AGILE: GIScience Series, 2, 33, 2021. https://doi.org/10.5194/agile-giss-2-33-2021
Proceedings of the 24th AGILE Conference on Geographic Information Science, 2021.
Editors: Panagiotis Partsinevelos, Phaedon Kyriakidis, and Marinos Kavouras.
This contribution underwent peer review based on a full paper submission.

Towards Conducting Reproducible Distributed Experiments in the Geosciences

Florian Ledermann and Georg Gartner

florian.ledermann@tuwien.ac.at

Research Division Cartography, Department for Geodesy and Geoinformation, TU Wien, Austria

Abstract. We present a system for running experiments and user studies that can be effortlessly distributed across multiple heterogeneous devices. By taking into account specific requirements of the geosciences (geovisualization, cartography, location-based services), and by providing a clear and simple conceptual model for defining experiments, this system can help researchers implement empirical studies in less time or with increased functionality, and can lead to increased transparency and reproducibility of studies for other researchers. The versatility of the proposed system is demonstrated in three case studies where the system is put to use in widely different application scenarios.

Keywords: user studies, map use, location-based services, empirical research, reproducible research

1 Introduction and Problem Statement

Any field in the geosciences that is concerned with researching something that is intended for use by *users* may eventually need to verify the applicability of its findings by conducting user studies in the form of practical experiments. In particular in the fields of cartography, geovisualization and location-based services (LBS), running such studies has a long tradition and has contributed to an improved understanding of how humans interact with spatial phenomena and their mediated representations. For the field of LBS, Huang et al. (2018) have recently argued that applications are becoming more complex and heterogeneous, and finding a framework for evaluating such usage scenarios is identified as a major research challenge.

Recently, the issues of reproducibility and repeatability have gained some attention in the research community (Nüst et al., 2019; Kosara and Haroz, 2018; Barba, 2018), partly as a consequence of the so-called *replication crisis*, referring to the fact that in some fields researchers have failed to replicate the results of well-known prior studies. One challenge that researchers who attempt to reproduce a study face is that it may be hard to distil the exact details about an experiment – input parameters, internal calculations, setup of distributed computing environments etc. – from the text describing the experiment, even if at first glance the description seems complete. The effort to analyse and re-implement experiments is a serious obstacle to reproducing research, particularly if it involves complex setups distributed across devices.

In the geosciences this problem is aggravated by the fact that conducting psychological experiments may not be the main focus of the researchers involved. Not all graduate students in the geosciences are naturally born software engineers who can program custom experiments from scratch, nor will readily available solutions always fulfil the specific requirements they may have. We believe that many researchers and projects in the geosciences would benefit from an improved and simplified way to design, develop and replicate experiments, informed by the requirements of their field of research.

1.1 Related Work and Current Practices

To our knowledge, the techniques and technologies employed for implementing experiments in the geosciences are heterogeneous, without clear standards or best practices. While for this early-stage report there was neither time nor space for a formal, exhaustive survey of the practices of the community, our own experience and informal exchange with other researchers has contributed to our understanding of current practices.

Strategies to implement empirical studies can be categorized along a continuum of the amount of control over the exact details of an experiment. Out-of-the-box solutions lie at one end of this continuum, and fully proprietary custom software, programmed by the researcher or under her direction, at the other end.

Towards the "ready-made" end of the continuum lie online survey system (such as LimeSurvey, SurveyLab, SurveyMonkey or even Google Forms), which are a popular way to gather participant responses, and are widely used both in lab settings and in online scenarios. These products offer interactive editing of surveys, which can be modified and customized for a wide range of purposes. All but the simplest of such tools offer ways to extend and customize the appearance of the user interface (UI) and presented information by uploading images or adding CSS and JavaScript to a predefined layout to alter its appearance and behaviour. Realizing fully interactive experiments or precisely controlling stimulus presentation in lab settings with such tools can be a difficult undertaking, and coordinated use across multiple devices or an experiment structure outside the "survey" metaphor is usually not possible.

For controlled stimulus presentation and participant feedback in lab settings, several tools exist that have been developed mainly in the experimental psychology community. These include commercial products such as E-Prime or EventIDE, and open source projects such as PsychoPy or Psychtoolbox. None of the tools we evaluated from this category supported interacting with geographic media (maps, geodata), mobile use or location-based interaction out of the box, and the paradigms at use are not necessarily tailored towards the needs of our community.

At the other end of the continuum sketched above are prototypes programmed by researchers for the purpose of a single specific experiment or a series of experiments. While this approach offers ultimate flexibility, it can be a tedious process to realize non-trivial experiments in this way. For some scenarios like distributed setups, multiple simultaneous users or non-standard user interaction, the researcher may face complex programming challenges. Consequently, when this approach is used, prototypes are often implemented with the minimum required functionality to run the experiment and extract the data.

Such a "bespoke prototype" scenario may offer the greatest flexibility and also allow for reproduction of the results by others, if the code is published open source. However, the source code of an experiment alone is often not sufficient for reproducibility – much may depend on the setup and configuration of software and hardware, which may not be part of the source code but distributed across various files or even devices. *Containerization* has in recent years gained popularity in research communities as a way to distribute not only isolated programming projects, but the complete configuration of a system. While indeed containerization can aid reproducibility further (Nüst and Pebesma, 2020), distributing the description of the complete computing environment for an experiment may obscure relevant details from less relevant system configuration and "boilerplate" code to set up the system. For mobile and distributed setups the possibility for containerization may currently also be limited.

In this paper, we report on our attempt to implement a system for designing and running experiments that takes into account some specific needs of the geoscientific community, allows for full flexibility and complex, distributed setups, while providing researchers with reusable building blocks and a runtime that removes unnecessary complexity from the development and prototyping process. We hope that by providing a programming model that affords a clear structure of an experiment's code in alignment with a conceptual model of the experiment's structure and dynamic behaviour, the proposed system may improve reproducibility, not just by allowing others to run the code, but by facilitating a better understanding and the possibility to not only reproduce, but extend and improve previous works in the spirit of open science.

2 Requirements and System Design

Based on the limitations of the various approaches to experiment implementation discussed in the previous section, we identified the following initial requirements for designing an experimentation framework:

- A conceptual model tailored towards behavioural experiments and user studies
- A centralized experiment specification that defines the behaviour of all (distributed) components, to aid peer review and reproducibility
- Distribution across heterogeneous devices should be effortless and defined at the experiment level (not in separate configuration files, subprojects or the devices themselves)
- Support of multiple configuration scenarios to aid development, simulation runs and different experiment configurations
- Simple composition of experiments from existing components, and simple extensibility, to allow for use in teaching and student's research
- Cater to specific requirements of research in cartography, geovisualization and LBS

Those requirements for research in the geosciences have been broken down further into the following aspects:

- Maps and location tracking should be readily available as components for experiments, with options to adapt those for specific needs
- Support for mobile devices and distributed setups for LBS experiments
- Specification of dimensions for on-screen rendering in real-world metric units, regardless of display resolution and size, for accurate reproduction of maps and stimuli used in cartography and geovisualization experiments
- Support for heterogeneous devices and platforms, as common in our domain, while still providing a concise definition of overall behaviour

2.1 Design Considerations and Implementation

Full flexibility for experiment design is only possible by specifying the experiment and tasks in a Turing-complete programming language (Bostock and Davies, 2013). However, due to the requirements of reproducibility and suitability for non-expert programmers, it is desirable to design an API that minimizes "boilerplate" code[1], complex data structures and object hierarchies, and allows users to express the relevant ideas concisely. A good programming framework keeps the user in a state of "selective cluelessness" (Ledermann and Gartner, 2015; Tulach, 2008) that hides away complexity that the user does not want to be concerned with (e.g. how exactly the code is distributed to the various devices of the overall setup), and highlights the relevant aspects of the design (e.g. how interaction with one component affects the overall state of the experiment).

Modern web browsers offer a versatile cross-platform runtime environment for interactive applications, providing a wide range of output modalities ranging from classic web pages to interactive graphics and even virtual/augmented reality displays (Karhu et al., 2014). JavaScript can be run in the browser or on a server through Node.js. It was therefore decided that the experiment framework should be implemented in JavaScript for both the server coordinating the experiment and the clients rendering its UI to participants.

Above considerations have been implemented in our prototype for a browser-based, distributed experiment framework, *stimsrv*. Upon start, the stimsrv executable reads an experiment description file written in JavaScript, launches a webserver, packages the experiment description together with all required libraries for the browser, and runs a central controller instance that coordinates the experiment. Client devices connect to the *stimsrv* server via a web browser, which shows the visual part of the experiment. (Other types of clients, such as custom hardware interfaces or specialized software, can also connect to the server through the WebSockets protocol). For mobile devices not equipped with a full modern web browser, such as older mobile phones, e-book readers or smartwatches, the experiment display can be rendered on the server using puppeteer [2] (a "headless" version of the Chromium browser, allowing the rendering of web applications without showing them on screen), and sent to the client as a series of images. This allows a wide range of devices to be incorporated in the experiments.

2.2 Terminology and Conceptual Model

The basic terminology used for creating experiments in stimsrv is based on the textbook on experiment design by Cunningham and Wallraven (2011) (concepts reflected in the stimsrv programming model are set in *italics* in the following paragraphs). The term *experiment* is used to denote all aspects contributing to a full experiment run, including participating devices, tasks to be run and data storage. An experiment typically runs a sequence of *tasks*, which usually present some kind of stimulus through a *user interface* to the participants and process some kind of *response*, potentially repeatedly in multiple *trials*.

On top of these fundamental building blocks, we introduce several concepts for managing and distributing the experiment's state and behaviour. A task's *context* represents the current circumstances under which it is run. The context may be modified only between tasks, and may contain parameters specific to each client device. For each trial of a task, a *condition* object specifies the parameters of the trial (i.e. the parameters of the stimulus, and the choice of possible responses). At the end of each trial, a *response* is yielded, which determines the next condition, and a *result* is generated, which may be stored as part of the overall experiment results.

[1] The program code needed to initialize and bootstrap a system before defining the actual desired *specific* behaviour. (https://en.wikipedia.org/wiki/Boilerplate_code)

[2] https://pptr.dev/

2.3 Formalizing Experiment State

One topic that has motivated development in web technologies in recent years is the insight that managing the overall state of a program, particularly when distributed among multiple components, is a complex undertaking and a common source of hard-to-find errors (Madsen et al., 2020). This has inspired the development of technologies like React.js, which allow application state to be modelled as an immutable singleton (an object that exists only once in the overall system and cannot be changed in an uncontrolled fashion), and recreate the complete application upon each state change. In the hope to aid novice and expert programmers alike to create consistent, bug-free experiments, the goal was to use a similar paradigm of centrally controlled, immutable state in our system.

Tasks in stimsrv generally cannot store internal state between runs, and can update the overall state (*context* and *condition*) only by sending *responses* to the server, which in turn broadcasts the new state to all devices – this removes many sources of complexity and errors, particularly in distributed scenarios. This also allows clients to (re)join while an experiment is already running, as they can immediately be updated to the current state of the experiment by simply receiving the active *context* and *condition* objects from the server.

This model also affords the explicit specification of changes of the context (between tasks) or conditions (between trials of the same task) at the experiment definition level, therefore aiding experiment authors to keep an overview of how the different "moving parts" of the experiment interact. Modern JavaScript constructs like arrow functions and generators allow for a concise definition of such dynamic behaviour (see Fig. 1 for example code).

```
const imageTask = require("stimsrv/task/image");
const random = require("stimsrv/controller/random");

module.exports = {
  tasks: [
    imageTask({
      image: random.sequence(["A.jpg","B.jpg","C.jpg"]),
      baseURL: context => "images/" + context.deviceId
    })
  ]
}
```

Figure 1: Example of a *complete* simple experiment specification. This experiment will show an identical random sequence of images on each participating client device, but loaded from a different subfolder for each client.

In addition to the explicit, functional model of state changes, a mechanism for distributing *real-time events* to all clients is provided that may be used for synchronizing volatile internal state during a single trial in a distributed setup. This mechanism can be used, for example, to synchronize map panning or location updates of tracked participants in quasi real time.

2.4 Setup Description

The *devices* participating in the experiment are described in the experiment specification as plain JavaScript objects, listing properties specific to each device such as screen resolution or available interaction methods. Such properties will become part of the *context* of each task on the device. Devices are assigned to *roles* in the experiment, specifying which user interface elements are available to them. This allows for devices to fulfil various roles in the overall experiment, such as displaying stimuli, collecting responses or displaying status information for the experimenter, or any combination thereof.

```
module.exports = {
  // ...
  devices: [
    { id: "supervisor",
      resolution: "1920x1080",
      pixeldensity: 96
    },
    { id: "participant",
      resolution: "1080x1920",
      pixeldensity: 441
    }
  ],
  roles: [
    { role: "supervisor",
      devices: ["supervisor"],
      interfaces: ["monitor", "control"]
    },
    { role: "participant",
      devices: ["participant"],
      interfaces: ["display", "response"]
    }
  ]
}
```

Figure 2: Example setup description, specifying devices and their properties (such as screen resolution) and the roles they may have in the experiment.

3 Three Case Studies Implemented With stimsrv

To explore the versatility of the proposed system, we implemented three initial case studies that demonstrate a wide range of application scenarios ranging from cartography to LBS (see Fig. 3). Two of these case studies replicate previous studies conducted at our group by graduate students – this way, the effort and complexity of implementing the experiment with the *stimsrv* prototype could be compared directly to the original version.

Figure 3: Three case studies implemented using stimsrv. Top: Map use study, asking participants to find places on a map using different interaction techniques. Middle: Map perception study, comparing map rendering on different mobile phone screens. Bottom: Location-based-services study, employing the "Wizard of Oz" prototyping methodology to compare indoor navigation instructions.

The first case study is a prototypical map use experiment, intended to compare the behaviour of participants when using different interaction modalities for zooming and panning on an online map (Huang, 2017). The participant is presented with a series of place names, which she is asked to find on the map by zooming and panning. After the completion of 5 trials, the participant is asked to change interaction modality, and complete another 5 trials. All interactions (zooming and panning gestures) are logged for later analysis.

The original experiment was implemented in JavaScript, using only the Leaflet library for creating the interactive web map. The code of the original implementation consisted of 386 effective lines of JavaScript code (without blank lines or comments), 24 lines of HTML, 114 lines of CSS and 110 lines of configuration specifying the place names and locations. The replicated experiment has been implemented for *stimsrv* with 96 lines of JavaScript, 4 lines of HTML, 28 lines of CSS and 102 lines of configuration[3], while extending the functionality of the experiment to include a supervisor screen on a second computer. Further analysis of the semantics of the JavaScript code shows that the original experiment code used 42 functions, 10 loops and 25 if/else statements, which were reduced to only 10 functions and 4 if statements in the *stimsrv*-based version. The replicated experiment has been implemented by a seasoned programmer in 3 hours.

While neither the number of lines nor the time needed for implementation is a good direct indicator for code complexity (the time needed to learn a new framework like *stimsrv* needs to be taken into account, too), the reduction to roughly one quarter of the original amount of code demonstrates how implementations can make use of the implicit functionality provided by stimsrv, and focus on the parts truly specific to the experiment, while at the same time gaining additional functionality like a supervisor screen "for free".

The second case study is an experiment that triggered the development of stimsrv due to its unique requirements, which could not be satisfied with any other experiment framework we evaluated. In this map perception study, map symbology is rendered on the screens of mobile phones of varying resolutions, and the limits of legibility are tested by presenting increasingly smaller stimuli using a staircase method (Cornsweet, 1962). The mobile phones are mounted behind bezels revealing only a small area of identical size of each screen – therefore, participant feedback cannot be entered on the screen on which the stimulus is presented, but needs to be entered on a separate device provided for this purpose. This experiment design requires a multitude of devices contributing to the experiment in a coordinated fashion. The implementation can nevertheless focus on the functionality specific to the experiment – rendering the various cartographic stimuli – while the *stimsrv* runtime takes care of coordinating devices, stimuli and responses, and logging the results.

The third case study is a replication of another experiment of a graduate student, as a continuation of

[3] *stimsrv* "configuration" is technically also JavaScript code – the distinction here is made between declarative-style configuration code of little complexity, and code defining dynamic behaviour.

earlier research in indoor wayfinding (Wang et al., 2017), inspired by Bakogiannis et al. (2019). In an LBS indoor navigation scenario, a "Wizard of Oz" prototyping strategy (Kelley, 1984) is employed by showing the user a hypothetical indoor navigation user interface, which is in fact a mock-up controlled by the experimenter, who is walking a few steps behind the participant. The original experiment used a native app and screensharing via TeamViewer, which severely limited the design and interaction possibilities. In the re-implementation, the experimenter can remote control the UI on the participant's phone, while not being forced to use a UI identical to that phone. The replicated experiment has been implemented with 107 lines of configuration and only 9 effective lines of JavaScript code (as such "Wizard of Oz" prototyping is facilitated as a standard application scenario).

4 Outlook and Future Work

The vision of *stimsrv* is to allow all aspects of an experiment to be defined in a single, well-structured code repository. Simple experiments may be defined in a single file, complex projects may split the experiment definition into multiple files such as individual task implementations. Still, we believe that even in those cases the structure and concepts provided by stimsrv may contribute to more transparency and better reproducibility and extensibility of experiments in the geosciences and beyond. However, thorough verification of such claims still needs to be provided.

The *stimsrv* server packages all code of an experiment, plus all libraries it uses, into a single JavaScript "bundle" for use by its clients. Such a bundle is a permanent, dependency-free, complete definition of the experiment. When archived together with its runtimes (i.e. as a container including the browser and Node.js executables), this would form a durable, complete archive of the experiment.

In scenarios where it is desirable to *prove* that specific results originate from a specific experiment design, a cryptographic hash of the experiment bundle could be calculated, and deposited together with pre-registration of hypotheses (Nosek et al., 2018), before the experiment is actually run. Results data could then be cryptographically signed with the experiment hash, to prevent tampering and ensure that the data indeed originates from a specific experiment configuration.

More on the user side, there is the idea to make use of the notebook paradigm (Nüst and Pebesma, 2020) for experiment specification. Observable[4] is a platform for JavaScript notebooks, and we plan to evaluate whether such notebooks can be used to specify and simulate experiments in an interactive, visual environment and to supply them to the *stimsrv* runtime for distributed execution for the actual experiment run.

In the immediate future, we are planning to put stimsrv to use in teaching in our LBS lab, and encourage graduate students to use it for their thesis work. In this process, we want to evaluate the advantages and difficulties of the system more thoroughly.

4.1 Software and Data Availability

stimsrv is available as open source software at https://github.com/floledermann/stimsrv. The code for case studies #1 and #3 is also available[5][6]. Case study #2 is still under active development, and will be made available once the experiment has been run and published. Further examples for stimsrv can be found at: https://github.com/floledermann/stimsrv-examples.

Acknowledgements and Author Contributions

This research was conducted using regular research funding by TU Wien. The authors thank Wangshu Wang and Andrea Binn for help with preparing the case studies and taking photographs.

Author roles according to the CRediT taxonomy:
F. Ledermann: https://orcid.org/0000-0001-7559-3531
Investigation, Conceptualization, Methodology, Software, Writing (original draft).
G. Gartner: https://orcid.org/0000-0003-2002-5339
Supervision, Writing (review & editing).

References

Bakogiannis, N., Gkonos, C., and Hurni, L.: Cartographic Visualization for Indoor Semantic Wayfinding, Multimodal Technol. Interact., 3, 22, 2019.

Barba, L. A.: Terminologies for Reproducible Research, arXiv:1802.03311, 2018.

Bostock, M. and Davies, J.: Code as Cartography, Cartogr. J., 50, 129–135,

[4] https://observablehq.com/
[5] https://github.com/floledermann/stimsrv-experiment-findonmap
[6] https://github.com/floledermann/stimsrv-experiment-wayfinding

https://doi.org/10.1179/0008704113Z.00000000078, 2013.

Cornsweet, T. N.: The Staircase-Method in Psychophysics, Am. J. Psychol., 75, 485–491, https://doi.org/10.2307/1419876, 1962.

Cunningham, D. W. and Wallraven, C.: Experimental Design: From User Studies to Psychophysics, 1st edition., CRC Press, Boca Raton, FL, 407 pp., 2011.

Huang, H., Gartner, G., Krisp, J. M., Raubal, M., and Weghe, N. V. de: Location based services: ongoing evolution and research agenda, J. Locat. Based Serv., 12, 63–93, https://doi.org/10.1080/17489725.2018.1508763, 2018.

Huang, S.: Comparing Mouse- and Touch-based Map Interaction for Target Search Tasks on Large Screens, Master Thesis, TU Wien, Vienna, Austria, 62 pp., 2017.

Karhu, A., Heikkinen, A., and Koskela, T.: Towards Augmented Reality Applications in a Mobile Web Context, Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, Oxford, UK, 1–6, https://doi.org/10.1109/NGMAST.2014.36, 2014.

Kelley, J. F.: An iterative design methodology for user-friendly natural language office information applications, ACM Trans. Inf. Syst., 2, 26–41, https://doi.org/10.1145/357417.357420, 1984.

Kosara, R. and Haroz, S.: Skipping the Replication Crisis in Visualization: Threats to Study Validity and How to Address Them, https://doi.org/10.31219/osf.io/f8qey, 2018.

Ledermann, F. and Gartner, G.: mapmap.js: A Data-Driven Web Mapping API for Thematic Cartography, Braz. J. Cartogr., 67, 1043–1053, 2015.

Madsen, M., Lhoták, O., and Tip, F.: A Semantics for the Essence of React, in: 34th European Conference on Object-Oriented Programming (ECOOP 2020), Dagstuhl, Germany, 12:1-12:26, https://doi.org/10.4230/LIPIcs.ECOOP.2020.12, 2020.

Nosek, B. A., Ebersole, C. R., DeHaven, A. C., and Mellor, D. T.: The preregistration revolution, Proc. Natl. Acad. Sci., 115, 2600–2606, https://doi.org/10.1073/pnas.1708274114, 2018.

Nüst, D. and Pebesma, E.: Practical Reproducibility in Geography and Geosciences, Ann. Am. Assoc. Geogr., 0, 1–11, https://doi.org/10.1080/24694452.2020.1806028, 2020.

Nüst, D., Ostermann, F., Sileryte, R., Hofer, B., Granell, C., Teperek, M., Graser, A., Broman, K., Hettne, K., and Clare, C.: AGILE Reproducible Paper Guidelines, https://doi.org/10.17605/OSF.IO/CB7Z8, 2019.

Tulach, J.: Practical API Design: Confessions of a Java Framework Architect, Apress, New York, 416 pp., 2008.

Wang, W., Huang, H., and Gartner, G.: Considering existing indoor navigational aids in navigation services, in: International Conference on Spatial Information Theory, 179–189, 2017.