Your System Gets Better Every Day You Use It: Towards Automated Continuous Experimentation

Conference Paper · August 2017

DOI: 10.1109/SEAA.2017.15

3 authors:

David Issa Mattos
Chalmers University of Technology
6 PUBLICATIONS **10** CITATIONS
SEE PROFILE

Jan Bosch
Chalmers University of Technology
426 PUBLICATIONS **10,982** CITATIONS
SEE PROFILE

Helena Holmstrom Olsson
Malmö University
85 PUBLICATIONS **1,466** CITATIONS
SEE PROFILE

Your System Gets Better Every Day You Use It: Towards Automated Continuous Experimentation

David Issa Mattos
Chalmers University of Technology
Gothenburg, Sweden.
davidis@chalmers.se

Jan Bosch
Chalmers University of Technology
Gothenburg, Sweden.
jan@janbosch.se

Helena Holmström Olsson
Malmö University
Malmö, Sweden.
helena.holmstrom.olsson@mah.se

***Abstract*—Innovation and optimization in software systems can occur from pre-development to post-deployment stages. Companies are increasingly reporting the use of experiments with customers in their systems in the post-deployment stage. Experiments with customers and users are can lead to a significant learning and return-on-investment. Experiments are used for both validation of manual hypothesis testing and feature optimization, linked to business goals. Automated experimentation refers to having the system controlling and running the experiments, opposed to having the R&D organization in control. Currently, there are no systematic approaches that combine manual hypothesis validation and optimization in automated experiments. This paper presents concepts related to automated experimentation, as controlled experiments, machine learning and software architectures for adaptation. However, this paper focuses on how architectural aspects that can contribute to support automated experimentation. A case study using an autonomous system is used to demonstrate the developed initial architecture framework. The contributions of this paper are threefold. First, it identifies software architecture qualities to support automated experimentation. Second, it develops an initial architecture framework that supports automated experiments and validates the framework with an autonomous mobile robot. Third, it identifies key research challenges that need to be addressed to support further development of automated experimentation.**

***Index Terms*—Automated experimentation, Software architecture, Autonomous systems, Controlled Experiments**

I. Introduction

Autonomous systems already play a big part in several areas, from financial markets, industrial robots, airplane navigation systems to the development of autonomous vehicles. These systems are deployed in uncertain environments and contexts with an ever-increasing demand to work more autonomously. As these systems become more and more complex, it is not clear how the developed features could be contributing to the system and whether they are delivering the expected value [1].

Previous studies show that in the development of systems, traditionally, the prioritization of a feature is usually driven by earlier experiences and beliefs of the people involved in the selection process [2]. The development of the new features should be, ideally, data-driven and done systematically [3]. The development of a full feature from conception to user deployment can result in inefficiency and opportunity cost if it does not have a confirmed value for the customer. Early customer feedback is important to determine and validate the feature value [4]. If a gap between the expected and the actual value of a feature is identified, the feature under development can be refined or abandoned. Additionally, companies continuously collect data from deployed software [5], [6]. Although not widely used in industry, post-deployment data can be used for both continuous optimization and improvements and to drive innovation in features of the existing products [6]. Testing and experimenting with customers and users are used as problem-solving processes and are critical to organizational innovation [7].

Frequently, complex systems and user behavior in development of software-intensive systems lead to opinion-based decisions captured in traditional requirements [8]. This mismatch between user behavior and opinion-based requirements is leading industry to change from requirements-based to experiment-based development [7].

Systematic experiments provide a level of understanding about what really works and leaving opinion-based decisions behind [9]. Web-facing companies report the use of experiments with customers and that these experiments can lead to a significant learning and return-on-investment [8]. Several companies report the use of A/B experiments for confirming feature value through hypothesis testing and for feature optimization [8], [10] (in short A/B experiment consists of comparing a variant A, the control, against a variant B, the treatment). In [7] online experiments are identified as a technique to drive innovation during development and post-deployment of a system.

However, traditional manual controlled experiments can be an expensive way to optimize a system. Validation runs online and can last for several months [8], the metrics used by different teams can lead to conflicting business goals, and it can be hard to reason on the system when dealing with hundreds of different observable variables [11]. In this scenario machine learning and artificial intelligence techniques can aid the research and development team to run optimization procedures to existing features more efficiently.

In customization and recommender systems there is an increase use of machine learning (ML) and data-driven approaches. Techniques such as deep learning, reinforce-

ment learning (RL), deep reinforcement learning, multi-armed bandits are getting large attention as companies (Google, IBM, Microsoft, Spotify, Facebook among others) successfully report the use of those techniques in their systems. Machine learning communities continuously report solutions to a vast range of difficult problems. However, ML also raises several software engineering problems, such as testing, system validation, supporting infrasctructure and increase of technical debt in the system [12]. Many ML algorithms have mathematical proofs but with the ever-changing landscape that those algorithms interact, system validation becomes a hard problem. Controlled experiments are used to evaluate both the ML value and system behavior [13].

As companies start to build autonomous systems with an increasing level of autonomy, experiments are seen as scientific way to learn and adapt the system behavior. The uncertainty raised by the environment, the interaction with humans and other agents all impact in the system behavior in unknown ways. It is unfeasible to grow the size of the R&D team with the increasing demand for experiments . This calls for the use of automated experiments, where the R&D team can build part of the functionality and guardrails where the system can experiment and learn from this process continuously and autonomously [1].

As companies are moving towards experiment-based development, they face several challenges such as: experiments in live field are time and cost expensive, experiments from different teams can lead to conflicting goals, and the lack of a systematic approach to run experiments in different domains leads to several experiment platforms. [1], [8], [11], [14]. To address some of these challenges the communities studying experiments in software can be viewed as moving towards automated experiments. Important research deals with automating data collection and data analysis [11], facilitating the deployment of new experiments in web systems [15], dealing with overlapping experiments [14], leveraging experiments with log data [16] and automated experiments from a algorithm perspective [17].

Automating tasks and allowing systems to perform adaptation online play a role part in the development of autonomous systems. Autonomous and adaptive systems are able to automatically adjust their behavior at runtime in response to changes in the operating environment [18]. Over the past fifteen years, different domains such as the self-adaptive systems community studied and developed several software engineering approaches and techniques for adaptation. While the ML is mostly concerned with algorithms and the theoretical basis of learning, these communities study the software engineering challenges.

Running continuously automated experiments in the already deployed systems, with both the aim of improving the system behavior and confirming the delivered value, requires novel software architectures and engineering approches [1]. Techniques from these different perspectives can serve as a basis for bridging from manual hypothesis experimentation and experimentation for optimization to automated experiments.

Although we recognize the important role ML and other artificial intelligence techniques have in the development of automated experiments, this work focuses on architectural aspects to support different techniques in automated experiments for both validation as well as optimization of features. Continuous optimization through automated experiments leads to systems that get better every day we use them.

The contribution of this paper is threefold. First, it identifies a set of software architecture qualities to support automated experimentation and analyzes several architectures from these qualities perspectives. Second, a framework to support automated experiments is developed based on the presented analysis. This framework presents a novel way towards automated experimentation, where manual hypothesis experimentation and fully automated experiments in optimization can be combined. This framework is validated in the context of an autonomous system in a human-robot interaction problem. Third, it identifies the key research challenges that need to be addressed to support the development of automated experiments.

The rest of the paper is organized as follows. Section II provides a background on some of the related areas, controlled experiments, machine learning applications for experiments and software architecture for adaptation of systems. Section III describes the research method. Section IV provides a list of desired architecture qualities to support automated experimentation. Section V analyzes several software architectures in light of the discussed qualities. Section VI shows an initial architecture framework that satisfies the discussed qualities for automated experimentation. Section VII provides an implemented version of the architecture framework in an autonomous mobile robot on the proxemics distance problem. Section VIII concludes and discusses future research challenges and future works.

II. Background

This section reviews some concepts on controlled experiments, reinforcement learning in experiments and software architectures for adaptation. These concepts are complementary to each other and form the basis of this work.

A. Controlled experiments

In controlled experiments, researchers manipulate independent variables in an experiment and observe the effect on the behavior by measuring the dependent variable.

In software development, experiments with customers can lead to a significant learning and return-on-investment [8]. The web provides an opportunity to test and evaluate development ideas using controlled experiments. Kohavi [8] provides an overview on A/B experiments controlled, techniques, study cases in a web environment and common

pitfalls. Controlled experiments provide a reliable experimental setup for understanding the system, however it can be an expensive method to evaluate new ideas [19].

When the sample cannot be considered to be independent (the case of some experiments social networks), networked A/B testing techniques [20] are used. When running several experiments at the same time, overlapping of experiments and confounding factors can influence the results. In [14], some strategies such as dividing the experiments in layers are discussed.

B. Reinforcement Learning in experiments

Reinforcement learning can be seen as a technique for solving sequential decision making problems [21] and largely developed by Sutton and Barto [22]. The method is based on agent that repeatedly interacts with the environment and is receiving feedback from it. The perception of the each state allows the system to continuously improve its state into a more optimal selection of actions. It is modeled as a Markov decision process. Reinforcement learning focuses on online improvement and presents the trade-off between exploration and exploitation. The exploration/exploitation problem consist of finding a balance on exploring different solutions to achieve an optimal solution and exploiting the best solution.

The explore/exploit trade-off is well studied with the multi-armed bandit problem. This problem consist of deciding which arm of a K-slot machine would maximize the total reward in a limited series of trials. Several algorithms for this problem were proposed and evaluated [23]. Experiments with a finite number of treatments can be formulated as bandit problems and A/B testing is mathematically equivalent to the ϵ-first strategy with equal division between the arms (ϵ-first strategy explore all arms choices before it starts to exploit). Complex strategies can take into account context information (contextual multi-armed bandits). Contextual bandits are wideley used in personalized/recommender systems [24], ad placement [16] and in search engines applications [25].

However, bandit problems differ from controlled experiments in the conceptual level. Controlled experiments focus on hypothesis testing and obtaining an understanding of the system with a statistical confidence level. Bandit problems focus on optimization and do not balance the experiments to accurately estimate the inferior treatment effects [26] .

C. Software architectures for adaptation

Adaptation refers to the ability of the system to automatically adjust their own behavior in response to changes in the operating environment [27]. Adaptation is seen as a key enabler for the development of autonomous systems and autonomic computing.

In 2003 Kephart, introduced the MAPE-K model with the IBM Autonomic Computing Initiative [28]. The MAPE-K model is a feedback loop and it stands for Monitoring, Analyzing, Planning and Executing over a Knowledge base. The MAPE-K loop forms the basis of self-adaptive systems feedback loops and has become the reference model in self-adaptation. Self-adaptive systems can be seem as an umbrella term to cover different areas involved in adaptation and are studied from different perspectives: software architecture, requirements engineering, middleware, component-based development, control systems theory among others [29]. These different perspectives solve their specific domain problems with different architecture configurations. However, most self-adaptive systems approaches can be mapped into one or more MAPE-K loop parts [29].

III. Research Method

The research method in this paper was conducted in three phases.

Phase I: In the first phase, we reviewed literature to identify relevant software architectures for experimentation and adaptation and how these architectures solve domain-specific problems.

For the literature search, we looked for software architectures in the different domains and applications related to experimentation: `( TITLE-ABS-KEY ( "software architecture" ) OR TITLE-ABS-KEY ( "software framework" ) ) AND ( TITLE-ABS-KEY ( "experimentation" ) OR TITLE-ABS-KEY ( "A/B tests" ) OR TITLE-ABS-KEY ( "split tests" ) OR TITLE-ABS-KEY ( "self-adaptive systems" ) OR TITLE-ABS-KEY ( "controlled experiments" ) ) .` This query was used in the indexing libraries SCOPUS and Science Direct, that cover the large research libraries. As a result of the literature search, we identified 410 papers with relevance to this search. From these papers, we identified 178 based on reading the abstract and identifying the relevance in relation to our research topic. From this we selected 34 papers that describe relevant concepts and software architectures for experimentation, adaptation and controlled experiments. In addition to the literature search based on the identified key search terms, through cross-reference we identified a set of 18 additional papers with relevance for our research.

The analysis of the selected works provided us two outcomes: (1) the identification of relevant architectures for experimentation, adaptation and controlled experiments and (2) identification of software architecture qualities that can support automated experimentation. The qualities are described in detail in Section IV.

Phase II: Based on the outcomes of phase I, we scoped the second phase to understand how each identified architecture implements the software qualities. In this phase, we revisited the identified architectures to identify how they implement or solve a problem related to each of the qualities. Our analysis, described in Section V, indicated that none of the architectures could be used as is with the automated experimentation problem but they can be used

to derive a new framework that could support automated experimentation.

Phase III: Based on the analysis performed in phase II, phase III focused on developing an initial framework that could support automated experiments. An initial version of this framework was instantiated in a proxemics distance problem in human-robot interaction. This human-robot interaction problem is currently being studied using manual experiments and therefore is a good candidate to try the developed automated experimentation framework.

IV. Architecture qualities

This section discusses the identified architectural qualities that support the idea of automated experimentation.

The identified qualities are outcomes of phase I of the research method, described in Section III. In our research, we identified six qualities that would constitute our approach towards automated experimentation. However, this list of qualities is not static. Further research on the area might extend this list to include different qualities. Additionally, the identified qualities are seen as desired qualities rather than required qualities. Excluding one or more of the qualities might be needed to implement a domain specific setup for automated experiments. Therefore, this list is ordered in terms of importance.

1) External adaptation control: Adaptation can be divided in two types of adaptation control: internal and external [30]. Internal adaptation refers to interlace application logic with adaptation logic. External adaptation refers to the use of an adaptation manager that coordinates adaptation with the use of sensors and effectors in the managed system (application logic). The use of an external manager increases maintainability of the system. However, it introduces an overhead to the system, penalizing performance.

Traditional controlled experiments are analyzed offline. All the data is extracted, analyzed to support a decision. Depending on the decision the system is manually modified to incorporate this decision. Systems running multi-armed bandits and other ML algorithms are usually incorporated in the feature application code. The external adaptation control was identified as an important quality for automated experiments, because it allows separating the application logic from the experiment logic. This auxiliates both automated experiments in controlled settings and in optimization settings, by facilitating to add automated experiments to existing features and removing it from features that already reached the static system loop [1].

2) Data collection as an integral part of the architecture: Software-intensive systems can gather large amounts of data in real-time, from the context, from the internal system as well as from users of the systems. Collecting data for online and offline analysis by both the system and the R&D should be an integral part of the architecture. An emphasis is given in this step as the development of software should be data-driven [2].

As most adaptive systems are based on the MAPE-K reference framework, data collection is usually seen as part of the monitor phase on the MAPE-K loop. The managed system exposes several observable internal states through the sensors touchpoints [31]. Data is collected continuously and filtered and later analyzed by the Analyze component to detect changes and decide whether adaptation is needed or not.

Machine learning systems are easy to develop but hard to maintain [12] due to hidden technical debt. One of the causes is the high diversity data and unstable data dependencies that rises by the extensive use of *glue code*. Strategies to create common API's allow a more reusable infrastructure supporting integral data collection and the external adaptation.

3) Performance reflection: As part of the learning process, the system should have performance reflection as an important part of the architecture. Performance reflection consists of evaluating the current behavior according to an expected value function. The experimental methods also use the performance reflection to validate the observed behavior.

In the conceptual solution presented in [1], evidence-based engineering refers to validate the deployed feature based on experiments and the value it delivers. Also, it was identified three levels of evidence-driven development feedback loop: the R&D loop, the dynamic system loop and the static system loop. Performance reflection occurs in the dynamic system loop in which the system can compare its delivered value with the initially expected value.

If the system does not keep track of its adaptation performance it is not possible to have an evidence that the experimentation and the learning process is increasing value to the system.

4) Explicit representation of the learning component: Most of the current approaches in self-adaptive systems recognize the importance of learning, but few of them support an online learning process [29]. Most architectures use predefined and reactive adaptation plans.

The field of ML provides several algorithms and techniques tailored to domain problems. Experimenting features in different domains requires modification of the learning algorithm. An explicit representation of the learning component facilitates introducing and maintaining (mitigating entanglement effects) learning algorithms in the optimization process. We believe that an explicit representation of the learning component is necessary because it reinforces the learning characteristic of the systems and facilitate the reuse of the learning component and learning algorithms in other features.

5) Decentralized adaptation: Centralized control refers to using a single adaptation manager to control the adaptation. Decentralization refers to each adaptation where each sub-system has a full adaptation manager. The use of decentralized adaptation control improves the performance of the system, splitting responsibility between sub-systems.

Hybrid approaches might use different patterns, such as the IBM hierarchical structures to allow adaptation [31].

Different systems running automated experiments will require different levels of decentralization. Centralized systems can grow quickly in complexity handling several features under experimentation. Traditional A/B experiments relly on centralized coordination. However, as the infrastructure grows decentralized patterns start to emerge. Tang et al. [14] describe an hybrid approach, dividing the system in layers with different levels of overlapping crossing multiple domains. We believe that automated experiments will require decentralized solutions to support the increasing number of concurrent experiments.

6) Knowledge exchange: Knowledge exchange can help systems to share learned and experimented solutions [32]. Collaborative learning is an increasing topic of interest in ML and in experimentation. Systems instantiated or users experimenting might not be completely independent or randomized. Algorithms for solving this sort of problem are studied within networked A/B tests [11] and counterfactual reasoning [16]. As the feature being experimented might also be evaluated by the development team in terms of the value it gives, the sharing component should also allow communication with the development team. Knowledge exchange can be seen in the work by [33] and in the area of collaborative feedback and Collaborative Reinforcement Learning [32].

V. Architecture analysis

Due to page limits constraint, this section discusses briefly the analyzed architectures and how they can contribute to the development of automated experiments. We analyzed the existing architectures for adaptation and experimentation in relation to the identified qualities as stated in the research method.

Architectures from different domains were selected in order to minimize the bias in selecting only a few. However, this list does not aim to be complete in respect of all existing architectures for adaptative systems. A description of the different approaches can be found in [29]. Table I provides an overview of architectures analyzed in this work.

Table I
Analyzed architectures

Approach	Architecture
Architecture-based	Rainbow framework [34], 3-layered approach [35], Archstudio [36]
Reflection-based	DynamicTAO [37] CARISMA [38]
Control-based Architectures	Model Predictive Control, MIAC, MRCA, Gain scheduling, Cascaded control [39]
Service oriented	SASSY [40] MetaSelf [41] MOSES [42] MUSIC [43]
Agent-based	CRL [32], Unity [44], MOCAS [45]
Learning systems	FUSION [46], Controller-Observer [47]
Requirement Engineering methods	LoREM [48] Zanshin [49]

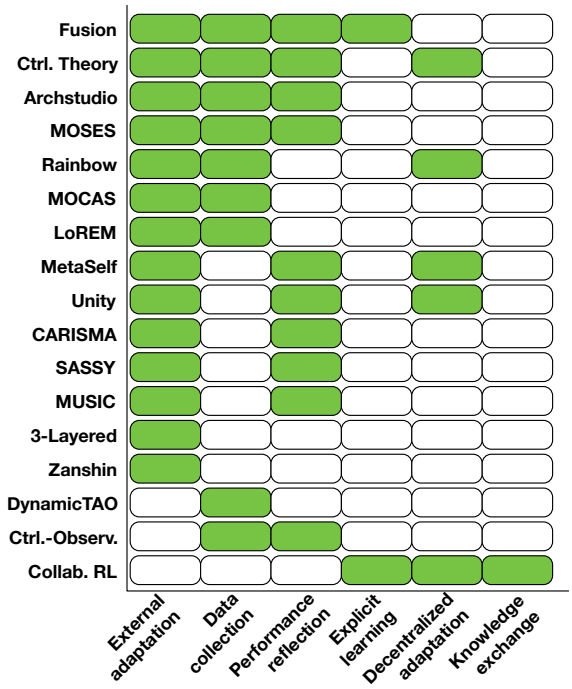

Figure 1. Summary of the classification of the architectures with respect to the architecture qualities. In green, are the qualities satisfied by the architecture. The architectures were ordered according to the relative importance of the quality.

The architectures were analyzed, classified and ordered according to the six qualities listed in section IV. Figure 1 shows the obtained classification of the analyzed architectures according to the listed qualities. Each of the six qualities is connected to the architectures that fulfill these qualities.

The summary table provides an overview of how the different architectures relate to the desired qualities. However, it is possible to see that most of the approaches deal with only a few of the identified qualities. The architecture that satisfies most of the qualities is the FUSION framework [46] and the developed architecture framework in Section VII is inspired by this framework.

The FUSION framework can be seem as a variation of the MAPE-K loop for architecture optimization. It uses a learning approach to drive adaptation of features. It is focused in optimization in the architectural level and it is independent of the learning algorithm. The FUSION framework uses a centralized external approach. The adaptation manager is divided into two cycles, the adaptation and the learning cycle. The learning process is based on measurements of the system. After the collection, the learning cycle identifies any emerging pattern and refine the induced model in a knowledge base. Then, the knowledge base is used in the

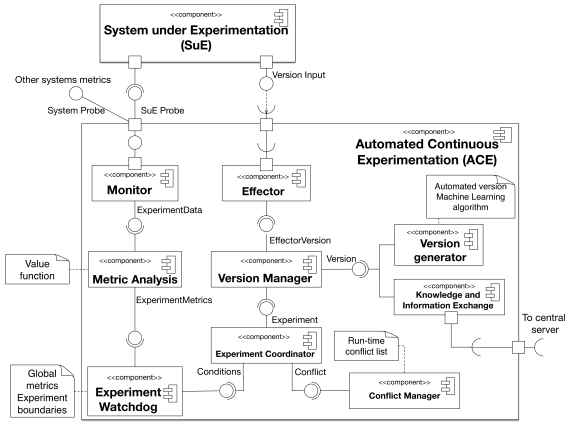

Figure 2. Automated Continuous Experimentation architecture framework

adaptation decision. Objective functions are used as metrics for the learning process and for the decision making. This framework brings together many of the identified qualities, except for the decentralized adaptation and the knowledge exchange. However, it focuses on optimizing from learning from existing features in the knowledge base in the architecture level. Automated experimentation is focused on learning and optimizing business goals to drive post-deployment innovation.

VI. Architecture framework

In this section, we describe an initial architecture framework that satisfies the identified qualities to support automated experimentation.

A. The architecture framework

The presented architecture framework is represented in Figure 2. This architecture is the result of the design decisions over several iteration processes and inspired by the analyzed architectures described in Section V.

The intention of this architecture framework is to provide for an existing system the capability of doing automated experiments decentralized in feature level, rather than providing an architecture for the whole system. Manual hypothesis testing is integrated as the hypothesis are still formulated by the R&D team.

The presented architecture modules are described next:

Monitor. This module is responsible for the data collection. The data collected come from the probes available in the system and in the SuE (system under experimentation), therefore both local and global behavior. This module is directly related to the data collection in the discussed qualities. Access to all the necessary data for experimenting requires proper instrumentation of the code. This module does not represent only a stream of raw data into the automated experimentation architecture framework. It represents data processed that add information to the system.

Effector. This module is responsible for interfacing with the managed system. Besides the monitor, it is the only point of contact with the rest of the SuE. This module requires that the managed system exposes interfaces for interaction with the system. This concept of not intermixing the experimentation code and the managed system code is directly related to the external adaptation quality. The same observations made to the monitor module are valid for the effector.

Experiment coordinator. This module is responsible for running the experiment and coordinating with the version manager. This module controls only the specific SuE, other experiments have their own experiment coordinator modules. The experiment coordinator can control experiments such as A/A (control variant A against the same variant for sanity checks), A/B/n (controlled experiment with more than one treatment), explore/exploitation and crossover experiments. This module keeps track on when to experiment, the number of experiments that should be run, which solution is more significant. This module receives inputs from the conflict-list manager if it is allowed to run an experiment or not. It also receives inputs from the experiment watchdog module, if the system is deteriorating any global metrics, if it went out of boundaries or if it still needs to perform more experiments.

Version Manager. This module is responsible for managing and generating different versions to experiment. This can be acting in parameters or replacing whole sub- component models. The version generator keeps a list of the versions used and accepts versions inputs from the Knowledge Exchange module and the Version Generator. This allows the system to experiment both automatically generated versions, as well as manual versions crafted by the R&D team. Although this module is not directly connected to a one of the design decision listed, this module is linked to both the functional and quality requirements

Version Generator. This module can accommodate different artificial intelligence algorithms that we might want to test. The generation algorithm is not specified, but it could include machine learning algorithms, such as reinforcement learning algorithms, genetic algorithms, parameter scheduling or randomized versions. This module is directly connected to the learning quality.

Experiment Watchdog. This module checks the conditions that the system can run the experiments, such as when the system should continue experimenting and when it should stop. If the system goes out the predefined boundaries or if there is deterioration in global metrics this module can stop the experiment and return the system to the "safe" version. Having a stop condition for global metrics prevents the system improving a local metric, but degrading a global metric. If any of the stop conditions is reached this module signals to the experiment coordinator to stop the experimentation process or to roll back to a safe version.

Conflict-list manager. This module keeps track in runtime of components that are being experimented with and which factors it affects. This is an important compo-

nent in a decentralized experiment environment. Several other systems of the robot can be experimenting. This manager keeps track of those systems in order to avoid confounding variables in the experiment. This is directly related to the decentralized adaptation quality. In a generic implementation, the conflict manager advertises its current state (experimenting or not) and listen to other conflict manager's states.

Metric Analysis. This module is responsible for keeping track of the managed system behavior and the value function. This module serves as a trigger to drive the adaptation through optimization or through keeping track of the validation process. In this module, we insert the value function and we run our statistical analysis. This module is directly related to the performance reflection quality.

Knowledge and Information Exchange. This module communicates with the optimization and experiment validation module and with the external world. This module is responsible for sharing discovered solutions in the experimentation process and also for sharing and learning the validated solutions from the experiment through a central server infrastructure. This also represents a way in which the R&D can interact with the system, either helping in the analysis step or proposing different versions not generating by the version manager, for example, testing different algorithms. This module is directly related to the Knowledge Exchange quality.

VII. Automated experiments in a human-robot interaction problem

In this section, we present an experimental implementation and evaluation of the automated continuous experimentation architecture framework. In this experimental scenario, we first validate the correctness of the architecture behavior. Second, our architecture for automated experimentation indicates a more cost-effective solution for running experiments compared to manual experimentation.

A. Proxemics distance in Human-Robot Interaction

Human interaction is based on several unwritten and subjective rules. One example is respecting other people's personal space. In human-human relations, several social factors play an important role in this interaction. Not conforming to these rules may cause miscommunication and discomfort. To have a good human-robot interaction, the robots must follow similar rules [50]. The large body of work in Human-Robot Interaction (HRI) identified several factors that come into play in proxemics distance, such as gender, age, personal preferences, technology involvement, crowdedness, the direction of approach, form factor and size [50]. Different works recognize some base distances and how they are influenced depending on a change of factor. However, this is still an open problem. The development of new robots and the deployment of these robots in very different contexts (e.g. different countries) require new manual experiments to validate and optimize the proxemics distance. The presented automated continuous experimentation framework can be used to allow the robot to try different distances and learn an optimal distance on the context that it is inserted. This would allow robots to adapt its proxemics distance and validate it without the need of designing costly experiments for each different context.

The developed architecture framework is instantiated in the open source mobile research platform Turtlebot 2[1]. This platform is similar in size and form factor with several commercial mobile companion robots.

The robot runs the Robot Operating System (ROS)[2] middleware. ROS is a widely used framework for writing robot software. It is a collection of tools, libraries and conventions that aim to allow creating complex and robust robot behavior across different robotic platforms.

The instantiated architecture consists of six ROS nodes that can be mapped to the architecture framework modules. Each node was implemented as a separate process, communicating through a mix of publish-subscribe and client-server methods as defined by ROS. Figure 3 shows the instantiated framework. In this initial step of the research we are focusing on implementing the framework in one system initially, therefore the knowledge exchange quality was the only one not contemplated in this experimental validation. Full implementation of the automated experimentation architecture can be seen in https://github.com/davidissamattos/david_ws.

Feedback monitor: this node can be mapped directly to the monitor module. It is responsible for capturing the human feedback. In this case, we receive input from both verbal feedback (e.g. "Too far") and non-verbal feedback (e.g. if it is too close the person steps back).

Metric Analysis: this node receives as input the value function of the system and analyzes input data. The value function we are using is the user satisfaction. We expect our user to be satisfied at least 70% of the approaches in the long run (after a minimum number of experiments).

Experiment Watchdog: this node verifies the current state of the system and the boundaries of our problem. For this system, we defined some boundaries such as, do not get closer than 20 cm from the human, do not experiment if the battery is low and if the experiment is performing poorly (e.g. less than 30% of the cases) we rollback to another version. This module works even if the version manager generates values outside the constraints of the experiment (by implementation/runtime errors).

Experiment coordinator: this node is responsible for keeping track of which experiment is going on, optimization, validation or running in static loop mode.

Version manager: the version manager node receives input from the experiment coordinator regarding which experiment is running. It the feature is running in safe-mode it uses a safe predefined distance. If the learning

[1]http://www.turtlebot.com/

[2]http://www.ros.org/

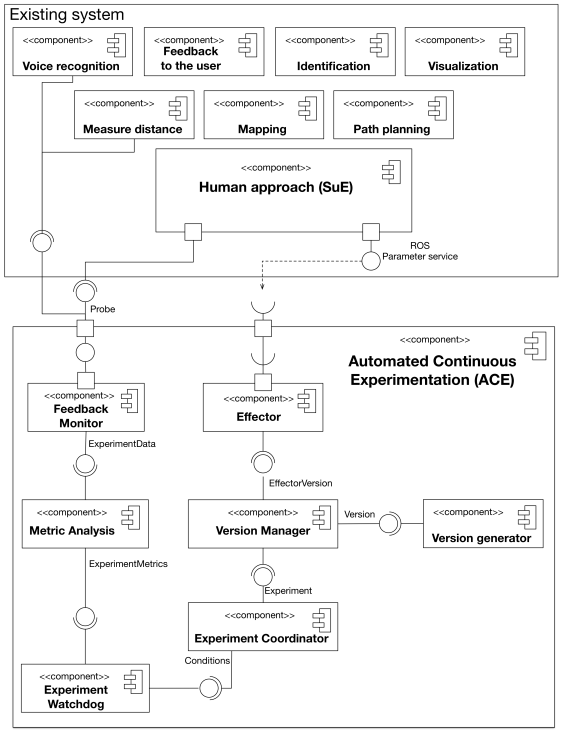

Figure 3. Instatiated version of the Automated Continuous Experimentation architectural framework in the case of a human-robot proxemics distance problem.

process has converged the experiment coordinator requests a static learned version. The version manager generates these versions either by static input or by calling a learning module to generate it (e.g. calling the machine learning module).

Machine learning: the version manager requests new versions to the machine learning module. This module uses the K-means clustering algorithm with the k-means++ initialization algorithm implemented in the scikit-learn [3] Python library. This learning algorithm is a particular solution of the k-armed bandit optimization problem.

Effector: this node is responsible for interacting with the human approach feature and modify in runtime its internal value for approaching. In the ROS context, this means changing parameters in the parameter server. The implemented architecture framework is completely external to the system. The system runs normally without the framework and even with if the instantiated framework crashes.

B. Experimental results

In this subsection, we describe the experimental conditions which we tested in this system. The robot was placed in an office environment. The participant was instructed to give verbal (speak), non-verbal feedback (stepping forward or backward), or both, to the robot if they think the robot is too far or too close (feedback value -1), and not to give any feedback if they think the robot is at a comfortable distance (feedback value 1). The robot always moves to a different location before approaching again. The participant did not have any previous experience with autonomous robots or knowledge of the internal system of the robot. The participant also didn't know that the robot was learning from the given responses. Situations, as when the system goes out of the defined boundaries, were tested manually during the experiment (by explicitly sending the system to a new state) without the knowledge of the participant. All cases were handled by the stop experiment module without any problems.

Fig. 4 shows three graphs representing the learning process of the system experimenting with different uniform random approaching distances. The first graphic shows the system exploiting the solution space trying different distances. The second graphic shows the system learning and refining the lower and upper distance boundaries using the clustering algorithm. The third graphic shows the system learning a static distance after several robot approaches. The static learned distance is the centroid of the cluster located between the boundaries. This graphic shows an online optimization through experiments on the proxemics distance. Changes in the user behavior are reflected in the learning process. Although this experiment shows only one experiment in an office environment, this could be extended to incorporate different factors that influences the robot behavior, such as incorporating the type of room in the experiment, or identifying different use scenarios. This would allow the robot to continuously experiment and learn new distances in different contexts, therefore increasing the value it delivers. This experiment suggests a more effective way of experimenting compared to manual experimentation. Each experiment requires time to run and set up and is valid in restrictive conditions. Although this experimenting method does not guarantee optimality, manually experimenting a matrix of solutions can be expensive in both terms of set-up cost and time.

VIII. Conclusion

Experiments in the field are used in a problem-solving process to drive both innovation and optimization of post-deployed systems. Companies are moving towards to experiment-based development, where experiments support the decision-making process. Several challenges, such as resources, the experiment architecture and novel engineering approaches arise when running experiments in a large scale. In this paper, we study automated experimentation to address these challenges.

This paper takes the software architecture perspective to understand and develop an initial architecture framework that supports both hypothesis validation and for optimization through experiments. Different architecture qualities are identified and it is proposed an initial architecture framework to support automated experiments.

[3] http://scikit-learn.org/

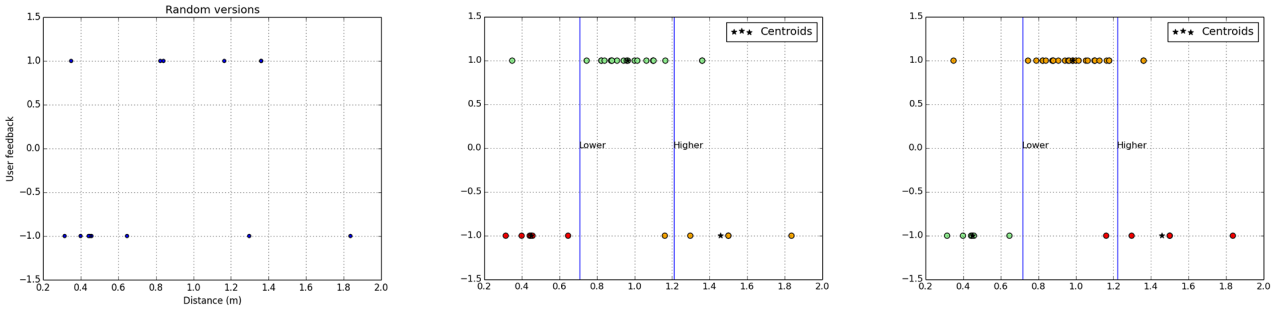

Figure 4. Three graphics representing the learning of the system. The first graphic shows the robot exploring the feedback-distance space. The second shows the robot learning an interval and refining it. The third shows the system after achieving a validated static value for the distance. This distance is represented by the centroid in the orange section.

The architecture framework is validated using an autonomous mobile robot establishing the optimal human-robot proxemics distance. The robot implements an optimization solution based on the explore/exploitation problem running automated experiments. The experimental validation not only assesses the correctness of the framework behavior but also suggests that this is a cost-effective way to run experiments.

A. Research Challenges

Automated experiment systems still have a long way to go before they are matured. Different domains develop solutions and algorithms that continuously push systems in the conceptual solution proposed in [1]. However, these different domains solve their experiment-specific problem without an unified view over the experimentation process itself. This work brings together different facets of automated experimentation and proposes a framework to support automated continuous experimentation. The framework was validated experimentally in the context of an autonomous system and is currently under validation in the mobile domain and in web-systems. One research challenge is to evaluate this and other architecture frameworks in different contexts. This will bring into perspective the challenges from the different domains that might reflect in both the desired architecture qualities and in the architecture framework.

A second research challenge is to combine manual experimentation with automated experimentation in R&D teams as part of the development process for the product. As much as the culture for experimentation changes the organization perspective [11], automated experimentation can change how the systems are developed.

A third research challenge in the process of fully automate experiments is the ability to formulate hypothesis automatically from the data. Manual experimentation requires significant effort for the hypothesis formulation from analyzed data. The proposed framework automates how to run and evaluate experiments, but it still does not support hypothesis generation. Creating hypothesis automatically from recorded data can leverage the amount of experiments the system can run and generate deeper insights on how the system works in the uncertain environment. This would allow the experiment innovation perspective to be an integral part of the system.

Concluding, although there are several challenges ahead, we are moving to a future where systems get better every day we use them through automated experimentation.

IX. Acknowledgments

This work was partially supported by the Wallenberg Autonomous Systems and Software Program (WASP).

References

[1] J. Bosch and H. H. Olsson, "Data-driven continuous evolution of smart systems," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems - SEAMS '16*. New York, New York, USA: ACM Press, 2016, pp. 28–34.

[2] J. Bosch, "Building products as innovation experiment systems," *Lecture Notes in Business Information Processing*, vol. 114 LNBIP, pp. 27–39, 2012.

[3] H. H. Olsson and J. Bosch, "The HYPEX Model: From Opinions to Data-Driven Software Development," *Continuous software engineering*, vol. 9783319112, pp. 1–226, 2014.

[4] A. Fabijan, H. H. Olsson, and J. Bosch, "Early value argumentation and prediction: An iterative approach to quantifying feature value," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9459, no. JANUARY, pp. 16–23, 2015.

[5] H. H. Olsson and J. Bosch, "Towards Data-Driven Product Development: A Multiple Case Study on Post-deployment Data Usage in Software-Intensive Embedded Systems," *Lean Enterprise Software and Systems - Proceedings of the 4th International Conference on Lean Enterprise Software and Systems, LESS 2013, Galway, Ireland, December 1-4, 2013*, pp. 152–164, 2014.

[6] ——, "Post-deployment Data Collection in Software-Intensive Embedded Products," *Continuous software engineering*, vol. 9783319112, pp. 1–226, 2014.

[7] P. Bosch-Sijtsema and J. Bosch, "User Involvement throughout the Innovation Process in High-Tech Industries," *Journal of Product Innovation Management*, vol. 32, no. 5, pp. 793–807, 2015.

[8] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: Survey and practical guide," *Data Mining and Knowledge Discovery*, vol. 18, no. 1, pp. 140–181, 2009.

[9] T. H. Davenport, "How to Design Smart Business Experiments How to Design Smart Business Experiments," *Harvard business review*, vol. 87, no. 2, pp. 68–76, 2009.

[10] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The Evolution of Continuous Experimentation in Software Product Development," in *to appear in: Proceedings of the 39th International Conference on Software Engineering ICSE'17*, Buenos Aires, 2017.

[11] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin, "From Infrastructure to Culture: A/B Testing Challenges in Large Scale Social Networks," *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, no. Figure 1, pp. 2227–2236, 2015.

[12] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, and D. Dennison, "Hidden Technical Debt in Machine Learning Systems," *Nips*, pp. 2494–2502, 2015.

[13] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "What ' s your ML Test Score ? A rubric for ML production systems," no. Nips, 2016.

[14] D. Tang, A. Agarwal, D. O'Brien, and M. Meyer, "Overlapping experiment infrastructure: More, better, faster experimentation," *Proceedings of the 16th ACM . . .*, pp. 17–26, 2010.

[15] E. Bakshy, D. Eckles, and M. S. Bernstein, "Designing and deploying online field experiments," *Proceedings of the 23rd international conference on World wide web - WWW '14*, pp. 283–292, 2014.

[16] L. Bottou, J. Peters, J. Quiñonero-Candela, D. X. Charles, D. M. Chickering, E. Portugaly, D. Ray, P. Simard, and E. Snelson, "Counterfactual Reasoning and Learning Systems," *Journal of Machine Learning Research*, vol. 14, pp. 3207–3260, 2013.

[17] A. Datta, M. C. Tschantz, and A. Datta, "Automated experiments on ad privacy settings: A tale of opacity, choice, and discrimination," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 1, pp. 92–112, 2015.

[18] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5525 LNCS, pp. 48–70, 2009.

[19] R. Kohavi, "Online controlled experiments," *Proceedings of the 1st workshop on User engagement optimization - UEO '13*, pp. 15–16, 2013.

[20] H. Gui, Y. Xu, A. Bhasin, and J. Han, "Network A/B Testing: From Sampling to Estimation Categories and Subject Descriptors," *Proceedings of the 24th International Conference on World Wide Web Pages*, pp. 399–409, 2015.

[21] C. Gatti, *Design of Experiments for Reinforcement Learning*, 2015, vol. 53, no. 9.

[22] R. S. Sutton and A. G. Barto, "Sutton & Barto Book: Reinforcement Learning: An Introduction," 1998.

[23] J. Vermorel and M. Mohri, "Multi-armed bandit algorithms and empirical evaluation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3720 LNAI, pp. 437–448, 2005.

[24] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A Contextual-Bandit Approach to Personalized News Article Recommendation," *Www 2010*, p. 10, 2010.

[25] L. Li, S. Chen, J. Kleban, and A. Gupta, "Counterfactual Estimation and Optimization of Click Metrics for Search Engines," *CoRR*, vol. abs/1403.1, pp. 929–934, 2014.

[26] S. L. Scott, "A modern Bayesian look at the multi-armed bandit," *Applied Stochastic Models in Business and Industry*, vol. 26, no. 6, pp. 639–658, nov 2010.

[27] M. Salehie and L. Tahvildari, "Towards a goal-driven approach to action selection in self-adaptive software," *Software: Practice and Experience*, vol. 42, no. 2, pp. 211–233, feb 2012.

[28] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, jan 2003.

[29] C. Krupitzer, F. M. Roth, S. Vansyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, no. PB, pp. 184–206, 2015.

[30] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, pp. 1–42, 2009.

[31] IBM, "Autonomic Computing White Paper: An Architectural Blueprint for Autonomic Computing," *IBM White Paper*, no. June, p. 34, 2005.

[32] J. Dowling and V. Cahill, "Self-managed Decentralised Systems Using K-components and Collaborative Reinforcement Learning," *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems - WOSS '04*, pp. 39–43, 2004.

[33] D. Fisch, E. Kalkowski, and B. Sick, "Collaborative Learning by Knowledge Exchange," in *Organic Computing — A Paradigm Shift for Complex Systems*. Basel: Springer Basel, 2011, no. July 2006, pp. 267–280.

[34] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self Adaptation with Reusable Infrastructure," *IEEE Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[35] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," in *Future of Software Engineering (FOSE '07)*. IEEE, may 2007, pp. 259–268.

[36] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 3, pp. 54 –62, 1999.

[37] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, C. Magalhã, and R. H. Campbell, "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB," *IFIP/ACM International Conference on Distributed systems platforms*, pp. 121–143, 2000.

[38] L. Capra, W. Emmerich, and C. Mascolo, "CARISMA: Context-aware reflective mlddleware system for mobile applications," *Software Engineering, IEEE Transactions on*, vol. 29, no. 10, pp. 929–945, 2003.

[39] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A Systematic Survey on the Design of Self-Adaptive Software Systems Using Control Engineering Approaches," *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 33–42, 2012.

[40] D. A. Menascé, H. Gomaa, S. Malek, and J. P. Sousa, "Sassy: A framework for self-architecting service-oriented systems," *IEEE Software*, vol. 28, no. 6, pp. 78–85, 2011.

[41] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky, "MetaSelf – An Architecture and a Development Method for Dependable Self-* Systems," *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*, pp. 457–461, 2010.

[42] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola, "MOSES: A framework for qos driven runtime adaptation of service-oriented systems," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1138–1159, 2012.

[43] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. A. Papadopoulos, "A development framework and methodology for self-adapting applications in ubiquitous computing environments," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2840–2859, 2012.

[44] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White, "A multi-agent systems approach to autonomic computing," *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '04)*, pp. 464–471, 2004.

[45] C. Ballagny, N. Hameurlain, and F. Barbier, "MOCAS: A state-based component model for self-adaptation," *SASO 2009 - 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pp. 206–215, 2009.

[46] A. Elkhodary, N. Esfahani, and S. Malek, "FUSION : A Framework for Engineering Self-Tuning Self-Adaptive Software Systems," *Foundations of software engineering*, pp. 7–16, 2010.

[47] M. Reichenbach, R. Seidler, D. Fey, and B. Pfundt, "Organic Computing — A Paradigm Shift for Complex Systems," *Work*, no. July 2006, pp. 179–192, 2011.

[48] H. J. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng, and D. Hughes, "Goal-based modeling of Dynamically Adaptive System requirements," *Proceedings - Fifteenth IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2008*, pp. 36–45, 2008.

[49] K. Angelopoulos, V. E. Silva Souza, and J. Pimentel, "Requirements and architectural approaches to adaptive software systems: A comparative study," *8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 23–32, 2013.

[50] T. van Oosterhout and A. Visser, "A visual method for robot proxemics measurements," in *Proceedings of Metrics for Human-Robot Interaction: A workshop at the Third ACM/IEEE International Conference on Human-Robot Interaction (HRI08)*, 2008, pp. 61–68.