undulate: Extreme continuous experimentation of separately deployable software

Timo Asikainen[a,*], Tomi Männistö[a], Eetu Huovila[a]

[a] *University of Helsinki, Department of Computer Science,*

Abstract

We introduce the undulate software for continuous experimentation in software engineering. The software covers all main phases of the experimentation process, from experiment design to the analysis of results. Further, the experiment results may be used automatically as a basis for further experimentation. The software is implemented using a microservice architecture, thus allowing parts of the software to be replaced with company-specific software when necessary. Also, the software under experimentation is assumed to follow a microservice architecture and run on a cluster.

Keywords: continuous experimentation, microservice architecture, data-driven software engineering, service mesh

1. Motivation and signifigance

During recent years, various continuous practices have become a significant trend both in the practice of and research on software engineering [1]. Simultaneously, *experimentation* has gained popularity as a way of acquiring knowledge from real users [2, 3]. The continuous principles have been applied to experimentation as well, leading to *continuous experimentation* [4, 5, 6]. Continuous experimentation has been studied from various points of view, including conceptual [2, 5, 7], adoption [8], deployment [7] and metrics [9, 10].

In addition to theoretical insights, reports on controlled experiments in the industry from various points of view have been published, see, e.g. [11**?** , 12]. However, despite the significant interest towards continuous, controlled experiments, there is a significant lack of open-source software covering all the

*Corresponding author.

Email addresses: timo.o.asikainen@helsinki.fi (Timo Asikainen), tomi.mannisto@helsinki.fi (Tomi Männistö), eetu.huovila@helsinki.fi (Eetu Huovila)

tasks required in continuous experimentation. Towards this end, we publish the undulate software developed as a part of our research project.

2. Software description

2.1. Software architecture

In this section, we describe the architectural components of the Undulate software package and dependencies on external software.

2.1.1. Systematic knowledge representation and process execution using nivel2

We employ components NIVEL[2] software as a part of the UNDULATE software. More specifically, experimentation data on various levels of abstraction is represented using NIVEL[2]: starting from defining the very notion of an experiment.

In addition to representing experimentation knowledge as NIVEL[2] entities, the nivel2 software is used to control the experimentation process. The observer component and running NIVEL[2] functions are used for this purpose.

2.1.2. Build pipeline – builder

The builder component is implemented in Python and provides an API for building Docker[1] images. The Flask application framework is used to enable access to the implemented services. In the production setting, the Flask application is run using the Nginx web server[2]. In practical terms, the services can be accessed through the HTTP (Hyper-Text Transfer Protocol) using a number of endpoints. To be able to build Docker images, the builder image uses the Docker software on the host machine, i.e. the machine running a container created from the builder image, through a socket. Naturally, Docker must be installed on the host machine in order for this to work.

The builder component connects to two external systems. First, the component retrieves the source to be built from a *code repository* using `git clone`. Second, the component pushes the Docker image it has built to a *Docker image repository* using `docker push`.

2.1.3. Automated management of containerised applications – Kubernetes

The UNDULATE approach is based on the ability to deploy components (or services provided by them) independently from each other and simply by running simple commands from the command line or by invoking the relevant process otherwise, e.g. through program code. The technological means to

[1] See `https://www.docker.com/`

[2] See `https://www.nginx.com/`

carry out these tasks have become widely known and generally available. For our purposes, we have selected the Kubernetes[3] product, provided through the Microsoft Azure cloud[4]. In the context of Kubernetes, the pool of resources (virtual machines etc.) is termed *cluster*.

2.1.4. Service mesh – Istio

In addition, being able to deploy software to the cluster is not sufficient – one must also be able to route requests to different destinations based on the content of the request, e.g. a cookie i.e. a kind of HTTP header. In general terms, a *service mesh* provides such functionality. Towards this end, we employ the Istio service mesh[5]

2.1.5. Deploying software and routing requests – cluster_control

The cluster_control component provides the functionality of deploying components (Docker images) to the cluster and working with the service mesh. The component is implemented in Python, with Flask and nginx used to provide an API similarly as for builder, see Section 2.1.2.

The cluster_control component operates on the cluster using the `kubectl` command line component. Files that can be directly applied to the cluster are passed through the API.

2.1.6. Data visualisation – undulate-app

For interactive applications for data analysis and visualisation, the R statistical software[6] and its Shiny package[7] are used. Compared with commercial business intelligence tools, the tools have the benefits of being free, easily programmable and can be readily containerised to accommodate the architectural style selected for undulate.

In the scope of the current research project, no holistic software for analysis and visualisation could be built. However, the prototype, undulate-app that was built, serves the purpose of incorporating concepts relevant to experimentation. Further, the implementation is enough to demonstrate that tools showing how data from experiments is accumulated in real-time can be built with moderate effort.

[3] See `https://kubernetes.io/`
[4] See `https://azure.microsoft.com/en-gb/`
[5] See `https://istio.io/`
[6] See `https://www.r-project.org/`
[7] See `https://shiny.rstudio.com/`

2.2. Software functionalities

In this section, we describe the main functionalities of the undulate software.

2.2.1. Representing experimentation knowledge

Software experimentation knowledge is represented using NIVEL^2 at all levels of abstraction necessary for designing and running experiments. In short, the level of abstraction are:

1. Basic concepts related to an experiment, such as Experiment and Group (either control or test)
2. Instances of the basic concepts that represent the design of an experiment, e.g. testing different link colours (ColourExperiment), where the groups would be identified by a different colour each
3. Execution data of the experiment, e.g. runs of experiment design with concrete start times and users assigned to the groups

We believe that representing experimentation data at all levels of abstraction using a multi-level modelling language such as NIVEL^2 adds rigour to work and thus makes the work more systematic. Once the basic concepts are defined, the nivel2 software makes it easier to adhere to the predefined structure of concepts. Also, the nivel2 interface can be used to navigate from experiment results (Level 3) to their design (Level 2). Without uniform data representation, data at Level 1 and possibly even Level 2 would be represented using a programming language or, alternatively, using configuration files without consistency checks. Therefore, we believe that resorting to multi-level modelling reduces the amount of coding required and makes the process, in this case, experimentation in software engineering, more accessible.

2.2.2. Implementation of the experimentation process

In the UNDULATE framework, experiments are seen more broadly as *experiment programs*, where new experiment groups can be added based on a predefined plan or based on the results from previously completed experiment groups. The notion of experiment programs enables using experimentation, e.g. to search for optimal settings in a search space.

In a research setting, in parallel with modifying the software that is running, also the user simulation must be modified in order to be able to see the effect of various experiment groups run. Towards this end, the simulient configuration is modified in accordance with the experiment program: in particular, user groups in the simulation are added and removed corresponding to the experiment groups being added and removed, respectively.

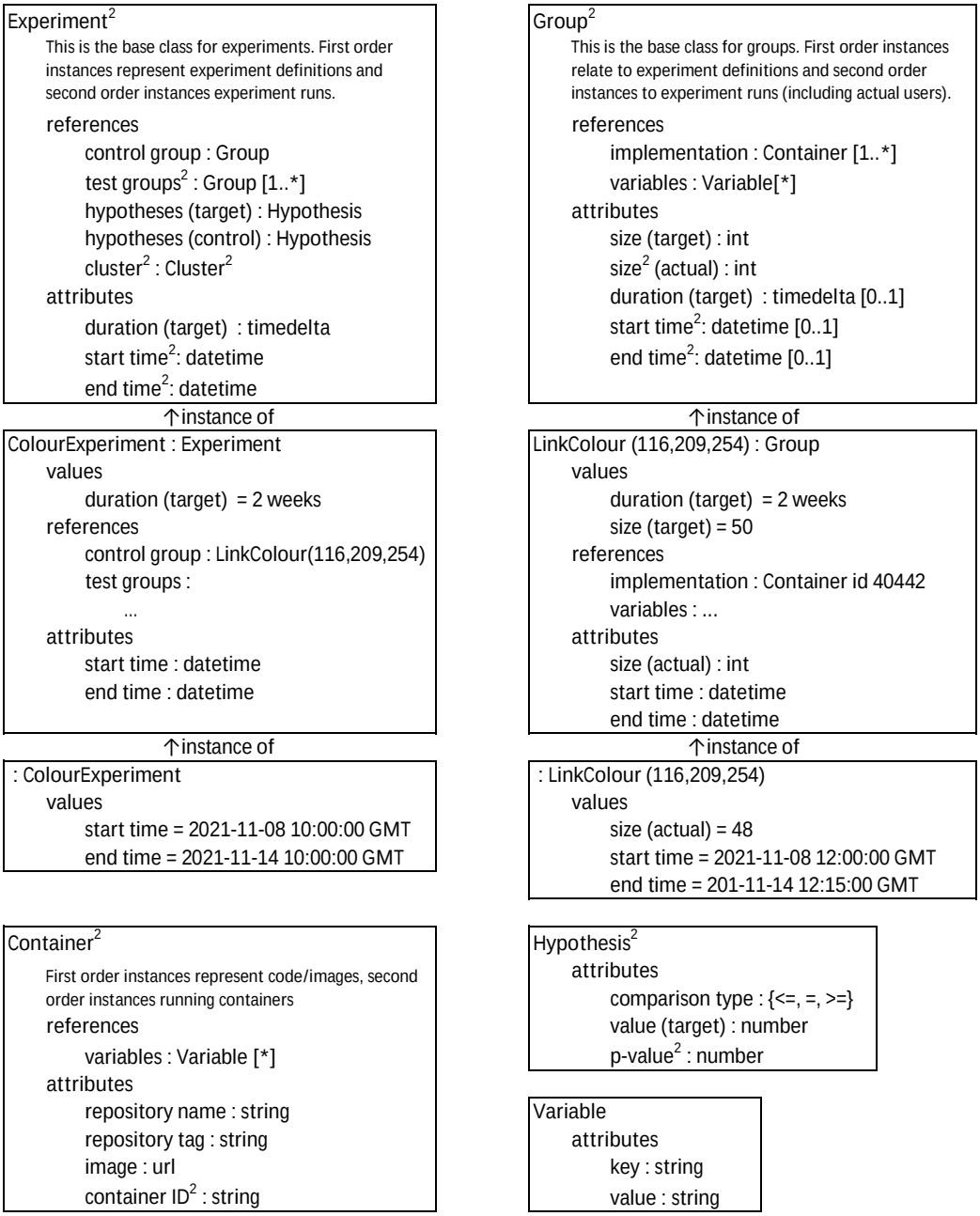

Figure 1: Concepts related to experiments and sample instances represented in NIVEL²

With the notion of experiment programs, running the experimentation process can be divided into two parts:

1. Modifying the NIVEL^2 data for the experiment
   (a) adding new groups and marking previous groups as completed
   (b) updating the simulient configuration
2. Updating the cluster to reflect the modified state of the experiment

When an automated experiment is running, the status of experiments is checked periodically (every n minutes). The check is done by the observer component, and the functionality to be run periodically is defined in the observation target obs_target_experiments. Point 1. above forms the payload of the function run *for each experiment* found by the observer under the target obs_target_experiments, and the implementation of the function related to the target is in procedure `xcese.sp_step_experiment_round`.

The implementation for Point 2. above is based on the two children of the target OBS_TARGET_EXPERIMENTS. The payload of the first child is to update for each experiment entity the value containing the active routings for each service related to the experiment: the routings contain the list of active groups in the experiment. The updated value is thereafter used when running the payload NIVEL^2 function of the latter child, where the following steps are executed for each service:

I The experiment group-specific images that are referred to in the current routings are built using builder
II A JSON that contains the *deployment* of the images built in Point I is generated using converter
III The JSON for the images generated in the previous point (Point II) is deployed to the cluster using cluster_control
IV A JSON that contains the *destination rule specification* of the service in Point I is generated using converter
V The JSON for the destination rule generated in the previous point (Point IV) is deployed to the cluster using cluster_control
VI A JSON that contains the *virtual service specification* of the service in Point I is generated using converter
VII The JSON for the destination rule generated in the previous point (Point VI) is deployed to the cluster using cluster_control

Above, destination rule and virtual service are service mesh constructs that are used to direct requests from an experiment group to the intended destination, i.e. a container with the implementation specific to the experiment group.

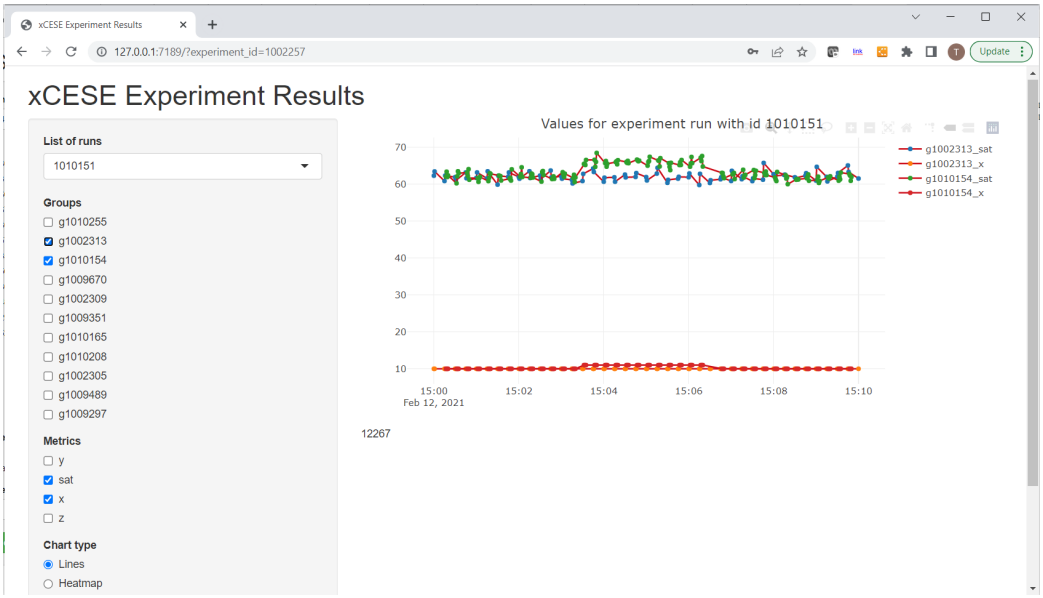

Figure 2: An example of experimentation results visualised using the undulate-app implemented in Shiny. The graph at the centre shows how the satisfaction of users in an experimental group (g1010154) has improved in comparison with the control group (g1002313) while the experiment took place, i.e. the value of the experiment variable x was changed from 10 to 12 as specified in the experiment.

Note that the automated experiment logic is encapsulated in the T-SQL procedure (`xcese.sp_step_experiment_round`) mentioned above. Whatever changes to the experiment design are made, they are applied to the cluster in the process of the second stage (Roman numerals). Although the experiment logic is currently embedded in the T-SQL code, the logic could be specified in other means, e.g. a strategy represented as a NIVEL[2] entity with parameters (attributes and values) that is merely interpreted in the procedure.

2.2.3. Visualising experiment results – undulate-app

Even if organised appropriately in a database, experiment results are not useful without the possibility of seeing the results visually. An example of the visualisation of the results is shown in Figure 2

2.2.4. Simulating user behaviour – simulient

In a research setting, recruiting users for numerous experiments is not feasible. Therefore, the only viable option is to simulate user behaviour using the software. Towards this end, we have implemented the simulient component in Python, with an API for running and otherwise controlling simulations implemented using.

The simulient component operates on a configuration defined in NIVEL[2], using a prespecified base type. A configuration includes a number of *user groups* that may differ from each other in terms of their behaviour, i.e. parameters. Further, a prespecified number of *users* may be instantiated from each user group. Each user has a *state*, defined in terms of *variables*: the initial distribution of variables for each state is defined for a user based on the group. During a simulation, the state develops based on the previous state and the outcomes of the simulation steps.

In each simulation step, an *action* is determined for each user based on its current state and predefined probabilities for each action. The action may also be a call to an endpoint provided by a service: this makes it possible to interact with the system under experimentation. Based on the action outcome (result) and current state, the user transitions to a new state according to *state transition formulas* specified in the group of the user.

Simulient reports all its activity to the log_data_api component at three levels of abstraction: the simulient run, the simulient step (e.g. old and new state, the action taken, the result of the action) and the simulation details (e.g. variable values).

3. Impact overview

The software described in this paper has been developed as a part of the xCESE (eXtreme Continuous Experimentation in Software Engineering) research project. The conceptual and theoretical aspects of UNDULATE have been reported in our previous paper [13]. No external partners or real-world software have been involved in the research project, and hence the software has not been applied in practical settings.

On the other hand, few, if any, complete frameworks for software experimentation have been released. The empirical work that has been reported stems from large-scale companies, such as Facebook and Microsoft, that have had the resources available for setting up the infrastructure required by experimentation. However, the situation is completely different in most software companies. We believe that releasing the Undulate software to the public will enable even medium or small software enterprises to run experiments systematically and thus further continuous experimentation practices in the industry.

4. Acknowledgements

The work was supported by the Academy of Finland (project 317657).

References

[1] B. Fitzgerald, K. J. Stol, Continuous software engineering: A roadmap and agenda, Journal of Systems and Software 123 (2017) 176–189. doi:10.1016/j.jss.2015.06.063.
URL http://dx.doi.org/10.1016/j.jss.2015.06.063

[2] H. Holmström Olsson, J. Bosch, Towards Continuous Customer Validation: A Conceptual Model for Combining Qualitative Customer Feedback with Quantitative Customer Observation, in: J. M. Fernandes, R. J. Machado, K. Wnuk (Eds.), Software Business, Springer International Publishing, Cham, 2015, pp. 154–166.

[3] D. I. Mattos, J. Bosch, H. Holmström Olsson, Your system gets better every day you use it: Towards automated continuous experimentation, in: Proceedings - 43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017, 2017, pp. 256–265. doi:10.1109/SEAA.2017.15.

[4] F. Fagerholm, A. Sanchez Guinea, H. Mäenpää, J. Münch, Building blocks for continuous experimentation, 1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014 - Proceedings (2014) 26–35doi:10.1145/2593812.2593816.

[5] F. Auer, C. S. Lee, M. Felderer, Continuous Experiment Definition Characteristics, Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020 (2020) 186–190doi:10.1109/SEAA51224.2020.00041.

[6] G. Schermann, J. Cito, P. Leitner, Continuous experimentation: Challenges, implementation techniques, and current research, IEEE Software 35 (2) (2018) 26–31. doi:10.1109/MS.2018.111094748.

[7] G. Schermann, D. Schöni, P. Leitner, H. C. Gall, Bifrost - Supporting continuous deployment with automated enactment of multi-phase live testing strategies, in: Proceedings of the 17th International Middleware Conference (Middleware'16), ACM, Trento, Italy, 2016, pp. 1–14. doi:10.1145/2988336.2988348.

[8] S. G. Yaman, M. Munezero, J. Münch, F. Fagerholm, O. Syd, M. Aaltola, C. Palmu, T. Männistö, Introducing continuous experimentation in large software-intensive product and service organisations, Journal of Systems and Software 133 (2017) 195–211. doi:10.1016/j.jss.2017.

07.009.
URL http://dx.doi.org/10.1016/j.jss.2017.07.009

[9] D. Issa Mattos, P. Dmitriev, A. Fabijan, J. Bosch, H. Holmström Olsson, An Activity and Metric Model for Online Controlled Experiments, in: M. Kuhrmann, K. Schneider, D. Pfahl, S. Amasaki, M. Ciolkowski, R. Hebig, P. Tell, J. Klünder, S. Küpper (Eds.), Product-Focused Software Process Improvement, Springer International Publishing, Cham, 2018, pp. 182–198.

[10] P. Dmitriev, S. Gupta, D. W. Kim, G. Vaz, A Dirty Dozen: Twelve common metric interpretation pitfalls in online controlled experiments, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Vol. Part F1296, 2017, pp. 1427–1436. doi:10.1145/3097983.3098024.

[11] A. Deng, X. Shi, Data-driven metric development for online controlled experiments: Seven lessons learned, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Vol. 13-17-Augu, 2016, pp. 77–86. doi:10.1145/2939672.2939700.

[12] R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, Y. Xu, Trustworthy Online Controlled Experiments : Five Puzzling Outcomes Explained (2012).

[13] T. Asikainen, T. Männistö, Undulate: A framework for data-driven software engineering enabling soft computing, Information and Software Technology (2022) 107039doi:10.1016/j.infsof.2022.107039.
URL https://doi.org/10.1016/j.infsof.2022.107039

Nr.	Code metadata description	Please fill in this column
C1	Current code version	For example v42
C2	Permanent link to code/repository used for this code version	For example: `https://version.helsinki.fi/xcese_public/undulate`
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	MIT, Istio included in the repository with Apache licence
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python (nginx), Microsoft SQL Server and Transact-SQL, R (Shiny)
C7	Compilation requirements, operating environments & dependencies	Docker, Kubernetes, Istio, Nivel2
C8	If available Link to developer documentation/manual	
C9	Support email for questions	timo.o.asikainen@helsinki.fi

Table 1: Code metadata (mandatory)