Using the Strategy Design Pattern for Hybrid AI System Design

Stephan Jüngling[1], Martin Peraic[2] and Cheng Zhu[3]

[1]FHNW University of Applied Sciences Northwestern Switzerland, School of Business, Peter Merian-Strasse 86, 4052 Basel, Switzerland

[2]FHNW University of Applied Sciences Northwestern Switzerland, School of Business, MSc Business Information Systems

[3]SZTU Shenzhen Technology University, Shenzhen, China

Abstract

The idea of design patterns originated in the architecture domain, subsequently shaped the standardization and communication of object-oriented system design for IT architectures, and facilitated the description of best practices in business process design. Recently, the idea of design patterns not only stipulated an initial collection and classification of machine learning patterns, but has also been used to structure and document machine learning based systems from a traditional software engineering perspective. We promote the idea of using design patterns as a general means to visualize the design of hybrid AI systems and present how the strategy design pattern in particular can be used for a passenger counting system by switching the implementation strategies from a standard YOLOv5 based object recognition with Deep Sort tracking to a customized head-based YOLOv5 detection in combination with a customized Deep Sort tracking strategy. In our example, the newly presented human head detector and tracker could significantly improve the overall accuracy of passenger counting in dense and crowded situations. Furthermore, we show, how rule-based symbolic decisions can be allocated to an AbstractStrategy class, while the sub-symbolic machine learning task is delegated to the most appropriate person- or head-based ConcreteStrategy class during run-time.

Keywords

design pattern, UML, passenger counting, hybrid AI, YOLOv5, Deep Sort, combining rules and machine learning

1. Introduction

Design patterns have been established as elements of reusable object oriented (OO) software design [1]. They are well known as Gang of Four design patterns (GoF-DP) and describe best practice solutions to typical software design problems with the help of different UML (Unified Modelling Language) [2] models such as UML class diagrams describing their structure or UML sequence diagrams describing details about their behavior. However, other UML models such as UML use case diagrams are also used to do requirements engineering and to specify the usage of IT systems from a very high abstraction level. As such, UML is already a shared

In A. Martin, K. Hinkelmann, H.-G. Fill, A. Gerber, D. Lenat, R. Stolle, F. van Harmelen (Eds.), Proceedings of the AAAI 2022 Spring Symposium on Machine Learning and Knowledge Engineering for Hybrid Intelligence (AAAI-MAKE 2022), Stanford University, Palo Alto, California, USA, March 21–23, 2022.

stephan.juengling@fhnw.ch (S. Jüngling); martin.peraic@students.fhnw.ch (M. Peraic); zhucheng725@foxmail.com (C. Zhu)

modeling language between business analysts and software developers, capable of modeling many different aspects of IT systems and describing their structure as well as their behavior. It seems a logical consequence to use UML and design patterns to describe not only the different architectures and design ideas of software architectures, but also how machine learning models are embedded into IT systems. Only recently, the idea of design patterns is also picked up by the the machine learning community by Lakshmanan et al. [3]. In their book about Machine Learning Design Patters (ML-DP), they present a comprehensive collection of tips and tricks from experienced machine learning experts to serve machine learning practitioners as a collection of best practice solutions to typical practical machine learning problems. Given that more and more hybrid systems as a combination of symbolic and sub-symbolic AI components are embedded into traditional software stacks, it would be beneficial to establish UML and design patterns as shared vocabulary and visual expression language between traditional software engineers, machine learning experts and knowledge engineers.

2. Related Work

Compared to the software development community, where Design Patterns (DP) are well settled to communicate design ideas, to improve the overall system design or enhance the maintainability and extendability of code, the use of DPs to communicate design ideas in the field of machine learning is quite new and not yet well established. While the positive impact of DP on the quality of software development is researched in a recent systematic literature review from different perspectives [4], their manifestations in AI are still quite diverse. In the aforementioned book about ML-DP, an initial set of 30 different patterns are presented in a standardized format of addressing problems, solutions, trade-offs and alternatives for each of the patterns, which is comparable to the GoF-DP template, which standardized their descriptions into *intent, motivation, applicability, structure, participants, collaborations, consequences, implementation, sample code and known uses.* The 23 GoF-DP are categorized into *creational, structural and behavioral* DPs, while the newly proposed ML-DP are categorized into *Data Representation, Problem Representation, Model Training, Resilient Serving, Reproducibility and Responsible AI.*

Both pattern collection types provide domain specific guidance in form of best practices of reusable problem solution pairs. However, the scope of the ML-DP is much broader and the complexity much higher. The GoF-DP can easily be described with UML class and sequence diagrams since these patterns focus on the collaboration of classes with appropriate interface designs. Aspects of the software development lifecycle (SDLC), deployment of components, aspects of performance or different data-types are out of scope of the GoF-DP, while the ML-DP cover such aspects in order to provide some guidance for machine learning practitioners. In the case of hybrid AI systems which combine sub-symbolic machine learning algorithms with symbolic rule-based components, the complexity of the domain increases even further. To visualize the interplay between data, symbolic and sub-symbolic components, Van Harmelen and Ten Teye developed the concept of a boxology as a set of compositional DP defining their own graphical syntax and semantics to describe all the possible solution designs of such hybrid AI systems [5].

There are also many ways, how GoF-DP and Machine Learning (ML) can be combined. Possible combinations range from applying ML algorithms to detect the GoF patterns in programming code [6] up to the opposite scenario of applying GoF DP to design the architecture of ML-based systems, such as proposed only recently by [7]. In their paper, Take et al. use different GoF DP such as *Factory Method* as a representative of creational DPs or the *Strategy* DP for implementing deep ensemble learning, which belongs to the structural GoF-DPs. Their Strategy interface *ML Model* exposes the methods *prediction()* and *train()* to the client code. We propose a slightly different approach for the implementation of the *Strategy DP* and factor the process of training the models out of the Strategy Interface and propose to reduce the API to the application of different strategies at run-time (e.g., focusing on the method *predict()*). Furthermore, we will show, how the choice of the specific run-time strategy can be delegated to an *AbstractStragegy* hosting the symbolic rules for the application of the specific *ConcreteStrategies*, which in turn apply the most appropriate sub-symbolic ML evaluations at hand.

3. Strategy Design Pattern in Hybrid AI Systems

We like to go a little bit deeper into the implementation details of applying the strategy DP in the context of combining symbolic and sub-symbolic components for the allocation of responsibilities to the typical classes involved in this structural DP. As stated by Gamma et al., the intent of the strategy DP is to define a family of algorithms, encapsulate them and make them interchangeable. Strategy and Context interact in order to determine the the most adequate algorithm, or in our case of hybrid AI systems, the most adequate ML model. When calling the Strategy, the Context either passes all the necessary data as arguments to the Strategy for calling the *predict()* method, or the Context might pass itself as an argument to the Strategy (see figure 1) in order that the Strategy itself can determine which of the concrete strategies maps best to the current prediction task at hand.

Different ML models can be encapsulated in *ConcreteStrategies* and as such, can change the behavior of the *Strategy* which is seen by the *Context*. However,the first thing when implementing a new extended functionality in a software system is to decide where to allocate the additional code. In object oriented programming one basically has only two choices, either to use inheritance or delegation. In case of using inheritance, you extend one of the existing classes to reuse the existing functionality and add additional methods or overwrite or override existing methods. When implementing software systems with OO languages such as Java, C++ or Python, it is also a good practice to reduce the coupling in the overall class design as much as possible, by programming client code towards an interface and not towards an implementation. In the case of hybrid AI system design, the forces are somehow different. Knowledge Engineering (KE), ML, and OO programming are mostly disjoint disciplines and some of the current shortcomings come rather from to little than to tightly coupling. The idea of optimizing the overall system design in terms of coupling and decoupling of individual components remains the same. But in the domain of ML, it is not yet defined how to model their domain specific characteristics, where part of the models can be "reused and extended" with the help of transfer learning, which would be kind of similar to inheritance in OO class design. However, what we can visualize, is the aspect of encapsulating the different ML models in form

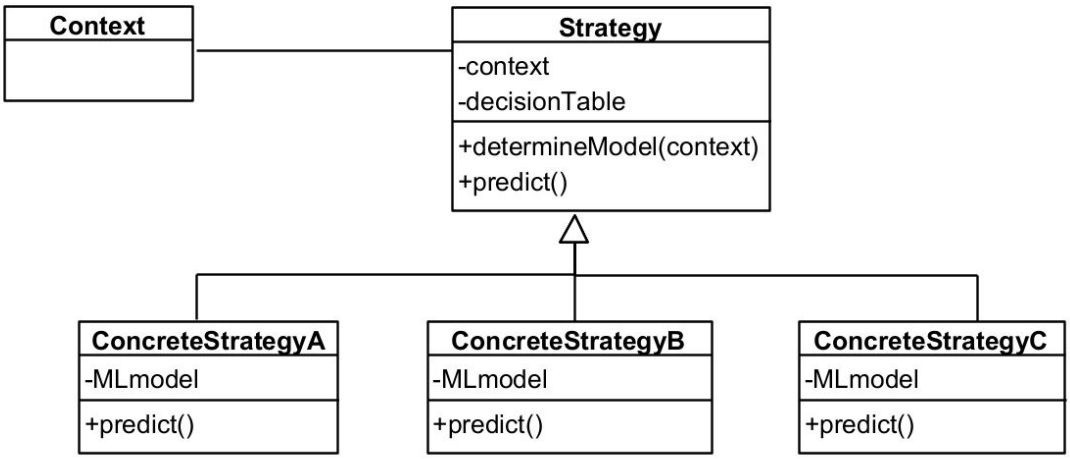

Figure 1: Strategy Design Pattern

of different *ConcreteStrategies*, where the Strategy DP can be used to modify the behavior of the ML model, as it is perceived by calling the interface of the Strategy of the Context class. However, in the long run, it would also be quite nice to show explicitly the reuse of pre-trained ML models, which can be adapted for the different business purposes. Nevertheless, we show in the following section, how the strategy DP and different *ConcreteStrategies* can be used to model and optimize the implementation of a passenger counting system.

4. Use Case: Implementation of Passenger Counting System

While the initial experiments of using video based passenger counting systems as cost effective alternative for passenger counting in public transportation systems has been investigated with Basler Verkehrsbetriebe BVB [8, 9], it turned out to be easier to continue experiments with real video material from Shenzhen, rather than doing similar investigations in Basel. The Shenzhen Metro serves over 5.5 million passengers daily and is one of the fastest growing and largest metro networks in the world [10]. During rush hours, large crowds of people commute in the network with 238 metro stations to change lines, to enter or to leave the metro system. Measuring and tracking these passenger flows within the metro stations in real time with AI-based models poses inherent problems. Due to the dense crowding, people are occluded by other people or objects and are thus not always immediately visible to the detector. This leads to objects being detected too late, twice or incorrectly.

4.1. Initial engineering and testing of the machine learning models

A collaborative student project between SZTU School of Urban Transportation and Logistics and FHNW School of Business [11] focused on exploring potential approaches for solving this

challenge. Initially, the task was defined as counting the persons within a defined view of a surveillance camera by augmenting a line. Any objects that traverse this line are counted. To achieve this contextual objective, a standard AI-based architecture for passenger detection and tracking was implemented. Based on the observations from various experiments with the available video data, possible solutions have to be designed, verified and tested within a single application in order to optimize the overall accuracy of the passenger counting system.

Having defined the methodology of this project, the individual approaches and their implementations shall henceforth be explained. To provide a starting point for this project, YOLOv5 [12] was chosen for detection and Deep Sort [13] for continuous object tracking. Both implementations were integrated with their standard configurations and models. No additional configurations were made for either YOLOv5 or Deep Sort. The objective of this standard implementation was to explore how capable the deployed AI architecture was to handle the passenger counting scenario. Subsequent experiments proved that many people are not detected by YOLOv5, particularly in large crowds. The selected angle of the surveillance camera causes the detection to ignore the bodies of people who are either located further back in dense crowds or are occluded behind objects. Based on the research of Hong et al. [14] and the initial experimental results, a decision to adapt the standard model of YOLOv5 was made. Rather than detecting entire humans, heads should be focused, as they appear to have a less higher probability to be occluded compared to bodies.

Figure 2: Standard Setup (left) vs. Custom Setup (right)

Approximately 1000 data samples were gathered and labeled from various recordings from different camera angles of the Shenzhen Metro. Subsequently, the YOLOv5 model was re-trained. Since the detector now passes bounding boxes of heads to the tracking, the feature extractor of Deep Sort must be adapted as well. To accelerate the training and embedding process, the concept and implementation described by Maiya [15] was adopted. Thereby a Siamese network [16] is used, which is ideally suited for feature mapping tasks. Another 5000 data samples were collected for the training of this network.

Once all the components had been merged together, it was time to run the experiments. For this purpose, various video sequences were selected from the available Shenzhen Metro data. Different sizes of passenger flows were taken into account and the number of people passing the augmented counting line was determined for each video. Using these video clips combined

with the ground truth of the number of passengers, both the standard models and the adapted models could now be examined for their counting accuracy. The customized setup achieved a Mean Average Error (MAE) of 1.7714 and the standard setup a MAE of 6.4857. The results as shown in figure 3 impressively illustrate how the adapted AI setup can significantly measure the number of passengers with higher accuracy compared to the standard setup. Even under very complex movement patterns and large crowds, the adapted system was able to perform highly accurate counts.

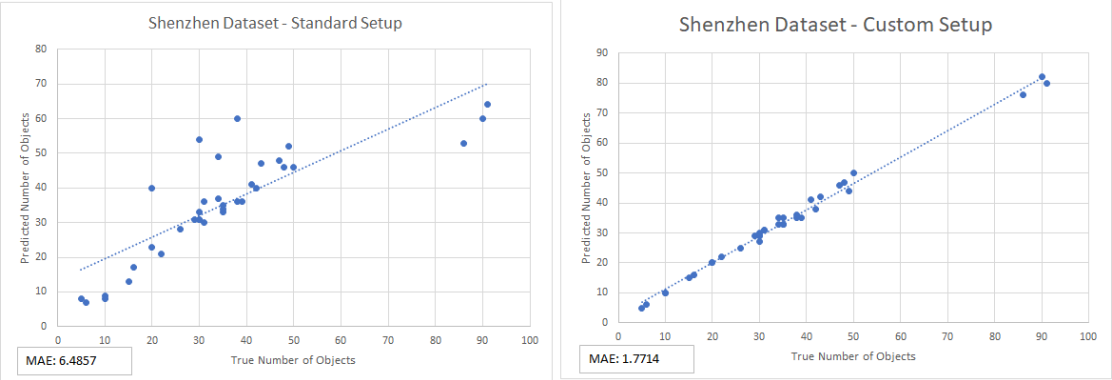

Figure 3: Evaluation between the Standard Setup and the Custom Setup

Besides the higher counting accuracy, the customized setup also recorded significantly more tracks per frame. Unlike the standard setup, which only achieved between 6-10 tracks per frame, the customized setup as illustrated in figure 2 was able to register up to 80 tracks per frame. This increase in active tracks could be achieved as assumed by reducing occlusion effects. Nevertheless, negative features of the adapted version could be observed as well. Since the object size is considerably reduced by the shift from entire bodies to heads, the number of pixels also decreases the further back the person is located in the image. Thus, especially in the background where the pixel density is small, a re-identification of the objects frequently occurs. This problem can be solved relatively easily by excluding parts of the area, so that only objects located in a certain range in front of the counting line are tracked.

4.2. Refactoring to Patterns

From a functional perspective, the implementation of the use case has been completed, but from a software engineering perspective, the code needs to be refactored in order to properly allocate responsibilities to classes and to improve the overall system design [17, 18]. In order to encapsulate the functionality of specific ML algorithms and to provide the flexibility of switching and comparing the different passenger counting approaches, the Strategy DP can be used, as illustrated in figure 4.

The *Context* class holds the attribute *frame* and counts values for each person passing the augmented line in one of the two possible directions (either the objects enter or leave the metro). Two separate strategies are used for the detection and tracking, which both can be implemented with the standardized as well as the customized version. In chronological sequence within the

application, frame by frame, this sequence of detection strategy followed by one of the tracking strategies can be repeated. Both abstract classes *DetectionStrategy* and *TrackingStrategy* expose

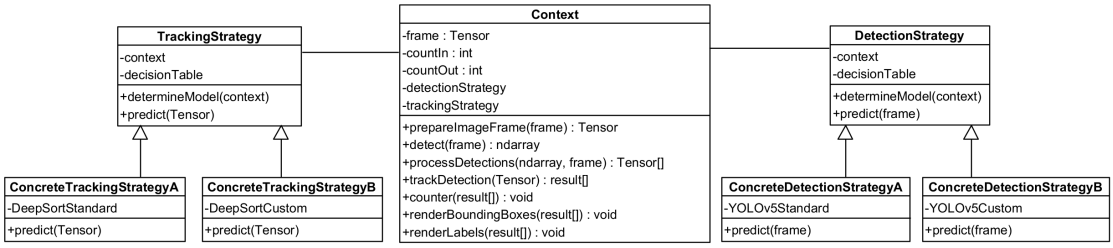

Figure 4: Strategy Design Pattern for Passenger Counting Use Case

the interface from the *ConcreteStrategies* and hold a reference to the *Context*, which can be passed at the method call *determineModel()*. This allows us to dynamically evaluate the concrete strategy based on different input parameters, which could be implemented in form of a decision table [19], which in turn is a typical representative of a rule-based system. With the help of the Decision Model and Notation DMN, entire decision hierarchies might be compiled, such as shown in figure 5, where the specific implementation strategy is determined as combination of a SystemContext and a BusinessContext.

The SystemContext might refer to specific technical circumstances such as the current hardware type, camera resolution or number of GPUs available, while the Business Context might consider different situations such as people density or even different business purposes such as counting the number of persons or counting the number of persons wearing a mask, as it might be of interest in the current situation of COVID-19.

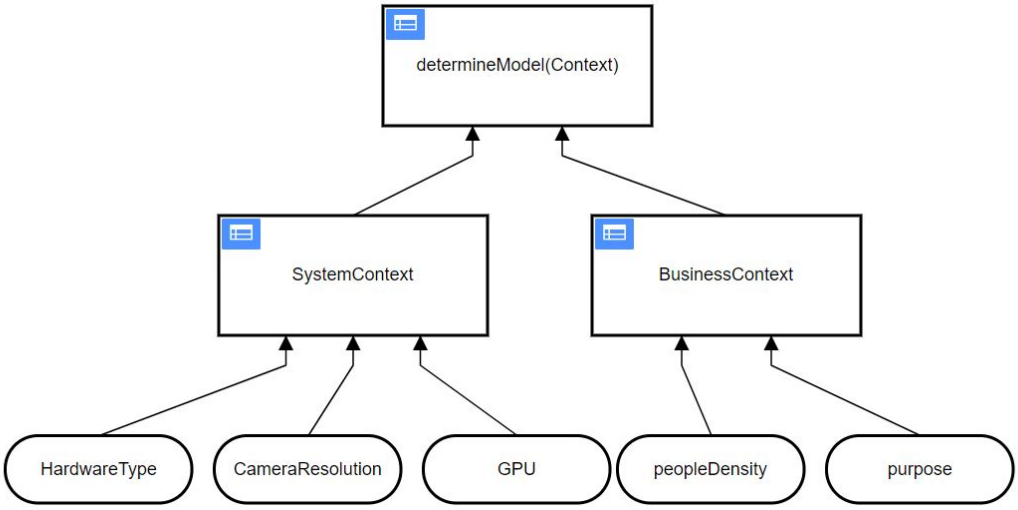

Figure 5: Strategy evaluation using Decision Requirements Diagrams (DRD)

4.3. Deployment Lifecycle

Although the development of hybrid AI-based software systems follow similar software development lifecycle stages and concepts as traditional IT systems, some of them are special due to the different nature of ML and KE. The ML lifecycle needs dedicated phases of data extraction, data preparation, model training evaluation and testing, until it can be deployed into operations. Not only the kind of reuse such as OO with established inheritance mechanisms, but also the maturity of these reusable concepts are different. Nevertheless, an increasing number of ML models are stored in standardized model repositories such as TensorFlow Hub [20], facilitating reuse in different BusinessContexts and SystemContexts. Hence, continuous delivery and automation pipelines can be built to enable the development and operation of large software systems, known as DevOps and MLOps, respectively.

Hybrid AI systems thus require a combination of disciplines from the selection and development of the model up to the final deployment. Figure 6 illustrates a possible solution path for this process. Once the model engineering is started, possible model reuse opportunities must be considered. Based on the requirements and the context, an existing model can be used either completely or as a starting point for transfer learning. This manually executed task facilitates the description of the abstract class using the Strategy DP. By using UML and standardized DP as proposed by this paper, a generally understandable blueprint can be developed, which can later be implemented in the system by different domain specialists. Subsequently, the necessary rules for the run-time strategy selection have to be defined. Based on the possible parameters of the context, a DRD diagram or a decision tree can be used as shown in Figure 5. To complete this first design phase within the lifecycle, all partial elements must now be tested for their compatibility and performance. Once the tests are successful, all symbolic and sub-symbolic elements can be deployed. Within the run-time environment, a monitoring can now be performed to determine whether the hybrid AI system is able to perform the given task. If the performance is either faulty or poor, or if the strategy requirement suddenly changes, adjustments can now be made dynamically to both the strategy and the decision model. Due to the standardization, these parts can be easily exchanged and deployed in a very short time.

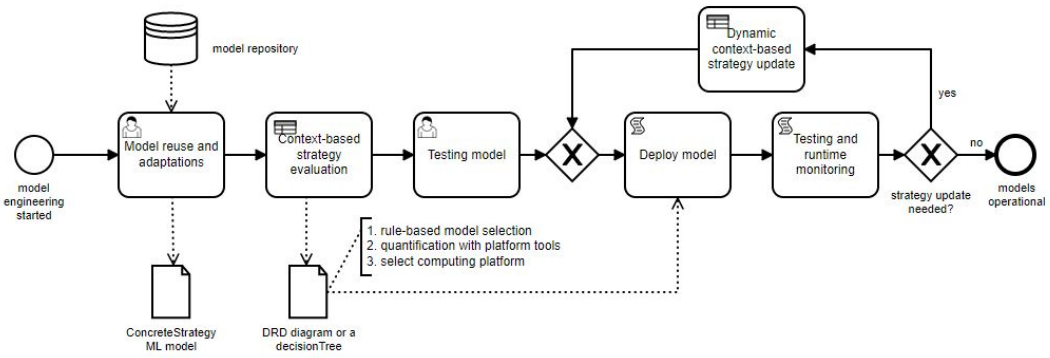

Figure 6: Deployment Lifecycle for the hybrid AI-based IT systems

Applied to the Shenzhen Metro use case, this lifecycle process was triggered by the need for

improved models for passenger detection and tracking. In this process, the already available models YOLOv5 and Deep Sort were optimized to fit the head detection scenario. Furthermore, the implementation was optimized according to the Strategy DP as described in Figure 4. Using a decision table, various parameters from the context such as recognition targets, accuracy, speed, density, etc. can be used to identify a concrete strategy. After successful integration testing, the hybrid system could be deployed followed by real-time monitoring. In case of accuracy degradations due to changing contexts or scenarios, strategies can be switched and the system can adjust to the new situation thanks to the standardized interface and the recursive optimization process.

5. Discussion

As shown in the implementation of the current application of passenger counting with the potential switch between different implementation strategies based on head or entire body detection, the structure of the strategy design pattern can easily be used to visualize the design idea off implementing different behavior of different machine learning algorithms in the run time environment. However, as described in the previous chapter about deployment possibilities, the complexity of the decision about the best choice of the deployment of the specific ML models can be huge and the number of facts and rules for proper selection by the DRD quite numerous. The overall lifecycle of the most adequate algorithm, which can be engineered, trained and validated based on the best practices of the ML-DP before it is deployed as strategy to the run time environment is quite challenging, and appropriate build pipelines from the design-time to the run-time environments, as they are known from the deployment of traditional software components remain to be standardized. Appropriate visualizations, which can be used for the abstraction of the different ML models are currently lacking. Could UML be extended by additional diagram types that could visualize the re-use and modifications of existing ML models? We raised this question already at the beginning when making the analogy of OO inheritance to transfer learning in ML. How could the different modifications to the architectures of existing ML models, which can be visualized as textual output in different ML models (e.g., the different layers of a Keras model with commands such as *model.summary()*) be visualized in kind of UML class diagrams? Up to our knowledge, appropriate solutions are currently not even proposed.

6. Conclusion and Outlook

While design patterns are state of the art for many years in software engineering, investigations in machine learning design patterns only started recently. In order to build hybrid AI systems, where machine learning and knowledge engineering should be combined within a single system, the glue that enables this combination is still designed with traditional programming, and appropriate software architectures need to be built up. Therefore current best practice solution architectures which are built with the Gang of Four object oriented design patterns can be used to embed the appropriate machine learning and knowledge engineering components. We use strategy design pattern and show how rule based components can be embedded in

the abstract strategy which in turn calls the most appropriate machine learning solution as a concrete strategy, solving the business problem at hand. In the use case of passenger counting from scenes with crowded people in the Shenzhen metro, we use YOLOv5 and Deep Sort as tracking and detection algorithms, we have shown that customized versions of both, which do the counting based on head tracking instead of entire person tracking Can be seen as different strategies showing superior performance in a given environment. We show how the rule based and machine learning components can be embedded in an overall IT system by using the idea of refactoring to patterns, which in turn can be visualized with UML. This strategy design pattern is only one of the 23 GoF design patterns, and for the remaining design patterns appropriate ways of how machine learning and knowledge-engineering components can be embedded and combined for the design of hybrid AI-systems remain to be researched. Additional investigations into how machine learning components that are reused in form of transfer learning, can be visualized in analogy to the successfully standardized inheritance mechanism from software engineering. Further investigations will be necessary in terms of mutual collaboration between the The different communities of software engineering, machine learning and knowledge engineering experts to foster common ways of finding optimal designs of hybrid systems. Other than in the single domain of software engineering, where the GoF design patterns originated from controlling typical defects of tight coupling of code, the same design patterns contribute to embed more coupling between currently quite heterogeneous fields of research.

References

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns - Elements of Reusable Object-Oriented Software, 1st ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[2] Unified modelleing language uml 2.5, 2015. URL: https://www.uml-diagrams.org/.

[3] V. Lakshmanan, S. Robinson, M. Munn, Machine Learning Design Patterns - Solutions to Common Challenges in Data Preparation, Model Building, and MLOps, 1st ed., O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2020.

[4] F. Wedyan, S. Abufakher, Impact of design patterns on software quality: A systematic literature review, IET Software 14 (2020) 1–17. doi:10.1049/iet-sen.2018.5446.

[5] F. Van Harmelen, A. Ten Teije, A boxology of design patterns for hybrid learning and reasoning systems, CEUR Workshop Proceedings 2491 (2019) 97–124. doi:10.13052/jwe1540-9589.18133. arXiv:1905.12389.

[6] S. Uchiyama, A. Kubo, H. Washizaki, Y. Fukazawa, Detecting Design Patterns in Object-Oriented Program Source Code by Using Metrics and Machine Learning, Journal of Software Engineering and Applications 07 (2014) 983–998. doi:10.4236/jsea.2014.712086.

[7] M. Take, S. Alpers, C. Becker, C. Schreiber, A. Oberweis, Software design patterns for ai-systems, CEUR Workshop Proceedings 2867 (2021) 30–35.

[8] M. Peraic, Feasibility Study : Al-based Passenger Counting System for Public Transit, (bachelor thesis). University of applied sciences and arts northwestern Switzerland, Basel, Switzerland (2019).

[9] S. Jüngling, M. Peraic, A. Martin, Towards AI-based Solutions in the System Development Lifecycle, Proceedings of the AAAI 2020 Spring Symposium on Combining Machine Learning and Knowledge Engineering in Practice (AAAI-MAKE 2020) - Volume I 2600 (2020).
[10] UITP, World metro figures statistics brief, 2018. URL: https://www.uitp.org/publications/world-metro-figures/.
[11] G. S. Lu, Y. S. Peng, M. F. Peraic, Ai based passenger conting systems - approaches in basel ans shenzhen: Independent learning module msc fhnw school of business and sztu school of urban transportation and logistics, 2021.
[12] J. Glenn, Yolov5 in pytorch, 2020. URL: https://github.com/ultralytics/yolov5.
[13] N. Wojke, A. Bewley, Deep cosine metric learning for person re-identification, in: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), IEEE, 2018, pp. 748–756. doi:10.1109/WACV.2018.00087.
[14] Z. Hong, L. Zhang, P. Wang, Pedestrian detection based on yolo-d network, 2018, pp. 802–806. doi:10.1109/ICSESS.2018.8663790.
[15] S. R. Maiya, Deep sort custom object tracking, 2019. URL: https://github.com/abhyantrika/nanonets_object_tracking/.
[16] G. Koch, R. Zemel, R. Salakhutdinov, et al., Siamese neural networks for one-shot image recognition, in: ICML deep learning workshop, volume 2, Lille, 2015.
[17] M. Fowler, Refactoring - improving the design of existing code, in: Addison Wesley object technology series, 1999.
[18] J. Kerievsky, Refactoring to patterns, in: XP/Agile Universe, 2004.
[19] Decision model and notation dmn v.1.3, 2021. URL: https://www.omg.org/spec/DMN/1.3/PDF.
[20] Tensorflow hub, 2021. URL: https://www.tensorflow.org/hub/.