

# The Indicator Browser: A Web-Based Interface for Visualizing UrbanSim Simulation Results

Yael Schwartzman and Alan Borning

Dept. of Computer Science and Engineering, University of Washington

Box 352350, Seattle, Washington 98195-2350

Email: {yaels,borning}@cs.washington.edu

**Abstract**—UrbanSim is a large scale land use and transportation simulator that models the possible long-term effects of different policies on urban regions. The output is presented using indicators, which are variables that convey information on significant aspects of the simulation results. To support their use, we designed and implemented the Indicator Browser, a web-based interface for visualizing and browsing indicator results in the form of tables, charts, or maps. This interface was designed and evaluated via design techniques including Value Sensitive Design, paper prototyping, and frequent user testing. The Indicator Browser was implemented on top of an evolving system, developed using agile test-first software development strategies. This paper presents the Indicator Browser as an illustrative e-government infrastructure case study. We describe its interface and system design, implementation process, and evaluation by a set of potential users, and discuss our plans for deployment.

## I. INTRODUCTION

The process of planning and constructing a new light rail system or freeway, setting an urban growth boundary, changing tax policy, or modifying zoning and land use plans is often politically charged. Our goal in the UrbanSim project is to provide tools for stakeholders to be able to consider different scenarios, and then to evaluate these scenarios by modeling the resulting patterns of urban growth and redevelopment, of transportation usage, and of environmental impacts, over periods of 20–30 years [1], [2].

UrbanSim, combined with transportation models and macroeconomic inputs, performs simulations of the interactions among urban development, transportation, land use, and environmental impacts. It consists of a set of interacting component models that simulate different actors or processes within the urban environment. For example, one component model, the Residential Location Choice Model, simulates the decision-making process of a household seeking a new place to live. Other component models include the Land Price Model (which captures key aspects of the real estate market), the Demographic Transition Model (which simulates changes in the overall demographics of the region, such as births, deaths, and moves in and out of the region), and a Real Estate Development Model (which simulates the actions of real estate developers as they build or renovate housing, office space, and so forth). An external travel model models trips, mode choice (whether the trip is by automobile, bus, bicycle, etc), and congestion. Typically, each component model is run on a simulated annual basis. To apply UrbanSim in a particular

area, users prepare a “baseyear database” that consists of the starting state of the region in a particular year. They then prepare alternate scenarios — packages of transportation investments, and policy and land use changes — and then simulate the effects of these scenarios over periods of 20–30 years. These scenarios are either realistic alternatives that are under active consideration, or diagnostic test scenarios to help evaluate how well UrbanSim performs on a particular region.

UrbanSim’s output is presented using indicators, which are variables that convey information on significant aspects of the simulation results. This paper describes the Indicator Browser, a web-based interface that facilitates indicator generation and visualization. We discuss the interface design and implementation process, in particular the integration of Value Sensitive Design [3] into user-centered interface design practices and a discussion of our recurring design issues and implemented solutions, as an e-government infrastructure case study.

The UrbanSim group has worked with regional government agencies in applying the system in the urban areas around Detroit, Eugene, Honolulu, Houston, Puget Sound (Seattle and surrounding cities), and Salt Lake City. There have also been research and pilot applications in Amsterdam, Paris, Phoenix, Tel Aviv, and Zurich. The Puget Sound application of UrbanSim, and our work with the regional planning agency, the Puget Sound Regional Council (PSRC), plays a central role in the system evaluation reported here.

The rest of the paper is organized as follows. Section II describes how indicators are used to present key results from UrbanSim simulations, while Section III summarizes prior work on indicators for UrbanSim, informed by the Value Sensitive Design theory and methodology, which underlies much of the research presented here. Section IV discusses some issues around the latest implementation of UrbanSim and implications for the Indicator Browser. Sections V, VI, and VII describe the interface design, system architecture and implementation, and evaluation of the Indicator Browser. Section VIII concludes.

## II. INDICATORS

As used in the planning literature, an indicator is a variable that conveys information on the condition or trend of one or more attributes of the system considered [4], [5]. The indicator will then have a specific value at a given time. Indicators form the principal means for presenting information

from an UrbanSim simulation to the users. Some examples of UrbanSim indicators are population, number of jobs, land price per acre, vehicle miles traveled per year, or greenhouse gas emissions from transportation. Generally we can produce values for these indicators for the region as a whole, and also for different subregions (for example, the population of the region, or of a particular county, city, or neighborhood). We can also find values for the indicator for each of the simulated years, and for different scenarios being investigated. The indicator values are then displayed as tables, charts, or maps. For example, we might track the greenhouse gas emissions for the region for each simulated year between 2005 and 2030, under two different scenarios. The greenhouse gas emissions for the entire region would best be displayed as a table or graph, while a spatially distributed indicator, such as population density, would typically be displayed as a choropleth map (a map that portrays different population densities as different shades, e.g. darker blue for higher densities). Figure 3 shows a variety of such indicator visualizations.

One important use of indicators is in policy analysis. For example, suppose that we are interested in fostering compact, walkable, more densely populated neighborhoods within the urban area, and curbing low-density, auto-oriented development. In the urban planning literature, population density is regarded as one of the key indicators of the character of development, e.g. dense urban, low-density suburban, rural, etc. We can then use UrbanSim to predict population density in different parts of the region, under alternate scenarios, and show the results as choropleth maps. In addition, modelers use UrbanSim indicators diagnostically, to learn about the system’s internal operation, to help assess whether it is operating correctly, and to debug problems. In the work reported here, we are concerned with both evaluative and diagnostic uses.

### III. VALUE SENSITIVE DESIGN OF INDICATORS FOR URBANSIM

In this section, we summarize prior work by Borning, Friedman, Davis, and Lin [6] that underlies much of the research presented here. The domain of urban planning is both value laden and rife with long-standing disagreements, including in particular the choice of indicators to present, and how they are presented and described. To approach these value issues in a principled fashion, we rely on the Value Sensitive Design theory and methodology [3].

Value Sensitive Design is an approach to the design of information systems that seeks to account for human values in a principled and comprehensive way throughout the design process. Key features are its interactional perspective, tripartite methodology, and emphasis on indirect as well as direct stakeholders. Value Sensitive Design is an interactional theory: people and social systems affect technological development, and technologies — such as urban simulation systems — shape (but do not rigidly determine) individual behavior and social systems. Value Sensitive Design employs a tripartite methodology, consisting of conceptual, empirical, and technical investigations. Conceptual investigations comprise

philosophically informed analyses of the central constructs and issues under investigation. Empirical investigations focus on the human response to the technical artifact, and on the larger social context in which the technology is situated, using quantitative and qualitative methods from social science research. Technical investigations focus on the design and performance of the technology itself. A third key aspect of Value Sensitive Design is its focus on both direct and indirect stakeholders. The direct stakeholders are the users of the system; the indirect stakeholders are those who don’t use the system directly, but who are affected by it — a group often overlooked in other design methodologies.

For UrbanSim in its current form, the direct stakeholders are the urban modelers and planners who use UrbanSim and manipulate its results. The indirect stakeholders are those who do not use the system directly, but who are affected by it. They include for example elected officials, members of advocacy and business groups, and more generally all the residents of the region being modeled, as well as residents of nearby regions. One of our goals in the UrbanSim project is to open the planning process and use of models to wider participation and direct use — in other words, to move more people from the indirect to the direct stakeholder category.

Early in our conceptual investigations, we made a sharp distinction between explicitly supported values (i.e., ones that we explicitly want to support in the model system) and stakeholder values (i.e., ones that are important to some but not necessarily all of the stakeholders). Next, we committed to several key moral values to support explicitly: fairness and more specifically freedom from bias [7], representativeness, accountability, and support for a democratic society. In turn, as part of supporting a democratic society, we decided that the system should not a priori favor or rule out any given set of stakeholder values, but instead should allow different stakeholders to articulate the values that are most important to them, and evaluate the alternatives in light of these values. For example, for one stakeholder economic values might be paramount, while for another environmental values, or for a third, issues of equity. These value concerns might lead to particular attention to different indicators (e.g. acres of vacant land available for development, greenhouse gas emissions, or distributions of wealth and poverty). In addition to the explicitly supported values listed above, we also identified comprehensibility, and subsequently legitimation and transparency, as key instrumental values to be supported explicitly. Legitimation in particular developed as a key value — if some stakeholders don’t perceive the use of UrbanSim as legitimate, they may never accept its use in the decision-making process, and may disengage from discussions involving it, reducing the diversity of stakeholders present at the table and undermining democratic participation. If stakeholders who do not see UrbanSim as legitimate nevertheless choose to stay at the table, their constant questioning of simulation results may detract from discourse about what really matters in the outcome of adopting a course of action.

We were concerned with usability issues as well, in particu-

lar with making information “ready-to-hand” [8], that is, easy to access in the course of interacting with UrbanSim, both as an end in itself and also in support of transparency.

Building on this analysis, we designed and built a series of prototype tools for browsing through the results from UrbanSim simulations. The key design problem we addressed was: how can we create an interaction design around indicators for UrbanSim that will provide improved functionality, support stakeholder values, enhance the transparency of the system, and contribute to the system’s legitimation? The final version presented in reference [6] included careful technical documentation of each indicator, a web-based browser for looking through the available indicators, and a separate Indicator Perspectives framework that supports different organizations in presenting their perspectives on which are the important indicators and how they should be interpreted from a policy perspective. The Technical Documentation for each indicator, with an eye toward providing useful, ready-to-hand, comprehensible information about each indicator, as well as minimizing perceptions of bias, includes sections such as a concise definition, a more formal specification, a discussion of how to interpret indicator results, and known limitations.

At the time this work was done, all UrbanSim output was stored in a SQL database, using the open-source MySQL database system, and indicator results were defined as SQL queries. The documentation included the SQL code to compute the results of the indicator, as well as tests that could be evaluated to check whether the SQL code was correct. These tests were automatically evaluated each time the indicator code was checked into the source code repository. The Technical Documentation was “live” in that the SQL code and tests were extracted directly from the code base each time they were displayed, guaranteeing that what the user read in the Technical Documentation was current.

In our empirical evaluation of the Technical Documentation, we found that users were able to perform information extraction tasks much more quickly using it than with their current work practices [6], which involved using a disjoint collection of reference material and tools. Further, users positively evaluated the inclusion of the SQL code and tests as part of the indicator documentation. These features support the values of transparency and legitimation: readers of the documentation could see exactly what is being computed, and further, could see the tests that were used for that code.

One significant gap in this work, however, was a graphical interface for browsing through the results of computing indicator values, as well as for requesting that additional values of indicators be computed. Instead, one had to evaluate a Python script to do this. It is this gap that the present work addresses (Sections V – VII).

#### IV. INDICATORS IN OPUS, THE OPEN PLATFORM FOR URBAN SIMULATION

In collaboration with an international group of urban modelers, the UrbanSim team recently developed a new open-source platform, named Opus, the Open Platform for Ur-

ban Simulation [9]. Opus is written in Python, and makes heavy use of efficient C++ array and matrix manipulation libraries, such as numarray. The latest version of UrbanSim, UrbanSim 4, is now written using the Opus framework. The Opus/UrbanSim 4 work was carried out after the work on value sensitive design of indicators had been completed, and while it had no major impact on the results of the conceptual investigations, it did result in major changes in the way that indicators were computed. The SQL queries that were used in UrbanSim 3 greatly enhanced transparency, but at a cost in efficiency. In particular, computing a set of indicators could take longer than the run time for the simulation itself. In Opus and UrbanSim 4, indicators can now be defined using Opus variable definitions in addition to SQL queries. This is a considerably less transparent format, but indicator values can now be computed in minutes rather than hours. As discussed in reference [3], there is a complex relationship between supporting human values and usability. Sometimes these goals align, sometimes they are independent, and sometimes in conflict. This situation is an example of a conflict: between supporting the value of transparency and enhancing usability (where the more transparent SQL indicators in some cases were so inefficient as to be nearly unusable).

Opus represents data using instances of the class DataSet, which in turn uses efficient numarray storage representations. Example datasets include the set of all households in a simulation, and the set of all gridcells (the basic geographic unit in our simulation). Households and gridcells in turn have attributes, such as each household’s income, or the number of people whose residence is in a given gridcell. There are two types of attributes: primary and computed. Primary attributes represent raw data (e.g. the gridcell in which a given household is placed, or the number of people in a household). Computed attributes are the result of computations or aggregations of one or more other attributes (e.g. the population in a gridcell, which is computed by summing the number of people in each household placed in that gridcell). Furthermore, attributes that aggregate multiple gridcells at different geography levels, such as zones or counties, are considered to be computed as well. Computed attributes are generated lazily — that is, their values are computed only when needed, and then cached in case they are requested again. Using this scheme, indicators can be simply represented as particular kinds of Opus attributes. (Note that in general, the running simulation won’t compute these particular attributes — in other words, the indicator values won’t be available until they are explicitly requested — unless some other part of the system happens to need them.)

As with indicators in UrbanSim 3, indicator values in UrbanSim 4 can be computed and visualized using scripts. We currently typically use scripts that generate an entire batch of indicator visualizations based on Opus attributes at one time. The script uses a Python dictionary to specify declaratively which indicators to visualize, the scenario, year(s), level of geographic aggregation, and visualization type. The script also allows the user to aggregate indicators from smaller geographies to larger ones and vice versa. For example, one

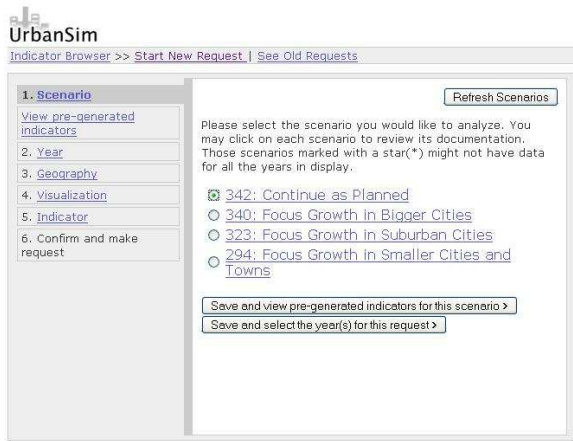


Fig. 1. Indicator Browser screenshot. The Session Bar on the left keeps track of users' selections and provides easy navigation throughout the interface.

can aggregate the values of an indicator at a gridcell level onto the entire region, or dis-aggregate an indicator that gets computed for the counties onto the respective districts within each county. Finally, the script gives the users the flexibility to alter the visualization code to, for example, fix the color range displayed on the maps or determine the desired range on charts. The currently available visualization types are charts, graphs, and maps using the matplotlib library (<http://matplotlib.sourceforge.net/>), maps using OpenEV, latex tables, and comma-separated-value files (which can then be read by any spreadsheet program).

Even though this interface provides considerable flexibility and power over the visualization outcome, some modelers and planners are not comfortable with writing and running code. They find that this interface produces unexpected side effects (such as launching sub-processes from the script that open unexpected command windows). In addition, the modelers still need to reformat and rerun the script, synchronize their code with the source code repository, set the right environment variables, and utilize various applications in order to create the visualizations. UrbanSim developers, on the other hand, feel very comfortable manipulating code, but can only access the visualizations from a file server located at the office instead of being able to access them through a web-browser at any location, and they need to edit and rerun the script even if all they want is a single additional indicator visualization.

## V. INTERFACE DESIGN

The Indicator Browser is a web-based tool for browsing through different UrbanSim runs, and indicator values that have already been computed for those runs and their visualizations. It also allows new indicator values and visualizations to be requested, using a web-based form. It is intended to make indicators easier to generate by the interface's target users. It is also intended as a step towards including more of the indirect stakeholders into the direct stakeholder group. Its design and implementation is strongly informed by the Value Sensitive Design work, in particular, the empirical results on

All Requests			
Delete	Id	Files	status
<a href="#">Delete</a>	1813 <a href="#">View Details</a> <a href="#">Make New Request Like This</a>	request_69: Scenario: 323:baseline with skims Indicators: zone_map_number_of_home_based_jobs_2002.png <a href="#">View Processing Log</a>	completed
<a href="#">Delete</a>	1814 <a href="#">View Details</a> <a href="#">Make New Request Like This</a>	request_67: Scenario: 323:baseline with skims Indicators: zone_map_total_housing_cost_2002.png <a href="#">View Processing Log</a>	completed
<a href="#">Delete</a>	1803 <a href="#">View Details</a> <a href="#">Make New Request Like This</a>	request_66: <a href="#">View Processing Log</a>	error
<a href="#">Delete</a>	1801 <a href="#">View Details</a> <a href="#">Make New Request Like This</a>	request_65: Scenario: 323:baseline with skims Indicators: zone_map_number_of_jobs_2001.png <a href="#">View Processing Log</a>	completed
<a href="#">Delete</a>	1797 <a href="#">View Details</a> <a href="#">Make New Request Like This</a>	request_63: Scenario: 323:baseline with skims Indicators: zone_map_population_2001.png zone_map_population_2002.png	completed

Fig. 2. Results Page, which contains information about current as well as previous requests, and provides commands to view details, delete requests, and make requests like previous ones.

the usefulness of providing tools that contain ready-to-hand information, and on transparency.

In addition, the design and testing of the Indicator Browser made extensive use of user-centered design techniques, including iterative design, paper prototyping, and frequent design discussions and user feedback. The intended users are urban modelers and planners, as well as software developers. Each iteration was designed and evaluated using interactive or paper prototypes keeping in mind the values that UrbanSim is meant to explicitly support. A brief description of all these methodologies, as well as their specific application for the Indicator Browser, follow below.

### A. Iterative User Interface Design

Iterative user interface design is by now standard practice in the Human Computer Interaction (HCI) community (see e.g. [10]). It is based on the realization that even the best designers will not design the ideal interface on the first attempt; instead, iterative testing, feedback, and refinement is essential in order to determine where and how to allocate design efforts. There is a continuing cycle of design, testing, and feedback to the design.

### B. Paper prototyping

Paper prototyping originated in the Participatory Design work in Scandinavia in the 1970s and 80s [11], but has since become widely used as part of iterative user interface design techniques [12]. It is used to evaluate interfaces in the early stages of the design process, with the goal of fixing as many usability problems as possible before implementing the system. Although there are certain interface affordances that paper prototypes simply cannot capture (such as the experience of clicking a button), addressing design decisions such as selecting feature sets and basic user-interface interactions using paper prototypes lowers the design and implementation costs while having tremendous impacts on the usability of the interface [13].

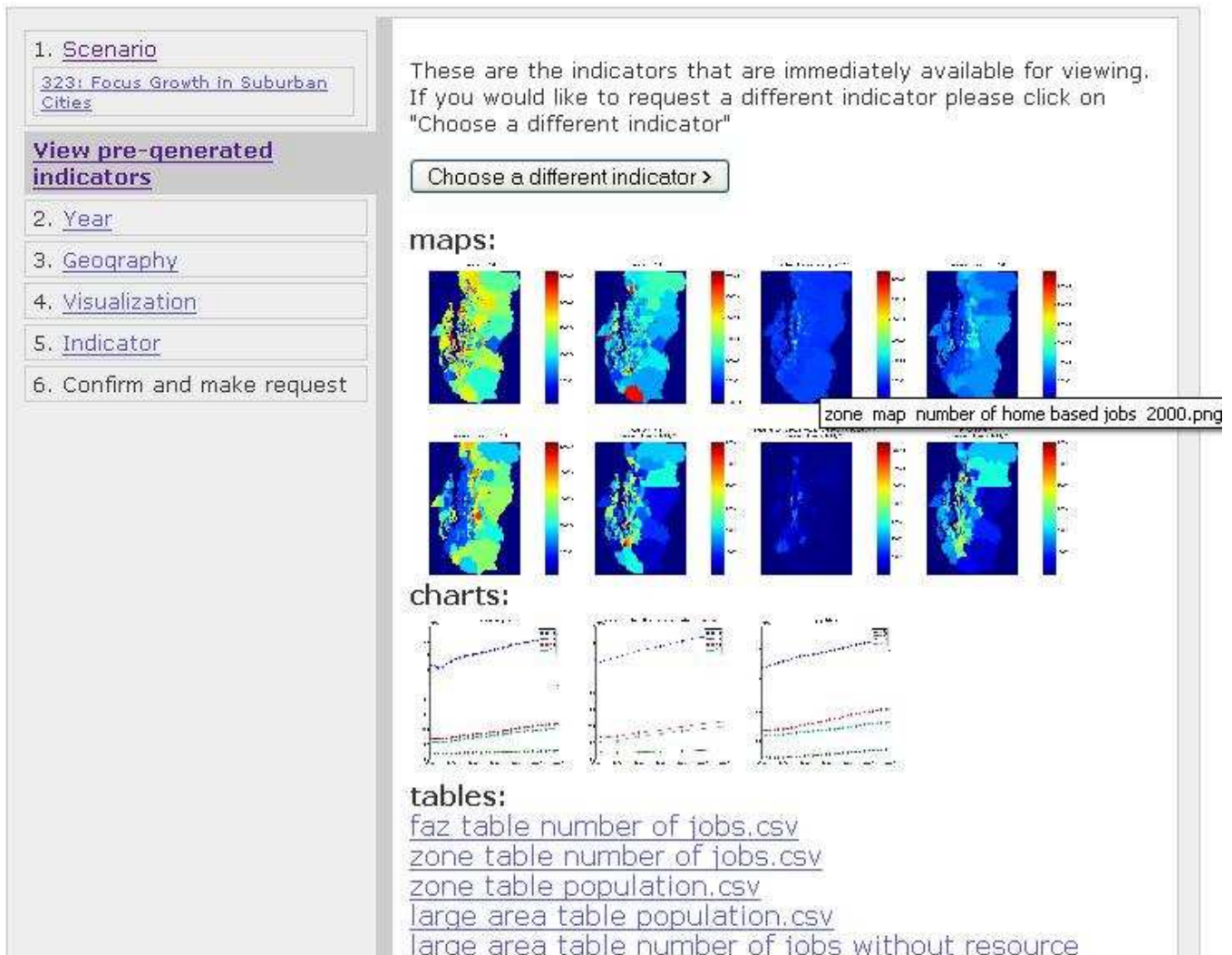


Fig. 3. Pre-generated Indicators page, which displays all the indicators that are immediately available for viewing.

The Indicator Browser design process combined usability testing, iterative user-interface design via paper prototyping, and the Value Sensitive Design methodology. The principal goal in the Indicator Browser design work has been for it to be usable by the target users while continuing to support UrbanSim's core values.

### C. The Design Process

The Indicator Browser was designed through iterative user-interface design. There was an informal initial discussion with modelers and planners at the PSRC, which resulted in an initial sketch of the interface. This interface would be a single web page that would let users select the scenario that they wished to visualize, along with specifications about the visualization, such as its type, years, geography and of course, the indicator(s) selection. After some technical investigations, the designers found the implementation of this interface challenging due to the dependencies between these selections, which are discussed in the next section. Therefore,

the designers moved to a sequential interface, in which the user's choices on one web page would affect the content and available options for subsequent pages. The first author then performed a small informal interface evaluation and fixed some usability issues to get the interface ready for user interface evaluations.

Finally, two rounds of interface evaluations using rapid paper prototyping were performed with eight urban modelers and planners at UrbanSim and PSRC in May and June of 2005, respectively. A team of researchers designed paper prototypes by printing the already existing Indicator Browser web pages, sketching new pages using pencil and paper, and using common stationary such as double-sided tape or stickers to create paper widgets, for instance, drop-down menus or radio button selections. The interface design was refined by using the paper prototype to attempt to complete a set of representative tasks. The tasks were developed using personas that represented our target users, using the knowledge that some of the researchers in the design team had about urban

planning and our target users.

The design team proceeded to evaluate the interface with volunteer evaluators at PSRC and UrbanSim. As is usual with paper prototypes, one person in the design team acted as a computer by reacting to user input while the rest of the team recorded the user's experience. We asked our users to think aloud, to say whatever was on their mind when they were using the interface, in order to determine the interface's usability. We were especially interested in determining which parts of the interface worked well, so that we wouldn't alter them during the next round of design, and which parts of the interface were confusing, in order to come up with better design alternatives. In addition, we got suggestions regarding the representative tasks, which we refined as well through the multiple evaluations. After the user completed the representative tasks we asked them if they had any general suggestions and we explored possible design alternatives together.

#### D. Recurring Design Issues

There were some prevalent design issues throughout the interface evaluation. These issues were discussed extensively and multiple design alternatives were created in order to address them.

**Providing ready-to-hand information.** This was a key goal, both to support the value of transparency and also to enhance usability. Its importance is supported by our earlier empirical work (Section III). In this context, the Indicator Browser should make all the information easily available for the user to make an informed selection of the indicator to be visualized. In a larger context, interfaces often suffer from lack of help and documentation in the right place [14].

Through the different stages in the design process, we found the need to present the user with ready-to-hand information about UrbanSim, the Indicator Browser, indicators and scenarios. The user wanted to know which indicators had been precomputed, how long did the indicators take to compute, the status of their request, what other requests had been requested and the current visualization's selections. Since it was somewhat difficult to assess how long were the indicators going to take due to the complicated interaction amongst models during the simulation, we decided to give the users a worst case estimate. While some users were satisfied with this information, some still believed more precision was necessary.

An example of our efforts to provide ready-to-hand information can be seen in the results page found in Figure 2, where the users can see the status, details, results, computation and error logs for all the requests. We also provided links to the scenario run configuration details and the indicator's Technical Documentation or code (in case the documentation was not available). All of this information was highly appreciated by the users at every step of the design process.

**Sequence.** As previously noted, the Indicator Browser was originally planned as a single-page interface. However, this was not possible due to the dependencies between selections. The indicator availability is dependent on the geography one wants to visualize, since there are some indicators that make

more sense when visualized at certain aggregation levels. For example, there are special Traffic Analysis Zones to study transportation-related indicators. Additionally, the visualization type depends on which geography is selected, since not all visualizations are possible for all geographies. (For example, it is not practical to create a table for gridcell-level indicators for most applications, since the table would be too large. The PSRC application, for instance, has approximately 800,000 gridcells.)

The order of the choices presented to the user was designed based on a combination of technical and empirical investigations. For example, the tests with paper prototypes indicated that users would prefer to select the indicator before selecting details about the years, visualization type, and aggregation levels (geography). However, once we moved to the implementation phase, we realized that constructing an Indicator required knowledge about the selected years and geography, so in the implementation the Indicator selection page was placed after these two selections.

**Versatility vs. simplicity of design.** The Python script, which is the current way of generating indicators, is very flexible and allows the user to create various indicators in addition to the ones that are available through the existing Opus attributes. In addition, users who feel comfortable manipulating code can access UrbanSim's code directly in order to manipulate the look and feel of the indicator visualizations. Therefore, the design team had to come up with a design that allowed for the most flexibility while maintaining its usability. Consequently, there were several design tradeoffs that limited the functionality of the interface in order to maximize its usability. One example being that we only allow users to compare alternative scenarios to the baseline scenario, rather than allowing them to compare any two available scenarios.

**Repetition reduction.** The interface is designed as a sequence of pages that present users with different alternatives in the form of radio buttons or check-boxes, in addition to giving the users the option to provide a title for the request and an email where to send the results. This meant that for every request they had to go through the same set of pages, even if they only wanted to change one or two alternatives. This proved to be very repetitive and inefficient, especially since urban planners and modelers usually analyze indicators starting with general information and then requesting more specialized information where they see fit. For example, they might initially want to see population maps at a county level and later request the actual numbers (or a table) at a district level for a particular year that looked interesting. Therefore, we decided to create a feature called *Make request like this*, which, when clicked, opens a new window with a pre-filled request that the users can modify. This feature was highly valued by the interface evaluators.

There were other shortcuts provided to reduce the amount of input repetition, such as an *All years* option in the years page that automatically checked all the available years, a *Results* page that allowed users to browse through other users requests and retrieve their visualizations. Finally, we also created a *Pre-*

generated *Indicators* page for the users to have immediate access to a list of pre-generated indicators after the scenario selection.

## VI. SYSTEM DESIGN AND IMPLEMENTATION PROCESS

### A. System Design

The system can be divided into three components: The mechanism that allows the user to make requests, the mechanism that manages the requests once they have been submitted; and the mechanism that actually generates the indicators and their respective visualizations. A general overview of the system can be found in Figure 4.

### B. Creating a Visualization Request

Creating an indicator visualization request is accomplished by allowing the user to define the request through a sequence of web pages. Each page in this sequence is channeled through a single CGI page that gathers user input from the current page, updates the web session, and generates and displays the subsequent page accordingly. Each web session is determined by an entry to the *Sessions* table in the Indicator Browser database. The database not only keeps track of the values that were selected, but also of the values that are available for future web pages based on previous selections. For example, suppose we have data for the year of 2000 and 2001 for Scenario A. If the user selects Scenario A in the *Scenario* page, then the only options that appear on the *Year* page are 2000 and 2001. A nice feature of this implementation is that if one would want to make more options available to the user, it is only necessary to add one more entry to the corresponding table on the Indicator Browser database. For example, we are planning to add support for visualizing the differences between scenarios. If we would want to allow the users to create these visualizations we would just need to update the *available tasks* table with information about this new task.

The page rendering is accomplished through a set of HTML templates that contain general information about how to display a page. A small number of global templates determine the HTML rendering that pertains to all web pages on the site, such as the CSS style-sheet link and common navigational links. A sidebar template contains the HTML framework for the part of the web page that displays the session's status (Figure 1). Finally, there is one template per web page that contains more specific information regarding the rendering of each page. Each web page has its *Page Builder*, a module that fills in the template blanks and renders the current state and page of the Indicator Browser.

The *Site Structure* module determines the site's sequence and the input values and types that one should expect at each page. This provides flexibility both for the order at which the existing pages are displayed and for including new pages into the sequence.

Since the rendering of a page usually depends on the previous user selections, there is a mechanism that determines whether or not there were changes made to the previous page. If there were changes, and these invalidate selections made

in subsequent pages, all the invalid values are removed to maintain coherence.

### C. Managing Visualization Requests

Once the user has selected the scenario run, year(s), indicator(s), geography, visualization type, request title, option for notification and email, a request is created and added to the visualization queue. The *Visualization Manager* is a Windows service hosted at one of UrbanSim's servers. This service is in charge of managing the request queue by waking up every so often to:

- Send new requests to the *Visualization Service*, the Windows service that generates the indicators and their corresponding visualizations.
- Attempt to complete requests that are being processed by assessing whether the visualizations ordered in the request are present in the corresponding location in the file system.
- Send email to users once the requests have been completed, if the users selected to be notified by email. This email contains the URL of the web page that displays the indicator visualization if the request was completed successfully, and a link to the error log otherwise.
- Send an email to the Indicator Browser maintainer if there was an error processing a request.

### D. Generating Indicators and Visualizations

The *Illustrator* is a module that takes in a visualization request, configures it by adding information that is specific to the Indicator Browser, and sends it to the *Indicator Factory*. The *Indicator Factory* is a module that retrieves all the scenario resources, adds them to the Indicator Browser specific information, creates a file with all this information, and forks a process for each of the indicators in the request. This process is called the *Indicator Maker*, which retrieves this file, all the necessary data from the results of running the scenario, and computes the values of the indicator. Once these values have been computed, the *Indicator Maker* creates a visualization and saves it both into the scenario cache and into the Indicator Browser web space, which is one of the specific items of information required from the *Illustrator* module.

### E. System Implementation

1) *Agile Test-first Software Development*: The Indicator Browser was developed using Agile Test-first software development [15], which is a technique that focuses on producing software systems in small steps driven by unit tests. The idea is to deliver working software early and continuously to enhance customer satisfaction. A developer should first write a test for a new piece of functionality before it is implemented (test-driven development). Both independent modules and interactions between modules are tested. UrbanSim developers are encouraged to commit their code as often as practical. Fireman, a program that runs all the project's tests every time the code is committed to the source code repository, also aids

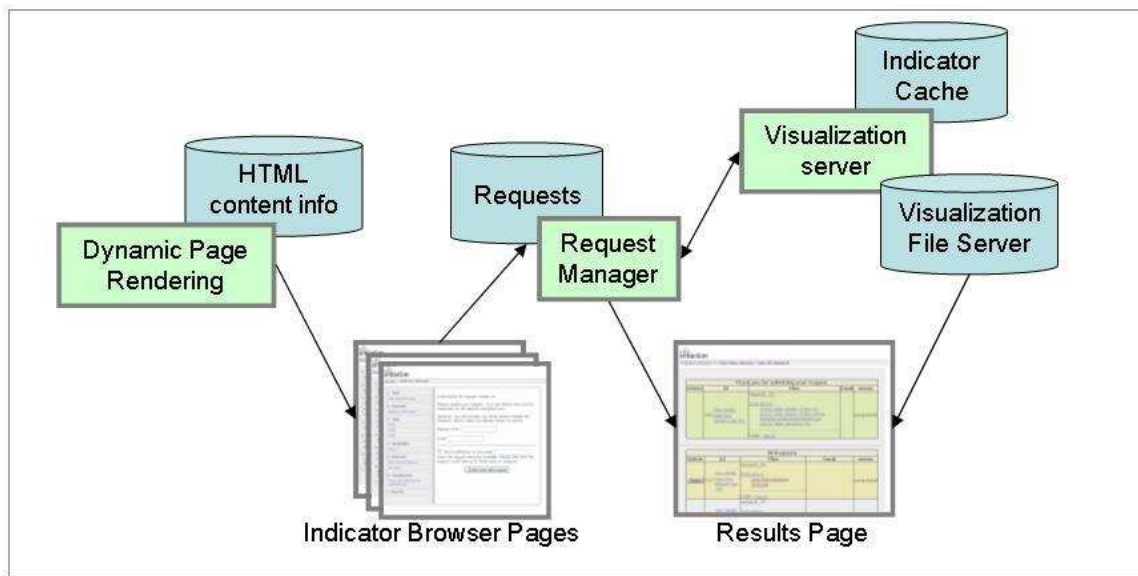


Fig. 4. System Overview.

in maintaining code quality. Agile development encourages constant communication amongst the people involved in the development process. Developers write “Today messages” [16] every day to let the rest of the development team know about their daily progress. In addition, software engineers and project leaders meet with customers on a frequent basis to communicate about the project’s progress and redefine requirements.

2) *System Reliability and Robustness*: Robustness refers to the ability of the software system to perform correctly even under extreme circumstances or when there are external or internal changes that affect system’s performance. Reliability refers to the ability of the software system to perform as intended for a long period of time. Opus is an evolving system that constantly redefines its requirements and whose code base is unstable. However, the Indicator Browser needs to be robust, reliable and flexible with respect to the changes in the Opus code. Therefore, there were certain measures taken in order to improve such reliability and robustness.

In order for the Indicator Browser to be reliable, all displayed indicators should be able to be generated and visualized. To achieve this goal, we created a set of Python unit tests that generate all possible indicator visualizations under all possible visualization types for a small test scenario run. The only indicators that are displayed on the interface are those that passed the aforementioned test, therefore considerably improving the chances that all the indicators displayed on the Indicator Browser can in fact be successfully generated and visualized. Currently, these tests and the set of available indicators are run and updated manually. However, these tests will be eventually added to the Opus nightly build and the available indicator list will be updated automatically.

## VII. EVALUATION

The Indicator Browser was evaluated by four urban planners and two UrbanSim developers in April 2006. The evaluation consisted of an observation of the use of the Indicator Browser in a natural setting, followed by an informal interview guided by five general questions. The users were asked to generate and visualize the indicators that they usually work with using our interface. If they were unclear about what indicators to generate, the evaluator prompted them to generate the indicators that were discussed on one of the PSRC-UrbanSim meetings that was held a few days before the evaluation. The users were asked how often and how they usually generate and visualize indicator visualizations. They were asked to compare the Indicator Browser with the tools they currently use. They were also asked whether and how they would use the Indicator Browser, and what changes needed to be made for the interface to be useful for them. Finally, they were asked to consider whether this interface could be used by a different set of users and if so, who would that group be. We were interested in gauging how commonplace it was for the users to generate or visualize indicators. Since there are two different types of users: urban modelers and planners on the one hand, and UrbanSim developers on the other, each group might have a different set of needs and uses for the Indicator Browser. Furthermore, we wanted to prioritize the usability issues found during the evaluation and to have an understanding of possible future steps.

### A. Usability and Value Issues

The evaluations showed that there are still some usability issues to fix before deploying the interface. Some of the interface changes made after the second round of usability testing, mainly due to internal design discussions, had not been evaluated before and showed the need for refinement.

For example, we added a feature that allowed users to see pre-generated indicators immediately after the scenario selection (Figure 3). If the indicator that the user wanted wasn't on the list there was a *Choose a different indicator* button that allowed them to continue making their request. This was a very useful feature that the users liked. The users showed interest once again in having more ready-to-hand information, such as having the scenario's name for each of the requests on the results page.

### B. Post-task Interview

After the users interacted with the interface for about half an hour the evaluator asked them the following five questions:

- How often and how do you currently generate indicator visualizations?
- How does the Indicator Browser compare to the tools you currently use to generate indicator visualizations?
- Would you use the Indicator Browser? If so, how?
- What would need to be fixed for you to use the Indicator Browser?
- Would you recommend this interface to someone else? If so, who?

The interviews revealed that each of the user groups generate indicators with different frequency and that they have different perspectives on how useful currently used tools are based on generation frequency and level of programming expertise. Their perception of the utility of the currently used tools had an impact in their perception of utility of the Indicator Browser.

One of the users is a general urban planner who usually does not generate indicators himself. The indicators are generated for him and he can access them through a file server. He then visualizes them using the FastStone Image Viewer tool (<http://www.faststone.org>), which allows contiguous visualization of up to four images, as well as synchronized image zooming and panning. Consequently, he did not find the indicator generation mechanism extremely helpful for his common work tasks. He mentioned he usually looks at the same set of indicators, and so he would use this interface to see the pre-generated indicators and would most likely not create some of his own. However, he suggested that the interface would be a useful way for the general public to access this information.

Two of our interface evaluators are urban modelers and generate indicators on a regular basis. Although they know how to create macros and how to program, they felt much more comfortable generating indicators through a web application. They found the Python script (described in Section IV) to have a high set-up cost. The Python script requires them to check out the correct projects from the source code repository, set environment variables, modify the Python script, and run it using multiple applications. Since several sub-processes are launched within the script, the users became confused whenever windows popped on their screen which they didn't know if they could close or what they were meant to do. This last user group found the Indicator Browser very useful for

indicator generation due to its low set-up cost, aggregation of functions into a single web-based application, and the elimination of the need to program. Their primary concern was with the pre-generated indicators section of the interface, which they found confusing, and mentioned they would use the interface after this issue was resolved.

Two of our evaluators are software engineers and developers with the UrbanSim project. They are very comfortable using and modifying the Python script and enjoy the flexibility that it affords. The Python script allows them to force color ramps on maps and create indicator difference, aggregations and dis-aggregations of data across multiple levels of geography. However, they do not have direct access to the file server at home. One of the evaluators mentioned he had to use Microsoft's Remote Desktop application to access the generated indicators from the UrbanSim file servers. Therefore, he found the Indicator Browser to be useful for accessing the indicators using a web browser from any location. The second evaluator liked the organization of information. He liked that he could see which scenarios were available and that he could access the indicators by selecting the scenario from a list on a webpage rather than having to find out what the scenario directory was through database tables and access it through the file server. He mentioned that once he analyzes the outcome of the big batch of indicators generated through the script he would use the Indicator Browser to create additional indicators instead of editing and rerunning the script. However, the other software engineer mentioned that he would not use the Indicator Browser since he enjoys the Python script functionality and does not like using multiple applications to complete a single task.

Some users mentioned they would recommend this interface to the general public and activist groups for use in urban planning courses, engaged citizens workshops, technical advisory committees and general regional policy inquiries.

All users found emailing the results or accessing the results through a URL to be very useful. Most of them mentioned they would use the Indicator Browser to create additional indicators to get more detailed information after analyzing the pre-generated indicators.

## VIII. CONCLUSION

This paper presented the Indicator Browser, a web-based application for generating visualizations of UrbanSim simulation results. We showed how a diverse set of techniques (Value Sensitive Design, iterative design, paper prototyping, agile software development) can be used in concert in the design and implementation of an interactive digital government application, in particular, how consideration of human values can be integrated with user-centered design techniques.

The evaluation described in Section VII had only 6 participants, and so these results are only preliminary regarding the usability of the system in practice. However, these results were mostly encouraging as far as they went. Users were especially interested in using the Indicator Browser to visualize the batch of indicators that are generated by the Python script,

and to create new indicators to further investigate interesting issues that arise from the visualization of the pre-generated indicators. There were some usability issues that still need to be resolved before deploying the interface but in general, users seemed very positive about the outcome.

Some of the disadvantages of this system are that it doesn't allow as much flexibility as some other indicator generation mechanisms, such as the Python script. UrbanSim developers who are comfortable writing code cannot modify the indicator generation code through the Indicator Browser, but they can through the Python script. In addition, the Indicator Browser only displays a subset of the available indicators which do not include the difference, aggregate and disaggregate indicators, among others.

The main advantages that the Indicator Browser has over other indicator generation mechanisms is that it provides all the functionality from within one application, a web-browser, which all of our users were comfortable with. The users can also access both custom and pre-generated indicators directly through the web-browser instead of having to access some remote file server or having to remote desktop to their work machine. The users can also email the URL of the indicator's visualization page directly from the interface, as opposed to having to compose an email and attach the indicator visualization. Most importantly, urban modelers and planners are comfortable with this interface since it doesn't require any programming.

The evaluations showed that the less comfortable the users were with respect to manipulating code and the more indicators they generate on a regular basis, the more appealing they found the Indicator Browser interface. It is better for single indicator generation than the Python script since one would need to edit and rerun the script. However, the Python script provides more flexibility and functionality than the Indicator Browser.

Users found the Indicator Browser to be a good source of ready-to-hand information. They were able to see information about available scenario runs, their particular configurations, indicator scripts and documentation, available geographies and indicator visualizations all within the same interface. Usually, our users would need to access the code, several databases and documentation in order to retrieve this information.

Browsing already-computed indicator results is much faster (and more satisfying) than submitting new requests. So in future work, we will investigate additional graphical interface support for configuring and editing scripts that produce batches of indicators, which can then be browsed using the Indicator Browser or a successor tool. We also plan to expand our target users to include engaged citizens as well as urban planners. (During the evaluation of the interactive system some users mentioned they would recommend this interface to engaged citizens and activist groups.) In addition, we plan to construct a web-based tool (U-Build-It) for configuring alternate scenarios to be simulated by selecting different transportation infrastructure options, zoning changes, and the like, from a palette of alternatives. The new alternative would then be

simulated using UrbanSim and then visualized. With this system, engaged citizens will have tools that we hope will aid them in making more informed decisions and participating more fully in the urban planning process.

#### ACKNOWLEDGMENTS

We would like to thank all of the UrbanSim research team members and collaborators, in particular Janet Davis and Sandra Fan for help with the initial rounds of design and evaluation of the system described here; Batya Friedman, Janet Davis, and Peyina Lin on their work on the design of the Indicator Browser and interaction mechanism, on which this work draws heavily; David Socha for software engineering help; Bjorn Freeman-Benson, Casey Huggins, and Jonathan Fuchs for their work on the previous versions of the system; and the participants in all of our user studies. This research has been funded in part by grants from the National Science Foundation (EIA-0121326 and IIS-0534094).

#### REFERENCES

- [1] P. Waddell and A. Borning, "A case study in digital government: Developing and applying UrbanSim, a system for simulating urban land use, transportation, and environmental impacts," *Social Science Computer Review*, vol. 22, no. 1, pp. 37–51, 2004.
- [2] P. Waddell and G. F. Ulfarsson, "Introduction to urban simulation: Design and development of operational models," in *Handbook of Transport, Volume 5: Transport Geography and Spatial Systems*, P. Stopher, K. Button, K. Haynes, and D. Hensher, Eds. Pergamon Press, 2004.
- [3] B. Friedman, P. H. Kahn Jr., and A. Borning, "Value Sensitive Design and information systems: Three case studies," in *Human-Computer Interaction and Management Information Systems: Foundations*. Armonk, NY: M.E. Sharpe, 2006.
- [4] G. C. Gallopín, "Indicators and their use: Information for decision-making," in *Sustainability Indicators: A Report on the Project on Indicators of Sustainable Development*, B. Moldan, S. Billharz, and R. Matravets, Eds. Chichester, England: John Wiley & Sons, 1997, pp. 13–27, SCOPE 58: published on behalf of the Scientific Committee on Problems of the Environment.
- [5] M. Hart, *Guide to Sustainable Community Indicators*, 2nd ed. PO Box 361, North Andover, MA 01845: Hart Environmental Data, 1999.
- [6] A. Borning, B. Friedman, J. Davis, and P. Lin, "Informing public deliberation: Value sensitive design of indicators for a large-scale urban simulation," in *Proc. 9th European Conference on Computer-Supported Cooperative Work*, Paris, Sept. 2005.
- [7] B. Friedman and H. Nissenbaum, "Bias in computer systems," *ACM Transactions on Information Systems*, vol. 14, no. 3, pp. 330–347, 1996.
- [8] T. Winograd and F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*. Norwood, New Jersey: Ablex, 1986.
- [9] P. Waddell, H. Ševčíková, D. Socha, E. Miller, and K. Nagel, "Opus: An open platform for urban simulation," Presented at the Computers in Urban Planning and Urban Management Conference, London, June 2005, available from [www.urbansim.org/papers](http://www.urbansim.org/papers).
- [10] J. Nielsen, "Iterative user-interface design," *Computer*, vol. 26, no. 11, pp. 32–41, 1993.
- [11] J. Greenbaum and M. Kyng, Eds., *Design at Work: Cooperative Design of Computer Systems*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1991.
- [12] M. Rettig, "Prototyping for tiny fingers," *Communications of the ACM*, vol. 37, no. 4, pp. 21–27, 1994.
- [13] J. Nielsen. (2003) Paper prototyping: Getting user data before you code. [Online]. Available: <http://www.useit.com/alertbox/20030414.html>
- [14] J. Nielsen and R. Mack, *Usability Inspection Methods*. New York: Wiley, 1994.
- [15] A. Cockburn, *Agile Software Development*, ser. Agile Software Development Series. Addison-Wesley, 2002.
- [16] A. B. Brush and A. Borning, "'Today' messages: Lightweight support for small group awareness via email," in *Hawaii International Conference on System Sciences (HICSS 38)*, Jan. 2005.