Leveraging Artificial Intelligence–enabled Workflow Framework for Legacy Transformation

Abdullah Al-Barakati
Faculty of Computing and Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia

***Abstract*—The rapid advancement of web technologies coupled with evolving business needs make legacy transformation a necessity for enterprises around the world. However, the risks in such a transformation must be mitigated with an approach that is flexible enough to allow for a gradual and low risk transformation process. This paper presents a Service Oriented Architecture (SOA) workflow-based legacy transformation approach that allows for phased transformation in which a legacy system is first transformed into self-contained modular services accessible via a dedicated service layer. These modular services are managed through an AI-enabled workflow management layer that interacts with improved UI frontend for the system's end users. This paper presents a hypothetical prototype in which an Oracle 5 legacy system is transformed using the proposed architecture. ASP .NET Core MVC as well as Pega business process management platform are utilized to practically assess the feasibility of the proposed approach.**

***Keywords*—Legacy systems; service oriented architecture; workflow management; legacy transformation; digital transformation; artificial intelligence (AI)**

I. Introduction

Legacy systems can be defined as applications that were built with old technology but are still in use in many business environments [1]. Despite the apparent advantages of transformation from legacy systems to modern web-based solutions, government and private enterprises consider the process risky and challenging and, therefore, show reluctance in initiating the change. Such reluctance can be attributed in part to the heavy investments that were associated with developing legacy applications [2]. Furthermore, enterprises incur heavy costs to train their employees and tailor their processes to benefit from legacy systems [3]. On the other hand, the legacy transformation and modernization process itself can be lengthy and costly. Major risks include the inherent complex designs of legacy systems [4], tightly coupled components, system performance issues, and difficulties in mapping current systems to target architectures and platforms [5]. Additionally, the underlying knowledge about such systems is usually scanty due to limited documentation and the unavailability of the developers who originally built these systems [4]. As a result, most legacy modernization tenures tend to begin with lengthy reverse engineering periods to document current systems before paving the way for technology transformation.

It can be argued that legacy transformation is inevitable with the rapid advancements that technology is witnessing, especially the digital transformation phenomenon. Digital transformation places special emphasis on legacy transformation as one of the cornerstones of successful transformation strategies [6]. While legacy transformation and modernization processes can be lengthy and costly as mentioned earlier, they can offer long term cost savings, increased efficiency, better resource utilization and the ability to adapt to the dynamic business needs of any given enterprise [7]. Therefore, enterprises need an optimal transformation approach that will enable them to part with legacy systems and take advantage of the possibilities offered by modern web-based technologies [8].

In this paper, we propose a legacy modernization approach that aims for gradual technology upgrade from legacy systems to modern web-based solutions without disturbing business operations. It is based on a Service Oriented (SO) architectural approach that wraps exciting legacy applications with an AI-enabled workflow management layer. The workflow management layer acts as a service orchestrater that reduces the risks of inadequate service mapping when migrating from legacy systems to target modernized systems. Workflow management functionality is achieved via Business Process Management Solutions (BPMS) that can sit on top of the legacy system services. This approach emphasizes service orientation where business logic is captured and managed in a dedicated middle service layer that can potentially integrate with any future core systems that may replace current legacy systems. While this approach can be technology agnostic, we are showcasing a hypothetical case study where Pega BPMS is utilized to manage the workflows of an Oracle 5 form-based legacy system while having ASP.Net Core MVC as the main technology for a dedicated service layer.

The reminder of this paper is organized as follows. Section II sheds light on some of the research in legacy system transformation approaches. Section III outlines the overall architectural approach, its layers, and the integration points with legacy systems. Section IV introduces the suggested technology stack related to the proposed architecture. Section V showcases a practical implementation of the proposed approach as a hypothetical proof of concept. Section VI presents the findings of this paper and highlights possible areas of future work.

II. Legacy Transformation Approaches

Due to the importance of legacy system transformation, several studies have focused on finding the best way for a safe and fast transformation process. In this context, the work produced by [9] examines several options for legacy system

transformation in which replacement is considered the best option for old systems which are undocumented, outdated, or not extensible. However, authors in [9] note that the replacement of such systems is often a resource-intensive and risky process. On the other hand, [6] present a lightweight agile approach for effective low-risk legacy transformation as opposed to waterfall-based transformation approaches. Such an approach can potentially address the technical and procedural complexities associated with legacy transformation.

In the work presented by [10], the authors showcased the process of transforming a legacy social services information system to a modern digital platform. This platform capitalized on advanced technologies (Artificial Intelligence [AI] and Machine Learning) for analyzing and processing big data. From an architectural perspective, this transformation was enabled via the utilization of cloud computing, big data innovations, and the emerging microservices architectural principles. In a similar manner, [8] proposed a tiered architectural approach for legacy system transformation. In this approach, component configuration is specified in Extensible Markup Language (XML) files to facilitate legacy service wrapping and integration. The work presented by [11] is in conformity with the approach presented by [8] as the author argues that legacy systems can be transformed by exploiting modern, faster, and cheaper technologies such as Java and XML. He also indicates that such an approach can shift focus to functionality not technology, hence allowing for better response to the evolving business requirements of any given enterprise. Furthermore, [11] presented a legacy transformation software tool (RescueWare) that acts to decompose business knowledge into self-contained e-components tasked with performing certain business functionalities. These components are defined within standard Application Programming Interfaces (APIs) which can be accessed by other systems and interfaces that can potentially replace the legacy system in question.

The work done by [1] emphasized a component-based approach for legacy transformation. Their methodology includes a reengineering process to transform legacy systems into new components with upgraded software architecture design. They adopted a reverse engineering approach that is based on the extraction of architectural information from the existing codebases of legacy systems. Based on the extracted information, in conjunction with business domain knowledge, modular system components were to be developed to replace the existing legacy system. Similarly, [12] present an interactive tool to extract database and business logic components from legacy systems. The aim is to minimize the complexity of the migration process by introducing a decomposition step. This step can help slice the legacy system into encapsulated components that can be migrated into a client-server platform.

The work in [13] encompasses a legacy migration approach based on the conversion of legacy system architecture into Service Oriented Architecture (SOA) within a systematic predefined process. The process that they suggest is feature oriented as it focuses on a reengineering approach to transfer existing legacy services into web services facilitated by the Web Services Composition Language (WSCL). The author in [13] validated their approach with a case study which presented a prototype based on a layered architecture comprising three main layers: Interaction Layer, Translation Layer, and Repository Layer.

Much of the previous research focused on the concepts of modularity and service orientation for successful transformation. However, to our knowledge, integration of workflow management layers as a part of the legacy transformation process is an area that is not yet fully examined. For this reason, we propose an AI-enabled workflow-based approach for legacy system transformation.

III. Proposed Architecture

To counter the risks and complexities that accompany legacy transformation tenures, a layered SOA-based architectural approach is proposed. SOA can play the role of a transformation and integration enabler for legacy systems [7]. Exposing legacy services in a service-oriented manner will provide for modular services that can be exploited by a variety of interfaces. Such interfaces can be frontend systems or other core systems that benefit from the services of the legacy system. Furthermore, Service Orientation - as a concept and a transformation enabler - will allow for greater interoperability for legacy services. More importantly, business logic transformation to a service layer will lead to decoupling the services of the core system to further facilitate legacy replacement/enhancement.

In addition to the advantage of transformation to service-oriented components, this model is complemented with a dedicated workflow management layer which is tasked with orchestrating the different business services of the legacy system in transformation. Such a model aims to facilitate the process of managing the usually complex services and workflows of a typical legacy system. Additionally, by having a dedicated workflow management layer, it will be possible to gradually move the legacy services from the core system to a service layer. In such a scenario, the workflow management layer will orchestrate uninterrupted business operations by managing the right mix of legacy services, external integration on the one hand and user interactions on the other.

Two main architectural principles will govern and shape the proposed legacy modernization architecture. Firstly, a microservices approach will allow for rapid delivery of the system's business services [14]. Microservices will also be an enabler for a robust technology stack that can be enhanced or modified when and if needed. Enhancement can be achieved by plugging in more service layers to cover any evolving business needs. In conjunction with the utilization of microservices architecture, the proposed solution focuses on the development of domain-specific services. Hence, the legacy application's services will be divided into self-contained modules based on business areas. This approach adheres to the principles of Domain Driven Design (DDD). Thus, business context will be divided into individual areas that can be developed and managed separately. Greater modularity and manageability can be considered as important advantages of this approach. Based on the proposed approach, the following architectural layers will be required as illustrated in Fig. 1.

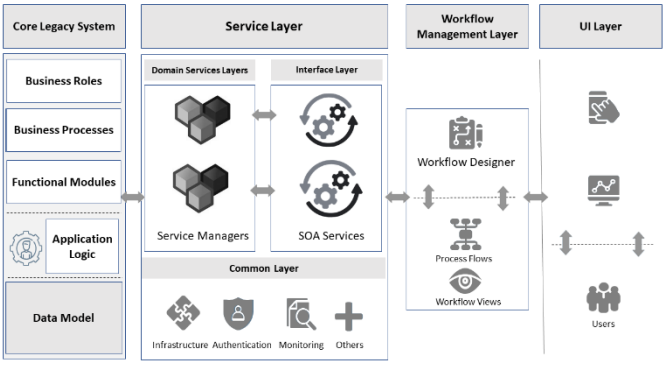

Fig. 1. Proposed Architecture.

A. Core Legacy System Layer

In the initial phases of transformation, the core legacy system will be considered as a backend system that the other system layers will be interacting with. As is the case with typical legacy systems, this layer will comprise application software and any associated databases.

B. Service Layer (Middle Layer)

To be able to adopt a service-oriented approach, most business logic should be separated from the core system into a dedicated API-based Service Layer (middle layer). This layer will contain the business logic such as all validations and custom roles related to the core legacy system. The service layer will be divided into the following sub-layers:

1) Interface Layer (API): This layer will be dedicated to exposing the services of the core legacy system into Representational State Transfer (RESTful) APIs that are consumable by the Workflow Management Layer. Furthermore, RESTful APIs will be exposed as web services that can be accessed by either internal or external systems.

2) Domain services layer: Following the principles of Design Driven Design (DDD), the Domain Services Layer will contain a number of business-oriented modules. These modules will correspond to the respective business areas in the core legacy system. Hence, each module will encapsulate several microservices that provide business-specific functions. In line with DDD implementation patterns, each module will have a service manager to manage its microservices. Moreover, the service manager will enable collaboration among the business modules as required. Service calling in this instance is achieved via the communication between the service managers of the different system modules.

3) Common layer: The Common Layer will provide generic services required by the application. For example, user management and authentication, integration with external parties, and data access. This layer will also comprise a dedicated Data Access Layer (DAL) that will provide all the data connectivity and access functionalities. Having a separate DAL is vital in the context of legacy transformation as it will enhance the adopted SOA approach by avoiding native access to databases.

C. Workflow Management Layer

One of the important elements of the proposed architecture is based on wrapping legacy solutions with Business Process Management (BPM) functionality. BPM functionality will be managed via a Workflow Management Layer that will act as an orchestration tool to manage the interactions between the legacy core system and its related end points. Following the proposed layered approach, a workflow management software component will sit on top of a dedicated service middle layer. Hence, the Workflow Management Layer will comprise three main components as follows:

1) Process workflows: The process workflows capture the workflows of the legacy system and manage the system services accordingly. It should be noted that domain driven workflows will not only capture the business-level models of the legacy system, but the approach adopted by [15] will be employed in which IT-level models will also be captured for effective workflow management. Hence, in addition to capturing the workflows of business processes, IT-level models will also be captured to address specific technical requirements such as infrastructure considerations and user access and authentication.

2) Workflow and data views: Workflow models constitute process models (views) that capture the actual sequence of activities/validations that a typical workflow process contains [16]. Workflow views will be used to create and manage the workflows that map the legacy system's functionality. Another layer of workflow modelling within the proposed architectural model is the data models (views). Data views contain the data objects required to define data fields, data field mapping, and connections to database. Hence, they bridge the connection between the backend database systems.

3) Artificial intelligence (AI) Layer: This layer will aim to streamline the system's workflows, reduce redundancies by intelligently handling large amounts of data, reduce user errors and increase the efficiency of routine tasks. It will offer the greater advantages of the legacy transformation process.

D. User Interface Layer

The User Interface (UI) Layer will provide users with the ability to interact with the backend legacy system to achieve the required business functionality. Frontends can be in the form of purpose-built desktop applications communicating with the Workflow Management Layer. They can also be in the form of web-based applications whether it be a website, web portal, tablet, or mobile applications.

IV. Technology Stack

There is a variety of technology solutions that can support a gradual transformation from legacy systems to modern web-based solutions. Based on the proposed workflow-managed SOA approach, the following software technologies can be utilized for a prototype implementation:

A. Pega Platform

Since one of the pillars of the proposed approach is the utilization of a Workflow Management Layer, the use of Pega

as a Business Process Management (BPM) platform is suggested. The low-code nature of Pega coupled with its App Studio that allows for business and IT cooperation in the design stage makes it a powerful tool for transformation and modernization projects [17].

Another reason to choose Pega is its wide range of data and integration capabilities that allows connecting Pega applications with distributed backend systems. The Pega platform also supports a wide spectrum of integration standards and protocols allowing for high connectivity levels with external systems [18]. Additionally, Pega offers a wide range of AI and machine learning tools that allow for optimized workflows and increased efficiency [19]. These capabilities are particularly important in relation to the proposed architecture which emphasizes communication with legacy systems via dedicated integration layers.

B. ASP.Net Core MVC

ASP.Net Core MVC is a modular and cross-platform development framework for developing web-based applications [20]. It provides a concrete framework for developing RESTful web services that can expose data operations [21]. This development framework was selected for the prototype implementation due to its ability to expose backend services as RESTful web services that can be consumed by other software layers (namely, the Workflow Management Layer in our proposed architecture).

Developing a middle layer using ASP.Net Core MVC can provide the required flexibility in terms of transforming a legacy system to a web-based system. Within this context, the main advantage of ASP.Net Core MVC is its ability to provide headless web services [21]. Headless API services do not have User Interface (UI) as they are meant to be consumed by other systems that may have their own UI elements. This approach provides the necessary flexibility to expose system services via different interfaces such as websites, web portals, and mobile applications.

C. Devart

Devart is a database connectivity tool that supports a variety of database platforms. To avoid direct (native) access to the legacy database, Devart can be a good tool for building Data Access Layers (DAL) that can provide the necessary interfaces to the service layer to access legacy databases. Furthermore, Devart's developer tools support reverse and forward engineering which makes it a suitable tool for legacy modernization implementation [22].

V. Solution Implementation

To identify the exact components of the proposed architectural approach, a hypothetical proof of concept is presented in this paper. We examine the effectiveness of the proposed approach through a prototype based on one of the common legacy transformation scenarios. This scenario is represented in the transforming of an Oracle forms-based system (Oracle 5) to a web-based application. In this proof of concept, the scenario of exposing a legacy HR system to the web is highlighted through the implementation of the proposed transformation approach.

A. Transformation Steps

A piece-by-piece transformation process is followed as opposed to a risky big bang approach where all system components are migrated at once to a new system/platform. Based on this gradual approach, two main transformation stages can be envisaged:

1) Phase I: Transformation to Service Orientation (SO): The main goal of this phase is to transform legacy services into modular services that can be accessed from a service-oriented middle layer (service layer). To achieve this goal, the legacy system will be analyzed and documented on an as-is basis. Then, business logic will be captured in the service layer that will directly interact with the legacy system and its database (acting as a backend system in this instance).

2) Phase II: Legacy system replacement: Since Phase I will separate business logic from the legacy core system, it will be relatively a lower risk process to replace the backend legacy system with a new system that will interact with the existing service layer. In such a scenario, business operations and end-user experience will not be interrupted as they will still be interacting with the same frontend systems. Such frontend systems can be either a workflow management interface as manifest by the proposed architecture, customized desktop applications or web-based applications.

B. Case Study

The prototype system includes four layers (Core Legacy System, Service Layer, Workflow Management Layer and UI Layer). In this context, it is assumed that the Legacy System's services were mapped into several clearly defined APIs that can loosely integrate with other systems. As illustrated in Fig. 2, the API services act as an entry point and perform the required services using the business modules' service managers. Furthermore, the API service layer will use the Common Layer for generic functions such as the management of user authentication, getting database context, etc. Additionally, API services will share the database context with all business services allowing the system activities to be handled in a few database transactions.

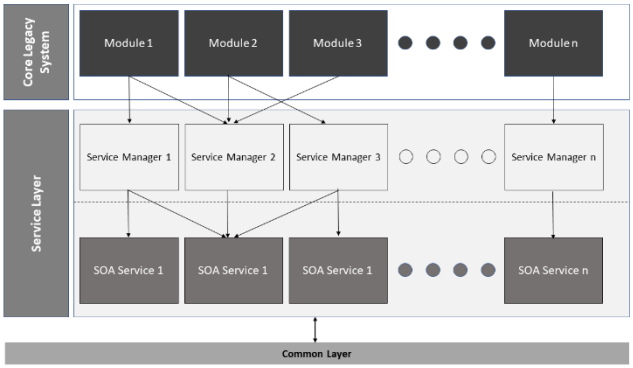

Fig. 2. Modules and Service Managers' Interactions.

C. System Services

Table I summarizes the main API Layer services that require the basic HR system functionality. If we take the example of a simple service to enter the details of a new employee, several RESTful API services can be used to populate the dropdown menus used in the data entry form within the system's UI. These services will be called via the designated service manager to pull the required data for use in the UI. Furthermore, business validation can be performed using services from several modules based on the action being performed by the end user.

D. Workflow Management

The legacy system functionality is decomposed into service-oriented workflows managed by the Workflow Management Layer. In our prototype, the Workflow Management Layer is represented in Pega BPMS, which is a low code workflow management platform that has the flexibility to integrate with a variety of backend systems. Fig. 3 illustrates the Pega-designed workflow for the employee addition process within the prototype HR system. This workflow contains three main steps: identification of employee details, employee addition, and closure.

TABLE I. API LAYER SERVICES

System Functions	**Service APIs (CommonDBContext = InfraSrctureManage.GetDatabaseContext)**
Search Function	API. Search Employee In this service, the user inputs the search attributes and then calls the following service to fetch the required results: HRSrvcMngr.GetEmployeeList
Add New Employee	API.NewEmployee: A data entry form allows the user to enter the attributes related to a new employee. The following services are used to fill the dropdown lists associated with employee attributes. For example: RefSrvcMngr.GetEmployeeTypes(CommonDBContext) RefSrvcMngr.GetContractTypes(CommonDBContext) RefSrvcMngr.GetContractTypes(CommonDBContext) RefSrvcMngr.GetDepartmentsList(CommonDBContext)
Update Employee Details	There are two steps in this process: 1. Show employee details, attributes are initiated by fetching the current employee data by using the service: HRSrvcMngr.GetEmployeeByID 2. Users can update the employee's details and then either save the record or cancel the process.
Delete Employee Details	API. DeleteEmployee The API.SearchEmployee can be used to fetch the employee details. Once an employee record is selected, it can be deleted by using the following service: SgnSrvcMngr.DeleteEmployee(CommonDBContext)
Save Details	API. SaveEmployee Saving an employee's details can be done through either of two processes: 1. Adding a new employee: HRSrvcMngr.CreateEmployee(CommonDBContext) 2. Updating employee details: HRSrvcMngr.UpdateEmployee(CommonDBContext)
Cancel Operation	This functionality will be achieved via the UI level by clearing the data entry form

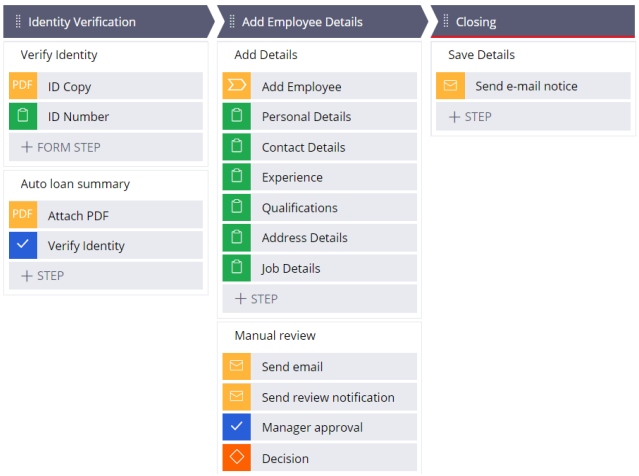

Fig. 3. Proposed Architecture.

The first step in the workflow involves the verification of the employee's identity before adding his details to the system. To verify the identity, it is assumed that Pega will integrate with a third-party provider to validate the employee's ID. In our prototype implementation, Pega capabilities are utilized to create the system's frontend in the form of a series of HyperText Markup Language (HTML) pages that correspond to the designed workflow.

Once the employee's identity is validated, the "Add Employee Details" step is invoked which will involve displaying the employee addition data entry form shown in Fig. 4. The different drop downs in the entry form will be populated with dynamic values pulled from the legacy system database. For example, the following services will be called to fetch Employee Types, Contract Types and List of Departments:

RefSrvcMngr.GetEmployeeTypes(CommonDBContext)

RefSrvcMngr.GetContractTypes(CommonDBContext)

RefSrvcMngr.GetDepartmentsList(CommonDBContext)

Once the employee details are added, they can be saved by invoking another service from the Service Layer:

HRSrvcMngr.CreateEmployee(CommonDBContext)

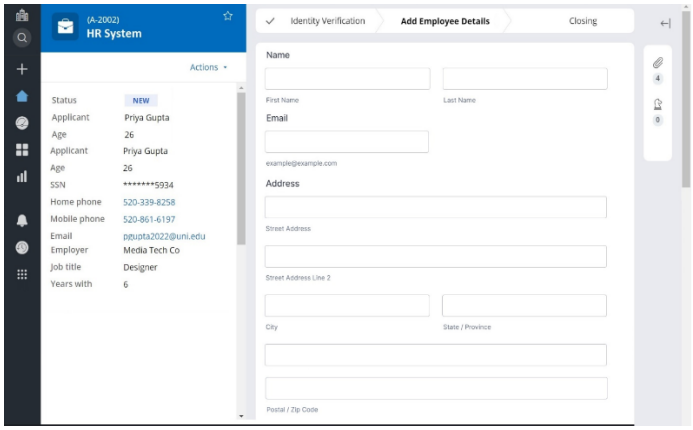

Fig. 4. Employee Addition Form.

E. Integration

Pega integration capabilities with RESTful APIs were utilized to integrate with the services provided by the Service Layer. In this scenario, Pega acted as a client application that uses HTTP protocols to access GET or POST methods to achieve the required functionality. An example of RESTful service consumption is the process by which the list of contract types is fetched to populate the relevant dropdown list in the employee addition form. HTTP GET requests are passed through service HTTP query strings that contain the required operations. In this example, GetContractTypes service is used:

https://www.legacyhr.com/GetContractTypes.php?operation=fetchtypes

The fetched data is formatted into JavaScript Object Notation (JSON) string that can be easily used in the system's frontend as illustrated in Fig. 5. Similarly, when there is a need to write data to the legacy system's database, HTTP POST operations can be used to pass the required data (for example, new employee's details) to the core system.

```
{
  "status"   : "SUCCESS",
  "count"    : "3",
  "contract_types"  : [
            {
              "id" : "1",
              "name" : "Fulltime",
              "category" : "contracts",
              "description" : "Contract for permanent staff"
            }, {
              "id" : "2",
              "name" : "Parttime",
              "category" : "contracts",
              "description" : "Contract for temporary staff"
            }, {
              "id" : "3",
              "name" : "Projects",
              "category" : "contracts",
              "description" : "temporary contracts for projects "
            }
             ]
}
```

Fig. 5. Contract Types JSON Sample.

VI. Conclusion and Future Work

A. Conclusion

The rapid advancement of web technologies coupled with evolving business needs make legacy transformation inevitable for enterprises around the world. However, the risks of such a transformation should be mitigated with an approach that is flexible enough to allow for a gradual and low risk transformation.

The proposed SOA workflow-based transformation approach offers several benefits in terms of legacy system transformation into web-based applications. The key advantage here is the adoption of a microservices architecture where the legacy system's functionality is decomposed into self-contained functional units. On top of that, an AI-enabled Workflow Management Layer orchestrates the system's functionality by calling the required legacy services from a dedicated Service Layer (middle layer). In our prototype implementation, we utilized ASP.Net Core MVC for the Service Layer implementation and Pega BPMS for the Workflow Management Layer.

B. Future Work

Future work will involve progressing further with the transformation approach by examining the process of replacing the legacy backend system with a new core system. The aim here will be to validate the success of full transformation by utilizing the suggested architectural approach and transformation steps.

References

[1] H. Kim and Y.-K. Chung, "Transforming a Legacy System into Components," In: Gavrilova M. et al. (eds) Computational Science and Its Applications - ICCSA 2006. ICCSA 2006. Lecture Notes in Computer Science, vol. 3982, pp. 198-205, 2006.

[2] H. M. Sneed, "Planning the reengineering of legacy systems," IEEE Software, vol. 12, no. 1, pp. 24-34, 1995.

[3] A. D. Ionita, M. Litoiu and G. Lewis, Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments, 1 ed., IGI Global, 2012.

[4] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen and J. Hage, "How do professionals perceive legacy systems and software modernization?," in ICSE 2014: Proceedings of the 36th International Conference on Software Engineering, Hyderabad, 2014.

[5] P. Gordon, R. Seacord, D. Plakosh, G. Lewis and J. Fuller, Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices, 1 ed., Addison-Wesley Professional, 2003.

[6] D. Goerziga and T. Bauernhansla, "Enterprise Architectures for the Digital Transformation in Small and Medium-sized Enterprises," in Procedia CIRP, Naples, 2018.

[7] H. M. Hess, "Aligning technology and business: Applying patterns for legacy transformation," IBM Systems Journal, vol. 44, no. 1, pp. 25-45, 2005.

[8] Y. Zou and K. Kontogiannis, "Migrating and Specifying Services for Web Integration," Lecture Notes in Computer Science, vol. 1999, pp. 253-270, 1999.

[9] S. Comella-Dorda, K. Wallnau, R. C. Seacord and J. Robert, "A Survey of Legacy System Modernization Approaches," Defense Technical Information Center, 2000.

[10] S. B. Popov and P. V. Khripunov, "Digital Transformation Legacy Social Service Information System," in Journal of Physics: Conference Series, Britsol, 2019.

[11] L. Erlikh, "Leveraging legacy system dollars for e-business," IT Professional, vol. 2, no. 3, pp. 17-23, 2000.

[12] G. Canfora, A. Cimitile, A. De Lucia and D. L. Giuseppe, "Decomposing legacy programs: a first step towards migrating to client-server platforms," The Journal of Systems and Software, vol. 54, no. 2000, pp. 99-110, 2000.

[13] S.-H. Li, S.-M. Huang and D. C. C. C.-C. Yen, "Migrating legacy Information," Journal of Database Management, vol. 18, no. 4, pp. 1-25, 2007.

[14] P. Krivic, P. Skocir, M. Kusek and G. Jezic, "Microservices as Agents in IoT Systems," in 11th KES International Conference, KES-AMSTA 2017, Algarve, 2017.

[15] M. C. Branco, J. Troya, K. Czarnecki, J. K¨uster and H. V¨olzer, "Matching Business Process Workflows," in Model Driven Engineering Languages and Systems, Innsbruck, 2012.

[16] W. Yang and F. Li, "Workflow modeling: a structured approach," in 8th International Conference on Computer Supported Cooperative Work in Design, Xiamen, 2004.

[17] Gartner, "Gartner Magic Quadrant for Enterprise Low-Code Application Platforms," 2019. [Online]. Available: https://www.gartner.com/en/documents/3991199/magic-quadrant-for-enterprise-low-code-application-platf. [Accessed 30 10 2021].

[18] S. Mangu, "Business Process Management: Robotic Process Automation Approach," International Journal of Advanced Research in Engineering and Technology (IJARET), vol. 11, no. 11, pp. 831-840, 2020.

[19] R. Walker, "Artificial Intelligence in Business: Balancing Risk and Reward," Pegasystems, 2018.

[20] J. Ciliberti, ASP.NET Core Recipes: A Problem-Solution Approach, 2 ed., Apress, 2017.

[21] A. Troelsen and P. Japikse, Pro C# 7: With .NET and .NET Core, 8 ed., Apress, 2017.

[22] H. Schwichtenberg, Modern Data Access with Entity Framework Core, 1 ed., Apress, 2018.