHome class, so any class extending AbstractHome (such as ShipmentHome) automatically has access to these operations. The onSave method, defined to sequentially call save and sendMailNotification, is treated as a composite operation, and its execution is represented by the onSaveTask individual.

By representing onSaveTask as an instance of the Task class, we can attach additional properties like duration, status, and sub-task relationships, thereby integrating it into a broader process management framework. This approach enables the integration of method-level functionality into coherent business process model, facilitating process validation, simulation, and eventual automated code generation for distributed PAIS systems.

5. BAB APPLICATION TO PAIS SYSTEMS AND AUTOMATIC VALIDATION

The ontological modelling of the AbstractHome facilitates the development of PAIS systems by ensuring that every data access operation follows a consistent semantic framework and enabling process simulation and validation resulting in a model that can be used for automatic template-based code generation. By encapsulating CRUD operations and transformation logic within the AbstractHome class, the ontology ensures that every data access operation follows a consistent semantic framework. For example, the restriction that each entity passed to the save method must be an instance of a class that inherits from AbstractEntity is enforced through ontological restrictions on the generic parameter T. Before generating the code, the BAB tool uses SPARQL queries to simulate the process flows defined in the ontology. Such a simulation helps with the early detection of inconsistencies (e.g. mismatched types or missing methods) and ensures that all business requirements are met. Once the ontology model is validated, it is used for automatic code generation (i.e. by Apache FreeMarker templates). This process ensures that any change to the ontology, such as updating restrictions on the AbstractHome class, is automatically propagated to the generated code, supporting agile development and continuous integration.

In our approach, a series of SPARQL queries is executed to validate the ontology, check constraints, and extract the necessary parts for code generation. These queries serve as the backbone for ensuring that the ontology is semantically consistent before the code generation process begins. The following example (Listing 2) illustrates the use of SPARQL queries to extract specific layers from the ontology, such as model classes, operator classes, session classes, DTO classes, view classes, and window classes.

```
<owl:ObjectProperty rdf:about="http://www.semanticweb.org/borivoj.bogdanovic/ontologies/2023/9/
BAB#executesMethod"> <rdfs:domain rdf:resource="http://www.semanticweb.org/borivoj.bogdanovic/
ontologies/2023/9/BAB#Task"/> <rdfs:range rdf:resource="http://rdf.webofcode.org/woc/Method"/>
<rdfs:label>executes method</rdfs:label>
</owl:ObjectProperty>


<owl:NamedIndividual rdf:about="http://www.semanticweb.org/borivoj.bogdanovic/ontologies/2023/9/
BAB#onSaveTask">
<rdf:type rdf:resource="http://www.semanticweb.org/borivoj.bogdanovic/ontologies/2023/9/BAB#Task"/>
<rdfs:label>onSave Task</rdfs:label>
<bab:executesMethod rdf:resource="http://www.semanticweb.org/borivoj.bogdanovic/ontologies/2024/2/
BABTest#onSaveMethod"/> </owl:NamedIndividual>
```

Listing 1. Task ontology for business processes

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX woc: <http://rdf.webofcode.org/woc/>
PREFIX bab: <http://www.semanticweb.org/borivoj.bogdanovic/ontologies/2023/9/BAB#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?class
WHERE {
  ?class rdfs:subClassOf ?restriction .
  ?restriction owl:intersectionOf/rdf:rest*/rdf:first ?component .
  ?component owl:onProperty woc:extends ;
    owl:onClass bab:AbstractEntity ;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger .
  { ?class rdf:type woc:Class . } UNION { ?class rdfs:subClassOf woc:Class . }
  FILTER (?class != owl:Nothing)
}
```

Listing 2. SPARQL query for extracting model classes

The successful execution of such queries confirms that each segment of the ontology meets its defined constraints. The extracted information is then used to generate code layer by layer, ensuring that the generated system mirrors the formal semantic structure. On the other hand, if any query returns unexpected results, the process flags these issues and halts code generation. This immediate feedback allows developers to refine the ontology and resolve inconsistencies before any erroneous code is produced.

6. CONCLUSION

As business requirements continue to change rapidly, the integration of ontological reasoning into development frameworks looks like a promising path toward building agile, reliable, and sustainable PAIS systems. In BAB ontology, abstract constructs, implemented through abstract classes, generic parameters, and well-defined interfaces, are rigorously formalized using OWL standards resulting in a comprehensive framework that facilitates process simulation and validation before any code is generated.

The AbstractHome class is a central element in the BAB ontology. Acting as a critical intermediary, the AbstractHome class serves as the backbone between domain entities (AbstractEntity) and their corresponding data transfer objects (AbstractDTO). It encapsulates the essential business logic needed to drive complex processes and streamlines the transition from high-level design to executable code through automated generation. That enable developers to reason about business processes at a high level of abstraction, ensuring that every constraint and relationship within the AbstractHome class is explicitly captured.

Despite these advantages, there are serious challenges ahead. The complexity of ontology modelling as well as the state-of-the-art integration of semantic tools with traditional development environments reduces the range of developers capable of using the BAB framework. Therefore, a short-term goal of future research is the development of more intuitive domain-specific languages (DSLs) and graphical tools for easier ontology management. Advances in AI provide the basis for advanced predictive validation and defect detection, and a medium-term research goal is to develop and integrate these techniques into the BAB framework. Finally, the development of LLMs opens up space for innovative tools highly customized to a wide range of users, which is the goal of long-term research in the BAB framework.

7. ACKNOWLEDGEMENTS

This research was partly funded by the Serbian Ministry of Science, Technological Development and Innovation through the Project no. 451-03-47/2023-01/200156 “Innovative scientific and artistic research from the FTS (activity) domain”.

REFERENCES

[1] M. Dumas, W. van der AalstT and A. H. M. ter Hofstede, "Introduction," in *Process-Aware Information Systems Bridging People and Software Through Process Technology*, M. Dumas, W. van der Aalst and A. H. ter Hofstede, Eds., Hoboken, John Wiley & Sons, Inc., 2005, pp. 3-20.

[2] B. Bogdanović, Đ. Obradović, M. Segedinac and Z. K. Konjović, "BAB Framework – Towards an Extensible Software Platform for AI-Augmented Process Aware Business Information Systems," in *Disruptive Information Technologies for a Smart Society (ICIST 2024)*, M. Trajanović, N. Filipović and M. Zdravković, Eds., Cham, Springer Cham, 2024, p. 197–212.

[3] B. Bogdanović, Đ. Obradović and Z. Konjović, "Bab (Business Application Builder) Framework for Rapid Development of Business Information Systems," in *International Scientific Conference - Sinteza 2023*, Belgrade, 2023.

[4] M. Atzori, M. Atzeni and M. Setzu, "CodeOntology," [Online]. Available: http://codeontology.org/about. [Accessed March 2024].

[5] F. Fonseca, "The double role of ontologies in information science research," *Journal of the Association for Information Science and Technology*, vol. 58, no. 6, pp. 786-793, 2007.

[6] J. Abonyi, L. Nagy and T. Ruppert, Ontology-Based Development of Industry 4.0 and 5.0 Solutions for Smart Manufacturing and Production - Knowledge Graph and Semantic Based Modeling and Optimization of Complex Systems, Cham, Swiss: Springer Cham, 2024.

[7] P. S. Sen and N. Mukherjee, "An ontology-based approach to designing a NoSQL database for semi-structured and unstructured health data," *Cluster Computing,* vol. 2027, p. 959–976, 2024.

[8] S. Bayne, "Digital education utopia," *Learning, Media and Technology,* vol. 49, no. 3, p. 506–521, 2023.

[9] C. Brys, I. Navas-Delgado and M. d. M. Roldán-García, "LEGO: Linked electronic government ontology," *Journal of Information Science*, vol. 0, no. 0, 2023.

[10] B. P. Bhuyan, R. Tomar, M. Gupta and A. Ramdane-Cherif, "An Ontological Knowledge Representation for Smart Agriculture," in *2021 IEEE International Conference on Big Data*, Orlando, France, 2021.

[11] L. E. Chan, A. E. Thessen, W. D. Duncan, N. Matentzoglu, C. Schmitt, C. J. Gondin, N. Vasilevsky, J. A. Ms-Murry, P. N. Robinson, C. J. Mungal and M. A. Haendel, "The Environmental Conditions, Treatments, and Exposures Ontology (ECTO): connecting toxicology and exposure to human health and beyond," *Journal of Biomedical Semantics,* vol. 14, p. 3(2023), 2023.

[12] M. Doerr, "Ontologies for Cultural Heritage," in *Handbook on Ontologies,* R. Studer and S. Staab, Eds., Springer Berlin, Heidelberg, 2009.

[13] Y. Wnad and R. Weber, "An Ontological Model of an Information System," *IEEE Transaction on Software Engineering*, vol. 16, no. 11, p. 1282/1292, 1998.

[14] Y. Wand and R. Weber, "An Ontological Analysis of Some Fundamental Information Systems Concepts," in *ICIS 1988 Proceedings.* 35, 1988.

[15] I. Seiji , I. Bittencourt, E. F. Barbosa, D. Dermival and R. Oscar Arajuo Paiva, "Ontology Driven Software Engineering: A Review of Challenges and Opportunities," *IEEE Latin America Transactions*, vol. 13, no. 3, pp. 863-869, 2015.

[16] M. P. S. Bhatia, A. Kumar, R. Beniwal and T. Malik, "Ontology driven software development for automatic detection and updation of software requirement specifications," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 23, no. 1, p. 197–208, 2020.

[17] S. K. Mishra and S. Anirban, "Service-oriented architecture for Internet of Things: A semantic approach," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 8765-8776, 2022.

[18] I. Lutsyk and D. Fedasyuk, "Analysis of Approaches to Design Ontological Models of an Adaptive Software System," *Computer Systems and Information Technologies*, vol. 3, pp. 13-20, 2024.

[19] D. Strmečki, I. Mgdalenić and D. Radošević, "A systematic literature review on the application of ontologies in automatic programming," *International journal of software engineering and knowledge engineering,* vol. 28, no. 05, pp. 559-591, 2018.

[20] D. Strmečki and I. Magdalenić, "An Ontological Model for Generating Complete, Form-based, Business Web Applications," *International Journal of Advanced Computer Science and Applications(IJACSA)*, vol. 10, no. 8, p. 34 – 38, 2019.

[21] S. Barakt, A. B. Sánchez and S. Segura, "IDLGen: Automated Code Generation for Inter-parameter Dependencies in Web APIs," in *Service-Oriented Computing. ICSOC 2023,* F. Monti, S. Rinderle-Ma, A. R. Cortés, Z. Zheng and M. Mecella, Eds., Cham, Springer, Cham, 2023.

[22] K. Lano and Q. Xue, "Code Generation by Example Using Symbolic Machine Learning," *SN Computer Scienca*, vol. 4, p. 170 (2023), 2023.

[23] D. Đurić and V. Devedžić, "Incorporating the Ontology Paradigm into a Mainstream Programming Environment," *Informatica*, vol. 23, no. 2, pp. 203 - 224, 2012.

[24] G. Tebes, L. Olsina, D. Peppino and P. Becker, "Specifying and Analyzing a Software Testing Ontology at the Top-Domain Ontological Level," *Journal of Computer Science & Technology*, vol. 21, no. 2, pp. 126-145, 2021.

[25] C. Pardo and J. Guerrero, "DevOps Ontology - An ontology to support the understanding of DevOps in the academy and the software industry," *Periodicals of Engineering and Natural Sciences*, vol. 11, no. 2, pp. 207-220, 2023.