

BAB software framework and micro-service technology as the basis for cooperative process-aware information systems on the Java platform

Borivoj V. Bogdanović^a, Vidan D. Marković^b, Milan T. Segedinac^c, Zora D. Konjović^d

^a Singidunum University, Belgrade, Serbia,
e-mail: borivoj.bogdanovic@gmail.com, **corresponding author**,
ORCID iD: <https://orcid.org/0009-0008-2025-176X>

^b Singidunum University, Belgrade, Serbia,
e-mail: vidan.markovic@singidunum.ac.rs,
ORCID iD: <https://orcid.org/0000-0002-5334-2237>

^c Faculty of Technical Sciences, University of Novi Sad, Novi Sad Serbia,
e-mail: milansegedinac@uns.ac.rs,
ORCID iD: <https://orcid.org/0000-0003-1743-9522>

^d Singidunum University, Belgrade, Serbia,
e-mail: zkonjovic@singidunum.ac.rs,
ORCID iD: <https://orcid.org/0000-0001-9997-1285>

 <https://doi.org/10.5937/vojtehg74-59807>

FIELD: computer sciences

ARTICLE TYPE: original scientific paper

Abstract:

Introduction/purpose: The objective of this research is to investigate the viability of using the Business Application Builder (BAB) software framework and microservice technology for the development of cooperative process-aware information systems (CPAIS) tailored for smaller business organizations with limited expert resources. The central hypothesis is that the BAB framework, based on an ontological model, combined with a microservice architecture, can facilitate automatic code generation, ensure rigorous consistency within an individual system, and enable seamless cooperation between multiple information systems.

Methods: The research methodology utilized the BAB ontology, implemented with RDF and OWL, as the conceptual model for the information system. Microservice technology was employed for communication between distributed applications, while the CodeOntology, in conjunction with predefined templates, was used for automatic code generation in the Java programming language on the Jakarta EE platform. The research design was validated through the implementation of a proof-of-concept CPAIS for a group of companies in the transport logistics sector, where each company's information system acted as a BAB instance.

Results: The implementation of the proof-of-concept system yielded significant quantitative improvements. Development time was reduced by nearly 60%, from an estimated 120 days for a traditional approach to 50 days using the BAB framework. Integration testing revealed a 75% reduction in errors related to the persistence and communication layers. The framework demonstrated high flexibility, allowing a new system node to be integrated in a single working day. The system successfully automated core logistics processes and enabled real-time data synchronization between distributed actors via standardized REST APIs.

Conclusion: The findings confirm that the BAB framework is a highly effective platform for the accelerated development of scalable, interoperable, and flexible cooperative process-aware information systems. The research validates the initial hypothesis, demonstrating that this approach provides a distinct advantage for agile business environments, especially small- to medium-sized enterprises, by drastically reducing development effort, improving software quality, and enabling rapid adaptation to changing business requirements.

Keywords: business information systems, PAIS, cooperative information systems, microservice architecture.

Introduction

Today's competitive business environment consists of business organizations (ranging from large to medium and small) made up of business processes and resources characterized by high changeability, heterogeneity, distributedness and global connectivity.

The research presented in the paper belongs to the field of information systems, more precisely to the application of semantic modeling and microservice architecture for the development of process-aware cooperative information systems. The characteristics of distributedness and heterogeneity of process-aware cooperative information systems highlight interoperability, especially semantic interoperability, as one of the main challenges (Park & Ram, 2004), while the changeability of business processes in modern business environments results in the challenge of continuous software development and delivery (Chen, 2015).

The result is a heightened need for continuous development and delivery of information systems based on business process models while ensuring collaboration and communication with external entities. Therefore, the construction of cooperative process-aware information systems (CPAIS) (Eshuis et al., 2009) becomes a key to the efficient use of resources and faster decision-making with the aim of improving

operations for business organizations of all sizes, including small- and medium-sized enterprises. In this context, the process-aware information systems (PAIS) concept (Dumas et al., 2005) provides process awareness, and the cooperation concept (Brodie & Ceri, 1992) provides collaboration and communication with external entities.

The development of the Internet, artificial intelligence and related technologies provides technological prerequisites for the development and operation of such systems. Moreover, cooperative information systems have reached high level of maturity and wide application, with a rapid increase in the utilization of artificial intelligence and semantic models utilization. However, the development and operation of such systems require significant resources, especially expert human resources that are rarely available in smaller business organizations.

The subject of research in this work is the possibility of developing CPAIS for smaller business organizations based on the BAB framework (Bogdanović et al., 2023; 2024) and micro-service technology (dos Santos Krug et al., 2024).

The research hypothesis is that it is possible to automatically generate Java code for CPAIS, including a rigorous check of its consistency, based on an ontological conceptual model of the information system extended by the ontology of the target programming language and generic structures and interfaces for working with microservices.

The research methodology applied in this work is as follows.

- The BAB ontology (Bogdanović et al., 2023; 2024; 2025), implemented through standardized Semantic Web technologies (RDF, OWL) is used as a conceptual model of an information system with generalized common business processes.
- The ontology CodeOntology (Atzeni & Atzori, 2017)], which is an ontology of the Java programming language and is also implemented through RDF and OWL, in conjunction with templates aware of generalized common business processes in the BAB ontology and ontological reasoning techniques, is used for automatic Java code generation.
- Microservices technology is used for cross communication between different applications and systems.
- The MVP (Minimum Viable Product) of the real-world cooperative information system that supports the logistic functions of transport in a waste management company is implemented, where each individual information system is represented as a BAB instance.

The paper consists of five sections, the first being this introductory section. Section two provides a brief overview of the fundamental concepts and technologies on which the research relies and of previous research in the relevant fields (process-aware information systems and cooperative information systems). Section three provides a brief overview of the software architecture of the generated information system and the software architecture of the BAB framework, along with a detailed description of the part related to working with REST services. Section four describes the process of applying the BAB framework for the development of cooperative PAIS systems including development of the MVP of the real-world cooperative information system that supports the logistic functions of transport in a network of waste management companies, and evaluation of the results. The fifth section, which is the conclusion, discusses the advantages and limitations of the proposed solution compared to other comparable solutions and identifies directions for further research.

Related work

This section presents the fundamental concepts and technologies used in the research, as well as a brief overview of previous research relevant to the paper. It places previous research on cooperative information systems and microservice architecture in the CPAIS context.

Fundamental concepts and technologies

The PAIS concept definition is provided in Dumas et al. (2005): "Process-aware information system (PAIS) is a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models" together with a detailed analysis of the concept and a classification of PAISs according to four orthogonal dimensions: (1) orientation (design-oriented and implementation-oriented), (2) participants involved (humans and software applications), (3) structure and predictability of the process (unframed, ad hoc framed, loosely framed, and tightly framed), and (4) organization scope (intra-organizational, inter-organizational).

The fact that in modern business systems the business process often extends beyond the boundary of the business system imposes a requirement for the cooperation of different business information systems that "live" in a distributed environment. Within such systems, two concepts are especially important: cooperation and communication, as a prerequisite for cooperation.

In the existing literature relevant to our research, these two concepts appear in the context of cooperative systems and service-oriented architecture (SOA), more specifically micro-service architecture. Cooperative systems represent frameworks in which several autonomous entities (software systems, services or organizations) work together to achieve a common goal, sharing information and resources, while Service-Oriented Architecture (SOA) is a software design approach that structures applications as a collection of loosely coupled, interoperable services.

Finally, the concept that is also relevant to our research is ontological modelling. In computing, ontology enables the machine readable representation of knowledge about the domain of discourse, consistency checking, queries and reasoning over the representation. When it comes to business systems, ontologies facilitate the description of the structure and behavior of the system (model and model execution) by transformation of informal models into formal, machine-readable ontological representations. Due to limited space, we briefly present here only two sources that most directly influenced the development of the BAB framework. The main difference between these studies and the BAB framework is that the latter contains a conceptual ontological model of a business information system that is further adapted to the needs of the specific application domain and explicit support for checking the consistency of the generated code, which is not the case in any of these studies. Source Strmečki & Magdalenić (2019) defines five ontologies for generating Java code using tools from the Semantic Web stack. It has a significant similarity with the BAB framework in terms of the information system's model (ontology) and the technology stack used. It differs from the BAB framework in the level of abstraction of the information system and the absence of explicit support for code consistency checking. The research presented in Đurić & Devedžić (2012) internalizes the ontological paradigm into the Clojure programming language, thus enabling semantically rich domain-driven programming on two basic platforms, Java and .NET. In addition to the absence of a conceptual ontological model of an information system, it differs from the BAB framework in terms of the target programming language because the BAB framework integrates knowledge of the target programming language into the BAB ontology and enables automatic code generation in that target language.

Implementation of those fundamental concepts in our research is done through suitable technologies: standardized Semantic Web technologies (RDF, OWL, SPARQL), microservice technologies (standardized REST services), and object programming (Java language).

Cooperative information systems and micro-service architecture in CPAIS context

The idea of cooperative (collaborative) information systems appeared in the early 1990s. Back in 1992, Michael Brodie and Stefano Ceri (Brodie & Ceri, 1992) provided the initial definitions of the key concepts of such information systems, which include multiple heterogeneous intelligent agents distributed over large computer/communication networks. Moreover, in De Michelis et al. (1998a) the authors state: "...The paradigm for the next generation of information systems will involve large numbers of information systems distributed over large, complex computer/communication networks. ...We call such next generation information systems cooperative information systems..."

Cooperative systems also have an organizational dimension, their components ("agents") relate to each other in the form of a social organization, share responsibilities and coordinate their participation in tasks (De Michelis et al., 1998b). In practice, this means that independent, federated information systems can operate autonomously, but also collaboratively execute processes to achieve common business goals. Typical examples of their application are inter-system and inter-organizational integrations e.g., the integration of multiple business applications within a company or the connection of information systems of different organizations to achieve an end-to-end supply process. Some examples of domains that strongly rely on such cooperative networks of systems are public administration (Rocheleau, 2005; Gacitúa et al., 2021; Chawwiang & Kiattisin, 2022), health care systems (Marref et al., 2022; Rinty et al., 2022; Kamal et al., 2023), education (Sanz-Angulo et al., 2025; Miranda et al., 2021), modern production systems and smart factory environments (Wang et al., 2016) and especially supply chains (Kakhki & Gargeya, 2019; McLaren et al., 2004; Gunasekaran & Ngai, 2004; Pan et al., 2021; Kot et al., 2020).

A key advantage of cooperative systems is the ability to share resources and knowledge among components, resulting in greater efficiency and innovation. These indisputable advantages of cooperative information systems are followed by significant challenges in their implementation (De Michelis et al., 1998b). Firstly, there is the complexity of integration and the issue of interoperability (Hodapp & Hanelt, 2022). The necessity for different information systems (often heterogeneous in terms of technology and structure) to communicate and exchange data smoothly requires standardized protocols and data models. In addition, security and trust are critical (Logeshwaran, 2021) since the data and

transactions are shared among multiple entities requiring authentication, authorization and permission management in a distributed environment. Process coordination is also a challenge since a collaborative process management is needed to synchronize the activities of several participants, often in real time (Steinau et al., 2021). Finally, managing changes in such systems is difficult. Each cooperative agent can evolve, so maintaining the integrity of the entire system requires good organizational coordination and flexible software architecture.

Micro-service architecture is a modern style of software design in which an application is not built as a single entity (monolith), but as a set of small, independently executable services that communicate with each other via lightweight network protocols (Wang et al., 2021). Unlike a monolithic architecture, where all modules are part of a single application and share the same environment and database, microservices are executed as separate processes, often in containers, communicating via REST API calls, messages or other network interfaces (Wang et al., 2021). This architectural approach is inspired by the principles of service-oriented architecture (SOA), but with finer granularity and a focus on the independent management of the life cycle of each service. Microservice architecture brings many well-documented advantages over traditional monolithic systems. Some of the key ones are (dos Santos Krug et al., 2024; Kumar, 2020): independent development and delivery, scalability and performance, fault tolerance, technological agility, faster delivery and development in large teams. Despite the advantages, the transition to micro-services brings serious challenges in the implementation and operational management of the system such as distributed complexity, communication delays and performance, data consistency, security, operational management and costs (dos Santos Krug et al., 2024; Kumar, 2020).

Cooperative systems and microservice architecture share a common philosophy of modularity and collaboration aimed at achieving system goals. In the context of PAIS, both cooperative systems and microservices support process orientation, the former through connecting different organizational units into a single process flow, and the latter through breaking monolithic BPMS into a series of service components. In fact, microservice architecture is recognized as one of the technologies that can support the construction of cooperative (process-aware) information systems.

Despite their complementary goals, the concepts are different. Cooperative systems emphasize organizational aspects, while microservices are purely software entities under the control of a single

organization. In other words, a microservice is typically not a separate IS at the enterprise level, but rather part of a wider system, while units in a cooperative system are often complete ISs or service systems belonging to different owners. Cooperative systems deal with a heterogeneous environment, while microservices are most often a homogeneous environment under a single orchestrator. Also, the granularity is different: a microservice is fine-grained, while a cooperative system can be coarser-grained.

The consequence is a difference in challenges. With cooperative systems, the dominant problems are standardization, interoperability and management between subjects, while with microservices, the challenges are more technical within software engineering. However, both paradigms must address distributed communication and coordination, just at different scales.

Cooperative systems and microservice architecture are not mutually exclusive but are often complementary in modern PAIS solutions. Research shows that an approach where each business process activity is implemented as a microservice brings scalability at the activity level and naturally distributed process performance (Cocconi & Villarreal, 2020; Schulte et al., 2015). Thus, PAIS can use microservices for the technical implementation of process steps, while the concept of a cooperative system ensures that these steps flow across the boundaries of different systems or organizations.

In practice, BPM systems are evolving towards microservice architectures to facilitate easier integration into cooperative environments such as the cloud (Cocconi & Villarreal, 2020).

System architecture

The BAB framework (Bogdanović et al., 2023) is designed to support the collaboration and integration of multiple software components and organizational units in a distributed environment. Based on JAKARTA EE specifications (Figure 1), it represents a modular and easily extensible approach to the development of business applications, enabling rapid development, continuous integration and automatic code generation, thus responding to the demand for continuous development and delivery. Through the use of key components such as the AbstractEntity class, AbstractDTO and, especially, the AbstractHome class (Bogdanović et al., 2025), the framework provides a standardized way to manage data and processes, thus facilitating integration with other applications and systems within a cooperative network.

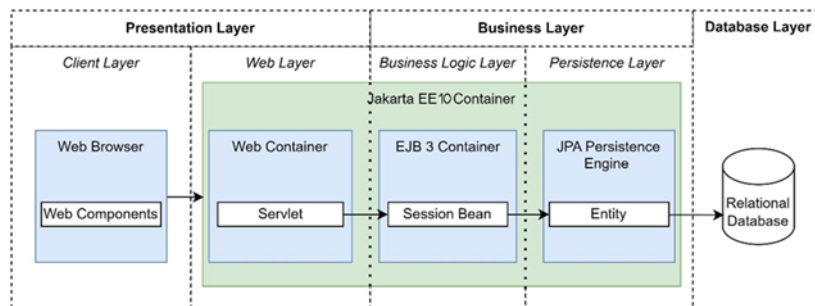


Figure 1 - Architecture of BAB based BPMS

By using ontologies and semantic technologies within the framework, BAB provides semantic interoperability, i.e., a mutual understanding of data and concepts between different systems. Standardized REST services further increase interoperability, enabling easy connection of different systems and organizational units.

Although initially developed as a monolithic application, the BAB framework fits naturally into microservice architecture concepts thanks to its clearly defined interfaces and modular internal structure. Each component of BAB can be isolated and implemented as an independent service, thus achieving easier scaling and maintenance.

In addition, BAB can be seen as the basis for a service in a microservice architecture if we define each service as a separate application instance based on the BAB framework. This approach allows the benefits of the BAB framework, such as automatic code generation, standardized operations, and rigorous validation, to be used directly within microservices.

The BAB core

At the center of the BAB framework lies a sophisticated core engineered to automate and streamline the creation of complex software by intelligently bridging the gap between the abstract concepts of business logic and the concrete implementation of source code. This is achieved through a synergistic interplay of three key components: the business-oriented BAB ontology, the foundational CodeOntology, and a powerful set of code generation templates. Together, these elements form a robust system that not only accelerates development but also enhances the consistency, correctness, and maintainability of the resulting software.

The code generation diagram (Figure 2) illustrates the following generation process:

1. First, the developer defines the application's structure and business logic by creating an ontology that imports and extends the BAB and CodeOntology. This ontology serves as a comprehensive model of the application to be built.
2. Next, the OwlBabGenerator tool parses this ontology and extracts all the relevant information about the classes, fields, relationships, and business rules that have been defined.
3. Finally, this information is fed into the FreeMarker templates, which use it to generate the complete and compilable Java source code for the application's entities, data transfer objects, session beans, and user interface views. These templates, which have the .ftl file extension, contain the boilerplate code for the various components of a BAB application. They are designed with placeholders that can be filled in with the specific details extracted from the ontologies.

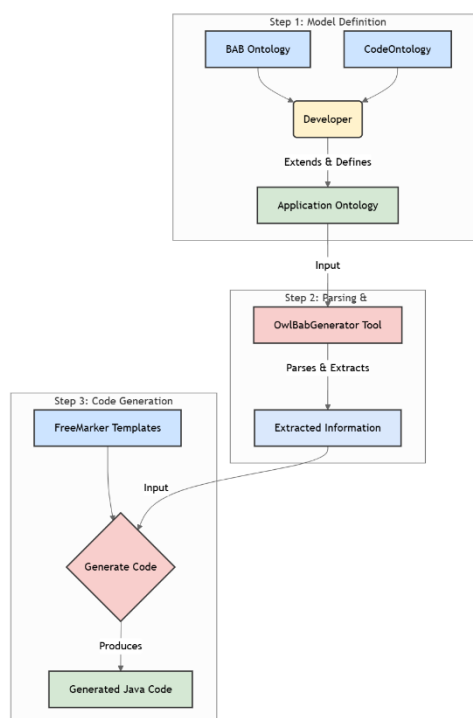


Figure 2 - Code generation diagram

The business-oriented BAB ontology is composed of several key components:

- Classes that represent the basis of an information system, i.e., atomic entities that are used in business processes. The most important ones, which can be extended by adding interface classes, are: AbstractEntity (represents the base for all subjects, later mapped to database tables and entities in Object-Relational Mapping (ORM)), AbstractDTO (principal concept for all different “looks” that one can have at data; used to define Domain Transfer Objects (DTO) and build SQL queries for populating views), AbstractHome (connects data from the former two classes by enabling transformation from entity to DTO and encapsulates business logic tasks).
- Classes that are a formal description of common web application elements: GenericMainView (used to represent the shell of the web app, responsible for navigation and security) and GenericView (an abstraction of one web page representing data and controls for a specific triplet of AbstractEntity, AbstractDTO and AbstractHome).
- Classes that are abstractions of entity properties, including their type, name, and access modifier. There are also classes that represent restrictions on those properties, like NotNull and NotEmpty. These properties are vital for enforcing business rules and data validation at the ontology level, which in turn influences the generated code’s reliability and robustness.
- The fourth type of classes are relationship description classes. Ontology supports the four common types of relationships, one to one, one to many, many to one and many to many.
- Apart from classes there are object properties that describe relations between classes. We mainly use them to explain how classes compose together.
- The last part of the ontology is a Task concept, which is used to model business processes. It consists of three parts, subject, predicate, and object, just like RDF itself. It usually has a duration, can be combined with other tasks to form a complex process, knows which business subject has privileges to start it, can require some external data (Resource) and has a status of competition.

This template-based approach to code generation offers several significant advantages. It ensures that the generated code is consistent, adheres to established best practices, and is closely tailored to the specific requirements of the business application as defined in the ontology. This

not only accelerates the development process but also results in a final product that is more robust, maintainable, and aligned with the needs of the business.

Generic structures and interfaces for working with REST services

The concept of cooperative business information systems directly relates to the ability of systems to programmatically communicate, collaborate and share resources. The BAB (Business Application Builder) library, although primarily focused on the rapid development of business applications with a user interface based on the Vaadin technology, contains a dedicated package `rs.co.bora5.programs.bab.session.rest`. This package, together with a series of specialized interfaces, provides the fundamental architecture for transforming a BAB application from a monolithic, user-oriented system into an open platform that can function as a service within a wider digital ecosystem. Using the standard Java technology for web services, JAX-RS (Jakarta RESTful Web Services), BAB enables developers to expose business logic and data via REST APIs in an elegant and standardized way.

The core of this functionality relies on three key abstract classes within the rest package, which define a pattern for consistency, security, and proper data formatting. The central component is `AbstractEndpoint`, which represents the basis for each specific REST endpoint. Instead of each developer writing an endpoint from scratch, it inherits `AbstractEndpoint<T>`, where T is the entity type being managed. The class is annotated with `@Produces(MediaType.APPLICATION_JSON)`, ensuring that all endpoints will return data in JSON format by default. Its most important aspect is the direct connection with `AbstractHome` beans, which encapsulate business logic and base operations. Through dependency injection (`@Inject`), the endpoint obtains an instance of the corresponding Home bean, making the API layer a thin facade that only exposes pre-existing, well-defined business logic.

Security is handled through `AbstractApiKeyRequestFilter`. This class, which implements `JAX-RS ContainerRequestFilter` and is annotated with `@Provider`, automatically intercepts all incoming REST requests. Its task is to check whether the request contains a valid API key in HTTP. If the key does not exist or is not valid, the request is rejected with the status 401 Unauthorized. Only validated requests are forwarded to the appropriate endpoint, thus protecting the data and business logic of the system.

The third key component is `AbstractObjectMapperContextResolver`, which controls the conversion of Java objects to JSON and vice versa. By implementing `ContextResolver<ObjectMapper>`, this class allows the developer to create and configure an `ObjectMapper` instance (from the Jackson library) for the entire application. Custom serializers, such as `LocalDateSerializer`, are registered here. This ensures that the JSON produced by the API is consistent, predictable, and easy to understand by client applications.

The success of the Rest package lies in its deep integration with other parts of the BAB architecture, which is additionally formalized through specific interfaces that contain the word "Rest" in their name. These interfaces define the contracts that the Entity and Home classes must fulfill to fit properly into the REST architecture.

At the data model level (`AbstractEntity`), interfaces such as `RestPublicIdEntityInterface` and `RestResourceEntityInterface` define basic requirements. The entity implementing `RestPublicIdEntityInterface` undertakes to have a `getPublicId()` method, providing a stable, publicly exposed identifier that is not directly tied to a primary key in the database. Similarly, `RestResourceEntityInterface` defines through inheritance and extension additional methods for monitoring the state of synchronization. Interface `RestResourceLocationInterface` additionally specifies how to form the complete URL of the resource with the `getRestResourceLocation` method.

At the level of business logic (`AbstractHome`), the corresponding interfaces ensure that Home beans provide the necessary methods for working with REST endpoints. `RestPublicIdHomeInterface` defines the contract for the Home bean that it must have a method `findByPublicId(String publicId)`, allowing an endpoint to easily find an entity based on its public identifier. On the other hand, `RestResourceHomeInterface` specifies methods for sending resources, such as the `send` method. This strict division of responsibilities, formalized through interfaces, guarantees that the system is predictable and easy to maintain.

When all the components come together, the flow of an API request is clear and robust. The client sends an HTTP request with an API key. `AbstractApiKeyRequestFilter` validates it. JAX-RS routes it to the appropriate endpoint it inherits from `AbstractEndpoint`. The endpoint, using an injected Home bean that implements e.g. `RestPublicIdHomeInterface` calls business logic (e.g. `findByPublicId(...)`). The home bean finds the entity in the database. An entity, which implements e.g. `RestPublicIdEntityInterface`, returns to the endpoint. In the end,

AbstractObjectMapperContextResolver ensures that the entity is properly serialized to JSON before sending the response to the client.

In conclusion, the rest package and associated rest interfaces are not just an add-on, but a key architectural element that makes the BAB library truly cooperative. They provide a robust, secure and standardized framework for creating REST APIs, promoting code reuse and clean design principles. In this way, the BAB application becomes not only a tool for internal users, but also a valuable resource that can communicate and collaborate with other systems, opening the door to more complex and powerful business integrations.

The BAB based implementation of cooperative PAIS systems

This paper investigates the application of the BAB software framework in the implementation of cooperative PAIS systems, with a special emphasis on practical implementation in Java.

Methodology

The methodology consists of several interconnected phases that ensure a comprehensive understanding of how the BAB framework is used to build distributed and cooperative information systems.

The first phase involves a detailed analysis of business and technical requirements for developing cooperative PAIS systems. In this phase, key challenges, such as the need for rapid development, scalability, interoperability and continuous integration, are identified. Based on literature analysis and case studies, the need for automation of the code generation process and business logic validation is particularly emphasized.

The second phase, preparing the development infrastructure, is a key step in the BAB framework development process, as it provides a stable basis for further implementation. It includes the following activities:

Configuration of development tools and environment: In this phase, a development environment based on the JAKARTA EE platform is initially set up, which provides a rich set of standard tools for business application development. An SQL database is integrated for reliable data storage and processing, as well as the Vaadin framework for developing modern, interactive web applications.

Installation and configuration of the BAB library: Since the BAB framework already exists as a complete, pre-prepared solution, proper integration of the BAB library into the development project in question

requires special attention. Within this activity, detailed adjustments of basic classes and structures such as AbstractEntity, AbstractDTO and especially AbstractHome should take place as well as adjustments of generic structures and interfaces definitions for working with REST services, which enables the automation of data exchange processes and integration with external systems.

Implementation

The implementation of a real process-aware business information system (PAIS) using the BAB framework took place through the following key activities:

Detail domain model creation: Based on defined business needs and goals, a detailed domain model was first created that includes entities, their mutual relationships and attributes necessary for business processes. The BAB framework provides tools for easy and efficient mapping of these entities to the persistence layer using JPA technology, thus ensuring data consistency, integrity and easier system implementation. This mapping enables efficient entity lifecycle management and optimized execution of data operations.

Implementation of business logic: To implement business logic, EJB components and JPA technology were used, which ensured high performance, scalability and transaction management of applications. The AbstractHome class of the BAB framework offers generic and optimized methods for performing CRUD operations. It supports lazy loading, automatic management of transactions and validations, which significantly simplifies application development and reduces the chance of errors. This component enables consistent implementation of business rules and facilitates future changes to business logic.

User interface development: The Vaadin framework¹ was used to create the user interface. The developed interface provides the ability to view, enter, modify and delete data in a clear and structured way. In addition to the basic functions, the Vaadin framework provides a real-time synchronization, which instantly shows all users the changes made in the system, thus achieving better interactivity and communication between users and contributing to more efficient development of business processes.

Verification and validation: The verification and validation of the system implemented using the BAB framework was carried out through a multi-layered testing strategy, designed to confirm the correctness of both

¹ vaadin.com

individual components in isolation, and of their mutual interactions within the overall architecture. The approach included unit tests, integration tests and performance tests, each tailored to the specifics of the BAB architecture.

The focus of unit testing is on isolating and verifying the key building blocks of the application: the business logic class (Home) and the API layer class (Endpoint). For this purpose, industry standards were used, JUnit 5² as an executive framework, Mockito³ for simulation (mock-up) of dependencies and Java Microbenchmark Harness (JMH)⁴ to test query performance.

Methods within the inheriting classes AbstractHome are tested in isolation from the database. Dependencies, such as EntityManager, are simulated to verify the correctness of business rules, validation logic and data transformations (e.g. from entity to DTO), without the need to establish an actual connection to the database.

The abstract nature of the class AbstractEndpoint required the creation of test-specific, concrete implementations. Through these implementations, and with the help of the Mockito framework, the operation of Home beans was simulated. This approach enabled the verification of key scenarios.

Creating Resources (POST): Using the Mockito.verify() construct, it is verified that the endpoint, after receiving the request, calls the save() method on the Home bean with the correct data and returns a 201 Created status.

Get a resource (GET): It was checked whether the endpoint returns a status of 200 OK when the simulated Home bean returns an entity, or 404 Not Found when it returns null.

Integration tests are designed to verify the entire flow of information, from the interface to the database, with special emphasis on cooperative functionalities.

The default send method from the interface RestResourceHomeInterface has been verified. These tests involved running a method on a real Home component that communicates with a test database (e.g. H2) and sends an HTTP request to a mock external server (e.g. WireMock). The test confirmed that after receiving a successful response (HTTP 200) from the fake server, the status fields on

² <https://junit.org/junit5/>

³ <https://site.mockito.org/>

⁴ <https://www.geeksforgeeks.org/advance-java/java-microbenchmark-harness-jmh/>

the entity (isSync, syncAt) are correctly updated and persisted in the database.

Using the Arquillian tool⁵ to run the application in a controlled container, the complete incoming flow is tested. The external HTTP client sent requests to the running application, which verifies the entire chain:

1. Correct functioning of AbstractApiKeyRequestFilter authentication components.
2. Proper deserialization of the JSON body of the request, guided by the configuration from AbstractObjectMapperContextResolver.
3. Successful running of business logic in Home bean and writing data into the database.
4. Sending the correct HTTP response back to the client.

To analyze the performance and scalability of the loaded system, the tools Java Microbenchmark Harness (JMH) and JUnit 5 were used. Scenarios were defined that simulate the work of a large number of competing users and systems, with a focus on:

- API Endpoint responsiveness: Measured response time for read and write operations at different load levels.
- Database efficiency: Analyzed the efficiency of JPQL queries, with special reference to the lazy loading mechanisms that AbstractHome uses by default, to ensure that N+1 problems do not occur when loading complex datasets.

This comprehensive, multi-layered testing strategy ensures that all aspects of the architecture implemented using the BAB framework, from the logic of individual methods to complex, transactional flows between systems, are thoroughly verified, which guarantees high quality, reliability and performance of the final PAIS solution.

MVP development and evaluation of results

The proof of concept is the implementation of a process- PAIS serving a group of companies dealing with transport logistics, where the roles of each actor (company) are clearly defined. The major business challenge of the group was ineffective communication within individual companies and difficult coordination when planning and executing joint actions, which led to delays, manual errors and a lack of transparency in work.

The implemented network of PAIS, based on the BAB framework, aimed to automate key processes and enable seamless cooperation between distributed systems, as well as the integration of legacy

⁵ <https://arquillian.org/>

applications that already function within companies (e.g., a material bookkeeping program).

Based on previously defined architectural and technical concepts, a system was implemented in which ready-made classes and services from the BAB library are used for processing business processes.

The implementation of the system is a multi-layered application (Figure 1), which includes:

- Persistent layer: Using JPA (Java Persistence API) and ORM tools (e.g. Hibernate), domain entities (such as *Announcement* and *TravelOrder*) are mapped to database tables. The BAB framework enables standardized management of these entities across classes *AbstractEntity*.
- Business layer: The implementation of basic operations (CRUD) and business logic is conducted through EJB components. Class *AbstractHome* serves as a central mechanism for managing entities and their DTO (Data Transfer Object) representations, thus enabling logic encapsulation and facilitating further integration.
- Web layer: Using the Vaadin framework, an interface was developed that allows users to view, enter, modify and delete data. This layer ensures user interaction and real-time synchronization between distributed system instances.
- REST communication: Class *AbstractEndpoint* implements RESTful operations, providing a unique interface for accessing services. By integrating REST principles, the system achieves stateless interaction, which contributes to scalability and interoperability.

The BAB framework, as part of its modular architecture, enables the integration of REST services through the abstraction of the *AbstractEndpoint* class. This class, using generic parameters, provides uniform processing of REST requests for creating, retrieving, updating, and deleting data. Thanks to the methods *create*, *findById*, *listAll* and *deleteById*, a consistent and easily maintainable approach to handling data within the system is provided.

AbstractEndpoint is the basis of the entire REST infrastructure, providing abstraction over CRUD operations, which in turn rely on *AbstractHome* methods. This class allows for easy extension of the system to new entities without the need for major changes to the existing infrastructure, which leads to a reduction in technical debt and easier code maintenance.

The `RestResourceHomeInterface` interface is used for dynamic synchronization. This interface allows sending data between different nodes of the system or external services. It defines the `send` method, which enables automatic forwarding of data to remote services, including the additional `Transformations` method for data customization before distribution, thus achieving greater interoperability between different components or external systems.

The key aspect of REST service implementation in the BAB framework is the configuration of data serialization and deserialization. For this, the Jackson library is used, whose configuration is implemented through the `AbstractObjectMapperContextResolver` class.

Managing the authentication and authorization of REST requests is another critical component of REST service integration in the BAB framework. The `AbstractApiKeyRequestFilter` class is used for this. This class checks the validity of API keys through cooperation with EJB components that implement the `OperatorHomeInterface` interface. The authentication process itself takes place by extracting the API key from the HTTP request header, then by checking its validity, and finally by allowing access to resources only to users with appropriate permissions. This ensures a high level of security and prevents unauthorized access to data and operations within the system.

`RestPublicIdHomeInterface` and `RestPublicIdEntityInterface` are used to identify entities, which define access to entities through their unique public identifier (`publicId`). This enables clear identification of each individual entity, thus facilitating work with data and increasing transparency in the operation of the system. `RestResourceEntityInterface` further extends these basic interfaces with attributes such as external entity status, synchronization information, and the time of the last successful synchronization, thereby providing additional information about the state and behavior of entities within distributed systems.

The combination of the above classes and interfaces results in a comprehensive solution for the implementation of REST services within the BAB environment. This integration enables not only standardized and simple access to data, but also high efficiency in the operation of cooperative applications. In addition, it provides flexibility for future extensions, and high data security.

Example implementation: Announcement and Travel Order

In this section, the application of the BAB framework is presented in detail through real-life examples of logistics and waste transportation, focusing on the entities Announcement and Travel Order.

The Announcement entity is a key part of the logistics system that serves to announce the place, time and method of waste collection. Its objective is to provide timely and accurate information to logistics operations regarding planned waste collection activities. As a specialization of the AbstractEntity class, Announcement uses the interface RestResourceEntityInterface, enabling automatic synchronization with external systems. This ensures consistency and up-to-date information shared between different applications or distributed system instances.

Class AnnouncementEndpoint extends AbstractEndpoint and provides REST API access to basic CRUD operations, and additional specialized validation (Listing 1) and invalidation (Listing 2) functions for announcements.

```
@POST
@Path("/validate")
public Response validateAnnouncement(@Context
HttpServletRequest request, AnnouncementDTO entity) {
    // Checking the validity of the carrier and order
    // Create or update the Announcement entity
    // Propagation of changes through the system using
    //the updateTables method
}
```

Listing 1 - Announcement validation method

```
@POST
@Path("/invalidate")
public Response invalidateAnnouncement(@Context
HttpServletRequest request, AnnouncementDTO entity) {
    // Checking the carrier and order
    // Remove invalid announcement
    // Update and synchronize data
}
```

Listing 2 - Method invalidate announcement

The validation process includes checking the carrier's data via PIB and checking the existence of the related Travel Order. If the validation is completed successfully, the Announcement entity is created or updated, and the change is propagated to all other instances of the system using

the `updateTables` method. In case of unsuccessful validation or identified data inconsistency, the invalidation method enables the announcement to be removed and the resulting changes to be synchronized.

`TravelOrder` is a central entity that aggregates transport information for a series of announcements, providing a consolidated view of all transport activities. It implements the interfaces `RestResourceEntityInterface`, `CheckPayoutEntityInterface` and `LocationInterface`, which enables not only external communication but also detailed tracking of operations and locations. This way, logistics operators can more easily manage and monitor transport activities.

The `TravelOrderEndpoint` class defines specialized REST operations for validating and invalidating travel orders. Validation checks the correctness of information about the carrier, driver, vehicle and all related announcements. If the validation is successful, the entity Travel Order is updated or created, and the changes are propagated through the system. If there are errors or inconsistencies, invalidation allows operations to be reversed, thereby maintaining data integrity across the distributed system.

An important aspect of the implementation is the close connection between the Announcement and Travel Order entities. When a travel order is created or validated, all related entities, announcements, are automatically updated. This integration ensures that all data is synchronized, which allows logistics personnel to have centralized access to all the necessary information about individual transport operations.

Real-time synchronization, enabled by methods like `updateTables`, is a key functionality of the BAB framework. Any change made on one node of the system is instantly propagated to all connected instances, thus achieving high interoperability. This approach is particularly useful in dynamic logistics environments where timeliness of information is critical to the efficiency of operations.

In the implementation of the examples of Announcement and Travel Order, the validation of business rules is of particular importance. Before any changes are forwarded to the system, a detailed check of the validity of the data is performed, including checks such as the identification of the carrier, the connection with the appropriate entities and the correctness of the transport details. This approach ensures high reliability of the entire logistics system and minimizes operational errors.

Finally, the implementation through the BAB framework enables easy extension and adaptation of these entities to specific business needs. This flexibility additionally contributes to the efficiency of the system and enables quick adaptation to market requirements or regulatory changes.

Evaluation of results

The benefits of the BAB framework were quantified through a direct, sequential comparative study designed to measure its impact on development efficiency, code quality, and system flexibility.

Evaluation methodology

The evaluation was conducted by a dedicated team of two developers who implemented the same PAIS handling business processes of a company in the waste management sector, in two consecutive phases:

1. **Baseline Development (Traditional Approach):** The team first built the complete system using a traditional development approach. This involved using the standard Jakarta EE platform without the BAB framework, requiring manual implementation of the persistence layer, communication logic, and other common business application features. The metrics collected during this phase, most notably development time, served as the baseline for our comparison.
2. **Framework-based Development (BAB Approach):** After completing the baseline system, the same team then redeveloped the exact same system with the same functional requirements. However, in this phase, they utilized the BAB framework. This allowed for a direct comparison of the development effort and outcomes when leveraging the framework's abstractions, code generation, and built-in functionalities.

This sequential approach ensures that the comparison is not based on estimates, but on real-world, measured data, while keeping the developer skill level constant across both phases.

Results and analysis

The comparative study revealed significant quantitative and qualitative advantages of using the BAB framework. The key findings are summarized in Table 1.

The data clearly demonstrates that applying the BAB framework reduced the total development effort from 120 working days to just 50, a time-saving of almost 60%. This acceleration is attributed to the framework's automation of repetitive tasks and the provision of standardized, pre-built components.

Furthermore, the quality of the final product was substantially improved. Quantitative analysis during the integration testing phase

showed that the standardization imposed by the BAB framework reduced the number of errors, mostly related to the persistence and communication layers, by over 75% compared to the baseline implementation. This highlights the framework's role in producing more robust and reliable code.

Table 1 - Comparative analysis of development metrics

Metric	Traditional Approach (Phase 1)	BAB Framework (Phase 2)	Improvement
Number of tables	133	133	None, same model
Number of classes	856	684	~20% reduction
Number of lines of code	136632	54799	~60% reduction
Development Time (person-days)	120	50	~60% reduction
Total number of reported errors (QA)	52	13	~75% less
New Node Integration Time	Not Applicable / Complex	1 working day	High Flexibility

The evaluation also confirmed the system's flexibility. The time required to integrate an entirely new node (e.g., a new company) into the cooperative system was just one working day. This rapid integration is a direct result of the framework's modular architecture and standardized REST interfaces. This capability was inherent in the BAB design, whereas achieving it with the traditional approach would have required significant additional, complex development.

These quantitative improvements are supported by the framework's rich, out-of-the-box feature set, which eliminated the need to develop or integrate third-party solutions for the following:

- User-specific app personalization;
- Notification services (SMS, email, and browser push notifications);
- Advanced security with support for authenticator applications;
- Real-time data synchronization using web socket technology;
- Application localization and standardized PWA APIs for hardware access.

Finally, the framework's reliance on stateless EJB components ensures the business layer is inherently scalable and ready for clustered environments, guaranteeing high performance from the ground up.

Conclusion

This paper presents the BAB framework intended for accelerating the development of business information systems that are process-aware and adaptable to changes. The key aspects of the BAB framework, its modular architecture, generic structures and REST service integration possibilities are analyzed. The advantages of using abstract models and ontological approaches, which enable automatic generation of program code and faster implementation of applications, are highlighted.

Evaluation results and comparisons with other popular frameworks such as Spring, Quarkus, and Jakarta EE show that the BAB framework offers a clear advantage for organizations that strive for simplicity, flexibility, and rapid adaptation to change. Although there are certain limitations, primarily poor available documentation and the size of the user community, the BAB framework provides a strong basis for further improvements.

Recommendations for future research include the development of more detailed and comprehensive documentation of the BAB framework, which would enable easier adoption, implementation and maintenance of this system.

In addition, integrating more advanced features to manage business processes and enhance security is recommended. The implementation of tools for automatic monitoring and optimization of business processes would enable more efficient management of resources. Improving security features, such as authentication, authorization and protection against common attacks, would increase the reliability and resilience of systems based on the BAB framework.

One of the concrete plans for further development is the creation of a new version of the BAB library called BAB-Q, which would be based on the Quarkus framework. This version would be an extension of the existing Jakarta EE specifications, allowing easier integration into modern microservice architectures and containers, with significantly better support for running in Docker-based environments and Kubernetes technologies.

Finally, as a long-term direction of the BAB development, the authors consider the inclusion of artificial intelligence methods and techniques, more specifically the application of large language models for generating OWL code.

References

- Atzeni, M. & Atzori, M. 2017. CodeOntology: RDF-ization of Source Code. In: *The Semantic Web – ISWC 2017: 16th International Semantic Web Conference*, Vienna, Austria, 21–25 October, str. 20–28. Cham: Springer. Available at: https://doi.org/10.1007/978-3-319-68204-4_2
- Bogdanović, B., Konjović, Z. & Obradović, Đ. 2023. Bab (Business Application Builder) Framework for Rapid Development of Business Information Systems. In: *Sinteza 2023: International Scientific Conference on Information Technology and Data Related Research*, Belgrade, str. 76–84. Available at: <https://doi.org/10.15308/Sinteza-2023-76-84>
- Bogdanović, B., Marković, V., Obradović, Đ., Segedinac, M. & Konjović, Z. 2025. Ontological Modelling and Reasoning for the AbstractHome Class of the BAB Framework for PAIS Systems. In: *Sinteza 2025: International Scientific Conference on Information Technology, Computer Science, and Data Science*, Novi Sad, str. 128–135. Available at: <https://doi.org/10.15308/Sinteza-2025-128-135>
- Bogdanović, B., Obradović, Đ., Segedinac, M. & Konjović, Z. 2024. BAB Framework – Towards an Extensible Software Platform for AI-Augmented Process Aware Business Information Systems. In: Trajanović, M., Filipović, N. & Zdravković, M. eds. *Disruptive Information Technologies for a Smart Society*. Cham: Springer Nature Switzerland, str. 197–212. Available at: https://doi.org/10.1007/978-3-031-71419-1_18
- Brodie, M.L. & Ceri, S. 1992. On Intelligent and Cooperative Information Systems: A Workshop Summary. *International Journal of Intelligent and Cooperative Information Systems*, 1(2), str. 249–289. Available at: <https://doi.org/10.1142/S0218215792000209>
- Chawvian, A. & Kiattisin, S. 2022. Sustainable Development: Smart Co-Operative Management Framework. *Sustainability*, 14(6), article 3641. Available at: <https://doi.org/10.3390/su14063641>
- Chen, L. 2015. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*, 32(2), str. 50–54. Available at: <https://doi.org/10.1109/MS.2015.27>
- Cocconi, D. & Villarreal, P. 2020. Microservices-based Approach for a Collaborative Business Process Management Cloud Platform. In: *2020 XLVI Latin American Computing Conference (CLEI)*, Loja, Ecuador, str. 128–137. Available at: <https://doi.org/10.1109/CLEI52000.2020.00022>
- De Michelis, G. et al. 1998a. Cooperative Information Systems: A Manifesto. In: Papazoglou, M.P. & Schlageter, G. eds. *Cooperative Information Systems: Trends and Directions*. Amsterdam: Elsevier Science, str. 1-37. ISBN: 8884250366.
- De Michelis, G., Dubois, E., Jarke, M., Matthes, F., Mylopoulos, J., Schmidt, J.W. & Woo, C. 1998b. A Three-Faceted View of Information Systems. *Communications of the ACM*, 41(12), str. 64–70. Available at: <https://doi.org/10.1145/290133.290150>

dos Santos Krug, D., Chanin, R. & Sales, A. 2024. Exploring the Pros and Cons of Monolithic Applications versus Microservices. In: *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024)*, Vol. 2, str. 256–263. Available at: <https://doi.org/10.5220/0012703300003690>

Dumas, M., van der Aalst, W.M.P. & ter Hofstede, A.H.M. eds. 2005. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Hoboken, NJ: Wiley-Interscience. Available at: <https://doi.org/10.1002/0471741442>

Đurić, D. & Devedžić, V. 2012. Incorporating the Ontology Paradigm into a Mainstream Programming Environment. *Informatica*, 23(2), str. 203–224. Available at: <https://doi.org/10.15388/Informatica.2012.357>

Eshuis, R., Mendling, J., Minor, M. & Weber, B. 2009. Agile Cooperative Process-Aware Information Systems (ProGility 2009) Workshop Report. In: *18th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, str. 219–220. Available at: <https://doi.org/10.1109/WETICE.2009.51>

Gacitúa, R. et al. 2021. Recent Models for Collaborative E-Government Processes: A Survey. *IEEE Access*, 9, str. 19602–19618. Available at: <https://doi.org/10.1109/ACCESS.2021.3050151>

Gunasekaran, A. & Ngai, E.W.T. 2004. Information Systems in Supply Chain Integration and Management. *European Journal of Operational Research*, 159(2), str. 269–295. Available at: <https://doi.org/10.1016/j.ejor.2003.08.016>

Hodapp, D. & Hanelt, A. 2022. Interoperability in the Era of Digital Innovation: An Information Systems Research Agenda. *Journal of Information Technology*, 37(4), str. 407–427. Available at: <https://doi.org/10.1177/02683962211064304>

Kakhki, M.D. & Gargeya, V.B. 2019. Information Systems for Supply Chain Management: A Systematic Literature Analysis. *International Journal of Production Research*, 57(15–16), str. 5318–5339. Available at: <https://doi.org/10.1080/00207543.2019.1570376>

Kamal, M.A., Ismail, Z., Shehata, I.M., Djirar, S., Talbot, N.C., Ahmadzadeh, S., Shekoohi, S., Cornett, E.M., Fox, C.J. & Kaye, A.D. 2023. Telemedicine, E-Health, and Multi-Agent Systems for Chronic Pain Management. *Clinics and Practice*, 13(2), str. 470–482. Available at: <https://doi.org/10.3390/clinpract13020042>

Kot, S., Ul Haque, A. & Baloch, A. 2020. Supply Chain Management in SMEs: Global Perspective. *Montenegrin Journal of Economics*, 16(1), str. 87–104. Available at: <https://doi.org/10.14254/1800-5845/2020.16-1.6>

Kumar, H. 2020. *Microservices vs. Monolithic* [online]. Available at: <https://sterling.com/microservices-vs-monolithic/> [Accessed: 12 March 2025].

Logeshwaran, J. 2021. AICSA – An Artificial Intelligence Cyber Security Algorithm for Cooperative P2P File Sharing in Social Networks. *ICTACT Journal on Data Science and Machine Learning*, 3(1), str. 251–253. Available at: https://ictactjournals.in/paper/IJDSML_Vol_3_Issue_1_Paper_2_251_253.pdf [Accessed: 12 March 2025].

- Marref, R., Amsha, K.A. & Lewkowicz, M. 2022. Practice-Centered Approach to Design Cooperative Healthcare Information Systems: Data, Architectural and Organizational Challenges. *Computing and Informatics*, 41(1), str. 191–212. Available at: https://doi.org/10.31577/cai_2022_1_191
- McLaren, T.S., Head, M.M. & Yuan, Y. 2004. Supply Chain Management Information Systems Capabilities: An Exploratory Study of Electronics Manufacturers. *Information Systems and e-Business Management*, 2, str. 207–222. Available at: <https://doi.org/10.1007/s10257-004-0035-5>
- Miranda, J., Navarrete, C., Noguez, J., Molina-Espinosa, J.-M., Ramírez-Montoya, M.-S., Navarro-Tuch, S.A., Bustamante-Bello, M.-R., Rosas-Fernández, J.-B. & Molina, A. 2021. The Core Components of Education 4.0 in Higher Education: Three Case Studies in Engineering Education. *Computers & Electrical Engineering*, 93, article 107278. Available at: <https://doi.org/10.1016/j.compeleceng.2021.107278>
- Pan, S., Trentesaux, D., McFarlane, D., Montreuil, B., Ballot, E. & Huang, G.Q. 2021. Digital Interoperability in Logistics and Supply Chain Management: State-of-the-Art and Research Avenues towards Physical Internet. *Computers in Industry*, 128, article 103435. Available at: <https://doi.org/10.1016/j.compind.2021.103435>
- Park, J. & Ram, S. 2004. Information Systems Interoperability: What Lies Beneath? *ACM Transactions on Information Systems*, 22(4), str. 595–632. Available at: <https://doi.org/10.1145/1028099.1028103>
- Rinty, M.R., Proddhan, U.K. & Rahman, M.M. 2022. A Prospective Interoperable Distributed e-Health System with Loose Coupling in Improving Healthcare Services for Developing Countries. *Array*, 13, article 100114. Available at: <https://doi.org/10.1016/j.array.2021.100114>
- Rocheleau, B. 2005. *Public Management Information Systems*. Hershey, PA: IGI Global Scientific Publishing. Available at: <https://doi.org/10.4018/978-1-59140-807-9>
- Sanz-Angulo, P., Galindo-Melero, J., De-Diego-Ponceta, S. & Martín, Ó. 2025. Promoting Soft Skills in Higher Engineering Education: Assessment of the Impact of a Teaching Methodology Based on Flipped Learning, Cooperative Work and Gamification. *Education and Information Technologies*, 30, str. 13463–13506. Available at: <https://doi.org/10.1007/s10639-025-13322-0>
- Schulte, S., Janiesch, C., Venugopal, S., Weber, I. & Hoenisch, P. 2015. Elastic Business Process Management: State of the Art and Open Challenges for BPM in the Cloud. *Future Generation Computer Systems*, 46, str. 36–50. Available at: <https://doi.org/10.1016/j.future.2014.09.005>
- Steinau, S., Andrews, K. & Reichert, M. 2021. Coordinating Large Distributed Relational Process Structures. *Software and Systems Modeling*, 20, str. 1403–1435. Available at: <https://doi.org/10.1007/s10270-020-00835-0>
- Strmečki, D. & Magdalenčić, I. 2019. An Ontological Model for Generating Complete, Form-based, Business Web Applications. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 10(8), str. 34–38. Available at: <https://doi.org/10.14569/IJACSA.2019.0100805>

Wang, S., Wan, J., Li, D. & Zhang, C. 2016. Implementing Smart Factory of Industrie 4.0: An Outlook. *International Journal of Distributed Sensor Networks*, 12(1), article 3159805. Available at: <https://doi.org/10.1155/2016/3159805>

Wang, S., Wan, J., Zhang, D., Li, D. & Zhang, C. 2016. Towards Smart Factory for Industry 4.0: A Self-Organized Multi-Agent System with Big Data Based Feedback and Coordination. *Computer Networks*, 101, str. 158–168. Available at: <https://doi.org/10.1016/j.comnet.2015.12.017>

Wang, Y., Kadiyala, H. & Rubin, J. 2021. Promises and Challenges of Microservices: An Exploratory Study. *Empirical Software Engineering*, 26(4), article 63. Available at: <https://doi.org/10.1007/s10664-020-09910-y>

Softverski okvir BAB i tehnologija mikro-servisa kao osnove za kooperativne procesno-svesne informacione sisteme na Java platformi

Borivoj V. Bogdanović^a, **autor za prepisku**, Vidan D. Marković^a, Milan T. Segedinac^b, Zora D. Konjović^a

^a Univerzitet Singidunum, Beograd, Srbija

^b Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Novi Sad, Srbija

OBLAST: računarske nauke

KATEGORIJA (TIP) ČLANKA: originalni naučni rad

Sažetak:

Uvod/cilj: Cilj ovog istraživanja je da se ispita izvodljivost korišćenja softverskog okvira „Business Application Builder“ (BAB) i mikro-servisne tehnologije za razvoj kooperativnih procesno-svesnih informacionih sistema (CPAIS) prilagođenih manjim poslovnim organizacijama sa ograničenim stručnim resursima. Centralna hipoteza je da BAB okvir, zasnovan na ontološkom modelu, u kombinaciji sa mikro-servisnom arhitekturom, može olakšati automatsko generisanje koda, obezbediti rigoroznu doslednost unutar pojedinačnog sistema i saradnju između više informacionih sistema.

Metode: Metodologija istraživanja koristila je BAB ontologiju, implementiranu pomoću RDF-a i OWL-a, kao konceptualni model informacionog sistema. Mikro-servisna tehnologija je korišćena za komunikaciju između distribuiranih aplikacija, dok je CodeOntology, u sprezi sa unapred definisanim šablonima, korišćena za automatsko generisanje koda u programskom jeziku Java na Jakarta EE platformi. Dizajn istraživanja je validiran kroz implementaciju bazičnog (MVP) CPAIS-a za grupu kompanija u sektoru transportne logistike, gde je informacioni sistem svake kompanije delovao kao BAB instanca.

Rezultati: Implementacija sistema za dokaz koncepta donela je značajna kvantitativna poboljšanja. Vreme razvoja je smanjeno za skoro 60%, sa procenjenih 120 dana za tradicionalni pristup na 50 dana korišćenjem BAB okvira. Integraciono testiranje je utvrdilo smanjenje grešaka u vezi sa slojevima za perzistenciju i komunikaciju za 75%. Okvir je pokazao visoku

fleksibilnost, omogućavajući integraciju novog čvora sistema u roku od jednog radnog dana. Sistem je uspešno automatizovao ključne logističke procese i omogućio sinhronizaciju podataka u realnom vremenu između distribuiranih aktera putem standardizovanih REST API-ja.

Zaključak: Nalazi potvrđuju da je BAB okvir efikasna platforma za ubrzani razvoj skalabilnih, interoperabilnih i fleksibilnih kooperativnih procesno-svesnih informacionih sistema. Istraživanje potvrđuje početnu hipotezu, pokazujući da ovaj pristup pruža izrazitu prednost za agilna poslovna okruženja, posebno za mala i srednja preduzeća, drastičnim smanjenjem napora u razvoju, poboljšanjem kvaliteta softvera i omogućavanjem brzog prilagođavanja promenljivim poslovnim zahtevima.

Ključne reči: poslovni informacioni sistemi, procesno-svesni informacioni sistemi, kooperativni informacioni sistemi, mikroservisna arhitektura.

Paper received on: 27 June 2025.

Manuscript corrections submitted on: 8 September 2025.

Paper accepted for publishing on: 3 October 2025.

© 2026 The Authors. Published by Vojnotehnički glasnik / Military Technical Courier (www.vtg.mod.gov.rs). This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/4.0/rs/>)

