BAB (BUSINESS APPLICATION BUILDER) FRAMEWORK FOR RAPID DEVELOPMENT OF BUSINESS INFORMATION SYSTEMS

Borivoj Bogdanović,
Zora Konjović*,
Đorđe Obradović

Singidunum University,
Belgrade, Serbia

Abstract:

This paper presents the BAB (Business Application Builder), a software framework for the rapid development of business information systems prone to frequent changes in business conditions and/or user requirements. The proposed framework is a programming pattern for a business information system in which a business information system is modeled as a set of related design patterns that implement common features of a class of business information systems. An object-oriented approach and a design pattern concept were used to model the framework, while the implementation is done using (but not limited to) selected technologies. The proposed solution is evaluated through the example of a business information system of medium complexity.

Keywords:

Business Information System, Meta-Model, Software Development, Rapid Application Development.

INTRODUCTION

Modern society has faced increased global interconnectedness facilitated by technological advancements for decades [1], [2]. To optimize business processes, automate tasks, and facilitate the work of employees, the information systems supporting the business logic and workflow of organizations have become increasingly complex. Modern information systems are required to enable users to work from spatially and globally distributed locations, use diverse devices [3], and interact with components (data, computer programs) that are logically and spatially distributed and, most often, largely heterogeneous [4]. Additional aspects that are especially important for the subject of this paper are two mutually opposed goals to be met: rapidly changing user requirements (both functional and non-functional) [5], and the severe requirement for continuous software integration and delivery [6]. This all means that, from a developer's point of view, a capacity for rapid development, easy maintenance, and a quick upgrade of software is of crucial importance. As an attempt to respond to these requirements, in this paper, we propose the BAB (Business Application Builder) Framework. The paper is composed of six sections.

Correspondence:
Zora Konjović

e-mail:
zkonjovic@singidunum.ac.rs

DOI: 10.15308/Sinteza-2023-76-84

Following this introduction, the second section presents the abstraction process and abstract component model of the BAB framework. The third section describes the BAB framework architecture, while the fourth one explains how to use the BAB framework and evaluates it through the building of a complex pattern of a business information system. The fifth section describes the implementation of the BAB framework. Section six brings the conclusion, containing the review of the results, the framework's constraints, and plans for further work.

2. ABSTRACT COMPONENT MODEL OF THE BAB FRAMEWORK

Starting from an abstract definition of an information system based on literary sources [4], [7], [8], [9]] and applying methods of analysis and synthesis to selected classes of business information systems, the characteristics of business information systems are distinguished, which are then mapped to the requirements that should be met by the BAB framework.

For our purposes, we use a slightly modified definition of an information system proposed in [7]: An information system is a system in which people and/or machines perform work (processes and activities) using information and information technology to produce information products and/or services for internal or external users. Starting from this definition, we can view an information system as a set of coordinated components that together enable the collection, processing, production, and distribution of information.

The following approach is used for the abstraction of the business information system underlying the BAB framework.

1. Literature sources describing functional and non-functional characteristics of the class of ERP information systems [10] are analyzed as this class of information systems best fits the application domain of BAB.
2. The specific business information system in operation is declared a reference information system and analyzed to collect knowledge about specific functional and non-functional characteristics and implementation technologies.
3. Additional information is collected from users of the reference information system through informal interviews and daily communications to gather additional knowledge about their habits and preferences.

Based on the first two steps of the used approach, a conclusion was drawn that, from the aspect of the business architecture of the specific information system, the appropriate basis for an abstract model would be a variant of the ERP system adapted to the specifics of the company's operations, i.e., that the BAB framework should enable the accelerated development of a customized ERP system in which some standard ERP functionalities are emphasized, while some of them are reduced or even eliminated. A component-based [11] and software pattern-based [12] approach to software development was chosen to enable easy reuse of the logic that is encapsulated in components and the description of a system functioning through orchestration. Based on the aforementioned conclusions, an abstract component model of the BAB framework was created, the description of which follows.

The components and their relations are maintained through the component called View which is the highest level of abstraction in BAB. A lower level of abstraction is the component used to tabulate and manipulate any descendants of the abstract ancestor. It also defines the conceptual deletion functionality with the condition that the object is free and additional conditions that can be defined as needed. As users are used to tabular display and manipulation of data through windows, the decision was made to implement the user interface through these components. In doing so, the basic operations are input, modification, and deletion of data, conditional transformation of data, and creation of "many-to-many" links. There are three more aspects of object manipulation, input/modification, and data transformation; this includes report generation as the most common use case. Therefore, we need three components for their realization. Firstly, we have defined placeholders for the insertion of additional logic at all functionally important points such as saving data before and after events. For the sake of handling sensitive data and security, operator roles are defined. Each abstract page defines a set of roles that a user must possess to be considered an administrator. Accordingly, the component will adjust and show/hide parts of the interface. In the main component that represents the application shell, a set of pages that the user can access is assigned to roles. The intersection of these two sets provides a conceptual solution to the problem of access rights for users from different sectors using the same data, which is the case with shared facilities. For example, the facility vehicle is important to users from the maintenance sector, the sales and support sector, and management.

Once we have defined the operators, we can create a component that will be bound to the user session and will take care of the identity and specific settings for each user individually, such as theme color, sounds, a predefined printer, important dates, etc. To enable synchronization of data between multiple sessions of a single user and between sessions of all users, event-handling components using the `Observer` pattern were deployed [13]. We have two components, one for user-related events and another for system events. We get the concrete event type by expanding the abstract event, which will be processed by type and additional information. We do concrete processing in the shell. Working with e-mail and printing is separated into detached components that can be included as needed. The table component includes a component with segments to run all additional functionality and optionally display additional information. To avoid formatting data individually for each column in the table, based on the data type, the table itself chooses the format. By using the same principle, we can generate a row in the header with fields for filtering data. The field type is directly determined by the data type. The table is joined by a data structure that stores the entered filtering and sorting parameters, which represents the interface between the user interface and the business logic. The initial layouts of the columns and their names are defined for each table and these names must match the description in the middle layer that supplies data from the database. An abstract type data collection and a condition description structure are all we need to connect these two layers. Based on this data, we can "on the fly" form queries to the database. An essential component for a system operation is also a component that will independently activate some processes under certain conditions. They are mostly timely conditioned.

Some of the examples of functions that are activated at a certain time, with the desired frequency of repetition, are sending an email notification of an employee's absence, updating, creating, and sending a report on the employment of employees, creating orders for minimum quantities for stock replenishment, etc.

3. BAB FRAMEWORK ARCHITECTURE

The BAB framework itself is, in essence, a business information system design pattern. It was created using an object-oriented approach and object-oriented design patterns for modeling the structure and behavior (Singleton, Builder, Template, Observer, Dependency Injection, Facade), and a functional programming approach used solely for the business logic. The basic structure of the BAB framework follows the EJB architecture [[14], [15], [16], [17]] shown in Figure 1 consists of three standard layers (presentation, business layer, and data management) where the presentation layer consists of two layers (client layer and web layer), which makes a total of 5 layers:

1. Database Layer (Database Layer) – the concrete database in which we store data.
2. Persistence Layer - JPA (Java Persistence API) annotated classes that describe the database structure and the JPA reference implementation of Hibernate. Business Logic Layer - Stateless EJB (Enterprise Java Bean) beans.
3. Web Layer (Web Layer) – A web container that houses servlets and the server part of the Vaadin Flow framework [18].
4. Client Layer (Client Layer) – Browser that contains Vaadin Flow, the client part of the Vaadin framework [19] and web components.

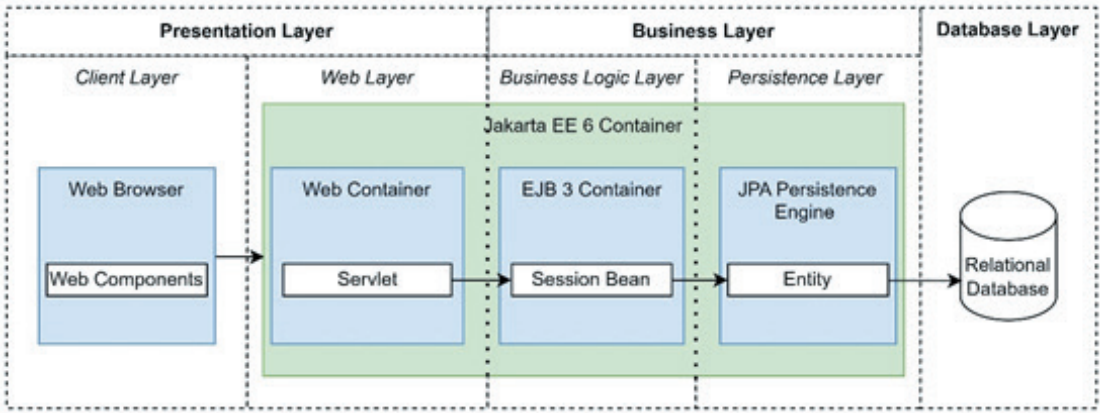

Figure 1 - EJB architecture.

3.1. PERSISTENCE LAYER

All objects of the process persistence layer are treated as objects with a common ancestor that determines their primary identity in the system. This ancestor object allows solving the following tasks: comparing two objects, forming sets of objects for presentation purposes, ensuring the uniqueness of objects in collections, sorting objects, and locking during concurrent modification of the object. All other common features and functionalities are formulated through interfaces. In this way, multiple inheritance is avoided, which simplifies the description of business logic without the burden that additional types would introduce in the utilization of the functional approach. As we work with SQL databases, the basic functionality to be implemented is to mark each row in the database with an artificial key that can be used later to sort the data; newer data will have a higher key value. The data type for the key, field ID, is Long, auto-increment, or identity, as MSSQL calls it. It allows for quick search and comparison, and prevents a user from causing an error because the database system itself takes care of the values. One can use it to calculate the hash value of objects, which is very important because all relations "to more" are realized with the help of the HashSet structure, which depends on the hash value. Another piece of data that each row must contain is the Version field, which is used for optimistic row locking necessary to support multi-user work. The implementation of the equals method, which is used to compare two objects, can already be implemented here (two objects are considered the same only if they are of the same type and have the same ID). These functionalities are described in the `AbstractEntity` abstract class. The `equals` and `hashCode` methods are essential for establishing relationships between objects and represent the basis of the entire framework. Specific cases that may occur in individual classes are described using interfaces. The selection of functionalities described by the interfaces derives from the most common use cases. It is expected that we will have a class of objects that will be bound to sessions and represent subjects that interact with the system. These objects must have fields for username, e-mail, and password. Many objects are described with a name that should be unique, or with numerical data. The same is true for natural keys, such as a serial number. With more sensitive objects, it is important to know when they were made and who preserved them. Much additional information can be introduced here like the arrival date and receiver of the original for the paper documents, approval and log of the changes made for financial documents, etc. A large part of the data we enter in the system will not be allowed to be deleted due to the referential integrity of the database and should be able to be marked as inactive. If a more complex pattern needs to be derived, it can be done by inheriting the `AbstractEntity` class. An example is the `Item` class. All items should have information about the manufacturer, an item from the corresponding codebook, the location where they are temporarily in the warehouse, a link to the items in the incoming and outgoing invoices, etc.

3.2. BUSINESS LOGIC

All business logic will be encapsulated in EJB objects that are stateless by nature. To use `AbstractEntity`, all the common functions that we can work with will be defined in the `AbstractHome` class. These include methods for storing, retrieving, modifying, deleting, and filtering objects. Specific functionalities will be implemented in concrete classes and defined in interfaces. An example is a class that manages users and requires an additional functionality to check the correctness of the entered logging data. The means to represent the conditions set by the user when filtering will be described in the next chapter, in the `QueryMetaData` class. For business logic, we force the functional programming paradigm to the greatest extent possible. The data is read from the database, then further transformed, and forwarded using the `Stream` API. Complex functionalities should be realized by composing functions that are considered atomic from the point of view of business logic.

3.3. WEB CONTAINER

The user interface (UI) configuration will be defined on this layer. Business systems rely heavily on a tabular display of data followed by windows for entering, modifying, and displaying details. Based on this, we can define a `GenericTable` that will contain rules for printing and formatting data based on a data type. One should be able to automatically generate filtering fields for each column. All this data is temporarily stored in the `QueryMetaData` data structure, based on which SQL queries are generated on the fly. For each column, it is necessary to define the name of the attribute in the JPA class, the name in the table, and the name in the DTO (Data Transfer Object) object that will serve the exchange of the data between the business layer and the Web layer. DTO is convenient because it allows us to

perform additional transformations and ignore the relationships between objects and represent them using one or more attributes that carry user-relevant information. So, if we have a DTO for a city, we can replace the connection with the state with an ordinary `String` field in which we will write only the name of the state. The initial layout and visibility of the columns should be defined in the annotation and the changes made by the user will be stored in the database. It is common to have a view table accompanied by input and edit windows that we can represent with a single two-state object because the set of edit fields is almost always a subset of the set of input fields. The field type depends on the attribute type in the entity class. An important part is the validation of the entered data. We can create validators for each specific use case and just bind them to the component. A typical example is a name validator that checks the uniqueness of the entered string. To make it universal, we will represent the object we are checking and the service with corresponding interfaces. In this generic window, one can also define some processes that need to be automated, such as recording the entry date and logged-in user, when the created object implements the interface that defines this functionality. Another example is setting focus on the first input field from the top. A generic table and a generic window can be grouped under the GenericView class, which represents a typical web page. Below the table will be a section for additional information such as the number of items in the table, and a section for input, edit, and delete buttons, and additional multi-purpose buttons. What follows is about resolving privileges and deleting an item from the table. Deletion should be possible if the user has the appropriate privileges and if the object has no relation to other objects of the multi-type. We can solve this by reflection. Additional conditions that need to be fulfilled in some cases or branching effects can be defined by redefining empty methods, hook methods, which are called before and after essential operations. Another type of input window we need is for manipulating many-to-many links. We will implement them through two tables. One will represent a set of all possible, free objects, and the other the assembly of the elect. The standard functions we will have represent the transfer of one/all objects between these tables with the possibility of defining validation steps and additional functionalities. The third type is windows for data manipulation and transformation where one can describe, for example, the generation of reports, the loading of XML files, and their processing. What is still needed is to define actions in detail for each row of the table. It can be a simple dialog, an object whose class inherits one of the two types of generic windows, a button to jump to a related table where filtering should be done immediately or a component that displays textual details if the text is longer and it does not make sense to display it in a table in completely. All those functionalities are implemented through the annotations that the corresponding view will contain. For the application to work like classic applications, it is necessary to refresh the tables for all users who view the same table on which someone made a change. This will be done by using the Observer pattern [13] with the help of the appropriate event type.

4. EVALUATION OF THE BAB FRAMEWORK

We need a pre-arranged set of classes that inherit from the frame classes to build each view. We need a class for mapping data to a base table that inherits the `AbstractEntity` class, denoted further on by the generic type T. Based on it and the data we want to display to the end user, we need to define a DTO object that will inherit the `AbstractDTO` class denoted by type D, which depends on T. Next is the class (generic type V) that will encapsulate the business logic and manipulation of objects in the database. This class inherits `AbstractHome`, which depends on T and D. The class of the concrete view inherits `GenericView` and depends on T, V, D and K which represents type of end user. In the same way, we define the class which defines the entry/modification window also dependent on T, V, D and K.

In the example of an information system that was built with the help of the BAB framework, one task is to define items. By item, we mean any material good or service that a company can buy, store, and/or sell. Accordingly, the basic requirements were that it should be possible for any item to enter and/or leave the company through the appropriate type of document. Therefore, the codebook for items should be defined, and the connection of the item with the organizational unit of the warehouse where it is physically located. The additional requirement arose from the structure of the legacy application as it was at the time since there were tables representing the items that needed to be brought under the new structure. The natural solution was to generate a new abstract class structure that is lower than `AbstractEntity`, yet higher than any individual item class. In this way, we can check the record structure at the level of restrictions that are written in the model classes, as has been done so far; we will still have one class

to one table mapping as described by the "merged table" strategy. Another option would be to use a "single table" strategy. These strategies are standardized by the JPA specification (Team, 2020). We make the form from two model classes, `Artikal` (Eng. Item) and `ArtikalInfo`, an abstract item, and the corresponding codebook. Next, we have their projections, where we will change the lower limit of the generic from `AbstractEntity` to `Artikal` or `ArtikalInfo`. According to the existing form, as before, we define the service for code books and items, also lowering the level of abstraction. We also define the view for code books as before with the same changes. A big extension of the form comes with the view, which will be the basis for all items. Instead of the previous four types, T, V, D and K, we will have 8 types: `ArtikliView<T extends Artikal, V extends AbstractArtikalHome<T, D>, D extends AbstractArtikliDTO<T>, X extends ArtikalInfo, Y extends AbstractArtikalInfoHome<X, Z>, Z extends AbstractArtikalInfoiDTO<X>, E extends Enum<?>, K extends AbstractEntity & OperaterEntityInterface> extends GenericView<T, V, D, K>`. As one can see, for each item, three additional types related to the codebook plus one that will be related to the item type should be defined. In this way, we can implement common functionality and checks in one place. Because of this, one can even define the design of the window for input/modification of items once and subsequently add specific fields in specific classes. The graphic representation of the extended pattern is given in Figure 2.

Using the extended pattern, we can derive a set of classes for the `Vozilo` entity entry as shown in Figure 3.

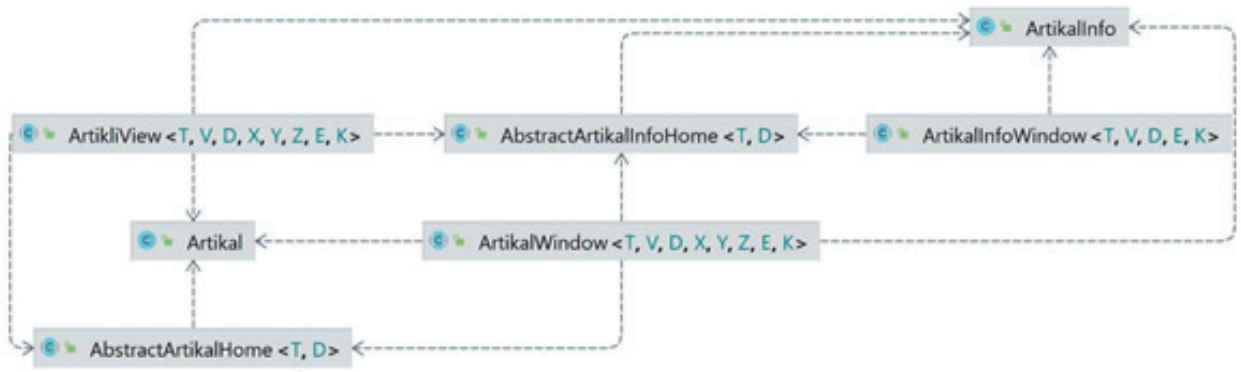

Figure 2 - Example - Arikal Entity.

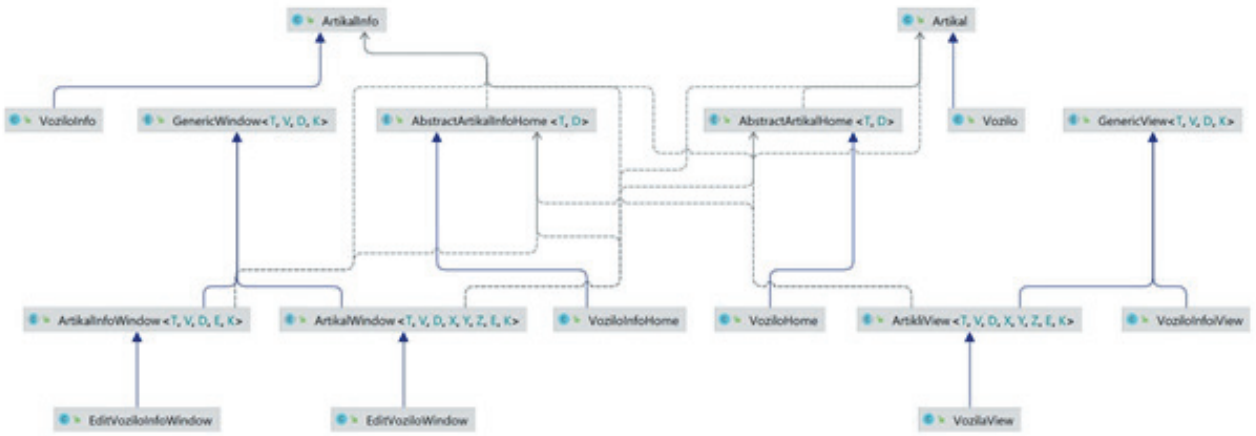

Figure 3 - Class diagram for the Vozilo entity.

5. SOFTWARE ARCHITECTURE OF THE BAB FRAMEWORK

The implementation technologies of the BAB framework were chosen so that the application meets the following requirements: internet capability with support for various terminal devices, portability, robustness and stability, easier development, and economy in production. The requirement of internet capabilities imposed the decision to implement a web application, and the requirement of the economy in production dictated the choice of open-source technologies. The programming language Java and the JAKARTA EE set of specifications for the development of business applications in Java [17], was used as the basic language. The reasons for this choice are stability and widespread, speed, available libraries, and portability. The relational database (here the MSSQL database) and JPA specification were chosen for persistence implementation due to the dominance of relational databases and the advantages JPA has over other persistence mechanisms. For the sake of easier development and "smoother" integration of layers, the Vaadin platform ([18], [19]) is used for the front end. Figure 4 [19] shows the generic framework for using web components in the Java environment.

This enables the description of the structure and functionality of web pages (HTML, CSS, JavaScript) in the Java programming language with the use of web components. The disadvantage that Vaadin potentially has as a server technology is related to the limited number of competing users. However, the study https://vaadin.com/vaadin-scalability-report reports the performance which is quite acceptable for midsize business systems. Conforming to Servlet API technologies allows the use of any application server that supports Servlet technology. In this specific case, the WildFly (formerly JBoss) server, a Red Hat open-source distribution, is used [20]. Hibernate was chosen as the ORM mapper due to its stability, large user base, and continuous development, as well as compliance with the JPA specification. Additional libraries for working with Excel, and XML files are from the Apache Foundation, while the JasperReports tool is chosen for report generation.

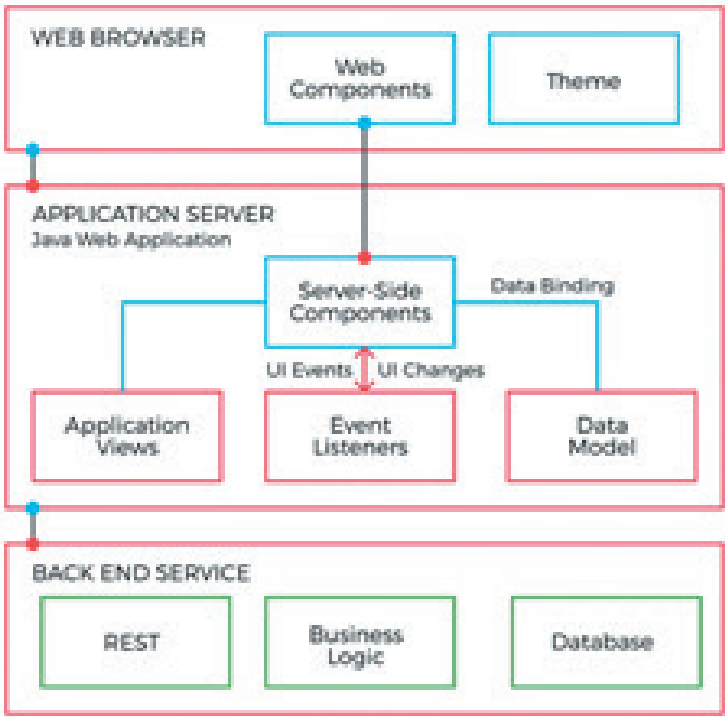

Figure 4 - Usage of web components.

6. CONCLUDING CONSIDERATIONS

BAB Framework is a software pattern that aims to accelerate the development of business information systems. It is based on an object-oriented approach and the design pattern concept. The BAB model is implemented using the Java programming language and the Jakarta EE, with functional programming in the business logic layer. The proposed solution is continuously evaluated by developing and maintaining the operative business information system of medium complexity. The results show that BAB Framework significantly accelerated the development process, reduced the workload and fatigue of programmers, improved code readability, and facilitated testing. One of the main BAB framework's deficiencies is the steep learning curve for programmers, as they must remember many conventions to describe the structure and functionality of the system. A domain-specific language, a code generator, and a graphical UML-based tool are under development to address the issue. Another deficiency is the significant effort needed to configure the infrastructure (operating system, database, and web servers), which can be time-consuming and/or require the commitment of specialized personnel. Container technology, like Docker, is envisioned to simplify configuration and ensure continuous delivery. Apart from the above-mentioned issues that are obvious and require short-term resolution, there is plenty of space for substantial improvement of the BAB Framework. Meta modeling enhanced with appropriate AI techniques (i.e., semantic technologies for semantic coupling and co-change of software components [21], explainable AI for defect prediction models [22], large language models [23] for automated models and/or code generation, and alike) is the road that should be hit in the future to improve substantially of the BAB Framework.

7. REFERENCES

[1] Y. Bakos, "The emerging role of electronic marketplaces on the Internet," *Communicatons of the ACM*, vol. 41, no. 802, pp. 35 - 42, 1998.

[2] Y. Bakos and H. Halaburda, "Overcoming the coordination problem in new marketplaces via cryptographic tokens," *Information Systems Research*, vol. 33, no. 4, pp. 1368-1385, 2022.

[3] T. F. Pereira, A. Matta, C. M. Mayeaa, F. Pereira, N. Monroy, J. Jorge, T. Rosa, C. E. Salgado, A. Lima, J. Ricardo, R. J. Machado, L. Magalhães, T. Adão, M. A. G. López and D. G. Gonzalez, "A web-based Voice Interaction framework proposal for enhancing Information Systems user experience," *Procedia Computer Science*, pp. 235-244, 2022.

[4] K. D. Schewe and B. Thalheim, Design and Development of Web Informaton Systems, Berlin - Heidelberg: Springer Verlag, 2019.

[5] N. Ali and R. Lai, "A method of requirements change management for global software development," *Information and Software Technology*, vol. 70, pp. 49-67, 2016.

[6] J. Stine, Design, Build, Ship: Faster, Safer Software Delivery, Oreilly & Associates Inc, 2021.

[7] S. Alter, "Defining information systems as work systems: implications for the IS field," *European Journal of Information Systems*, pp. 448-469, 2008.

[8] K. E. Pearlson, C. S. Saunders and D. F. Galleta, Managing and Using Information Systems: A Strategic Approach, Hoboken: Wiley, 2019.

[9] S. Haag, M. Cummings and M. D. J., Management Information Systems for the Information Age, New York: McGraw-Hill, 2002.

[10] E. Monk and B. Wagner, Concepts in Enterprise Resource Planning 4th Edition, Boston: Cengage Learning, 2012.

[11] K. Whitehead, Component Based Development: Principles and Planning for Business Systems, Boston: Addison-Wesley, 2002.

[12] M. Fowler, Patterns of Enterprise Application Architecture, Boston: Addison-Wesley, 2002.

[13] S. J. Metsker and W. C. Wake, Design Paterns in Java(TM) (Software Paterns Series), Boston: Addison-Wesley Professional, 2006.

[14] M. Sikora, EJB 3 Developer Guide: A Practical Guide for developers and architects to the Enterprise Java Beans Standard, Birmingham: Packt Publishing, 2008.

[15] Oracle Corporation, "Enterprise JavaBeans (EJBs)," Oracle Corporation, Austin, 2022.

[16] Oracle Corporation, "Kodo™ 4.2.0 Developers Guide for JPA/JDO," Oracle Corporation, Austin, 2015.

[17] Jakarta Persistence Team, "JAKARTA EE Jakarta Persistence," Eclipse Foundation, Ottawa, 2020.

[18] Vaadin Ltd, "Develop Web Apps in Java," Vaadin Ltd, 2023.

[19] Vaadin Ltd, "Creating a Flow Application," Vaadin Ltd, 2023.

[20] L. Stancapiano, Mastering Java EE Development with WildFly, Birmingham-Mumbai: Packt Publishing, 2017.

[21] N. Ajienka, A. Capiluppi, and S. Counsell, "An empirical study on the interplay between semantic coupling and co-change of software classes," *Empirical Software Engineering*, vol. 23, p. 1791–1825, 2018.

[22] C. Tantithamthavorn and J. Jiarpakdee, Explainable AI for Software Engineering, Monash University, 2021.

[23] S. I. Ross, F. Martnez, S. Houde, M. Muller, and J. D. Weisz, "The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development," in *IUI '23: Proceedings of the 28th International Conference on Intelligent User Interfaces,* Sydney, 2023.