Draft

Behavery: Providing human and artificial behavior for experiments

Alexander Funcke

May 15, 2015

Draft

BEHAVERY: PROVIDING HUMAN AND ARTIFICIAL BEHAVIOR FOR EXPERIMENTS

Alexander Funcke

Behavery is a cloud-based system designed for rapid implementation and execution of behavioral experiments. Unlike other systems for running behavioral experiments Behaverly also allows run the experiments as simulations with artificial agents. Through its web interface, the experimentor create and interlink stages for the subject to interact with, a programming background is thus typically superfluous. The system is built to handle multi-subject experiments, promote re-use through modularity and repositories. The system is equally capable to run with subjects recruited from online labor markets as in a controlled laboratory setting. The most innovative aspect is, however, the introduction of scriptable or trainable automatic or artificial intelligence subjects that allow an experimentor to mix experimental and simulation methodology.

This paper introduce the design, building blocks, as well as some key features and limitations of the system.

Keywords: behavioral experiment software, experimental economics, social psychology, agent based simulation.

1. INTRODUCTION

The reason behavioral experiments typically are computer-aided is not only because of convenience, though the reduction of costs is important, but also as it makes it easier to replicate and limits human error. Recently, experimentors have turn to the Internet, utilizing online job markets to find, collect data from, and pay subjects. This has made behavioral experiments cheaper and more readily available. No longer do a researcher need a lab and pay students for long sessions. Instead, subjects can be recruited online for short tasks and compensated accordingly.

The quality of Internet collected data is however still debated, but per dollar it does look promising (Berinsky et al., 2011; Horton et al., 2011; Rand, 2012). Online experiment tools have often been limited to dynamic

questionnaires, hence many types of experiments have been unfeasible to implement online, or at least have required a considerable software development effort.

Behavery is a system for rapid implementation and execution of computer-aided behavioral experiments. It is designed to be accessible for the experimentor that lack programming background. The paradigm is one of using web forms to create objects representing aspects of the experiment and then relating them to each other. Further, the environment is designed for collaborative work, letting multiple users work concurrently on a project. Projects can also be shared to ease replication, to simplify the development of new treatments and to provide supplementary material for articles. This implies that an experimentor often can re-use, or at least base her work on, an existing experiment. Projects can also be combined, and recombined, to create new experiments. The software uses a web interface both for the implementation and the actual execution of experiments. This implies that experiments can be run out of any web-enabled device, such as in a mobile environment off e.g. tablets or phones.

The system allows for subject interaction whether the experiment is conducted online via the Internet or in a controlled lab environment. Most innovative is perhaps however the introduction of a scriptable and trainable artificial subject. This opens a whole new space of possible inquiry, on the borderline between experimental and simulation methodology. Experimental methods can be applied where only part of the population is actual agents.

In this article, the architecture of the software will be sketched, as well as how projects with-in the software are implemented. Lastly, we speculate about the new questions posed by the possibility of human/artificial agent interaction.

2. SYSTEM DESIGN

Behavery is a cloud-based system, this implies that there is no need to install software on either the device used to create experiments nor on the devices used to run it; saving the experimentor the administrative tasks. The Behavery server is instead run off a server controlled centrally and accessed via the Internet. The software is accessed from a web server and implemented using an array of open-source components such as the Django framework, the MySQL, and Mongo databases. For all implementation of content and logics, the Behavery use the designer-friendly templating language Jinja2.

The only capabilities required by the client-side is Internet connectivity, and a web browser, preferably a modern one, such as Google's Chrome, Mozilla's Firefox, Apple's Safari or Microsoft's Edge. This implies that clients, experimentors and subjects, could access the system with a variety of devices, such as computers, laptops, tablets, or smartphones. [1]

The experiment system include multiple machines of any sort, possibly separated by continents, interconnected via public networks, and as we know from usual Internet usage, sometimes there are problems. Reliability has thus been a core design principle of the system. Subjects' client devices are therefore made "dumb"; i.e. there is no state kept at the client-side. This implies that whether the Internet connection goes down, the device is rebooted, or there is any another problem with the client device; a subject can still safely reconnect and continue where she left off. Even if a device in say a lab permanently malfunction, the user can resume seamlessly from another machine. State of the experiment and its data is directly written to the cloud server's database. Further implying that in the unlikely event of the Behavery server having a problem, being restarted or otherwise tem-

[1] A stand-alone server can also be set up in cases where Internet connectivity is spotty.

porarily disconnecting, all clients seamlessly can continue where they were as soon as the server is reachable again.

Users assume either of two roles, administrator or subject. Only subjects are mandatory, both roles can be automated using Robots.

2.1. *Laboratory*

Despite being a cloud-based, the Behavery system was originally developed for use in a laboratory setting. There is thus a great deal of conveniences created for this very use case. This implies that a proctor for an experiment will have less stressful work, and will require less training.

To set up an experiment for a lab setting the experimentor creates an *E*xperiment Setup and associate it with the laboratory in question. The subjects in a laboratory may be divided into groups. Experiments may then run as if parallel for each group, or even interact or reshuffle group memberships. The Experiment Setup not only associate a particular Experiment with the laboratory, but also how to run it: what language to use? How the groups are to be divided, and how large they are to be? If there are any automatic subjects, or not. Further, the experimentor can assign a "Backup experiment" such that if not a multiple of the group size enters the lab, those not assigned to a group get to participate it in that instead. The proctor of the experiment will typically not need to do anything beyond starting an instance of the Experiment Setup and directing the subject's web browser to the dedicated URL for the laboratory in order to launch the experiment at hand.

2.2. *Online*

Being cloud-based, the Behavery system is by definition online. This implies that collaborating experimentors are working on the same implemen-

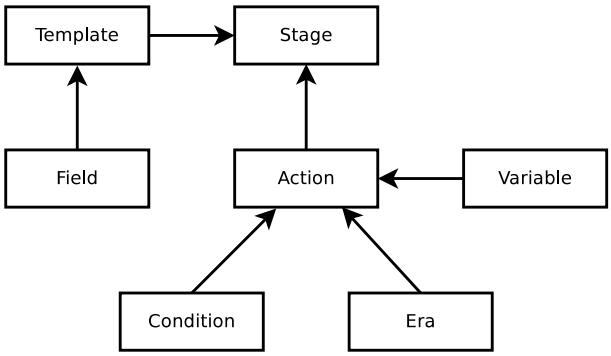

FIGURE 1.— Overview of the building blocks and their relations.

tation wherever they may be based. It also allows the experimentor to set up the system for subjects in a multitude of ways.

Subjects can participate in single or multiple subject experiments. They can require subjects to be logged in, and thereby tie the experimental data to the users demographic information. Log-in can either be done using a Facebook account or by creating an account for the particular user.

To set up an online experiment, the experimentor create a *P*laza Task around a non-laboratory assigned Experiment Setup. The Plaza Task allows the experimentor to specify for how many repetitions the experiment shall be run. It may also specify that the experiment is to be run in generations, such that the behavior in the first generation will change variable values in the second generation, and so on.

At this point, the user base of the Behavery is typically insufficient for efficient recruitment. Recruitment can however efficiently be done by a third-party service such as Amazon's Mechanical Turk.

3. THE BUILDING BLOCKS

Reliability may be the first design priority, but easy-of-use was not trailing far behind. The Behavery tries to strike a balance such that it is usable for experimentors with no programming experience but still be efficient for

those with plenty. This is primarily done by having standard experiments be possible to set up without writing much, if any, code, whilst retaining the possibility to do.

To create a new experiment one typically either fork a pre-existing experiment by duplicating it, or start out from scratch. Creating a new project entails using the graphical user interface's forms to create each of the *Stages* that the subjects are to transverse. After creating the Stages, you assign a *Template* to each. The Template defines the content shown to the Subject in the Stage, it fills in the "blanks" of the *Theme* chosen for the experiment. The Theme defines the look and feel of the page presented. Most administrator choose one of the pre-defined ones, but the flexibility to fork an existing or roll your own, anything that can be done in a website can be used for an experiment.

A third way of creating an experiment is to combine existing ones. Each experiment in the system may also be used as a module in a *macro experiment*.

The logic of transitioning between the Stages are set up using *default redirect*s, or as redirect instructions in *Action*s set to run as one enters or leaves a Stage, called *pre-action*s and *post-action*s respectively. Further control of subject's flow between Stages is done by *Era*s, which are used to synchronize subjects, to wait for each other or proceed simultaneously. The experiment starts out by not being in any Era. Each subject may enter an Era as the consequence of an Action, but it is not until all subjects are in the same Era that the experiment enters that Era. A Stage may be set up to transition to another Stage as an Era is entered, technically this imply that whilst in the first Stage the web browser will poll the server about the current era and issue a redirect accordingly. Note that when using groups, Eras may also be defined on a group-level.

Every state that the experimentor want to keep in the experiment, beyond

FIGURE 2.— A simple example of the flow for an admin

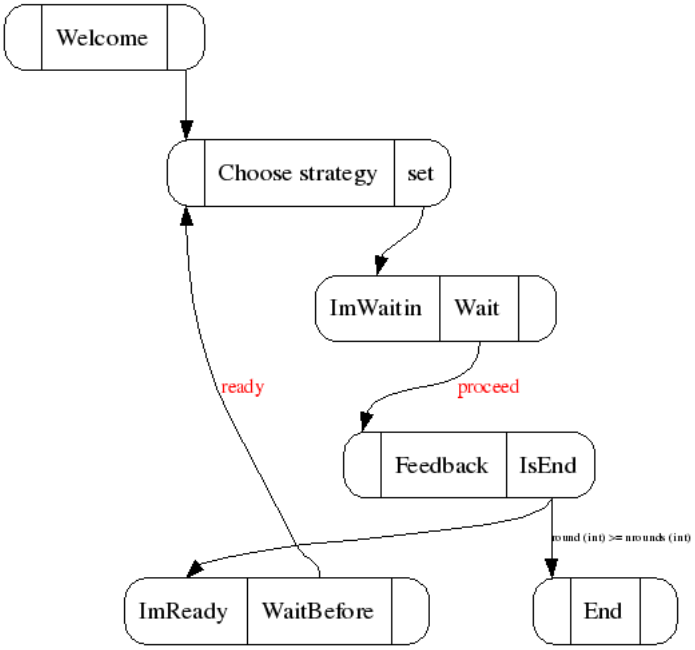

FIGURE 3.— A simple example of the flow for a subject

its flow, is set up using Variables. Variables are defined as either for each user, each group or as experiment-wide. Actions can have steps that which execution is conditioned using *Conditions*, e.g. such that it is only executed if a Variable is greater than a particular value. There are more possible steps of an Action than to control redirection and the current Era of subjects. E.g. to write all Variables defined to be "logged" to the result database, set the value of Variable, or run arbitrary Jinja2 code.

4. FLOW

Behavery offers an auto-generated graphical representation of the Stages and their interlinking. Figure 2 and 3 is an example of a simple experiment. Each Stage is depicted as a rounded edge box with its name in the middle segment. In the left-most segment is the name of the pre-action to be run as a subject enters the particular Stage, and in the right-most segment the post-action that is to run as she leaves. The black arrows indicate stage transitions, the arrows lacking an explicit condition are default redirects. If they do have a caption, they are conditional redirects defined in Actions. The red arrows indicate Eras waited for, such that if the specified Era is entered by the experiment while in the Stage a transition occurs to stage the arrow points to.

In the example in Figure 3 we can see how the subject first enters a "Welcome" Stage informing her about what the experiment is about. Next, she enters a "Choose strategy" Stage, with a Field associated with its Template. As she leaves the Stage the Action "set" will be executed, saving the value of the Field into a Variable. The subject then transition into a "Wait" Stage, with a pre-action Action "ImWaitin" setting the Era of the subject to "Waitin". As all subjects have entered the "Waitin" Era, the experiment will do so too. As we can see in Figure 2, it will then trigger the admin user. She is waiting for the "Waitin" Era to run an action calculating the results. The action writes to the database, updates what round we are in, as well as sets for each user the Era to "proceed", thus making the experiment go into the "proceed" Era. This thus pushes all subjects to the "Feedback" Stage as the "Wait" Stage was waiting for the "proceed" Era. In the "Feedback" Stage, information is given and the post-action checks whether the "round" Variable is greater than the "nround" Variable. The "nround" variable is set up by the experimentor as she starts the experiment. If it is larger, we go to the "End" Stage, and the experiment is over. If not, we go to a "WaitBe-

fore" Stage that works just like the "Wait" stage and is there to ensure that no subject spend more time in the "Choose strategy" Stage than anyone else.

5. RE-USE AND MODULARITY

Another design principle was to ease collaboration and re-usability. This is primarily realized by the use of Themes, modularity of Experiments and the sharing, duplicating or forking of Experiments.

Themes define the look and feel of a Stage's Template and define the blocks that may be filled on a per-template per-language basis. Templates and Themes are written in HTML with the possibility to add logics using the Jinja2 language. Allowing the reuse of existing code in e.g. HTML5, Javascript or Flash, and content, e.g. images, audio, and video.

Perhaps more important than Themes for the re-usability is the modular design. Each experiment can either be run by itself or be used as a module in a large macro experiment. This encourages the development of experiments in modules that can be re-used in later experiments, simplifies running treatments. In a macro experiment, the subjects continue to the next experiment as she enters the Stage that is flagged as "last".

Finally, Experiments can be duplicated, and the copy can be modified, this allow existing experiments to be forked. The possibility to share your experiments with other experimentors thus allows an ecosystem of standard games and beneficial conventions to develop.

6. SIMULATED SUBJECTS

This is still in an early phase, but a promising development. In its first incarnation, simulated subjects can be used for testing purposes. They naively hammer away on the available choices and speed-up the development process. This functionality is also interesting for stress testing ordinary exper-

iments too.

The final purpose is however quite different. Agent-based modeling often uses both behavioral experiments and simulations. The latter tend to be called for as the former does not scale cheaply. This might partly be alleviated. In the next generation a neural net can be trained for a specific demographic profile and then act accordingly. This can be used to render simulations without the need to re-implement the setting. However, more interestingly it allows to run massive multi-subject online experiments, where only a minority of the subjects are humans. This promises to open up experiments for new applications, e.g. we may allow markets to clear.

7. SUMMARY

Experimental economics and social psychology are plagued by the expensive data points, and few replications, this system might partly alleviate the problem. Today there are a handful of multi-purpose systems that aim to do the same, it is, however, unique in that it helps collaborative efforts, reuse, online execution and easy setup. The system is free to use with assigned users or in a laboratory setting, for online payments we charge a small fee on top of the cost of the transfers to support the servers. If Behavery has been used to produce an article we ask you to cite this article.

There are a number of other software systems to conduct experiments, one of the most prolific is the Fischbacher (2007) software z-Tree, it does however not handle online environments. An interesting effort towards similar goals as Behavery is TurkServer (Mao et al., 2012).

REFERENCES

Adam J Berinsky, Gregory A Huber, and Gabriel S Lenz. Using mechanical turk as a subject recruitment tool for experimental research. *Submitted for review*, 2011.

Urs Fischbacher. z-tree: Zurich toolbox for ready-made economic experiments. *Experimental Economics*, 10(2):171–178, 2007.

John J Horton, David G Rand, and Richard J Zeckhauser. The online laboratory: Conducting experiments in a real labor market. *Experimental Economics*, 14(3):399–425, 2011.

Andrew Mao, Yiling Chen, Krzysztof Z Gajos, David Parkes, Ariel D Procaccia, and Haoqi Zhang. Turkserver: Enabling synchronous and longitudinal online experiments draft: April 2012. 2012.

David G Rand. The promise of mechanical turk: How online labor markets can help theorists run behavioral experiments. *Journal of theoretical biology*, 299:172–179, 2012.