Process Pattern-Based Flexible Federated Learning

Marius Take[1], Sascha Alpers[1,2], Christoph Becker[1], Maryline Irma Mengoualeu Majiade[1] and Andreas Oberweis[1,3]

[1]FZI Research Center for Information Technology, Haid-und-Neu-Straße 10-14, Karlsruhe, 76131, Germany
[2]Heilbronn University of Applied Sciences, Max-Planck-Straße 39, Heilbronn, 74081, Germany
[3]Karlsruhe Institute of Technology, Kaiserstraße 12, Karlsruhe, 76131, Germany

Abstract

Federated Learning (FL) is an increasingly adopted approach for training Machine Learning models with geographically distributed clients. Multiple clients can benefit from an enlarged cross-location training dataset without sharing any potentially sensitive data, thereby improving data privacy compliance. Various frameworks exist for implementing the basic FL approach. However, the definition and implementation of entire FL systems is cumbersome. In particular, different requirements must be addressed flexibly, as they can differ from use case to use case. Even within a single FL system, requirements may vary between individual clients, for example, in terms of data pre-processing or the execution of training procedures. This paper presents a novel modular approach that incorporates the use of FL frameworks and the specification language Business Process Model and Notation (BPMN). It adopts the concept of process patterns to facilitate the flexible design and implementation of FL systems. For this purpose, the general FL process is modeled in BPMN. The process patterns provide templates for the flexible extension of the general FL process model, for example, the process pattern "Scheduling" for customization of client-specific local training procedures. To further facilitate the implementation of FL systems, this paper demonstrates how the BPMN process is interwoven with an underlying FL framework instance that encapsulates the actual FL. Finally, this paper shows how this approach is applied to two case studies by modeling the FL processes and deploying and executing them in a workflow engine such as Camunda.

Keywords

Federated Learning, Business Process Modeling, Process Patterns

1. Introduction

Federated Learning (FL) has gained immense popularity in recent years. FL is a Machine Learning (ML) approach where multiple entities (clients) collaboratively train a shared prediction model under the orchestration of a central server, while keeping the training data private [1]. The approach was initially introduced by McMahan et al. [2]. In FL, various steps are iterated so that the server aggregates model updates from clients to produce a new global model (see for example [2, 3, 4]). Depending on the source, the subdivision of these steps and the steps themselves may vary slightly. Firstly, the model to be trained is initialized on the server side. The server then distributes the model to the connected clients (step 1), after which the clients train this model locally with their local data (step 2) and send the locally trained model updates back to the server (step 3). The server in turn aggregates the model updates received (step 4) and the steps 1-4 are repeated until convergence. This decentralized approach is particularly useful in domains like healthcare, where strict data protection regulations and privacy concerns limit the sharing of sensitive data, and for scenarios where data is distributed across various devices, such as mobile phones, or across institutions such as hospitals or banks [5]. Further information about the general FL can be found in [2, 4, 5]. In recent years, several frameworks have been developed to facilitate the implementation of FL, including but not limited to Flower [1], PySyft[1], TensorFlow

PMAI - 4th International Workshop on Process Management in the AI era
take@fzi.de (M. Take); sascha.alpers@hs-heilbronn.de (S. Alpers); christoph.becker@fzi.de (C. Becker); mengoualeumajiade@fzi.de (M. I. Mengoualeu Majiade); oberweis@fzi.de (A. Oberweis)
https://www.fzi.de/team/marius-take/ (M. Take); https://www.hs-heilbronn.de/de/sascha.alpers (S. Alpers); https://www.fzi.de/team/christoph-becker/ (C. Becker); https://www.fzi.de/team/andreas-oberweis/ (A. Oberweis)
0000-0002-3419-8380 (M. Take); 0000-0001-6568-8924 (S. Alpers); 0000-0001-8843-2649 (C. Becker); 0009-0004-7075-9722 (M. I. Mengoualeu Majiade); 0000-0002-5304-8433 (A. Oberweis)

[1]PySyft - OpenMined: https://blog.openmined.org/tag/pysyft/

Federated (TFF)[2], NVFlare [6], FATE[3], and OpenFL [7]. The framework *Flower* is used in this paper.

As discussed in Lo et al. [8], a lot of research has already been carried out in FL on the ML side, but architectural design aspects have so far been neglected. According to Lo et al., a kind of non-coding platform for creating and adapting FL systems is also particularly helpful, so that FL systems can be configured without coding (see pattern 15 in [8]). The approach presented below focuses on the user-friendly, simple, flexible design and implementation of process-based (modular) FL systems and therefore also addresses these gaps.

Connected clients of an FL system can have different requirements, for example regarding the communication with the server or the local training procedures. If the data representation varies from client to client, pre-processing steps may be necessary for some clients. There may also be different requirements for each client in terms of deployment of new models. If a connected client uses the ML model in production, additional steps are usually required in terms of verification before the model is actually used. In test systems, these steps are less important. The client processes of an FL system can therefore vary from client to client and must be implemented accordingly. There are also various customization options on the server side, for example, to protect the FL system against attacks, to centrally evaluate the models or to take other parameters into account during model fusion. The modeling and implementation of this variability should be easily possible without major programming effort. To achieve this, we have developed a new process-based approach. Process-based systems have the advantage that, in addition to clarity, they enable the maintainability and re-usability of activities (modules). A typical specification language is Business Process Model and Notation (BPMN)[4]. The approach presented in the following combines the advantages of BPMN with the principle of FL to make it as easy as possible to design and implement individual processes for different FL systems. For this purpose, the general FL procedure is modeled as a BPMN process and the associated activities are initially implemented. For the individualization of the processes, so-called process patterns are provided for both the server side and the client side. These enhance the respective processes with specific properties to meet individual requirements and enables customizations and configurations with as little additional coding as possible. The approach presented here therefore has overarching advantages, both in terms of the resulting FL system:

- Customizability: FL systems can be flexibly designed and implemented to meet specific requirements.
- Clarity: The process-based representation enables a clear presentation of the workflows, which ensures a common understanding.

and the approach itself:

- Extensibility: The approach can be easily extended by integrating new process patterns.
- Easy to use: By focusing on graphical process modeling and pre-implemented fragments, the amount of code programming is reduced.

The related work and existing frameworks are discussed in Section 2. Section 3 introduces the approach in more detail, followed by the presentation of the process patterns in Section 4. The underlying implementation is presented in Section 5. For evaluation purposes, two exemplary FL processes were modeled and implemented for one ML model each. This is presented in Section 6, before a brief summary is given in Section 7.

2. Related Work

This section discusses existing FL frameworks on the one hand and briefly presents current work focusing on process-based FL and design patterns related to FL procedures on the other hand.

[2]TensorFlow Federated: https://www.tensorflow.org/federated
[3]FATE - An Industrial Grade Federated Learning Framework: https://fate.readthedocs.io/en/latest/
[4]OMG - BPMN: https://www.omg.org/spec/BPMN/2.0/

2.1. Federated Learning Frameworks

There are various frameworks for implementing FL systems. The frameworks are briefly described in the following, as they already provide the basic functionality of the FL procedure used by the approach presented here. Specifically, the framework *Flower* [1] is used for the implementation in Section 5. Flower is, according to the project documentation[5], an open-source FL framework that enables many different configurations of the basic FL procedure. Two separate scripts are needed to execute a typical Flower workflow: A server script that defines a customizable aggregation logic and a client script that defines how local training is executed on private datasets. Clients connect to the server over a network to receive the global model parameters and return updated parameters after performing local training steps. Flower enables the adaptation of existing ML training procedures into an FL setup [1]. Flower thereby focuses on usability, scalability, and provides an environment that accommodates a broad range of ML frameworks[6]. This paper builds on the FL functionality of Flower so that the advantages of Flower are combined with the advantages of BPMN to provide an approach for creating flexible, modular FL systems. Along with the pure FL, corresponding FL systems also address preparatory and follow-up steps, as well as additional functionalities that expand the base FL algorithm without belonging to it.

In addition to Flower, there are several other frameworks. For example, *OpenMined PySyft*[7], which is a Python library that extends Tensorflow and PyTorch, or *TensorFlow Federated (TFF)*[8] also facilitate ML on decentralized datasets. *NVIDIA Federated Learning Application Runtime Environment (FLARE)*[9] is a software development kit designed to help researchers and data scientists to transform their existing ML and Deep Learning workflows to an FL procedure [6]. *Federated AI Technology Enabler (FATE)*[10] and *Open Federated Learning (OpenFL)*[11] are two other open source FL frameworks. FATE is, according to the project documentation[10], the "first industrial grade FL open source framework to enable enterprises and institutions to collaborate on data while protecting data security and privacy", and OpenFL, for example, provides support for hardware-based trusted execution environments (TEEs) while also offering software-based approaches[11].

2.2. Process based Federated Learning

In addition to concrete frameworks for the implementation of FL systems, there are already a few scientific papers on process- and pattern-based FL. For example, Lo et al. [8] presented a collection of architectural patterns for the design of FL systems, which address recurring design challenges. Their work stems from a systematic review of the literature and identifies fifteen different patterns. The patterns are arranged along an FL model life cycle and guide software architects in designing suitable FL systems.

Yin et al. [9] consider FL architectural options aimed at predictive business process monitoring, focusing on challenges such as data distribution heterogeneity, alignment of event logs, and appropriate prediction model aggregation. This paper addresses the combination of process mining with FL and argues that FL can preserve data privacy while enhancing model performance through cross-organization collaboration and model aggregation.

Verlande et al. [10] present an approach in which an FL system is directly integrated into a BPMN workflow for HR recruiting scenarios. They demonstrate how FL can be incorporated into an organization's recruiting process to provide models for the detection of potential malware while complying with GDPR requirements. By modeling the recruiting process and security checks through the provided model in BPMN, they show where and how FL computations occur in combination with the original process. This work demonstrates how FL can be embedded into a real-world business process.

[5]Flower: https://github.com/adap/flower?tab=readme-ov-file
[6]Flower: https://flower.ai/
[7]PySyft - Open mined docs: https://docs.openmined.org/en/latest/index.html
[8]TensorFlow Federated: https://www.tensorflow.org/federated
[9]NVIDIA FLARE: https://nvflare.readthedocs.io/en/main/index.html
[10]FATE: https://github.com/FederatedAI/FATE
[11]OpenFL: https://openfl.io/

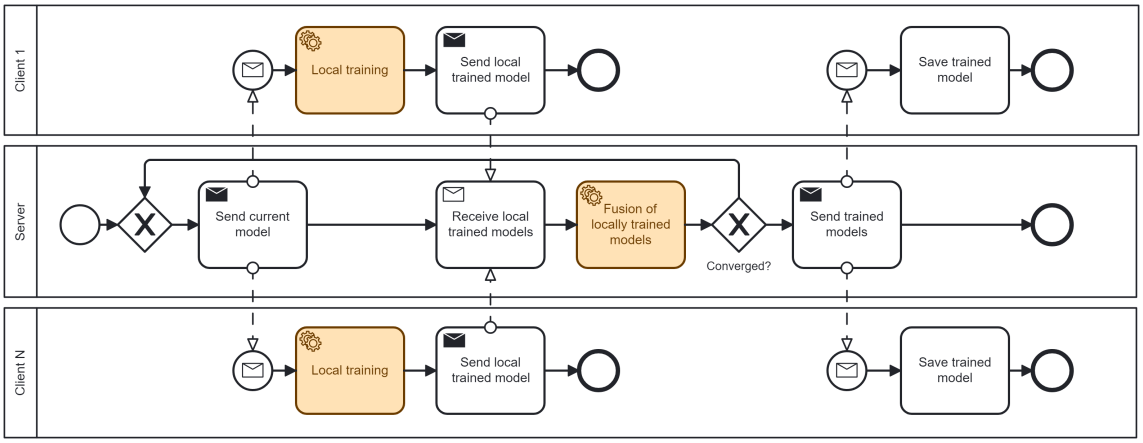

Figure 1: General FL process modeled in BPMN.

Several other papers also demonstrate the use of FL in various practical contexts (see [11, 12]). However, the reviewed papers do not focus on modeling the FL workflow as a BPMN process which could be flexibly expanded at certain breakpoints. Despite extensive research, we have not found any work specifically addressing the design and implementation of a BPMN and process pattern-based flexible FL approach.

3. Basic Concept

In order to be able to adapt FL processes flexibly to different requirements of different use cases, we consider process patterns. This approach is related to the approach of Lo et al. [8], in which architectural patterns for the different points in the FL model life cycle were identified through a literature review. Process patterns are a special form of software design patterns and are characterized by us as process snippets, small process models modeled in BPMN (for a detailed description, see Section 4). These process patterns can be integrated into existing process models, can form the basis for new process models and can be combined. All process patterns presented in this paper can be used (individually or in combination) to extend the general FL procedure. The general FL procedure (see Section 1) is therefore also modeled in BPMN, as shown in Figure 1, to enable seamless customization.

As suggested in the publication of Lo et al. [8], the general FL process can be extended with patterns at various points. Therefore, the BPMN model of the general FL process can be extended with placeholders. These placeholders indicate where process patterns can be applied to extend the process. In our concept, this takes place between the central FL steps (see Figure 2 – the placeholders are highlighted in purple, the training and fusion tasks in orange). A total of five different process pattern collections have been developed – (similar to the grouping of the various patterns in Lo et al. [8], but structured differently):

- **Orchestration (O)** Contains process patterns that are executed before the FL process is carried out.
- **Adapter (A)** Contains process patterns that are executed before the respective local training procedure.
- **Security (S)** Contains process patterns that are executed after receiving the models on the server side and before the model fusion.
- **Quality assurance (Q)** Contains process patterns that are executed before distributing the resulting model.
- **Deployment (D)** Contains process patterns that are executed before commissioning at the end of the entire FL process.

This set of process pattern collections can be adjusted in the future, for example, by further subdivision or supplementation, such as with an additional placeholder for process patterns after local training of models. Initially, each collection contains three to four different process patterns. These process

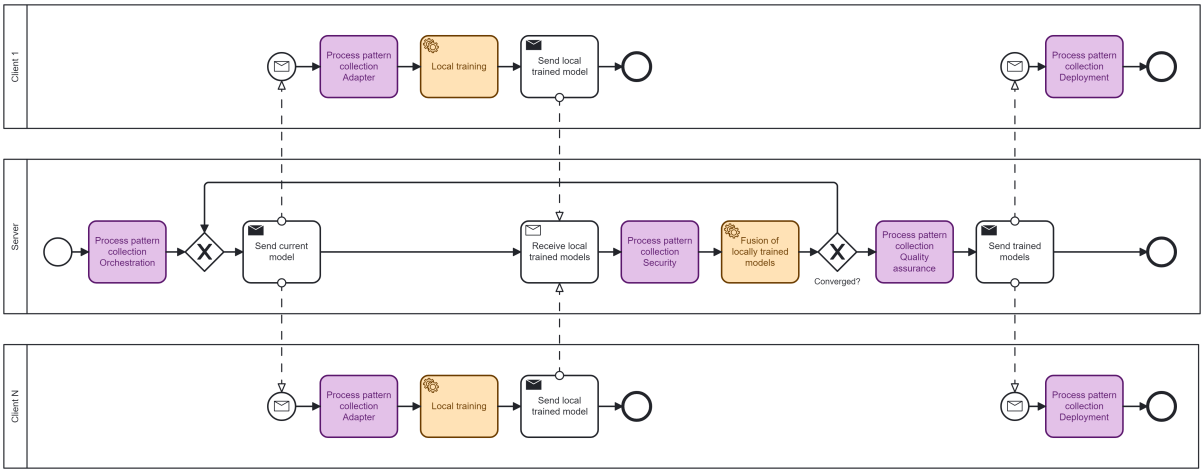

Figure 2: General FL process modeled in BPMN, with placeholders for the process patterns for application-specific customization.

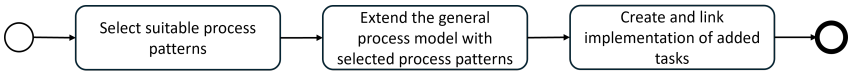

Figure 3: The steps of the flexible approach developed to implement application-specific FL systems.

patterns are briefly presented in Section 4. The general FL process model can be adapted to a use case by selecting and inserting the process patterns relevant to the use case at the positions of the corresponding placeholders.

The advantage of modeling FL processes in BPMN and customizing them using process patterns is that it provides a clear visualization of the adaptation and the resulting system. In addition to visualization, BPMN processes can also be executed using workflow engines. Modeled and individualized FL processes can therefore be used directly to build the FL system, simply by deploying the processes with their corresponding tasks. The approach presented in this paper supports implementation by offering pre-implemented tasks providing the basic FL functionality. The basic FL functionality is available through existing frameworks. The automated execution provided by the frameworks is divided into individual parts so that the framework execution stops before each process pattern placeholder, waits for all process pattern tasks to be executed and then continues the FL framework sequence. This implementation concept is presented in Section 5. The modularization based on process patterns enabled by this implementation concept offers further advantages in terms of implementation, such as the ability to reuse modules that have already been implemented and the creation of more maintainable code. To summarize, our approach to creating application-specific FL systems comprises the three steps shown in Figure 3.

4. Process Patterns

As presented in the previous Section 3, process patterns are intended to provide templates or ideas for the individualization of the general FL process. The concept of process patterns is not new – a number of process model patterns have already been presented in the literature, which have been collected in an online catalog by Fellmann et al. [13]. Fellmann et al. [13] define business process model patterns as "a description of a proven solution to a recurring problem that is related to the creation or modification of business process models in a specific context". This is supplemented by the requirement that the description must be standardized and structured. The process patterns presented in this paper visualize the solution as small BPMN process models that ensure specific properties. They can therefore also

be regarded as a recommendation for the implementation of recurring requirements. For the process pattern collections (Orchestration (O), Adapter (A), Security (S), Quality assurance (Q), Deployment (D)), the following process patterns have been developed:

- **O1 – Server-side startup**: Additional communication between clients and server to enable server-side orchestration.
- **O2 – Query additional parameters**: Collection of additional parameters of the clients that are used within the FL procedure.
- **O3 – FL process management**: Support in handling different FL configurations.
- **A1 – Data preparation**: Adaptation of the training data to the model architecture.
- **A2 – Data sovereignty**: Ensuring that the decision-making authority for training data provision or training participation remains with the respective client.
- **A3 – Scheduling**: Flexible scheduling of the local training procedures (see Figure 4).
- **S1 – Central test dataset**: Checking the received models for required quality criteria with a dataset managed in the server (see Figure 5).
- **S2 – Distributed testing**: The models received are forwarded to other clients for testing. The test results are then collected and aggregated centrally.
- **S3 – Client Authentication**: Upstream authentication of the clients.
- **S4 – Model Backup**: Saving the received models, for example to ensure the traceability of procedures.
- **Q1 – Central evaluation**: Evaluation of the fused models using a central evaluation dataset.
- **Q2 – Dealing with model degradation**: Interruption of the process depending on the evaluation results, for the execution of manual measures (e.g. protection against distribution of model versions with deteriorated quality).
- **Q3 – Backup new model**: Saving the final model, for logging and versioning.
- **Q4 – Forwarding the evaluation results**: Additional communication between clients and server to transmit the evaluation results.
- **D1 – Local evaluation**: Execution of local evaluations with client-internal datasets.
- **D2 – Regulatory review**: Ensuring regulatory review before models are replaced by the newly received models.
- **D3 – Conditional replacement**: Ensure active approval before models are replaced by the newly received models.

The listed process patterns address basic requirements for extensions regarding information exchange, process adjustments, safeguards, as well as relevant preparatory and follow-up steps. However, the process patterns are only examples derived from theoretical considerations and experimental implementations, and serve merely as an initial starting point to illustrate the general approach. The set of process patterns should therefore be continuously modified and supplemented in the future based on findings, particularly those derived from practical implementations. Due to space limitations and the fact that the specific design of the process patterns is not decisive for the presentation of the general approach, only two process patterns are presented in more detail below. Process pattern *Scheduling* from process pattern collection *Adapter* is shown in Figure 4 and is motivated by the work of Hehnle et al. [14]. The idea behind this pattern is that local training procedures should only be started once sufficient (ecological) computing resources are available. If an FL system client wants or needs to make corresponding scheduling, this process pattern can be used for this purpose. However, in order not to hold up the entire process for too long, appropriate application-specific timeout rules should be defined.

The second exemplary process pattern – *Central test dataset* from process pattern collection *Security* – is shown in Figure 5. Once the locally trained model updates have been received, they are checked with a central dataset. If clients maliciously, intentionally send extra bad models to the server, for example, these are filtered out in this way. The other process patterns can be represented analogously by a respective BPMN model.

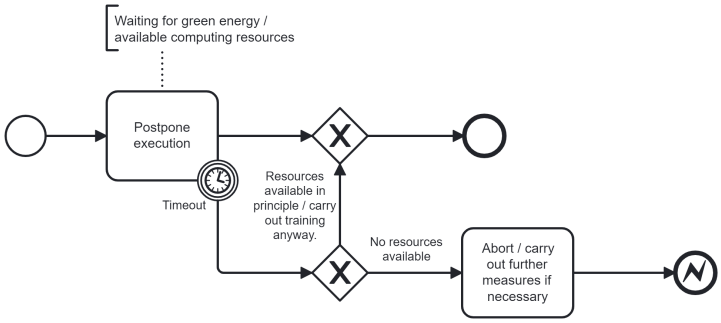

Figure 4: Process pattern *Scheduling* from the process pattern collection *Adapter*. Idea based on [14]. The start event and the unspecified end event define the points of connection to the higher-level process model.

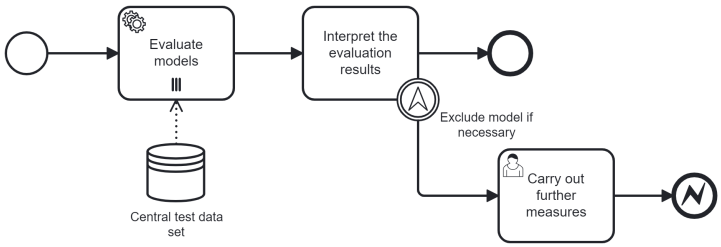

Figure 5: Process pattern *Central test dataset* from the process pattern collection *Security*. The start event and the unspecified end event define the points of connection to the higher-level process model.

5. Implementation

In general, FL offers relevant advantages for many areas of application, so that there are also various frameworks for implementing FL processes, as described in detail in Section 2. In order not to completely re-implement the basic functionality of the FL and to build on already implemented functionalities (such as fusion strategies), we use an already existing FL framework to implement the basic scaffolding (see Figure 1) – in our case framework *Flower*. As already mentioned in Section 3, the framework execution is split into individual parts according to the process pattern placeholders (see Figure 2). As soon as the FL-related tasks preceding the respective placeholder have been completed, FL execution is paused until the corresponding tasks of the selected process patterns inserted into the placeholder have been completed – then the FL execution is resumed. The implementation is visualized in Figure 6 for the client and in Figure 7 for the server. As illustrated on the right hand side of Figures 6 and 7, the FL execution takes place in a separate container and not directly in the workflow engine. This means that the sequence flow is controlled in both the execution of the FL framework and the workflow engine. Consequently, there is a need for synchronization between these two executions. As soon as the FL process is to be started, a corresponding service task (see (1) in Figures 6 and 7) is executed. In the associated java implementation of the service task (2), an interface endpoint is called to start the FL script based on the FL framework (3). The FL Code is then executed in the docker container until the first process pattern (4). During this execution, the workflow engine (the java implementation) regularly queries the interface to check whether the execution of the first part of the FL script has already been completed (5). As soon as this is the case, the execution of the process patterns starts (6). During this time, the FL script calls the interface until the process patterns have been successfully completed (7). This procedure is repeated until the entire FL process (and all added process patterns) have been completed. It should be noted here, as the process excerpts in Figures 6 and 7 already indicate, that the modeling of the general FL process in Figure 2 differs from the process model used here for the actual implementation. Some tasks had to be added around the placeholders or replace the tasks of the basic FL functionality (shown in red in the Figures 6 and 7) in order to realize the communication with the FL container accordingly.

The approach presented here enables basic FL communication to remain via established frameworks,

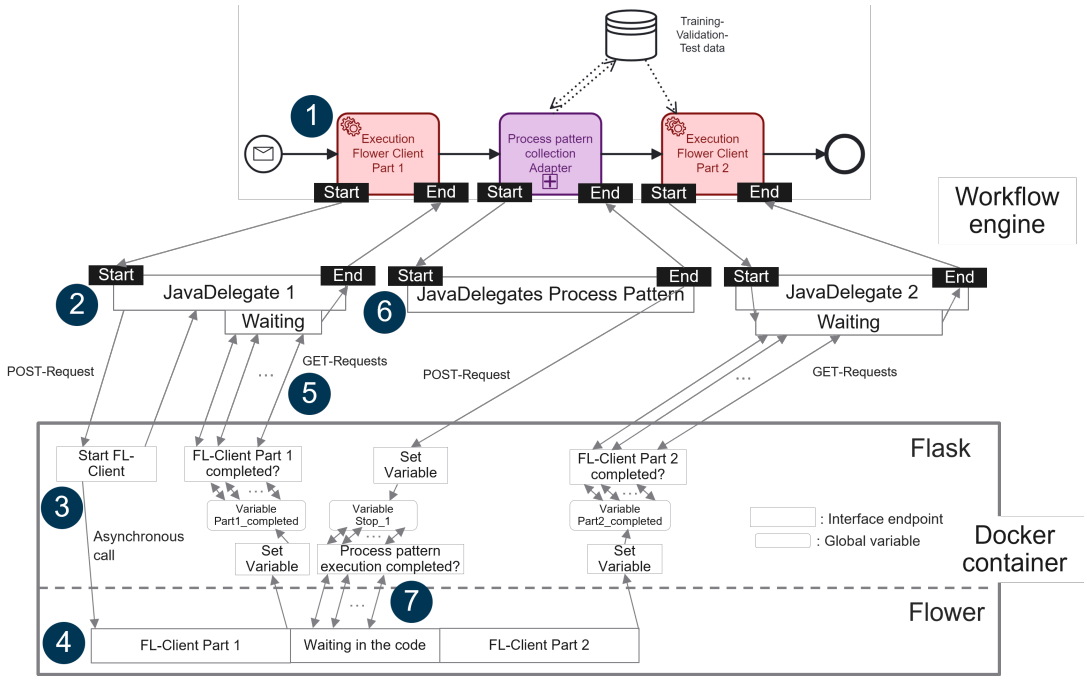

Figure 6: Implementation concept for the client. Starting the FL client (see 3) only takes place at the beginning of the first iteration. This is not necessary in all further iterations of the FL procedure.

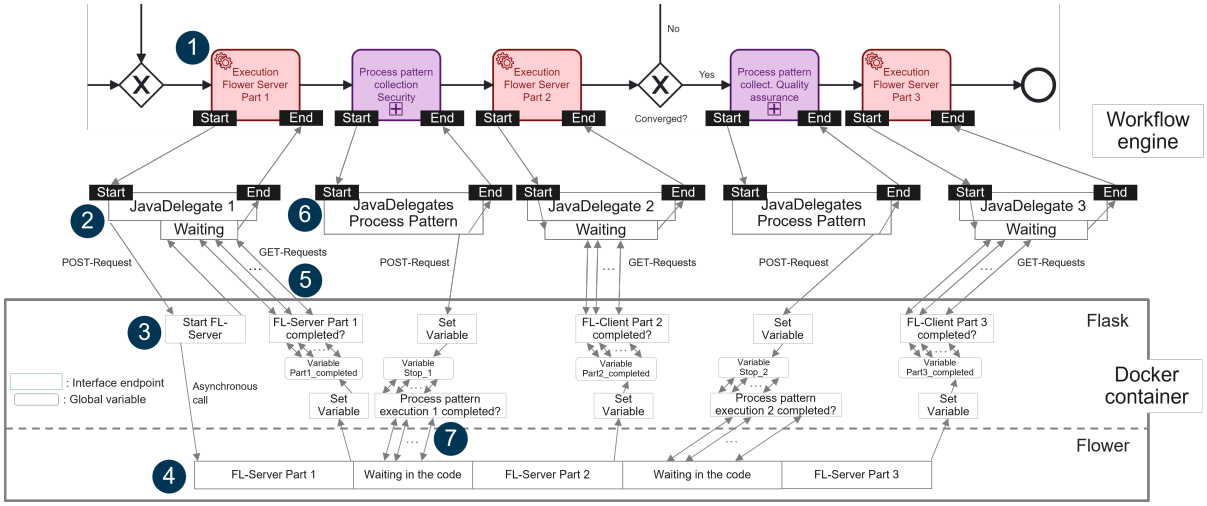

Figure 7: Implementation concept for the server. For the sake of clarity, only the sequence flow for converging models is shown. Flower Server Part 3, without the prior execution of the process patterns for quality assurance, is executed analogously directly after Flower Server Part 2 in the event that the model has not yet converged. Starting the FL server (see 3) only takes place at the beginning of the first iteration. This is not necessary in all further iterations of the FL procedure.

but still allows these FL processes to be easily (model based) expanded in a process modeling tool such as the Camunda Desktop Modeler[12]. Any additional tasks can be executed at the breakpoints and the FL procedure is only continued once these have been fully processed. However, the implementation described is to be considered as a proof of concept and conceptual improvements would be useful in the future. For example, a suitable concept for shared file and data storage between the docker container and workflow engine should be added and the queries of the execution status could be replaced by an event-driven approach in the future.

[12]https://docs.camunda.io/docs/8.7/components/modeler/desktop-modeler/

6. Evaluation

To evaluate the approach, two different uses cases are considered which, in addition to the datasets used, differ in particular with regard to the requirements of the process. The two use cases are briefly presented in the following section. Depending on these use cases, specific FL systems were modeled and implemented using the approach presented in this paper. The results are presented at the end of this section. The evaluation demonstrates the functionality of the approach. Studies on practicability and user-friendliness are planned for future work.

6.1. Use cases

The basic FL implementation offers the same variety of supported ML frameworks and models as the framework Flower, because the Flower execution is not changed but only partially interrupted (see Section 5). In this section, we present the models used for the evaluation. In addition, the requirements that the FL systems should fulfill for each use case are discussed below. In general, the respective client implementation expects a model, data, and corresponding operations, which are provided by four functions:

- *load_model*: Loads the model used for training and evaluation.
- *load_data*: Loads the dataset for training and evaluation.
- *train*: Executes the training of the model on the local training dataset.
- *test*: Evaluates the model on the local test dataset.

6.1.1. HAM10000 Model

The HAM10000 dataset [15] used in the first use case contains images of skin lesions and the label whether the image shows a malignant or benign lesion. AI models trained on this data can be implemented as assistance systems for dermatologists. The following requirements (R) are exemplary requirements that a corresponding FL system should fulfill. To evaluate the approach, these should be addressed accordingly in the resulting FL system. On the one hand, the training procedure should only be performed by the client (e.g. a clinic) if sufficient computing resources are available (R1.1). It is also important to ensure that newly trained models are not automatically used in a productive system (R1.2). Other useful requirements in this use case, such as protection against attacks, are omitted here in order to reduce the complexity of the use case. Server-side requirements are explicitly addressed in the second use case (see Section 6.1.2).

For the model, we defined a Convolutional Neural Network (CNN) to classify the medical image data from the HAM10000 dataset. The network architecture consists of two convolutional layers, each followed by max-pooling layers, and two fully connected layers. For training, a cross-entropy loss function is used alongside the Adam optimizer with a learning rate of 0.001.

6.1.2. MNIST Model

The MNIST dataset [16] contains images of handwritten digits and can therefore be used to train models for digit recognition. To evaluate the approach, we considered requirements for a corresponding FL system, such as first querying the respective amount of training data from the clients (R2.1). For example, the data should then be preprocessed locally by cropping the images to a uniform size (R2.2), after which the local models should be documented before fusion (R2.3). To ensure that only models that meet certain quality criteria are rolled out, a corresponding safeguard is also to be implemented (R2.4). In order to also evaluate the flexibility with regard to different client realizations within an FL system, the client requirement R2.2 should only be addressed for one client.

For the model we defined a simple neural network (NN). The model is trained for multi-class classification, distinguishing between ten digit classes (0-9). The NN consists of an input layer with 784 neurons (corresponding to the flattened 28x28 pixel images), two hidden layers, and an output layer

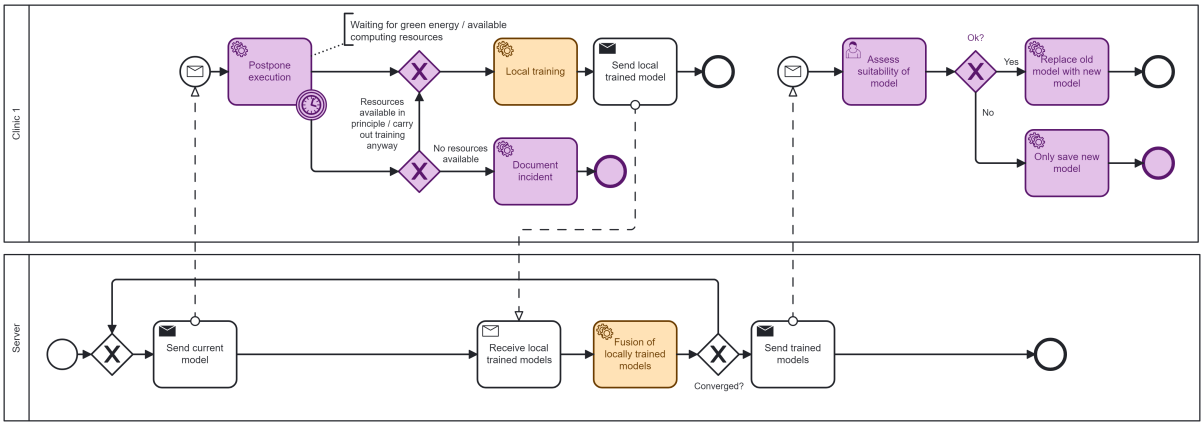

Figure 8: Resulting process model of the FL system for use case *HAM10000*. For the sake of clarity, implementation details were omitted and only one client is shown. The tasks of the added process patterns are shown in purple.

with 10 neurons (representing the output classes). The training is conducted using stochastic gradient descent (SGD) with a learning rate of 0.001 and the cross-entropy loss function.

6.2. Results

Depending on the requirements and models mentioned, specific FL systems were modeled and implemented for both use cases. This was done using the approach presented. The starting point in both use cases was the general FL process model depicted in Figure 2. For the requirements listed in 6.1.1 and 6.1.2, suitable process patterns were selected from the process pattern collections and inserted at the placeholders. For use case *HAM10000* (denoted as "Requirement": "Selected process pattern"):

- R1.1: Scheduling (A)
- R1.2: Conditional replacement (D)

For use case *MNIST*:

- R2.1: Query additional parameters (O)
- R2.2: Data preparation (A)
- R2.3: Model backup (S)
- R2.4: Dealing with model degradation (Q)

The process model of the FL system resulting from the insertions for use case *HAM10000* is shown in Figure 8 and for use case *MNIST* in Figure 9. For the sake of clarity, implementation details (additional service tasks for synchronization – see Section 5) were omitted in both models. The implementation of the basic FL functionality, as well as the required additional communication between workflow engine and Flower, are already given by the approach (see Section 5), so that only the concrete training functionality (for the local training) had to be linked to the existing code and the tasks of the inserted process patterns had to be implemented. The implementation effort is therefore generally not greater than that required for the development of a classic FL system with corresponding functionality and can be further reduced in the future by reusing already implemented process patterns. For this evaluation, the process pattern tasks were implemented in a simplified form either directly in the Java Delegates or by calling additional Python scripts. After carrying out these steps, the modeled processes could be deployed and successfully executed in the Camunda workflow engine (exemplary for two clients and different number of iterations – sufficient to evaluate the functionality of the approach, as the FL framework was not customized). The FL execution waits for the process patterns to be executed and vice versa. The additional communication effort between the workflow engine and Flower creates overhead, which can be reduced by the improvements already mentioned in Section 5. As the evaluation focused on feasibility, a corresponding detailed performance analysis remains to be conducted.

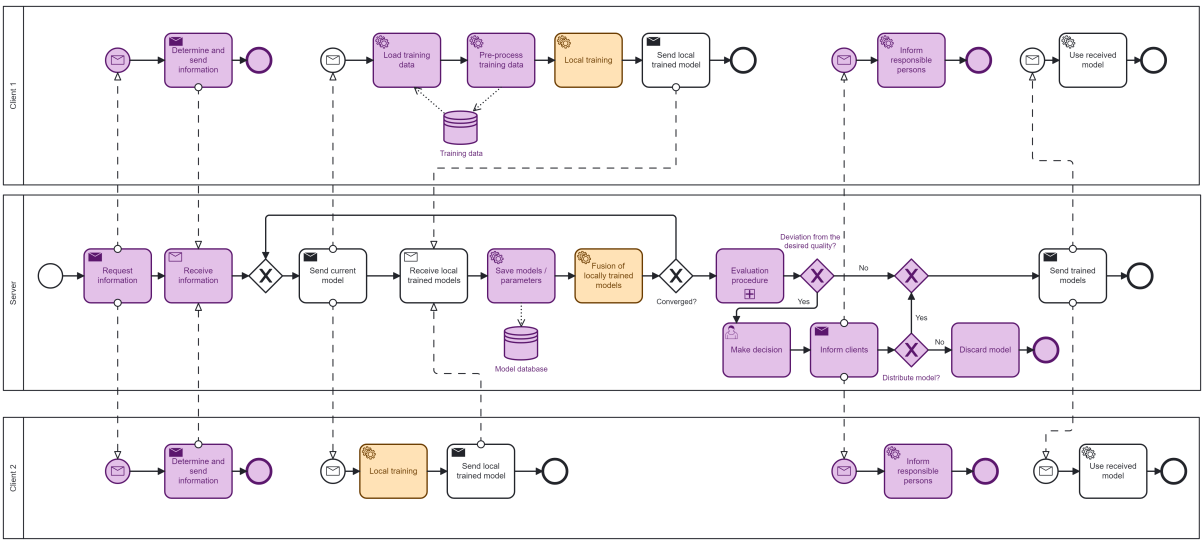

Figure 9: Resulting process model of the FL system for use case *MNIST*. For the sake of clarity, implementation details were omitted. The tasks of the added process patterns are shown in purple.

7. Conclusion

The process based FL approach introduces the concept of flexibility by embedding FL workflows into process modeling frameworks like BPMN. In this work, we extend the functionality of the *Flower* FL framework by flexibly integrating process patterns for both clients and servers into the overall FL procedure, which is also represented as a process model. This integration is enabled by pausing the FL framework execution at predefined breakpoints, allowing the execution of use case specific tasks (as introduced with the process patterns). The evaluation has shown that FL systems and their architectures can thus be easily adapted to specific requirements of different use cases. The approach presented in this paper, particularly through the use of BPMN, enables the customization of FL systems with reduced programming effort. This paves the way for the future development of a corresponding low-code tool.

In future work, the approach presented here can also be expanded alongside the extension to a corresponding tool. So-called subprocess templates, which were introduced in [17] and represent a kind of construction specification for subprocesses, can be used to further specify the resulting process models. This would integrate implementation details directly into the process model. Furthermore, the current approach still needs to be expanded to include additional FL functionality. For example, an adjustment per iteration of the client set is not yet supported. In addition, the implementation of the synchronization of the two sequence flows between workflow engine and FL framework could be improved and a detailed evaluation of the approach in terms of user-friendliness, usefulness, and performance overhead should supplement the functional evaluation that has already been carried out.

Acknowledgments

The work described in this paper is financed by the Ministry for Social Affairs, Health and Integration from state funds approved by the Baden-Württemberg state parliament.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT and DeepL in order to: Text translation, grammar and spelling check, paraphrase and reword. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, N. D. Lane, Flower: A friendly federated learning research framework, 2022. doi:10.48550/arXiv.2007.14390.

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: A. Singh, J. Zhu (Eds.), Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics, 2017, pp. 1273–1282. doi:10.48550/arXiv.1602.05629.

[3] T.-M. H. Hsu, H. Qi, M. Brown, Measuring the effects of non-identical data distribution for federated visual classification, 2019. doi:10.48550/arXiv.1909.06335.

[4] P. Kairouz, et al., Advances and open problems in federated learning, Foundations and Trends in Machine Learning 14 (2021) 1–210. doi:10.1561/2200000083.

[5] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, W. Zhang, A survey on federated learning: challenges and applications, International Journal of Machine Learning and Cybernetics 14 (2023) 513–535. doi:10.1007/s13042-022-01647-y.

[6] H. R. Roth, et al., NVIDIA FLARE: Federated learning from simulation to real-world, IEEE Data Engineering Bulletin 46 (2023) 170–184. doi:10.48550/arXiv.2210.13291.

[7] P. Foley, et al., OpenFL: the open federated learning library, Physics in Medicine and Biology 67 (2022) 214001. doi:10.1088/1361-6560/ac97d9.

[8] S. K. Lo, Q. Lu, L. Zhu, H.-Y. Paik, X. Xu, C. Wang, Architectural patterns for the design of federated learning systems, Journal of Systems and Software 191 (2022) 111357. doi:10.1016/j.jss.2022.111357.

[9] W. Yin, Y. Sun, X. Li, R. Song, Process predictive monitoring based on federated learning: Challenges and architecture proposal, in: Proceedings of the 6th International Conference on Information Management and Management Science, Association for Computing Machinery, 2023, pp. 1–5. doi:10.1145/3625469.3625507.

[10] L. Verlande, U. Lechner, S. Rudel, Design of a federated learning system for it security: Towards secure human resource management, in: Proceedings of the 11th Latin-American Symposium on Dependable Computing, Association for Computing Machinery, 2023, pp. 131–136. doi:10.1145/3569902.3570179.

[11] K. M. J. Rahman, F. Ahmed, N. Akhter, M. Hasan, R. Amin, K. E. Aziz, A. K. M. M. Islam, M. S. H. Mukta, A. K. M. N. Islam, Challenges, applications and design aspects of federated learning: A survey, IEEE Access 9 (2021) 124682–124700. doi:10.1109/ACCESS.2021.3111118.

[12] S. Bharati, M. R. H. Mondal, P. Podder, V. S. Prasath, Federated learning: Applications, challenges and future directions, International Journal of Hybrid Intelligent Systems 18 (2022) 19–35. doi:10.3233/HIS-220006.

[13] M. Fellmann, A. Koschmider, R. Laue, A. Schoknecht, A. Vetter, A taxonomy and catalog of business process model patterns, in: Proceedings of the 22nd European Conference on Pattern Languages of Programs, Association for Computing Machinery, 2017. doi:10.1145/3147704.3147725.

[14] P. Hehnle, L. Weinbrecht, M. Behrendt, The Camunda 8 connector for carbon-aware process execution, https://camunda.com/blog/2023/07/carbon-aware-process-execution-connector/, 2023. Accessed: 2025/05/27.

[15] P. Tschandl, C. Rosendahl, H. Kittler, The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions, Scientific Data 5 (2018) 180161. doi:10.1038/sdata.2018.161.

[16] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (1998) 2278–2324. doi:10.1109/5.726791.

[17] M. Take, C. Becker, S. Alpers, A. Oberweis, Modeling the integration of machine learning into business processes with BPMN, in: X.-S. Yang, R. S. Sherratt, N. Dey, A. Joshi (Eds.), Proceedings of Eighth International Congress on Information and Communication Technology, Springer Nature Singapore, 2024, p. 943–957. doi:10.1007/978-981-99-3236-8_76.