An Approach for Platform-Independent Online Controlled Experimentation

Florian Auer(✉) and Michael Felderer

University of Innsbruck, Innsbruck, Austria
{florian.auer,michael.felderer}@uibk.ac.at

Abstract. Online controlled experimentation is an established technique to assess ideas for software features. Current approaches to conduct experimentation are based on experimentation platforms. However, each experimentation platform has its own explicit properties and implicit assumptions about an experiment. As a result, experiments are incomplete, difficult to repeat, and not comparable across experimentation platforms or platform versions. Our approach separates the experiment definition from the experimentation platform. This makes the experimentation infrastructure-less dependent on the experimentation platform. Requirements on the independent experiment definition are researched and an architecture to implement the approach is proposed. A proof-of-concept demonstrates the feasibility and achieved level of independence from the platform.

Keywords: Online controlled experimentation · Continuous experimentation · Experimentation platform · Experimentation infrastructure

1 Introduction

Online controlled experimentation is an established approach commonly used by organizations to make data-driven decisions about changes in their product. Fabijan et al. conducted in [9] a survey in which they observed that most organizations use in-house built experimentation platforms. Similar in literature, large organizations report of their self-built experimentation platforms, like Microsoft [15] or Google [23]. However, the development of an experimentation platform is a resource-intensive and error-prone project [16]. Thus, many organizations cannot afford to develop a platform. Alternatives are third-party experimentation platforms. But, these platforms do not support all aspects of experimentation [5] and focus more on the technical execution of experiments. For example, not all platforms (proprietary as well as open-source) support the definition of a hypothesis or criteria to automatically shut down an experiment based on business-critical metrics. Thus, it seems that organizations have to choose between high upfront costs of developing an in-house experimentation platform or to reduce their requirements on experimentation and use a third-party experimentation platform. Moreover, the experiment definitions do not include the implicit assumptions made by the used experimentation platform (e.g. the segmentation algorithm). Hence, the definitions are incomplete and the described experiments are difficult to repeat.

The separation of the experiment definition from the experimentation platform could combine the benefits of both approaches. It would allow organizations to select a cost-effective third-party experimentation platform to execute the experiment while developing independently of it the remaining infrastructure to support the organization's experimentation process. This would, for example, allow developing an infrastructure to assure the quality of the experiments without having to depend on the feature set provided by the experimentation platform.

This research aims to propose an architecture for platform-independent online controlled experimentation by releasing the experiment definition from the experiment platform. As a consequence of the separation, the experiment definition becomes an independent artifact. Moreover, it allows developing experimentation infrastructure independently of the used experimentation platform.

The remainder of this paper is structured as follows. Section 2 provides background information on online controlled experimentation. Section 3 describes the applied research method. Section 4 presents the findings. Then, Sect. 5 presents the architecture, and Sect. 6 the evaluation of it. Section 7 discusses the study. Finally, Sect. 8 concludes the paper.

2 Background

In this section, an overview of the research this paper is based on is given. The overview starts with the general concept of online controlled experimentation. Next, the research on the characteristics of online controlled experiment definitions is outlined, and finally, research on domain-specific language (DSL) to define online controlled experiments is discussed.

2.1 Online Controlled Experimentation

Online controlled experiments are a technique to evaluate software changes based on data [17]. The change could be a novel feature, a performance optimization, modified elements in the user interface, and many more. A version of the software with the change (treatment) is deployed in addition to the unchanged software (control). Thereafter, requests on the software (e.g. user interaction) are split between the two versions (segmentation). In addition to the regular processing of a request, both software versions collect relevant data about the processing and the request. After a predefined duration or number of requests, the collected data (telemetry) is used to calculate metrics. Finally, the telemetry is analyzed, and based on the success criteria that are defined before the execution of the experiment, the experiment is evaluated.

In [2] the authors highlight the technique's potential to improve software quality assurance for modern technologies like machine learning or the internet of things—areas that are challenging for traditional software testing with offline testing techniques.

The experiment lifecycle by Fabijan et al. [8] gives an overview of the activities related to an online controlled experiment (see Fig. 1). First, during the ideation phase, a hypothesis and its implementation are developed. Thereafter, the design of the experiment is specified. This includes amongst others, the user segmentation across the different software versions, and calculations about the size and duration of the experiment. Next, the experiment is executed. Besides the deployment, the instrumentation and monitoring of the deployed software are important. Thereafter, the collected data is analyzed and data-driven arguments for decisions are provided. The last activity is learning, in which the experiment metadata is captured and institutionalized.

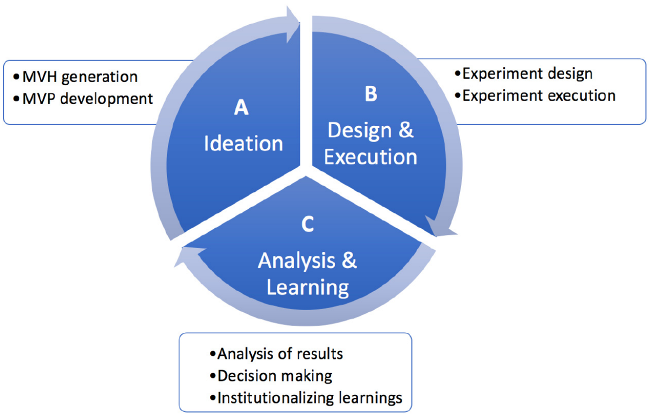

Fig. 1. Experiment lifecycle by Fabijan et al. [8]. It describes the lifecycle of an experiment from its ideation by the generation of a minimal viable hypothesis (MVH) and the development of a minimal viable product (MVP) over the design, execution and analysis to the decision making and institutionalization of the learnings.

A more detailed view of experimentation and its process is given by models for continuous experimentation like the RIGHT model by Fagerholm et al. [13] or the HYPEX (Hypothesis Experiment Data-Driven Development) model by Holmström Olsson and Bosch [20]. There is also research about the required infrastructure [12] and guidelines about experimentation in general [17].

In summary, the process of online controlled experimentation received a lot of attention in research. Process models of continuous experimentation [8,13], experimentation platforms [12] and guidelines [17] for experimentation are researched amongst others.

2.2 Characteristics of Online Controlled Experiments

Although the experimentation process and specifics of it are researched well [1,21], there is little research explicitly on the experimentation definition and its characteristics. Nevertheless, the definition of experiments is fundamental for experimentation. A taxonomy of its characteristics allows experiment owners to choose the necessary characteristics of it for a concrete experiment. Without such an overview, experiment owners are at the risk to miss important characteristics or to define experiments incomplete. Therefore, the authors reviewed in [3] the literature on characteristics of experiment definitions. It revealed 17 properties that were grouped by common themes among the properties.

However, the authors expected that there are additional properties used in practice. Thus, based on the results on the characteristics of experiment definitions, the more detailed study reported in [5] was conducted. The study covers the analysis of existing open-source as well as proprietary experimentation platforms. In [5] the results are combined to one taxonomy of experiment definition characteristics. Figure 2 visualizes the identified characteristics for each phase of an experiment.

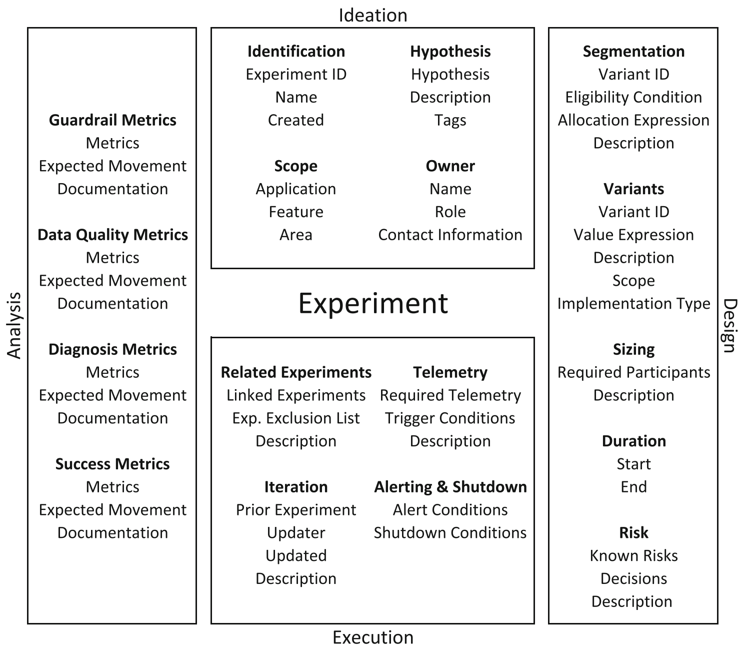

Fig. 2. Experiment definition characteristics taxonomy [5]. It enumerates for every phase of the experimentation lifecycle the characteristics (bold) and their properties (below each characteristic).

To conclude, the authors presented in [5] a taxonomy of the known characteristics and properties used in the definition of online controlled experiments.

However, the application of the taxonomy, or its usefulness to describe experiments was not known.

2.3 Experimentation Definition Language

The taxonomy of experiment definition characteristics [5] represents the characteristics that are used in literature and experimentation platforms. However, it is not fully clear whether this set of characteristics is useful to define concrete experiments. Thus, a DSL was developed in [4] that was built on the taxonomy.

The language allows to describe an experiment with the characteristics of the taxonomy. As the host language, the most commonly used exchange format observed during the analysis of experimentation platforms, JSON (Javascript object notation) was selected. Listing 1.1 provides an experiment defined in the language and shows that the structure follows the taxonomy closely. Each characteristic and its properties can be defined using the JSON syntax.

Listing 1.1. Structure of an experiment written in EDL. It follows closely the structure of the experimentation characteristics taxonomy (see Fig. 2).

```
{
  "Ideation":{
    "Hypothesis":...,
    "Owners":...
  },
  "Design":{
    "Variants":...,
    "Segmentation":...
  },
  "Execution":{
    "AlertingAndShutdown":...
  },
  "Analysis":{
    "SuccessMetrics":...,
    "GuardrailMetrics":...
  }
}
```

A technology acceptance study [4] revealed that the language and the idea of describing an experiment in a structured form, according to the characteristics were accepted by the majority of participants. Moreover, for most participants, the language was considered easy to use. However, the data too showed that there is a relationship between the participant's assessment of the language's ease of use and the participant's background (i.e. business or software engineering).

As a result, the research on a DSL for the definition of an online controlled experiment shows that the developed taxonomy with its characteristics and properties is considered useful. However, the representation of the definition as a DSL hosted in JSON may not be beneficial for all stakeholders.

3 Research Method

This study aims to propose an architecture for platform-independent online controlled experimentation. It is based on the idea of separating the experiment definition from the experimentation platform. Therefore it is necessary to study which elements an experiment definition includes (0.), for what an experiment definition is used during the experiment lifecycle [8] (1.) and what the qualities of an experiment definition are to ensure reliable experimentation (2.). Next (3.), an architecture needs to be designed that meets the identified requirements of (0.) and (1.). Finally, it is necessary to evaluate whether the approach is feasible and beneficial (4.).

The first objective (0.) was mentioned for completeness. It is already researched in [5], in which the authors studied the characteristics specified in an experiment definition. Moreover, in [4] a DSL for an experiment definition was proposed and evaluated. The results are summarized in the Background Section. It follows the objectives researched in this study.

1. *Roles of experiment definitions.* After having studied what characteristics an experiment definition describes, it is necessary to identify the roles that an experiment definition takes in each phase of the experiment lifecycle. The roles describe the applications of the information stored in the experiment definition. Moreover, they make visible the requirements of the experiment definition on the proposed architecture.
2. *Qualities of experiment definitions.* The qualities of an experiment definition are requirements that need to be fulfilled to ensure reliable experimentation. The separation of the experiment definition from the experimentation platform should not impact the quality of an experiment definition or the experiment itself.
3. *Architecture.* An architecture is proposed that separates the experiment definition from the experiment platform. It shows what infrastructure components have to be provided to support all requirements imposed by the identified roles and qualities.
4. *Feasibility.* Finally, the proposed architecture is evaluated about its feasibility and its potential to mitigate the dependency on the experimentation platform. Therefore, a prototypical implementation for an experimental scenario is presented that retains the essence of the problem in an industrial setting. Additionally, the migration to another experimentation platform in the context of the scenario is discussed to evaluate the architecture's independence to the experimentation platform.

The roles and qualities are inferred from the results of the previously conducted literature review [3], observations made during the analysis of open-source as well as proprietary experimentation platforms in [5] and adjustments made during the evaluation. Note that the identified qualities and roles constitute our proposed architecture. However, they are not static nor expected to be complete. Further research on roles and qualities might extend the enumerations about additional roles and qualities.

4 Experiment Definition's Qualities and Roles

In the following the qualities and roles of experiment definitions that were identified from the results of the literature review [3] and observations made during the analysis of experimentation platforms [5] are presented.

4.1 Qualities

Four qualities of experiment definitions were identified.

Knowledge exchange of experimentation results and their implications support the collaborative optimization of systems [19]. Improving the *institutional memory* of experimentation [7] also prevents from accidental repeating already conducted experiments. Therefore, Fabijan et al. [7] suggest building an archive of executed experiments. It should summarize an experiment with metadata like its hypothesis, execution date, and results. A requirement to enable knowledge exchange is that experimentation decisions (like the selection of the learning component [18]) are explicitly documented.

Reproducibility and replicability are two important qualities of an experiment [6]. Reproducibility means that experiments can be independently replicated by another experimenter. Therefore, the context of the experiment and a detailed description of all steps are necessary. Furthermore, Buchert et al. [6] note that "the description of an experiment has to be independent of the infrastructure used". Replicability refers to the act of repeating an experiment under the same conditions, which will lead to the same results.

Traceability. Experiment iterations allow to gradually improve the system under experimentation by iterative adjustments of the parameters in order to maximize a metric of interest [22]. Hence, experiments are commonly part of a series of iterative evolving experiments. Specifications of experiments should therefore highlight the relationship between experiments to improve the traceability.

All of these qualities are supported by the experiment definition language (EDL) [4]. Required characteristics and properties are provided by the language to define reproducible and replicable experiments. Moreover, the language itself is based on the data exchange format JSON which ease the information exchange. Additionally, each characteristic has properties to document the decisions behind the chosen property values. Finally, properties are included that can be used to reference to previous versions of an experiment and document the changes made (see Sect. 2.2).

4.2 Roles

Concerning the roles of an experiment definition, the analysis of the selected papers [3] and observations among experimentation platforms identified that the definition serves various purposes throughout an experiment. Each phase uses the experiment definition in another way (see Fig. 3).

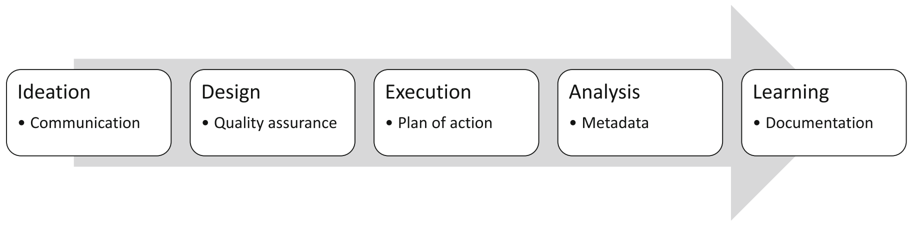

Fig. 3. Main role of the experiment definition in each phase of an experiment. In each phase of the experiment the definition serves another main role.

Communication. A central purpose of the experiment definition is its usage as a communication tool between stakeholders. Online controlled experimentation involves multiple stakeholders that need to exchange information between them. Fagerholm et al. [13] enumerate various stakeholders that are involved in the process of experimentation. The business analysts and product owners that create ideas for experimentation, the data scientist that ensures rigor experimentation, software developers that implement the necessary modifications, quality assurance to verify the software changes, DevOps engineers to deploy the changes, and many more. Although Fig. 3 indicates that communication is used mostly during the ideation of an experiment, activities in all phases can be found that use an experiment definition artifact as a communication tool. For example, the hypothesis made by a business analyst is used by a data scientist to define a fitting segmentation that is used by a DevOps engineer to adapt environment variables. To conclude, all stakeholders use the experiment definition as a tool to manifest and share their decisions on an experiment.

Quality Assurance. The results of an online controlled experiment can have a significant impact on the decisions made by an organization. Hence, it is important to ensure reliable and comprehensible results [19]. Therefore, the quality of experimentation needs to be assured. A structured experiment definition improves the constructive quality of an experiment. It can, for example, limit the number of possible values for a property. In addition, analytical quality approaches can be applied to a definition. Examples are tests to ensure required properties, or sanity-checks on the respective experiment design (e.g. is a user segment assigned to every variant). The definition is used in each phase of an experiment to improve the experiment's quality. For example, in the ideation phase, constructive quality approaches on the definition ensure a solid definition of an experiment idea. During the design phase, analytical quality approaches support the data scientist in the specification of experiment parameters. Moreover, the quality of the experiment execution benefits from a well-structured definition that allows automating previous manual steps. Similarly, the analysis and the learning phase rely on trustworthy information that benefits from a reliable experiment definition. As a result, the experiment definition considerably influences the quality assurance of experimentation throughout each experimentation phase.

Plan of Action. The execution of an experiment requires the accurate execution of a sequence of actions to ensure a trustworthy result. An explicit plan of action that lists all steps of an experiment supports the execution of an experiment. Moreover, it improves the experiment's reproducibility, which is a fundamental quality aspect for experimentation [6]. The experiment definition is implicitly used as a plan of action in the execution phase of an experiment. In this phase, the specified properties are translated into the required actions to set up and execute the experiment. However, other phases use the definition too as the plan of action. For example, during the design phase, development might have to implement changes to the software according to the definition. Similar, analysis, for example, is directed amongst others by the specified metrics and success criteria stated in the definition. To conclude, the experiment definition serves for many activities during an experiment as the plan of action.

Metadata. The definition of an experiment serves as metadata about an experiment [14] during the analysis phase. Data scientists that analyze the collected data are dependent on complete and trustworthy metadata of an experiment to draw conclusions about the collected data. Metadata about an experiment is not only used during the analysis but also, for example, in the ideation phase. Previous experiments could be searched by properties similar to a planned experiment to find relevant experiments and consult their results and learnings [19]. In summary, the experiment definition serves as metadata about an experiment.

Documentation. The prerequisite to draw lessons learned from an experiment is to document it. It is the essential difference between a sequence of independent experiments and continuous experimentation. Although documentation is necessary in each phase of experimentation [14], it is especially relevant for the learning phase. In this phase, the definition serves as a description of the conducted experiment, its idea, the decisions made, and the steps taken. Institutional learning can use this information and draw conclusions from it [10]. For instance, a series of experiments that explore the user habits may reveal that fundamental assumptions about users are no longer true. In addition to learning, documentation can also be useful in other phases of experimentation. For example, during the analysis of an experiment, the documentation of reasons behind decisions made in the design or execution of an experiment gives additional insights into the data. As a result, the experiment definition represents a documentation of an experiment that gives insights into the executed steps and the reasoning behind them.

5 Platform-Independent Experimentation

In this section, the development of the architecture for experimentation platform-independent experimentation is presented. First, the requirements resulting from the roles and qualities identified in the previous sections are discussed. Thereafter, the architecture itself is presented.

5.1 Requirements

The requirements are inferred from the previously identified roles and qualities. Figure 4 visualizes the experimentation lifecycle, the related roles and the requirements on the experiment definition. The definition itself is expected to describe the characteristics of an experiment according to the taxonomy developed in [5]. The qualities are considered in the inferred requirements, which is why they are not explicitly visualized in the figure. In the following, each requirement is discussed in detail.

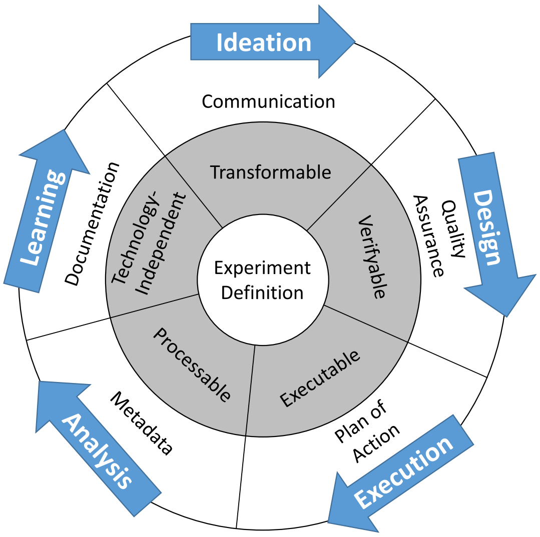

Fig. 4. Roles and requirements of the experimentation definition throughout the experiment lifecycle. The experiment definition (center) is surrounded by its qualities in each phase of the experimentation lifecycle and its main role.

Transformable. The experiment definition not only needs to contain all information relevant for each stakeholder (e.g. hypothesis for the product owner, or segmentation for data scientists) but also be accessible for each stakeholder. The definition needs to be presented in a form that is interpretable and usable for the respective stakeholder and its professional background. The varying fields of expertise of the stakeholders (e.g. business, user experience, development, ...) suggest providing the information in different representations with varying level of detail. For example, developers may prefer a more technical representation (e.g. JavaScript Object Notation), whereas business analysts may prefer a human-readable textual description. Nevertheless, both should work on the same artifact to ensure a single point of truth.

A technology acceptance model study reported in [4] indicates that it is not sufficient to provide one DSL for all possible stakeholders. In the study, a DSL based on the JavaScript Object Notation (JSON) was evaluated. Participants with a strong business background were not as convinced of the language as participants with a technical background.

As a result, the architecture is required to provide information about an experiment in different formats.

Verifiable. Constraints on the structure and the content of the experiment definition are necessary to ensure reliable experimentation. The syntactical verification of the definition is necessary to assure that the properties and values specified in the definition are syntactically correct. Without syntactical valid experiment definitions, the information exchange becomes infeasible. Additional to the syntactical verification, the semantical verification further improves the quality assurance of an experiment. Rules that verify the semantic of a definition complete the verification of an experiment definition. An example of a rule could be that the sum of the user partitioned upon the variants sum up to all users available. Another example could be the enforcement of an organizational rule that for each experiment two owners with emergency contact information need to be defined. Thus, the architecture is required to provide a syntactical verification and the capability to define rules on an experiment definition.

Executable. In the execution phase of the experiment lifecycle, it is required of the experiment definition to provide enough information in a level of detail to infer the plan of action – the steps necessary to execute the experiment. This can include activities like the deployment of a software variant, the collection of data, or the monitoring of shutdown criteria for an experiment (see Fig. 2, Execution). Hence, the definition is required to provide enough information in the necessary level of detail and the architecture is required to interpret the experiment definition and execute the necessary actions to run the experiment.

Processable. The data stored in an experiment definition needs to be in a format that supports the exchange of data between programs. Given that it is the source of information about an experiment, the experiment definition is used by many programs. In order to ease the access of the stored data, the format for the experiment definition artifact should be commonly supported by programs and programming languages.

As a result, the architecture is required to provide the artifact in a commonly supported data exchange format, like XML, JSON, or CSV.

Technology-Independent. The experiment definition should be independent of the concrete technology used to implement the experiment execution, analysis, or archival. The separation between the infrastructure and the experiment definition requires the architecture to provide transformations of the definition of infrastructure specific actions (execution phase) and formats (ideation phase, analysis phase). Nevertheless, the architecture allows creating a robust experiment definition, that is beneficial to documentation, allows interpreting experiment definitions independently of the technology used to execute them, and makes experiments even portable across different experimentation infrastructures.

As a consequence, the architecture is required to define experiments independent of the used technology to conduct the experiment.

5.2 Architecture

The architecture is designed to make a clear distinction between the experiment definition and the experimentation infrastructure (e.g. monitoring service, deployment service). Additionally, it considers all discussed requirements of the experimentation lifecycle on experiment definitions. Note that it is an architecture and not a description of a concrete implementation of a framework. Thus, it focuses on the structure of the system's components it describes. The architecture is visualized in Fig. 5. In the following, the elements of the architecture are described.

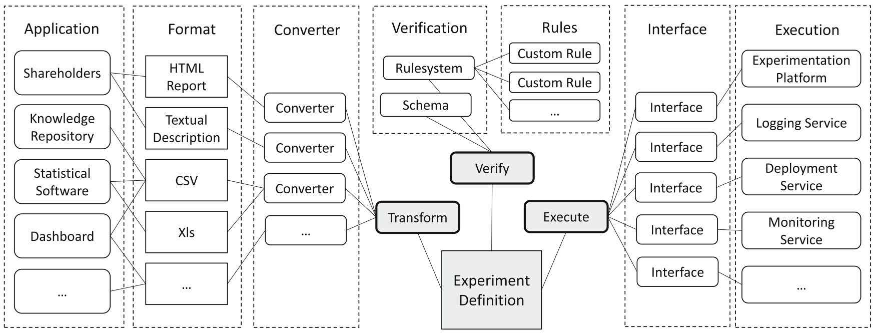

Fig. 5. Architecture for platform-independent online controlled experimentation. The four main elements are the experiment definition, the Transform tool, Verify tool and the Execute tool.

Experiment Definition. In the center of the architecture is the experiment definition artifact. It documents all relevant information of an experiment (e.g. hypothesis, segmentation, or success criteria) according to the taxonomy of experiment characteristics [5]. Therefore, the artifact stores characteristics and their related properties in a systematic way to ensure that the artifact allows systematic access to individual experiment characteristics. Hence, a general data exchange format or a DSL like [4] is suggested as a data format.

Transform. On the left side in Fig. 5, the components that support the transformation of the definition to application-specific formats can be found. The component responsible for this is called *Transform.* It delegates a requested transformation to the appropriate converter. For example, a data scientist may request the metadata of an experiment in the CSV-format for the statistical software R. In this case the Transform component selects among the known converters the appropriate one and executes it. The transformation could also be from an application-specific format to the experiment definition. For a meeting, for instance, the experiment is transformed into an interactive form that allows editing the properties. After the meeting, the form is saved and transformed back to the experiment definition. Note, that the list of formats is exemplary. It depends on the concrete experimentation infrastructure in place and the stakeholders' needs. As a result, the component is extendable by arbitrary converters.

Verify. On top of the experiment definition in Fig. 5 is the *Verify* component. It consists of two subcomponents, namely *Schema* and *Rulesystem.* The *Schema* component verifies the structure of the experiment definition. Most data exchange formats (like XML or JSON) provide a language to describe the structure and verify a document according to it. This technology can be used by the component. The other subcomponent is *Rulesystem.* It is a lightweight, modular system that allows to register custom rules that verify a document syntactically or semantically. A rule, for instance, could be that each experiment has to have a hypothesis following a specific template, like "Based on [qualitative/quantitative] insight, we predict that [change X] will cause [impact Y]" [11]. Rules are a mechanism provided by the architecture to support an automated quality assurance of the experiments. Note that the rules allow to verify an experiment independent of the platform and prior execution of an experiment. Furthermore, they could be used as quality gates that, for example, enforce organizational requirements on an experiment.

Execute. On the right side of the experiment definition in Fig. 5 is the *Execute* component. It is responsible for the interface between the experiment definition and the execution of an experiment. The architecture itself does not include components for the execution or monitoring of an experiment. These are traditional tasks in which experimentation platforms excel [5]. The alternative, to develop custom components that cover tasks like segmentation, is resource-intensive and error-prone as reported in the literature (e.g. [16]). Therefore, the architecture delegates these tasks to individual services or platforms that provide the respective functionality.

6 Evaluation

In this section, a prototypical implementation and an evaluation of the experimentation platform-independent architecture is presented. In the experimental scenario, first, the feasibility of the architecture is evaluated. Second, the claim of platform-independence is validated by changing the experimentation platform and discussing the changes necessary. Finally, the result of the experiment is summarized.

6.1 Scenario

The experimental scenario represents a common infrastructure of an organization developing an Internet service. Thus, common approaches, tools, and methods for the development of an Internet service are assumed. The fictional organization follows the agile development process and uses the Internet service Trello[1] as Kanban board. The developed software is deployed with Docker[2]. Additional assumptions about the scenario are not necessary, given that the experiment focuses on the feasibility and the experimentation platform-independence. Thus, for the scenario, it is not of importance which programming language, libraries, or frameworks are used for the development of the Internet service or possible experiments of it. As experimentation platform the proprietary platform Optimizely[3] was selected.

6.2 System Overview

The implementation consists of three tools, namely `transform`, `verify` and `execute`. For the experiment definition artifact the EDL [4] was selected. It is a DSL based on JSON, which eases the processing of it. As programming language python was used, because of the major ecosystem of libraries and software development kits for third-party applications. An overview of the developed system is visualized in Fig. 6.

The `transform` tool is modular structured and allows adding arbitrary converters in the form of python scripts with the name schema `to-<format>.py` that are located at a specific folder. For the experiment, three converters were implemented that are based on python libraries to convert the information stored in the experiment definition JSON to the respective format.

The `verify` tool is based on two submodules namely `verifySchema` and `verifyRules`. The first, `verifySchema`, provides syntactical verification of the

[1] https://trello.com.
[2] https://docker.com.
[3] https://optimizely.com.

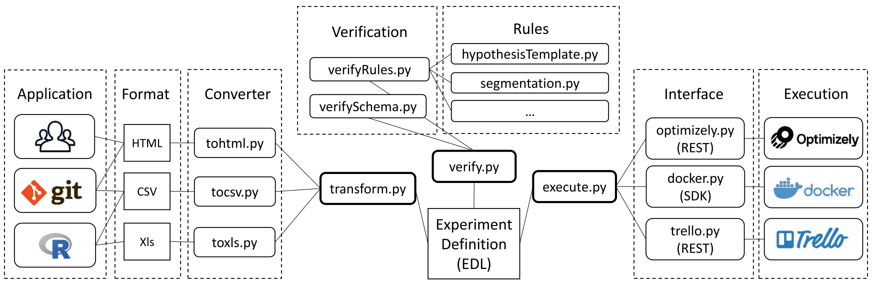

Fig. 6. System overview of the experimental implementation. It demonstrates concrete adaptions of the architecture for the exemplary scenario.

experiment definition. Therefore, the JSON schema definition of the EDL is used to automatically verify the syntax of the definition artifact. The second submodule, `verifyRules` verifies the artifact semantically by executing custom rules against the experiment definition. Rules are python scripts that are located at a specific folder. They can be specific to the project (e.g. 20% is the minimum allocation of users for the unmodified variant) or organization (e.g. two experimentation owners at least).

The `execute` tool is similar implemented as the `transform` tool. It allows adding arbitrary scripts that interpret the experiment definition and execute the related interface calls. In the experimental scenario, three interfaces were considered. An interface to the experimentation platform Optimizely to deploy the experiment on the experimentation platform, another to docker to deploy the software version under experimentation and the third to Trello to create a task for Operations to monitor the experiment. All three of them are built on SDKs or REST APIs that are provided by the tools. The prototypical implementation is available at GitHub[4].

6.3 Feasibility

The feasibility is evaluated by the implementation of the described experimental scenario and the researchers' observations doing so. Therefore, in the following, each components' development is discussed.

Initially, the format and language of the experiment definition artifact had to be selected. A common data exchange format is beneficial, given that the information stored in the artifact needs to be processed by multiple programs. The EDL [4] was selected because it is based on JSON and provides a Schema with all necessary characteristics and properties of an experiment. Given that the verification and interpretation of the artifact are delegated to the `verify` and `execute` tool, a generic experiment definition language like EDL can be used without modifications. Project-specific interpretations or verification rules

[4] https://github.com/auerflorian/platform-independent-experimentation-prototype.

can be implemented with the extension of the respective tools. This eases the decision of the format for the experiment definition.

Next, the tool `verify` that verifies the definition was implemented. It is supposed to verify the definition syntactically and semantically. The syntactical verification is in the case of EDL already provided by the JSON Schema that is defined for it. For the semantic verification, the rule system was implemented. It is a lightweight, modular approach to implement reusable rules. Thus, the verification of experiment definitions can be reused and improved across projects and for different infrastructures without additional implementation effort.

The `transform` tool has a modular architecture and calls the appropriate converter provided as a script within a specific folder. For the implementation of the specific converters, the selected format of the experiment definition artifact was beneficial. JSON stores data objects and thus provides a rich structure of the data, which is used by the EDL to provide the characteristics and properties of an experiment in a structure of hierarchical objects. The additional information by the hierarchy of the individual properties ease the conversion. For example, the hierarchy of the properties could be translated to headings for a report in an HTML report.

Finally, the `execute` tool that redirects calls to the appropriate interface was implemented. For the experimental scenario, an interface to Optimizely, docker, and Trello was implemented. All three systems provide REST APIs or SDKs. Thus, the interface's main complexity was in the interpretation of the experiment definition and translation of it into system-specific function calls. For example, for the experimentation platform Optimizely, the initial implementation created an experiment on the platform according to the experiment definition. In the next iteration, the interface, first verified that there is not already an experiment wit the same ID on the platform. A future iteration could consider to update the experiment specification according to the experiment definition. This demonstrates that the implementation of an interface to an experimentation platform is not a trivial task, if all possible states of the experiment definition and the experimentation platform have to be considered. Note, however, that the proposed architecture does not specify where in the process of experimentation or of the software development process the tools are executed. Thus, with additional call arguments and the integration of the tools at the right places within the development process the complexity of the interfaces could be reduced. Nevertheless, the integration of third-party tools through interfaces introduced the most complexity in the implementation of the proposed architecture.

6.4 Platform-Independence

The platform-independency of the proposed architecture is evaluated with a theoretical modification to the experimental scenario. Therefore, the following addition to the scenario description is assumed:

After a year of experimentation, the organization reevaluates the infrastructure used to identify possible optimizations. The analysis of the infrastructure components revealed that there is another more cost-effective experimentation platform available. Thus, the experimentation platform needs to be changed.

This scenario can lead to considerable migration costs without the application of the proposed platform-independent architecture. All experiment definitions are stored implicitly within the platform. Moreover, the process of experimentation is coupled to the platform and its implicit experimentation lifecycle. Thus, with the change of the platform not only the existing knowledge base of experiments may be lost, but also the process of experimentation, that requires an expensive adaptation of the process to the new platform. Additionally, verification rules that were implicitly in the previous experimentation platform may no longer exist in the new platform or may have changed. To summarize, the migration to another experimentation platform has a considerable impact on the whole experimentation process.

In contrast, with a platform-independent architecture, the migration is reduced to a new implementation of an experiment platform interface. Metadata about existing experiments is not affected and would still be "executable". Moreover, the process of experimentation is not affected. Verification, for example, follows the same organization-defined rules as with the previous experimentation platform.

Note, that in both cases the migration to another experimentation platform may require changes in the software, deployment, or infrastructure. For instance, the interface for the platform has to be implemented and the related code sections within the software that request the experimentation platform to decide which variant to show, have to be adapted. Nevertheless, neither the experiment definition nor the generators or the verification should be affected by the migration.

6.5 Experimental Result

The experimental scenario of an organization developing an Internet service was presented. An implementation of the proposed architecture demonstrated the feasibility of it. The description of the development indicates the implementation effort of its components and may allow reasoning about the possible return of investment when compared, for example, to the outlined benefits in the case of a migration to another experimentation platform.

Finally, the scenario was adapted to portrait the possible impacts of a migration. Thereby, it was argued that the proposed architecture is experimentation platform-independent by considering the changes that are necessary in the case of a migration to another experimentation platform.

7 Discussion

The study identified the roles of an experiment definition throughout the experimentation lifecycle. It shows that in each phase of the lifecycle, the definition of an experiment plays an important role. Moreover, the described qualities and requirements on the experiment definition make the strong impact of the definitions on the success of an experiment visible. For example, its appropriateness as a tool for communication in the ideation phase for each shareholder, the precise representation of the experiment for verification, or its availability in a processable form for the analysis of the collected data.

In addition, the study indicated how dependent the experimentation process is on the experimentation platform that commonly provides the (implicit) experiment definition. Furthermore, the implicit experiment definition of third party experimentation platforms introduces a risk of vendor lock-in. A data-exchange format for experiment definitions does not exist. Thus all metadata about experiments is platform-specific and may not always be exportable. Thus, it is not surprising that most organizations do not use third party experimentation platforms, as the survey [9] among practitioners indicates. However, the development of a self-built experimentation platform is not feasible for every organization. The high upfront cost of time and resources to develop a reliable experimentation platform [16] are not manageable for every organization.

The proposed experimentation platform-independent architecture mitigates the impact of a platform on the experimentation lifecycle. Despite the use of a third-party experimentation platform, the organization can define and adjust its experimentation lifecycle. Moreover, the migration to another experimentation platform becomes feasible as discussed in the experimental scenario of a migration.

Limitations. Even though possible threats to validity were considered during the design and execution of the study, the findings of this experiment have to be interpreted within their limitations. The main limitation of the study is the evaluation of the proposed architecture. Although the technical feasibility was evaluated by a proof-of-concept implementation, the organizational feasibility of the approach cannot be demonstrated with this method. Thus, the evaluation does not show whether the approach would also be feasible to be followed by a team. However, the construction of the architecture that is based on the requirements on the experiment definition is expected to have guided the development of the architecture to be also organizational feasible. The second point of evaluation was the platform-independence. This was evaluated by the discussion of the impacts of a migration to another experimentation platform to stress the dependency of the architecture to the experimentation platform. Even though the evaluation was only done by the discussion of the theoretical implications, the impacts are arguable sufficiently predictable on the architecture to use this evaluation technique.

8 Conclusions

Organizations that use third-party experimentation platforms are in the risk of a vendor lock-in. The implicit experimentation lifecycle enforced by the platform and the predefined definition of an experiment requires the organization to adapt its experimentation process to the platform. To mitigate this risk, an experimentation platform-independent architecture is proposed.

The proposed architecture separates the experiment definition from the experimentation platform. Therefore, the qualities and roles of experiment definitions were studied to develop an architecture that separates the definition from the platform without mitigating a role or a quality of the definition. The conducted evaluation suggest that the architecture is feasible and mitigates the impact of the experimentation platform on experimentation.

Interesting future research directions are the conduction of a case study to observe the architecture in an industrial setting. This could further improve the evaluation of the architecture and show the benefits as well as disadvantages of the approach.

References

1. Auer, F., Felderer, M.: Current state of research on continuous experimentation: a systematic mapping study. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE (2018)
2. Auer, F., Felderer, M.: Shifting quality assurance of machine learning algorithms to live systems. In: Software Engineering und Software Management 2018 (2018)
3. Auer, F., Felderer, M.: Characteristics of an online controlled experiment: preliminary results of a literature review. arXiv preprint arXiv:1912.01383 (2019)
4. Auer, F., Felderer, M.: Evaluating the usefulness and ease of use of an experimentation definition language. In: 2020 32th International Conference on Software Engineering and Knowledge Engineering. KSI Research Inc. and Knowledge Systems Institute Graduate School (2020)
5. Auer, F., Lee, C.S., Felderer, M.: Continuous experiment definition characteristics. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE (2020)
6. Buchert, T., Ruiz, C., Nussbaum, L., Richard, O.: A survey of general-purpose experiment management tools for distributed systems. Fut. Gener. Comput. Syst. **45**, 1–12 (2015). https://doi.org/10.1016/j.future.2014.10.007
7. Fabijan, A., Dmitriev, P., Olsson, H.H., Bosch, J.: The evolution of continuous experimentation in software product development: from data to a data-driven organization at scale. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE (2017). https://doi.org/10.1109/icse.2017.76
8. Fabijan, A., Dmitriev, P., Olsson, H.H., Bosch, J.: The online controlled experiment lifecycle. IEEE Softw. **37**, 60–67 (2018)
9. Fabijan, A., Dmitriev, P., Olsson, H.H., Bosch, J.: Online controlled experimentation at scale: an empirical survey on the current state of a/b testing. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 68–72. IEEE (2018)
10. Fabijan, A., Dmitriev, P., Olsson, H.H., Bosch, J., Vermeer, L., Lewis, D.: Three key checklists and remedies for trustworthy analysis of online controlled experiments at scale. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 1–10. IEEE (2019)

11. Fabijan, A., Dmitriev, P., Olsson, H.H., Bosch, J., Vermeer, L., Lewis, D.: Three key checklists and remedies for trustworthy analysis of online controlled experiments at scale. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE (2019). https://doi.org/10.1109/icse-seip.2019.00009
12. Fagerholm, F., Guinea, A.S., Mäenpää, H., Münch, J.: Building blocks for continuous experimentation. In: Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, pp. 26–35 (2014)
13. Fagerholm, F., Guinea, A.S., Mäenpää, H., Münch, J.: The right model for continuous experimentation. J. Syst. Softw. **123**, 292–305 (2017)
14. Gupta, S., Ulanova, L., Bhardwaj, S., Dmitriev, P., Raff, P., Fabijan, A.: The anatomy of a large-scale experimentation platform. In: 2018 IEEE International Conference on Software Architecture (ICSA), pp. 1–109. IEEE (2018)
15. Kevic, K., Murphy, B., Williams, L., Beckmann, J.: Characterizing experimentation in continuous deployment: a case study on Bing. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), pp. 123–132. IEEE (2017)
16. Kohavi, R., Deng, A., Frasca, B., Longbotham, R., Walker, T., Xu, Y.: Trustworthy online controlled experiments: five puzzling outcomes explained. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 786–794 (2012)
17. Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. Data Min. Knowl. Discov. **18**(1), 140–181 (2008). https://doi.org/10.1007/s10618-008-0114-1
18. Mattos, D.I., Bosch, J., Holmström Olsson, H.: More for less: automated experimentation in software-intensive systems. In: Felderer, M., Méndez Fernández, D., Turhan, B., Kalinowski, M., Sarro, F., Winkler, D. (eds.) PROFES 2017. LNCS, vol. 10611, pp. 146–161. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69926-4_12
19. Issa Mattos, D., Dmitriev, P., Fabijan, A., Bosch, J., Holmström Olsson, H.: An activity and metric model for online controlled experiments. In: Kuhrmann, M., et al. (eds.) PROFES 2018. LNCS, vol. 11271, pp. 182–198. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03673-7_14
20. Olsson, H.H., Bosch, J.: From opinions to data-driven software R&D: a multi-case study on how to close the open loop problem. In: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 9–16. IEEE (August 2014). https://doi.org/10.1109/seaa.2014.75
21. Ros, R., Runeson, P.: Continuous experimentation and A/B testing: a mapping study. In: Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering (RCoSE), pp. 35–41. ACM (2018). https://doi.org/10.1145/3194760.3194766
22. Tamburrelli, G., Margara, A.: Towards automated a/b testing. In: Le Goues, C., Yoo, S. (eds.) SSBSE 2014. LNCS, vol. 8636, pp. 184–198. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09940-8_13
23. Tang, D., Agarwal, A., O'Brien, D., Meyer, M.: Overlapping experiment infrastructure: more, better, faster experimentation. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 17–26 (2010)