em-arch: A software architecture for reproducible and extensible collection of human travel data

Anonymous Author(s)

ABSTRACT

Humans, whether citizens, planners or policy makers, are key to understanding and managing urban spaces. Most current smart city platforms focus on instrumenting urban *infrastructure*. Platforms that instrument *human behavior* require augmenting automatically sensed data with human-reported information. Creating such platforms requires builders to work directly with end users instead of the institutions that manage the infrastructure, and address new questions around control, utility, generalizability and privacy.

Researchers such as travel demand modelers, urban planners, economists and psychologists have extensive experience in working with human data, and are also most likely to deploy such platforms in their work. Bridging the gap between platform *deployers* and platform *builders* leads to several technical and social challenges. Technical challenges include processes for customization and extension by deployers, and techniques for augmenting inference results with human input. Social challenges include deployer skepticism about the utility of platforms, and builder reluctance to delve deeply into application domains.

In this paper, we outline an architecture for extensible platforms that instrument human behavior. This architecture is generalized from anonymie, an open-source, extensible platform for human travel data. We validate the architecture with usage patterns from three separate use cases (or "apps") from applied projects. The use cases also serve as a proof of concept for resolving the social challenges involved in bridging the noted gap. It is an open question whether broader deployer communities will coalesce around and contribute extensively to such platforms as opposed to simply using them.

CCS CONCEPTS

•Software and its engineering → Software usability; Open source model; Layered systems; *Data flow architectures; Application specific development environments;*

KEYWORDS

software architecture, human centered, extensibility, usability

BuildSys, Shenzhen, China

DOI: 0

ACM Reference format:
Anonymous Author(s). 2018. em-arch: A software architecture for reproducible and extensible collection of human travel data. In *Proceedings of The 5th International Conference on Systems for Energy-Efficient Built Environments, Shenzhen, China, Nov 7-8, 2018 (BuildSys),* 10 pages.
DOI: 0

1 INTRODUCTION

Humans, whether using infrastructure, demanding changes to it, or planning how those changes should happen, are at the heart of understanding urban areas. They determine the shape of urban areas by accepting or rejecting emerging services and products. Even services designed by experts in urban theory must survive contact with user preferences in practice. Therefore, a holistic understanding of urban areas is tightly intertwined with the understanding of the human decision making process, their preferences, and how they evaluate options. This allows their observed behavior to be modeled for predicting the performance of future hypothetical changes.

Recent innovations in Information and Communication Technology (ICT), typically grouped under the Smart Cities umbrella, hold out the promise of instrumenting urban areas in order to improve them. Most current smart city platforms focus on instrumenting urban *infrastructure* - vehicle flow on roads, or water consumption of buildings. Platforms that instrument human *behavior* - end-to-end multi-modal trips, or the water consumed by each family member in the shower - require augmenting automatically sensed data with human reported information.

Creating smart city platforms for human behavior requires platform builders to engage directly with end users instead of the institutions that manage infrastructure, and address new questions around control, utility, generalizability and privacy. Infrastructure sensors passively observe all behavior within their field and users cannot opt out of participation, but data is typically not associated with one individual. In contrast, human sensors are closely associated with an individual, and humans can actively engage with the sensing, but they control the data collection and can opt out at any time.

Researchers such as travel demand modelers, urban planners, economists and psychologists have extensive experience in working with human data, including *mixed methods* that combine quantitative and qualitative data. They are also most likely to deploy such platforms. Inter-disciplinary research involving both platform *deployers* and platform *builders* can help overcome the challenges associated with human behavior sensing, and build impactful platforms that also rapidly advance the state of the art in the application domains.

However, there is a gap in approaches between these communities which has to be bridged first. *Deployers* use the new technology as a *tool* to understand other phenomena, whether

it is travel behavior or food consumption, so they want to put minimal effort into it. It is not the focus of their research or expertise. In the absence of an existing standardized platform, this results in non-generalizable one-off apps with shortcuts that exactly meet the minimum requirements for that study. *Builders* tend to focus on *generalized analysis* of infrastructure, such as traffic counts or police reports in which the data can be collected using common techniques. While this is undeniably useful, it can suffer from measurement bias because the inferences cannot take human behavior into account. A data collection platform that supports mixed methods, and provides robust core functionality while being open to extension with little effort, would provide a bridge between these two approaches.

Building such easily extensible bridge platforms comes with a host of new technical and social challenges. Technical challenges include: (i) identifying frequently customized modules along with processes to ease customization by deployers, (ii) identifying core inference results which will be reused by multiple analyses along with processes for the deployer community to iteratively improve them, and (iii) techniques for augmenting core inference results with human interaction while respecting user attentiveness budgets. Social challenges include: (i) deployer skepticism about the utility of platforms, (ii) deployer unfamiliarity with open source community building, and (iii) builder reluctance to engage deeply with application domains and act as unpaid tech support.

In this paper, we outline an *architecture* for extensible platforms that instrument human behavior. This architecture is generalized from anonymie, an open-source, extensible platform for human travel data. In the spirit of bridge building, we envision that this architecture will be useful for both deployers and builders. The structured set of components and shared vocabulary allows deployers to evaluate comparable platforms and their features, and choose the one that is most relevant to their needs. It also allows builders to understand the core modules that they need to support, and the features that they may want to incorporate into their platform. And if the platform and its extensions are open source, the collected data is comparable and the analysis is reproducible. This outcome has significant implications on studying the dynamics of human behavior as a function of cultural, social, environmental and economic indicators of urban areas around the globe.

The rest of this paper is structured as follows. In Section 2, we examine prior work from both the deployer and builder communities, and describe their published architecture. In Section 3, we describe an overview of the architecture and the key challenges for each tier. In Sections 4, 5 and 6, we outline the architecture of the client, server and analysis tiers, respectively. Finally, in Section 7 we evaluate how this platform has been used in three different use cases - (i) a classic travel study, (ii) a crowdsourcing initiative for accessibility metrics, and (iii) a behavioral study on incentivizing sustainable transportation.

2 RELATED WORK

We divide the related work into three main sections. Research from *deployers* is focused on the *uses* of the collected data, from *builders* is focused on collecting and aggregating data. We further subdivide builders into *platform builders* who focus on aggregating large quantities of data, and *sensing platform builders* who focus on collecting data. We consider examples of the software architecture in papers of each type, and show how each of them only addresses one part of a complete system.

2.1 Deployers

Deployers tend to be focused on the data collection (the "app") and not the underlying architecture. Almost all prior projects are closed source, one-off studies [1, 4, 7] - the first open source project [9] was just uploaded to github in June 2018. Outside the research community, there is a strong tradition of hiring consultants who deploy their closed source solution, recruit participants, collect data and provide data to the hiring agencies. As expected, most deployers focus on the study performed using the tool and not the underlying architecture, or future reusability.

A classic example of a one-off study is the Future Mobility Survey (FMS) [1], which contains two high level architecture diagrams, and 7 UI design figures. A classic consultant study is rMove [5, 6], which briefly describes the data collection process in text. All of the figures are focused on comparing manual and smartphone based data collection.

DataMobile [9], is a recently (June 2018) open sourced app for travel data collection. It incorporates duty cycled data collection on multiple mobile OSes, a one-time customizable survey on installation, and the authors have solicited partners for data collection. While it is not a full platform, since it only connects to servers at the author's lab, the data collection modules are not clearly delineated, and there is no user interaction beyond the initial survey, it definitely represents one, fairly basic, point in the platform space. It would be interesting to see whether the app evolves into a full platform and if so, whether the result is consistent with the architecture described here.

2.2 Platform builder

A classic example of platform building [8] examines the use of the Internet of Things (IoT) to instrument urban infrastructure. While this paper contains 5 figures solely devoted to software architecture, none of it involves human interaction, collecting qualitative information, or extensibility. The overview software architecture and the example focus on a one-way flow from sensors to humans.

2.3 Sensing platform builder

The closest related work is prior attempts to build platforms that mix sensing and user interaction. Two examples of these are ohmage [11] and participACT [2]. Both are purportedly open source, and both mix qualitative and quantitative data collection. Their combined architecture is fairly similar to the one described here, which provides additional validation for our choices However, there are some key limitations, which we outline here.

Three examples of missing or obscure specifications are described below.

No true multi-method ohmage does not support combined passive sensing and self-reporting, e.g. a mobility app that prompts users when the trip is done. This would require a local processing module that would generate events for self-reporting. The passive sensing is also at fairly low frequencies (1 minute or 5 minutes) which implies that there is no local duty cycling. In contrast, MSF, participACT's sensing architecture paper [3] describes a clear event architecture that could potentially be used to combine methods. Unfortunately, it only works on android and does not address the limitations on background processing in iOS.

Unclear analysis architecture In both ohmage and participPACT, it is not clear what their server and analysis architecture is. How is the data analysed? Is there a pipeline? How can others reproduce the analysis? How can they extend it? This obscurity extends to the actual source code. Although participACT is touted to be open source, for example, their server code is only available "upon request" and it is unclear how others can contribute to it[1].

UI channels and installation For ohmage, in two of their three use cases, participants were provided with phones with the app pre-installed. In the third (Mobilize, 2013), participants were asked to use their own phones, but the installation process is unclear, since all students were also developing surveys using the platform. For participACT, all participants were provided with smartphones, presumably with the app pre-installed.

3 ARCHITECTURE OVERVIEW

As we have discussed in Sections 1 and 2, systems for for urban areas need to understand the interaction between humans and their environment. We have also seen (Section 2) that there is a need for a shared system architecture to understand such interaction and how it can be observed and modified.

These interactions typically consist of:

[1] http://participact.unibo.it/infoen/?page_id=48

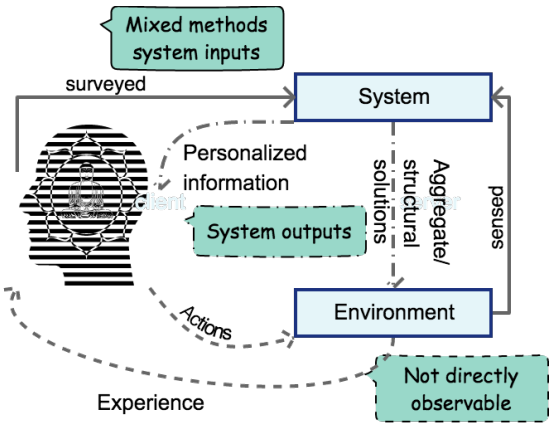

Figure 1: Interaction between humans and the environment, mapped to a computer system

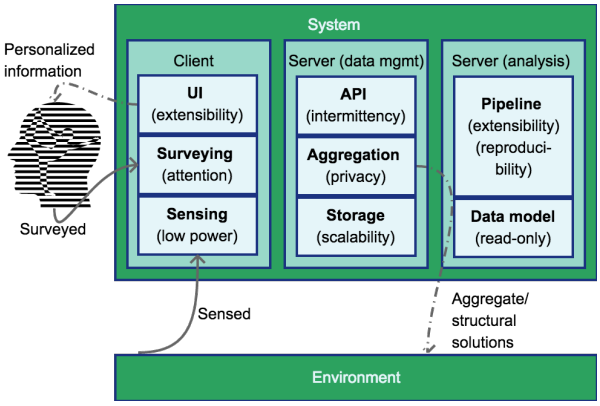

Figure 2: High-level components of the system from Figure 1, and their primary challenges

actions performed by humans (e.g. taking trips). These actions can have both *observable* (e.g. trip mode) and *unobservable* (e.g. trip purpose) properties. Actions can be based on a complex set of factors (e.g. trip purpose, trip benefit, trip chaining, etc). Actions also modify the environment (e.g. depending on the trip mode, a trip affects traffic flow on streets or occupancy in public transit) and are dependent on the environment (e.g. mode choice depends on the relative attractiveness of the alternatives).

experiences of humans when they perform actions in a particular environment (e.g. automotive traffic congestion, public transit delays, or close calls while bicycling). A person's experiences affect their perception of the environment, and in turn affect the actions that they perform in a given environment.

It is interesting to note that neither of these interactions can be completely observed in their entirety by current computer systems (Fig. 1). Instead, what we observe is a *representation* of the interactions, either through background *sensing* for actions and properties for which sensors are visible, and through user *surveys* otherwise. Similarly, the interactions can be *modified* by either influencing the human's decision making process directly, or by changing the environment so that the decision-making landscape is modified.

This interaction flow leads to a 3-tier architecture (Figure 2) with multiple sub-modules per tier. Each of these sub-modules faces a different challenge that needs to be addressed for completion.

We discuss each of these sub-modules in detail in Sections 4, 5, 6, describe the challeges, and outline solution(s) at various points in the solution space for each challenge.

4 CLIENT ARCHITECTURE

As we have seen from Section 2, most prior projects related to travel diaries have focused on the smartphone app and its ability to sense location and accelerometer data. However, focusing only on sensing ignores the human component that is an important component of urban studies. To incorporate the human component using mixed methods, we develop the

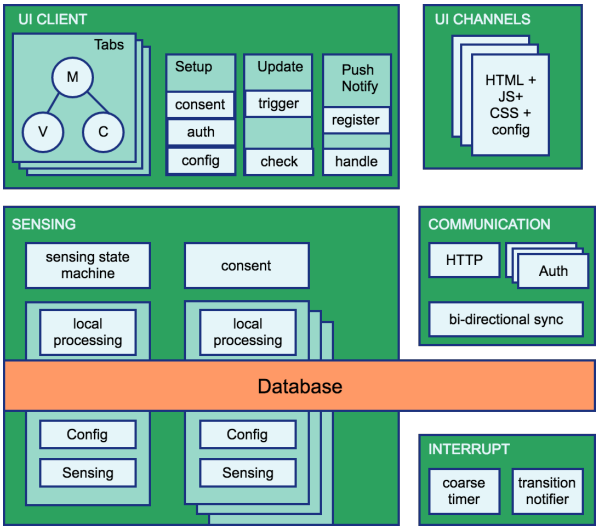

Figure 3: Client architecture, including modules for configurable sensing, robust communication and customizable UI

more complex architecture outlined in Figure 3 that includes modules for configurable sensing, robust communication, and a customizable User Interface (UI). This section describes each of these modules and their desired functionality in greater detail. The three canonical use cases that we consider primarily modified the UI, the transition notifier and the server instance. In order to support *human in the loop* applications, these modules need to be particularly easy to extend.

4.1 Sensing

The sensing module is conceptually simple - it reads and stores sensor values, automatically, in the background. However, power and latency considerations are important while choosing a particular point in the design space.

Local buffering The primary storage tradeoff relates to the frequency at which the data is uploaded to the server. While it may seem intuitive to use the server directly as storage by uploading the data as it is read, the radio draws significant power when turned on, so data should be buffered locally as much as possible. In the case of primarily passive data collection, such as for travel diaries, it is sufficient to upload data after a trip is complete.

Buffering also reduces data loss due to poor connectivity, and decreases the latency of computations on locally sensed data. However, it increases the latency of aggregate operations computed as the server, such as traffic speeds or counts for particular segments.

Local processing The primary processing tradeoff involves latency versus flexibility and complexity. Local processing on buffered data has the lowest latency but the least flexibility, since it has to be implemented in native code for each mobile OS (e.g. android, iOS, ...) that the platform supports. Local processing is also useful if the data volume of the sensor overwhelms the buffer.

Some examples of local processing in the current platform are: (i) *for low latency,* basic filtering of location points for use in the location state machine, and (ii) *for large data volume,* accelerometer-based gesture detection for shakes or bumps.

Location state machine Restrictions on background execution on iOS imply that all background sensing has to piggyback on location tracking. Given that requirement, the primary tradeoff is between continuous sensing, which increases power consumption, and turning off tracking while at rest, which loses the first few minutes of a trip.

If we do turn off tracking while at rest, we need to have a state machine for location sensing that automatically transitions between `WAITING_FOR_TRIP_START` and `ONGOING_TRIP`.

Consent Most mobile OSes already require explicit consent for access to privacy-sensitive sensors such as location. However, academic deployers may be required by their IRB to provide a more explicit consent. All sensing should be stopped until explicit consent is received, and the consent should be documented for future reference.

4.2 Communication

The communication module deals with automatic upload of collected data and download of recently computed data for improved performance. This module needs to handle all aspects of communication, including establishing connections, authentication, and dealing with errors.

Auth All API calls to the server that transmit or receive personal data should be authenticated. The most basic form of authentication is to send a stored password, entered by the user, from the app to the server. While this is intuitive and easy to use, it should be combined with verification to avoid email hijacking.

For short studies with significant researcher interaction, an alternative is to pre-generate a list of random tokens and hand them out to participants. The researcher then does not need to know the users' email and can just use the unique token for all indexing.

For longer studies, the OAuth standard specifies the generation of encrypted tokens (JWT) with configurable expiry times. OAuth JWT tokens can be generated using open source auth servers such as Keystone, or by integrating with third party sign-in provides such as Google or Facebook.

bi-directional sync The main consideration for the bi-directional to/from data transfer is the **D**urability component of ACID transactions. Since any data transfer can be unreliable, the transfer should handle both poor connectivity and potential server errors without losing data.

One technique to accomplish this is to delete buffered data only after a `push` call fully succeeds. This may result in duplicate data from partial retransmissions but will not lose data.

HTTP calls There should be a standard code path to establish connections, refresh the JWT token, and make the appropriate HTTP calls to the server. iOS allows apps to run in the background for no more than 30 seconds, so this code path should use parallel, async calls and rate limiting to speed up execution.

4.3 Interrupt handler

The interrupt handler deals with external triggers. Two current examples are:

Coarse timer We need to have a timer interrupt fire periodically to perform regular maintenance and recover gracefully from unexpected situations. For example, we may want to: (i) push any pending data that was retained in the buffer from previous partial retransmissions, or (ii) reset the location state machine if it is an inconsistent state.

This may appear to be a trivial requirement, since most standard OSes include a timer interrupt and several periodically scheduled bookkeeping threads. However, in order to reduce power drain, many mobile OSes have limits on background operation that make it harder to include such an interrupt. The android limits are still workable, but have been tending towards more restrictive. The iOS limits are much stricter, and typically require server intervention (e.g. *silent push notifications*) for reliable operation.

Event notifier The module also needs to deal with context-specific user notifications. While various events can be detected by the local processing module, deployers should be able to choose the message and actions in the displayed notification. Since the event is detected in the background, when the configurable UI is suspended, there needs to be a native code module to listen to the transitions and display the user-visible message.

4.4 User Interface (UI)

The primary tradeoffs for UI are performance versus effort. Native UIs have better performance, but more effort. This increased effort is intrinsic - native UIs will require a different implementation for each mobile OS that needs to be supported. If we want to support UI channels (Section 4.5), apps would need to ship with all possible channel UIs embedded in them. In contrast, using a hybrid app approach, based on Apache Cordova, allows the core modules to be in native code while the UIs use standard web technologies (HTML+CSS+Javascript). This approach allows a single, consistent UI to be reused across multiple mobile OSes, while channel specific UIs can be dynamically downloaded on demand.

The UI can be split into a core component that configures and sets up other modules, and a set of Model-View-Controller (MVC) tabs that offer other functionality. A brief description of the core components follows.

Setup There needs to be an onboarding process that introduces the app, acquires consent, and authenticates the user. Other initial steps, such as choosing a username, or collecting demographic information, can also be included here.

UI update There needs to be a mechanism (potentially triggered by the *coarse timer interrupt*, Section 4.3) that periodically checks for updates to the channel and applies them, potentially asking the user for confirmation.

Push notify The app needs to register for remote push notifications on startup - both for prompting for qualitative information and for triggering the coarse timer when subject to OS restrictions 4.3.

4.5 User Interface (UI) channels

Extensible components need to be easily configurable for the greatest utility. Each deployer who uses the platform needs to be able to specify the configuration that works for their needs. Since we aim to support mixed methods, this configuration should be able to extend to the user interface. Typical customization requirements include the information displayed, the qualitative input solicited and the controls visible to the end user.

In order to provide maximum flexibility, generality and reproducibility for this customization, it should be possible to support separate UI *channels* that the end-user can switch to. Each UI channel can have a completely different look and feel and can specify completely different configurations for the various modules.

Supporting dynamic UI *channels* also includes several other benefits:

Randomized trials It is easy to conduct randomized behavior trials by randomly directing end-users to different channels as they install the app.

Custom server support Since module configuration is included in the UI channel, installs using different channels can send their data to different servers. This makes it easier for deployers to collect data without having to go through app store review.

Standardization Particular deployer communities (e.g. travel survey groups) can develop canonical user interfaces for their particular use cases. This makes it easier to launch new examples of that use case, and also shortens the methods section of the resulting papers.

Reproducibility Such standardization would be difficult for behavioral studies, in which the goal is to innovate new methods of interaction. However, once the new interaction method has been embodied in a published channel, the study can be generalized or reproduced by recruiting new users and asking them to use the channel.

5 SERVER ARCHITECTURE

Although sensing (Section 4) is typically the focus of the related work, most deployers will also want to upload the data to a server for long-term storage, shared access and complex analysis. The architecture of this server software is typically elided from platform descriptions. For example, a review of Experience Sampling Software([10], Table 1) indicates that only ohmage includes a server component. We need a software architecture that outlines modules for storage, data communication and analysis and groups them into tiers for additional scalability. This section describes such an architecture in greater detail. Having a well-defined architecture with an open source implementation allowed every one of our three canonical use cases to collect data on their own servers. While short-term studies with in-person recruitment also found the server support useful, it was critical for long-term use with online recruitment.

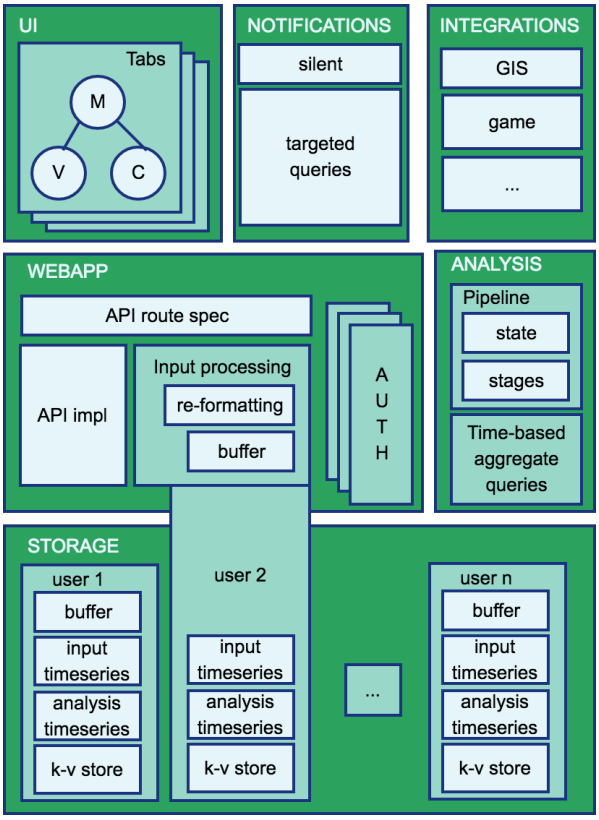

Figure 4: Server architecture, including modules for storage, communication and integration.

5.1 Storage

Storage is the key component of the server architecture. However, the actual storage instance or database product chosen depends on multiple factors like the number of users, the response time expected and the resources available for the data collection. So we instead focus on the types of data collected and the broad storage catgeory for each data type. These broad types are listed below.

Input timeseries The input data received from the smartphone app is stored in a timeseries database. Our data model and analysis pipeline (Section 6) treat this data as *read-only*.

The data can be conceptually viewed as separate user databases, each with multiple streams of data (e.g. `location`, `transition`). Most processing will work on one user at a time, multi-user queries will be aggregated across user databases. This formulation is compatible with privacy sensitive implementations.

Note that we actually have intermittent timeseries data because the sensors on the phone are not typically guaranteed to be periodic. And even if they were, if we use the state machine for lower power drain, there will be no data for long stretches of time. However, we still consider it to be a timeseries, since the primary querying method will be for a time range, and the primary index should be the timestamp associated with each data point.

Analysis timeseries Analysis results generated after processing the input data are stored in a separate timeseries database. While the volume of this data is not likely to be as high as the raw data, it is also time-indexed, and using the familiar timeseries interface allows us to stack analysis results (Section 6) in a consistent fashion.

K-V store Modifiable objects (e.g. `profile`, `config`) are conceptually modifiable objects associated with a particular user by a key. If the deployers would like versioning, and don't want to install two separate database packages, this data can also be stored in the timeseries database - the entry with the most recent timestamp is valid. However, this data will be looked up by key and not by time range, unless somebody requests an audit. So it does not need to be indexed on the timestamp.

Incoming buffer Since the background operation on iOS is time-bound (Section 4.2) we want the data received from the phone to be stored as quickly as possible. As server and database loads grow, directly storing incoming data into a potentially distributed timeseries database could introduce high latency. Instead, we can dump the incoming data into a separate, potentially local buffer, and move it into the timeseries before processing. This additional step also allows us to run pre-processing steps that unify the data model before inserting into the timeseries.

5.2 Other components

The analysis component of the server architecture is fairly complex, and is described in detail in Section 6. The other components of the server architecture are fairly straightforward and their novel features are described in brief in this section.

Webapp The webapp layer defines the API routes used by all clients, including the smartphone app, and any browser-based UIs. The webapp layer also authenticates all user-specific API calls, and needs to support the same set of authentication methods as the smartphone app (Section 4.2.

Push Notifications This module integrates with push notification services to send both *targeted surveys*, which send a link to a survey based on user travel patterns, and *silent notifications*, which are used as coarse timer interrupts on the smartphone (Section 4.3).

Integrations This module handles external integrations. Current examples include `OpenStreetMap`, for GIS lookups, and habitica, for gamification.

6 ANALYSIS ARCHITECTURE

Similar to the sensing (Section 4 module, aggregate analysis exposed through dashboards is a highly visible component of Smart City applications. However, the raw sensor data still needs to be cleaned and post-processed to provide the inferred data that is included in these dashboards. These inference algorithms need to be transparent and reproducible so that they can be understood and improved by the research community. We meet these goals by defining a data model and algorithm structure for reproducible analysis. This section describes the modules required for such a system, and how the resulting data can be transferred and explored by multiple researchers.

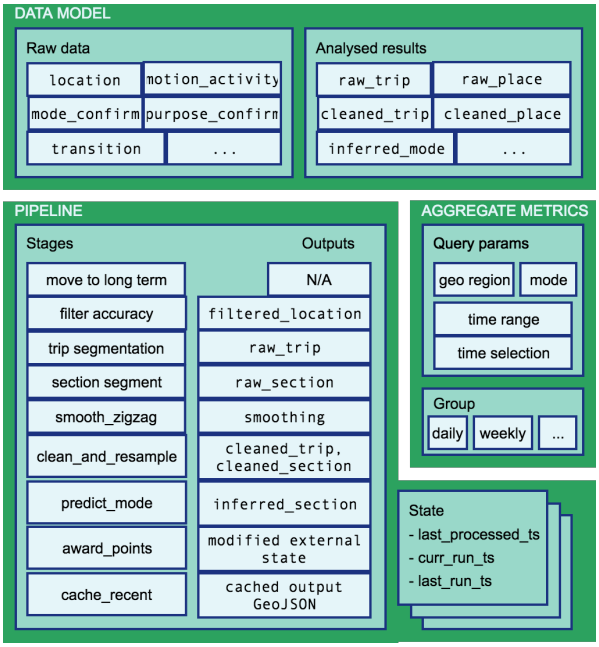

Figure 5: Analysis architecture, including modules for processing the data in idempotent stages, a data model that supports such an algorithm, and aggregate queries.

In one of the three canonical use cases that we consider, the data was collected on a server where the pipeline was not run. The analyst was then able to download the collected data and run the pipeline on her laptop. This shows how well-defined modules are important for reproducibility.

6.1 Pipeline

The analysis component is structured as a *progressive system* in which the only permanent state is the input data received from the smartphone app. The algorithm is structured as a pipeline with a set of *stages*, and the input to each stage is the output from the previous stage. Since each stage only modifies its output, the stages are idempotent. This implies that, given the same inputs and the same algorithm stages, the results will be the same.

This has several important implications.

Reproducibility Analysts can reproduce results from any version of the algorithm simply by running the code on a fresh set of inputs. If the code is versioned properly in a source control system (e.g. `github`), then reproducing results at a previous time t is as simple as: (i) downloading the raw data, (ii) checking out the version v of the source code at time t (iii) running v on the raw data This allows analysts to reproduce prior results even as the codebase has evolved beyond the time that the data was collected.

Extensibility If a researcher develops a new algorithm for a particular stage, she can run both the current state of the art and the new algorithm against the same input data and compare the results. If she chooses to publish the algorithm implementation, other researchers can reproduce her results by running the published algorithm against the raw data.

6.2 Data model

This algorithm design needs a data model that does not require modifying any fields. This can be accomplished in at least two possible ways:

Carry forward The data from the previous step is carried forward to the next step. For example, every trip can store the location points associated with it. Unfortunately, as the number of inferred objects increases, this can get increasingly unwieldy. For example:

(1) since each multi-modal trip can be split into multiple *sections*, should every section also store the location points associated with it? Or should trips retrieve all their sections in order to determine the set of associated location points?
(2) How can we store ground truth for a trip as it ends, potentially before the pipeline has run and generated a trip object?

Time association The newly created objects for each step are associated with start and end time information. We can then associate raw or processed inputs with any object by querying for entries within that time range. This addresses most of the concerns with the *Carry forward* method. For example:

(1) Each section and each trip will have start and end timestamps. The set of points associated with a particular section or trip is then just the set of points between the start and end timestamps. Note that the same kind of query can be used to retrieve (i) *reconstructed locations*, which are resampled at a known frequency for consistency (ii) *filtered locations*, which are raw locations locally filtered for accuracy (Section 4.1) (iii) *locations*, which are raw locations with no filtering. This careful tracking of provenance makes it easier for researchers to experiment with alternate algorithm implementations. Since ground truth is a user input, it should not contain a reference to inferred data. Instead, the ground truth object also contains the time range for which the user has confirmed the sensed data, potentially by specifiying `mode` or `purpose`. While determining the ground truth for an inferred trip, we would simply look up the `confirm` object that overlaps with the inferred time range. This approach can be used for both trips and sections, depending on how much editing power the deployer wishes to provide to the user.

6.3 Aggregate metrics

In general, aggregate metrics are fairly straightforward - the user provides a time range and a grouping, and gets back mode-specific aggregates. Some non-obvious details to keep in mind are:

Geo queries Analysts may want to retrieve data for a particular region for better performance. This implies that all aggregate queries should include an (optional) geo-region as

part of the query parameters. Not all timeseries storage supports both extended time and geo-queries, so deployers should use one that is compatible if they desire such functionality.

Time selections Most timeseries will support range queries where the range is specified in UTC. While this is perfectly adequate for internal use (Section 6.2), deployers may actually want to query by time slices instead. For example, a deployer might want to study evening commute time travel patterns, in which case, she would want to access data from 2pm - 5pm for the month of April. This kind of query is challenging to specify with timestamps since the required time ranges are disjoint, not continous. Storing expanded times, specially in the local time, can help overcome this limitation.

7 EVALUATION

In this section, we evaluate our architecture by comparing it to the challenges associated with bridging the builder-deployer gap. Our evaluation is based on its use in three separate use cases (or "apps") from deployer projects. We show that although the use cases initially appear to be different, they reuse several common modules without any modifications, and are able to extend other modules to meet their needs. We also show that the development time for the projects was much shorter than building one-off apps from scratch. Finally, these projects are an existence proof that it is possible to overcome the social challenges associated with inter-disciplinary platform building. However, in the absence of a rigorous user study, we have no knowledge of negative cases (e.g. deployers who are unconvinced by platforms). Further, although the platform enhancements have reduced as the platform matures, requests for documentation, particularly from non-CS deployers, are increasing with adoption. Therefore, the long-term viability of such platforms is still an open question.

7.1 Metrics

When presenting the idea of a platform to deployers, there was skepticism about the benefits of a platform. Some of the questions that have been raised are:

(1) Is there enough common functionality that it can be abstracted out?
(2) What is the difference between an app and a platform? What is wrong with a one-off project? How much time will using a platform actually save?
(3) Will non-CS communities embrace open source platforms? Why not continue to use consultants instead?

In order to answer these questions, we use the following metrics:

extensibility: We examine the system components identified by the architecture, and see how they are used by the system. Does the architecture ensure that that common functionality is reused and all customization is restricted to customizable modules? Is it possible to extend common functionality without rewriting the entire platform? Which modules are core, and which are customizable?

utility: We compare the time required to create a one-off app from scratch with the time required to customize a platform.

adoption: We measure external contributions to both core modules and customizations, especially if the customizations were re-used by other projects.

7.2 Use cases

We consider three, very different use cases that used the anonymie platform. All of the projects provisioned their own server and collected their own data. In all three cases, the actual customization was done by undergraduates with CS backgrounds; the undergraduates were from three different universities. The undergraduate who worked on the `cci-berkeley` project had worked with anonymie the prior summer; the others had no prior direct experience.

`cci-berkeley` The Center for Community Innovation (CCI) is instrumenting travel patterns of low-income households in order to study the effects of gentrification on overall Vehicle Miles Travelled (VMT).

They use a classic travel survey with a stripped down UI that only included the travel diary. They also removed several of the controls from the profile, notably, the option for "Medium accuracy", and the entire Developer Zone. Since they had GSRs recruit participants in person, they chose to hand out a unique, randomly generated token for authentication instead of having users sign in with an email ID.

They added the ability for users to specify mode and purpose ground truth from the diary screen, including a rich set of modes such as carpool, shared ride, etc.

They initially used the event notifier to pop-up a survey at the end of every trip, but turned it off after negative feedback during the pilot. They also added a survey that would link the user token to the user UUID, but ended up not using it when they switched to token-based authentication.

They are not running the analysis pipeline on the data collection server. Instead, the data analysts pull subsections of the data onto their own laptops and run the analysis there.

`opentoall` The Taskar Center for Accessible Technology (TCAT) is documenting barriers to accessibility - bumpy or non-existent sidewalks, blocked routes, etc.

While they have a regular trip diary, they prompt the user at the end of every trip their experience of the trip, including any barriers that they encountered that are not already in the `opentoall` dataset. They use OpenID connect, linked to their own keystone server for authentication. This allows them to associate trips taken by an user with trips recommended by the `opentoall` trip planner.

They are interested in gamification to prompt crowdsourcing of barriers, as well as adding local processing for bumpy sidewalk detection using the accelerometer.

`tripaware` A group of undergraduates participating in an Undergraduate Research Apprentice Program (URAP) is studying the difference between emotion and information in motivating sustainable behavior.

They are conducting a Randomized Controlled Trial (RCT) so participants are randomly assigned to the emotion, information or control channels, and automatically download the appropriate UI for their group.

Feature	cci-berkeley	opentoall	tripaware
Local buffering	✓	✓	✓
Local processing	✓	✓	✓
Location state machine	✓	✓	✓
Consent	✓	✓	✓
Auth	Pre-created token ↑	OpenID connect ↑	Google auth
bi-directional sync	✓	✓	✓
HTTP calls	✓	✓	✓
Coarse timer	×	×	✓
Event notifier	Removed after pilot	✓	×
Setup	✓	✓	✓
UI update	✓	??	✓
Push notify	×	×	✓
UI channel	✓	✓	✓
Input timeseries	✓	✓	✓
Analysis timeseries	Offline, on laptop	✓	✓
K-V store	✓	✓	Added leaderboard tier position ↑
Incoming buffer	✓	✓	✓
Webapp	✓	✓	New API endpoint for suggestions ↑
Push notify	×	×	✓
Integrations	GIS for mode	GIS for mode, opentoall trip planner	GIS for mode
Pipeline usage	Analyst runs offline ✓	✓	New stage for *tiers, happiness*
Reproducibility	Multiple analysts work with subsets of data ✓	×	Investigate errors in mode inference ✓
Algorithm Extensions	×	×	×
New data model objects	mode_confirm ↑	survey result ↑	×
Aggregate metrics	×	×	×

Table 1: Three projects and their usage of various components of the architecture. Usage key - ✓: used without modification, × not used, ↑ enhancement contributed by this project

They retained the classic trip diary for the control group. For other groups, they added a leaderboard, and modified the summary dashboard based on intervention. For the information group, they provided summary statistics and a set of suggestions for alternatives. For the emotion group, they showed a polar bear that grew or shrank, and was compared to the others in your leaderboard tier.

7.3 Extensibility + adoption

The usage matrix 1 indicates that most of the components were used without modification in a majority of the projects. Most changes were to customizable modules. The only external enhancement to the core modules was the addition of a new auth by the opentoall project. Further, many of the contributions to customizable modules can re-used by other projects. For example, the enhancement that allowed users to specify mode and purpose, introduced as part of the cci-berkeley project was adapted for use in the opentoall project.

Notable exceptions to these general results include:

Auth Every project used a different authentication mechanism. Having a configurable authentication mechanism allows deployers to easily switch between mechanisms, as well as allowing projects to contribute auth plugins that they needed for later re-use.

Coarse timer/Push notify 2 out of 3 projects did not turn on the silent push notification based coarse timer on iOS. Since the data can also be uploaded at trip end, the data collection still worked since both projects were based in the United States, which has reasonable connectivity. They also did not use targeted push notifications.

Algorithm extensions No group has yet contributed algorithm extensions. The CCI group is actively analysing their collected data and might contribute improvements if they develop any. Since the architecture and data model are now clearly documented, we hope that researchers who work on inference algorithms in the future will contribute them to the platform.

Aggregate metrics Since all the projects so far have been focused on small-scale data collection, they have not explored the aggregate analyses possible. The opentoall crowdsourced dataset could be an instance of such analysis, although they

currently plan to use it to publish inferred quality to OpenStreetMap (OSM) instead.

7.4 Utility

This metric is hard to assess because one-off deployer projects that did not publish source code did not publish their development time either. The commercial rMove app [6] took 5 months to develop, but it is unclear how large the development team size was. The one-off Quantified Traveler project [7] involved a development team of 5 in addition to the authors, but it is unclear how many contributions there were, and how long the development took. DataMobile [9] is open source, but it only recently (June 2018) created github repositories through code bulk upload, so we are unable to see the commit history.

All of the anonymie changes so far have taken < 3 months with CS undergraduates working part-time. Less ambitious changes are possible with one undergraduate, RCTs with multiple UIs need a larger team. One of the authors has made minor changes and deployed new channels to collect data at time-limited events such as conferences in less than 24 hours.

Although comparisons to prior projects are not meaningful given lack of reliable data on their development timelines, 3-4 months of part-time undergraduate effort to develop a new app does not seem onerous.

cci-berkeley ≈ 6 weeks of full-time work by one CS undergraduate with prior anonymie experience + ≈ 2 weeks of part-time work by another CS undergraduate to change text and colors.

opentoall ≈ 1 month of full-time work by one CS undergraduate for extending auth + ≈ 3 months of extremely part-time effort for UI changes + integration.

tripaware ≈ 3 months of 6-10 hrs/wk by 6 CS undergraduates to design 3 custom UIs for RCT + server changes for leaderboard and polar bear

Note that this estimate only accounts for deployer, not builder effort. While the pace of platform enhancements has slowed as it has matured, requests for clarification and documentation are on the rise. These requests are particularly numerous when non-CS deployers are involved - for example, although `cci-berkeley` UI changes took only ≈ 2 months, it took another month and a half for the research group to install the server, including 2 weeks just for the SSL configuration. It is unclear how to generate documentation that meets the needs of non-CS audiences without overwhelming platform builders, especially in resource-constrained research environments.

8 CONCLUSION

Humans are key to understanding how urban spaces work. Most current smart city platforms focus on instrumenting *infrastructure*. We outline an architecture for extensible platforms that instrument *human behavior*. This architecture is generalized from anonymie, an open-source platform for instrumenting human travel data. It is validated through three different use cases: (i) a classic travel study, (ii) a crowdsourcing initiative for accessibility metrics, and (iii) a behavioral study on incentivizing sustainable transportation.

The evaluation shows that the architecture is: (i) *extensible*, since all customization was to non-core modules; all module extensions could be performed without rewriting other modules, and (ii) *useful*, since the time taken to create a custom "app" for a new project was < 3 months of part-time undergraduate time.

However, the evaluation also reveals some open questions. (i) All the extensions so far have been to the UI, none of the projects have contributed incremental improvements to the core algorithms. (ii) In the absence of a comprehensive user study, it is unclear whether the deployer community will embrace extensible platforms, and contribute meaningfully to them. (iii) All deployments so far have required substantial builder assistance to create documentation and clarify concepts. It is unclear how best to balance builder and non-CS deployer effort to improve usability, especially in resource–constrained research environments.

We anticipate that we will gain more clarity around these questions as the anonymie platform usage expands, and other similar platforms for human sensing (e.g. [9]) are developed and used.

REFERENCES

[1] Caitlin D. Cottrill, Francisco Camara Pereira, Fang Zhao, Ines Ferreira Dias, Hock Beng Lim, Moshe Ben-Akiva, and P. Christopher Zegras. 2013. The Future Mobility Survey: Experiences in developing a smartphone-based travel survey in Singapore. *Transportation Research Record: Journal of the Transportation Research Board* 2354 (2013), 59–67. DOI: http://dx.doi.org/10.3141/2354-07

[2] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, and Luca Foschini. 2014. The participact mobile crowd sensing living lab: The testbed for smart cities. *IEEE Communications Magazine* 52, 10 (Oct. 2014), 78–85. DOI: http://dx.doi.org/10.1109/MCOM.2014.6917406

[3] Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, and Dario Maio. 2013. MSF: An Efficient Mobile Phone Sensing Framework. *International Journal of Distributed Sensor Networks* 9, 3 (2013), 538937. DOI: http://dx.doi.org/10.1155/2013/538937

[4] Andre Carrel, Raja Sengupta, and Joan L. Walker. 2016. The San Francisco Travel Quality Study: tracking trials and tribulations of a transit taker. *Transportation* (2016), 1–37. http://link.springer.com/article/10.1007/s11116-016-9732-4

[5] Leah Flake, Michelle Lee, Kevin Hathaway, and Elizabeth Greene. 2017. Use of Smartphone Panels for Viable and Cost-Effective GPS Data Collection for Small and Medium Planning Agencies. *Transportation Research Record: Journal of the Transportation Research Board* 2643 (2017), 160–165. DOI: http://dx.doi.org/10.3141/2643-17

[6] Elizabeth Greene, Leah Flake, Kevin Hathaway, and Michael Geilich. 2016. A Seven-day smartphone-based GPS household travel survey in Indiana. Transportation Research Board, Washington, D.C. http://docs.trb.org/prp/16-6274.pdf

[7] Jerald Jariyasunant, Maya Abou-Zeid, Andre Carrel, Venkatesan Ekambaram, David Gaker, Raja Sengupta, and Joan L. Walker. 2015. Quantified Traveler: Travel Feedback Meets the Cloud to Change Behavior. *Journal of Intelligent Transportation Systems* 19, 2 (April 2015), 109–124. DOI: http://dx.doi.org/10.1080/15472450.2013.856714

[8] Jiong Jin, Jayavardhana Gubbi, Slaven Marusic, and Marimuthu Palaniswami. 2014. An Information Framework for Creating a Smart City Through Internet of Things. *IEEE Internet of Things Journal* 1, 2 (April 2014), 112–121. DOI: http://dx.doi.org/10.1109/JIOT.2013.2296516

[9] Zachary Patterson and Kyle Fitzsimmons. 2016. DataMobile. *Transportation Research Record: Journal of the Transportation Research Board* 2594 (2016), 35–43. DOI: http://dx.doi.org/10.3141/2594-07

[10] Veljko Pejovic, Neal Lathia, Cecilia Mascolo, and Mirco Musolesi. 2015. Mobile-Based Experience Sampling for Behaviour Research. *arXiv preprint arXiv:1508.03725* (2015). http://arxiv.org/abs/1508.03725

[11] H. Tangmunarunkit, J. Kang, Z. Khalapyan, J. Ooms, N. Ramanathan, D. Estrin, C. K. Hsieh, B. Longstaff, S. Nolen, J. Jenkins, C. Ketcham, J. Selsky, F. Alquaddoomi, and D. George. 2015. Ohmage: A General and Extensible End-to-End Participatory Sensing Platform. *ACM Transactions on Intelligent Systems and Technology* 6, 3 (April 2015), 1–21. DOI: http://dx.doi.org/10.1145/2717318