software studio

design review: project 1

Daniel Jackson

why write stuff down?

why not just think about design and hack code?

articulating your ideas

helps you clarify them

- separate concerns
- identify key concepts
- recognize mistakes
- find ways to simplify
- get new ideas

lets you communicate

- with clients
- with implementors
- with outsourcers

helps you remember

- easy to forget why you did X
- make same mistake again

purpose & goals

Purpose and Goals

The purpose of this project is to develop a simple web analytics app, capable of tracking visits to different pages and maintaining an average over page visit durations. This tracking should be automated via JavaScript or some scripting language. The motivation for development is to get comfortable with the pertinent web technologies (Rails, HTML5+CSS, JS +jQuery+AJAX), to develop some intuition on good design (through the submission/evaluation cycle), and to practice making RESTful, DRY apps on the web.

Purpose and Goals

In this project, I created a simple web analytics service that registers individual page visits on sites that the service is tracking. Currently, many web analytics services are overly complicated for the average user who just wants to see how many people are visiting their websites, and for how long. The simplest services that currently exist are:

1. hit counters (where a website will have a small widget that shows all visitors the total visitor count so far)
2. Google Analytics

Hit counters are too public, too limited in functionality, and may make the user seem narcissistic, whereas Google Analytics may have too much data and jargon that the average user doesn't need. This project aims to serve as a middle ground between the two options.

1. Overview

WebAnalytics 1.2 is an application designed to track the frequency and duration of visits to any website. Services such as Google Analytics or Stat Counter provide similar services, but WebAnalytics makes it simple to obtain the most critical tracking data quickly. As opposed to more heavy-weight solutions, this application requires no account registration.

Overview:

In this project, I built an analytics website which allowed companies to track their user data. The purpose was to track visits to various pages of a company's website, along with some visit duration, and display the data to the companies. This service is useful for companies that want to track their data, see how often certain pages are viewed, and adjust their architecture/advertising accordingly. Ideally, this analytics will have many more features including some graphics and more customization tools.

Overview

My solution for project one had one goal in mind: make it easy for the user. With this in mind I decided that users do not have to register their site with us before using the service; instead, if I encounter a request from a site I haven't seen before I create the necessary entries into the database to handle requests from this site in the future. This means site "registration" is automatic.

I. Brief description of system to be built

The system is a web analytics service that provides people who run websites with data about the frequency and average duration of visits.

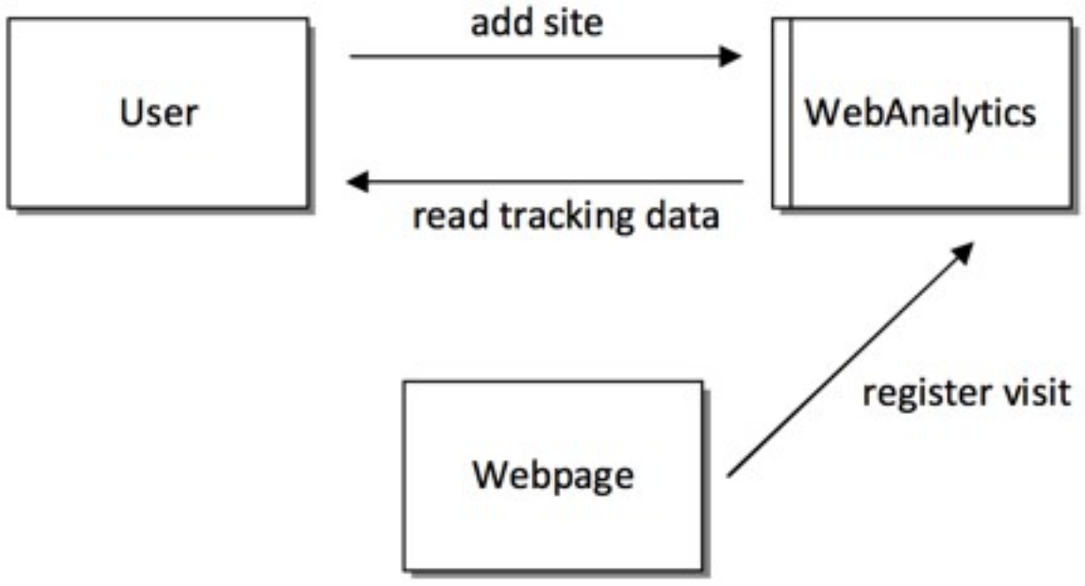
II. Key goals and purpose

The goal is to keep track of which sites have been visited, which pages of said sites have been visited, how many times and for how long on average. The purpose is so that this service will function as a basic, working analytics engine.

III. Motivation for development (e.g. deficiencies of existing solutions)

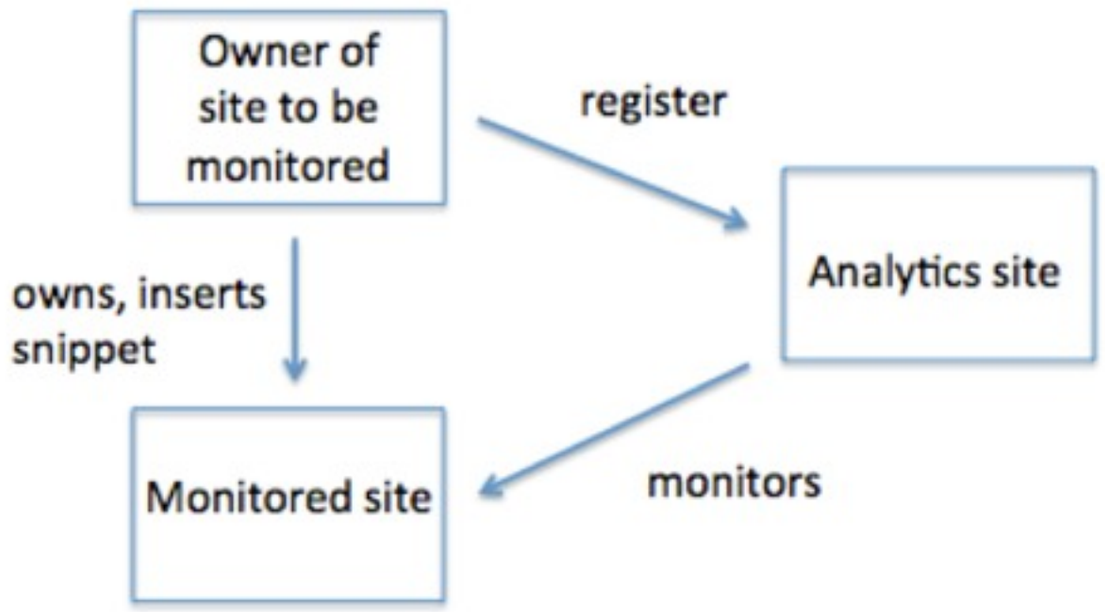
The motivation for development is to learn more about Rails, Cross-Origin Resource Sharing, JavaScript, separation of concerns, HTTP requests, etc. This project's minimum requirements allow the developer to sample a little bit of all of these, while still providing a useful service.

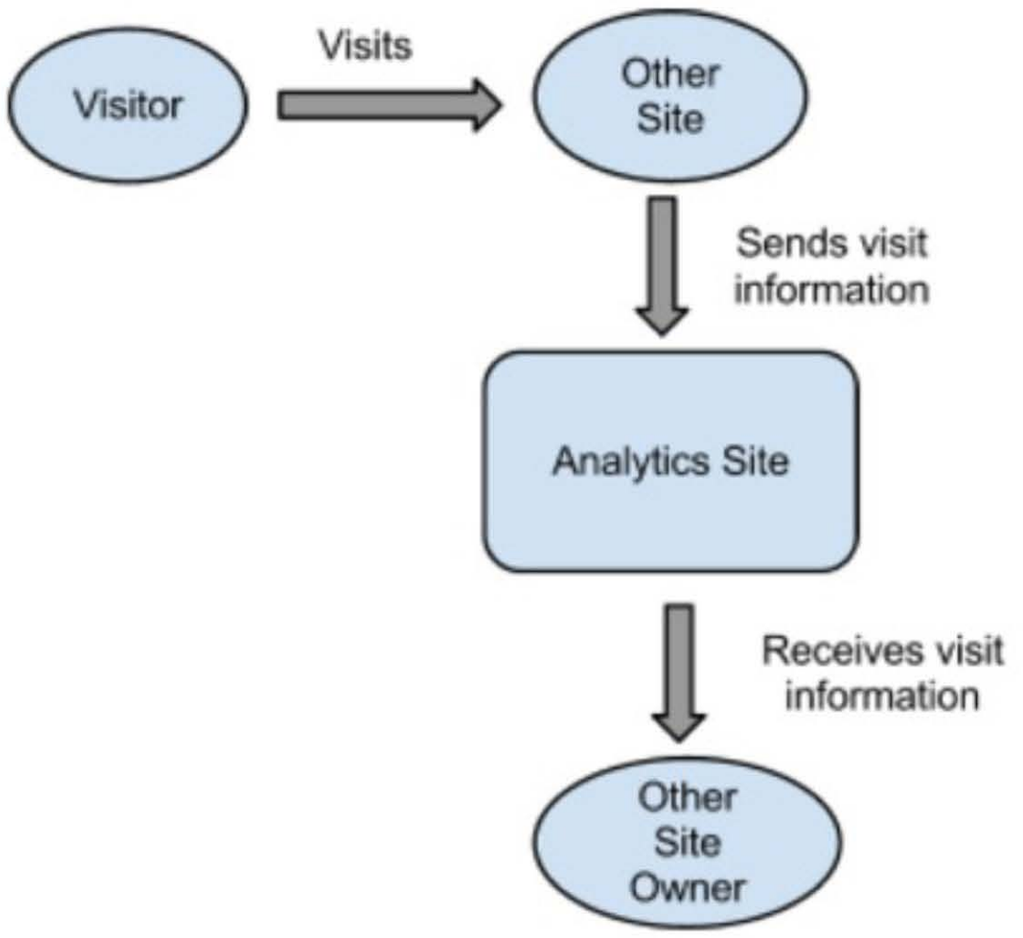
context diagram



show user of webpage too?
page vs server?

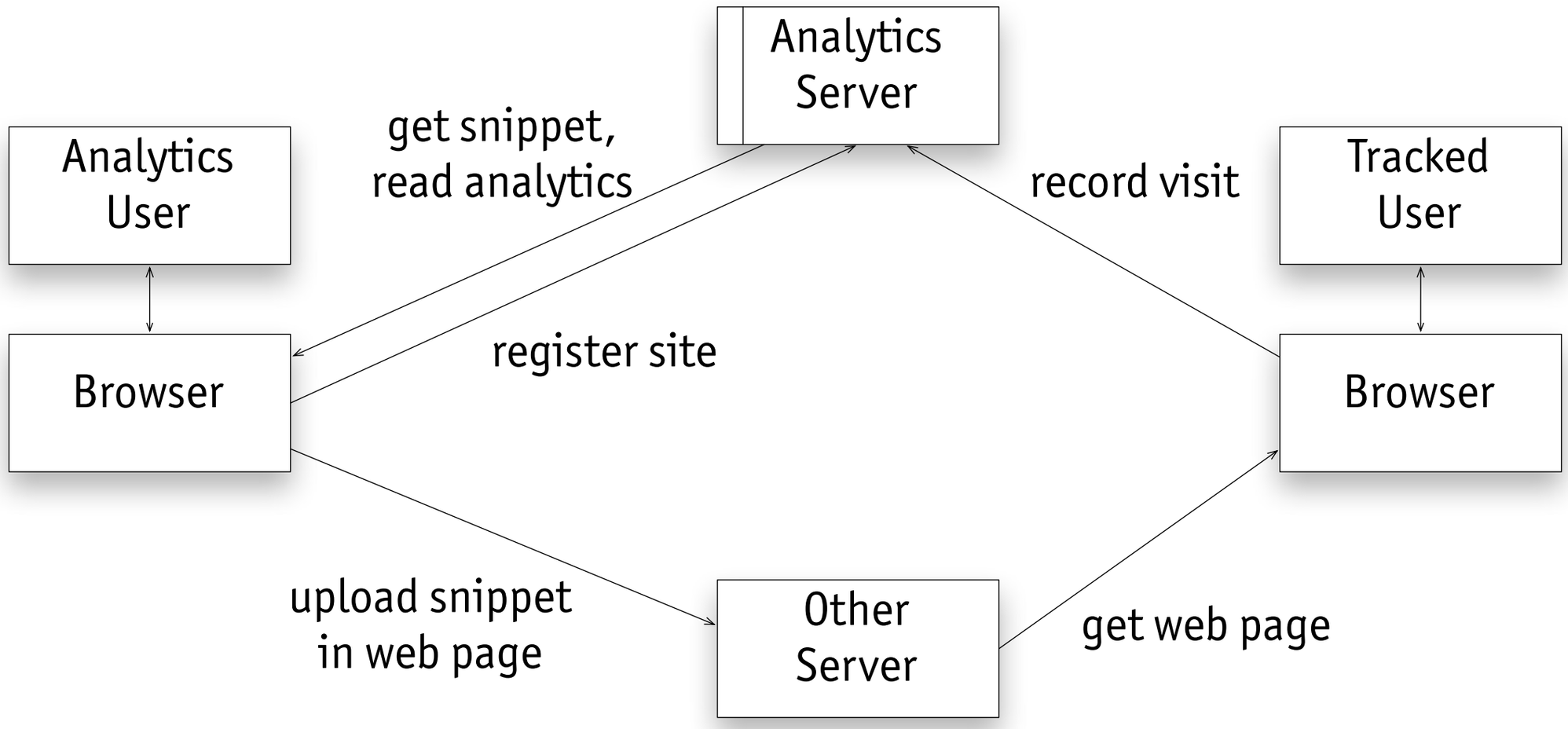
what do arrows indicate?
reading visit data?





includes visitor!
what do arrows mean?

a context diagram



concepts

The analytics site models the following concepts:

- **Site:** Represents one website and contains pages, which track visits individually.
- **Page:** Represents a page on a website. Tracks visits to that page.
- **Visit:** Represents a visit to a page. Tracks the duration of the visit as well as the IP address of the visitor.

The key concepts are:

a. A *site* is a collection of *pages* under a common domain or path.
b. A *page* is a web document, usually formatted in HTML
c. A *visit* is an abstraction over a HTTP (GET) request by a user of an HTML page. A visit is said to *start* when the page requested has been received and loaded into a viewer (e.g. browser). A visit is said to *end* when the page is no longer being used (e.g., window or tab is closed).

Key Concepts

The key concepts in the design are sites, pages, and visits. A site consists of a set of pages, and both a site and its pages are created by the user via form submission. There is no required correlation between the names of the sites and pages in the web analytics service, and the actual names of the site and pages the user wants to track. So, for example, if a user wants to track the pages www.google.com, maps.google.com and docs.google.com, they can create a site named "Google" and pages called "Search", "Maps", and "Docs". The user creates their own mapping between the webpages they administer and the names of the sites and pages in the analytics service.

2. Concepts

The key concepts in the design of WebAnalytics are **sites**, **pages**, and **visits**. A site consists of multiple pages (zero or more). Each page in turn can be associated with multiple visits. Visits include information specific to each individual visit on the webpage, such as duration. The object model is shown below.

some redundant info

Concepts

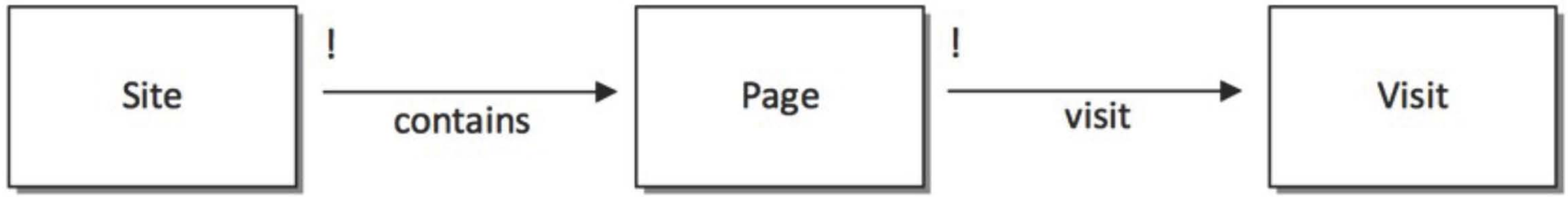
The key concepts in the design of SWA are sites, pages and visits. A site consists of a set of pages and each page contains multiple visits. A visit has a start and an end time.

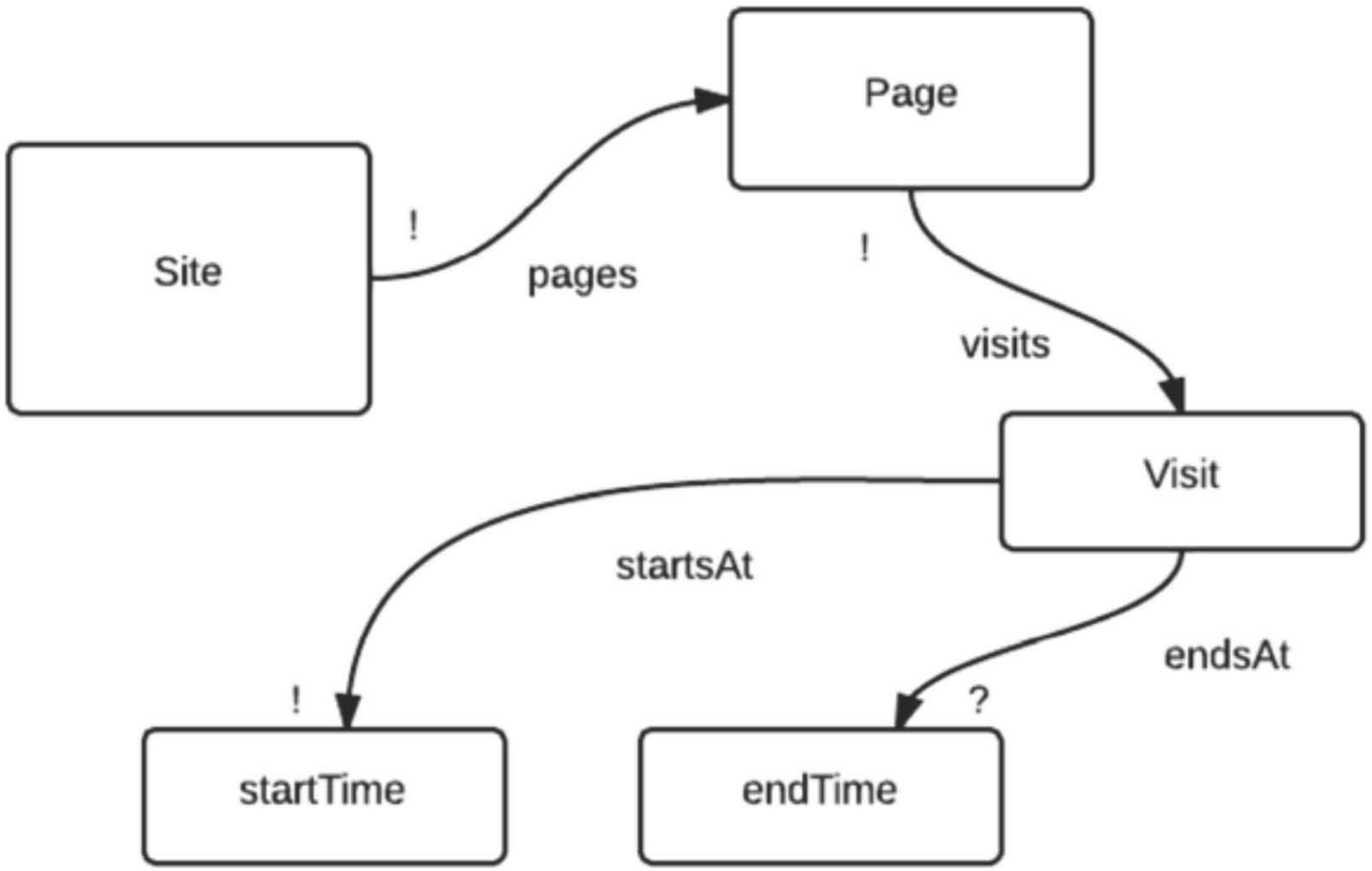
2 Concepts

The key concepts in the design of Web Analytics are Sites, Pages which are associated models. Each site has many pages and can register many visits. A page can only belong to one site and has many visits. The main differnce between a site and a page is their string names; a site contains the domain name of a website and a page, the website's path. A visit can only belong to a page and in turn the page's associated site. I use polymorphic association to allow visit to neatly belong to site and page on a single association. This is possbile by assigning to each visit a visitable_id and a visitable_type to differentiate if a visit belongs to a page or a site. I make this distinction so that we can we can optimize on our database by avoiding duplicated entries. The con of this approach is that it requires more iterated steps though pages to compute the average duration of visits to a site. I will discuss later how we can overcome this. Figure. 1 shows the relations between the models. Note that I use the arrow notation similar to disjoint sets to indicate that a visit can only belong to either a site or a page but not both although semantically, a visit to a page is also a visit to a site.

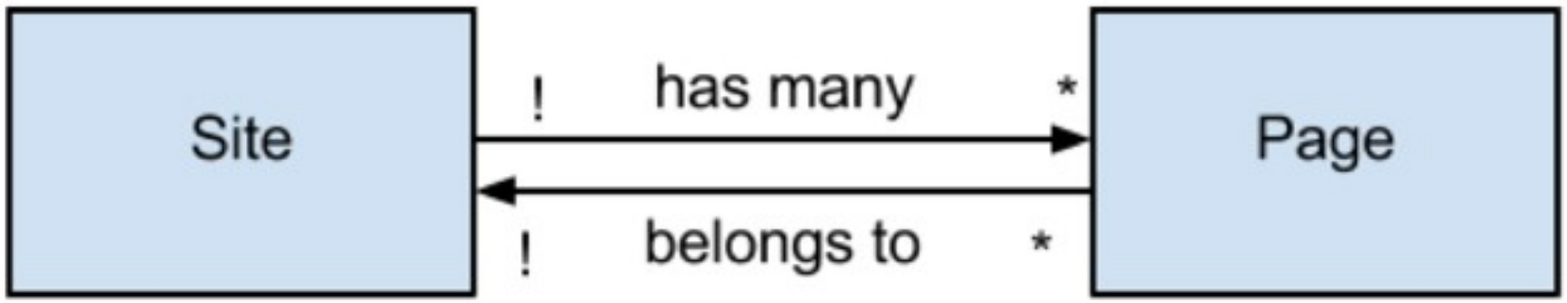
need separation of concerns!

object model

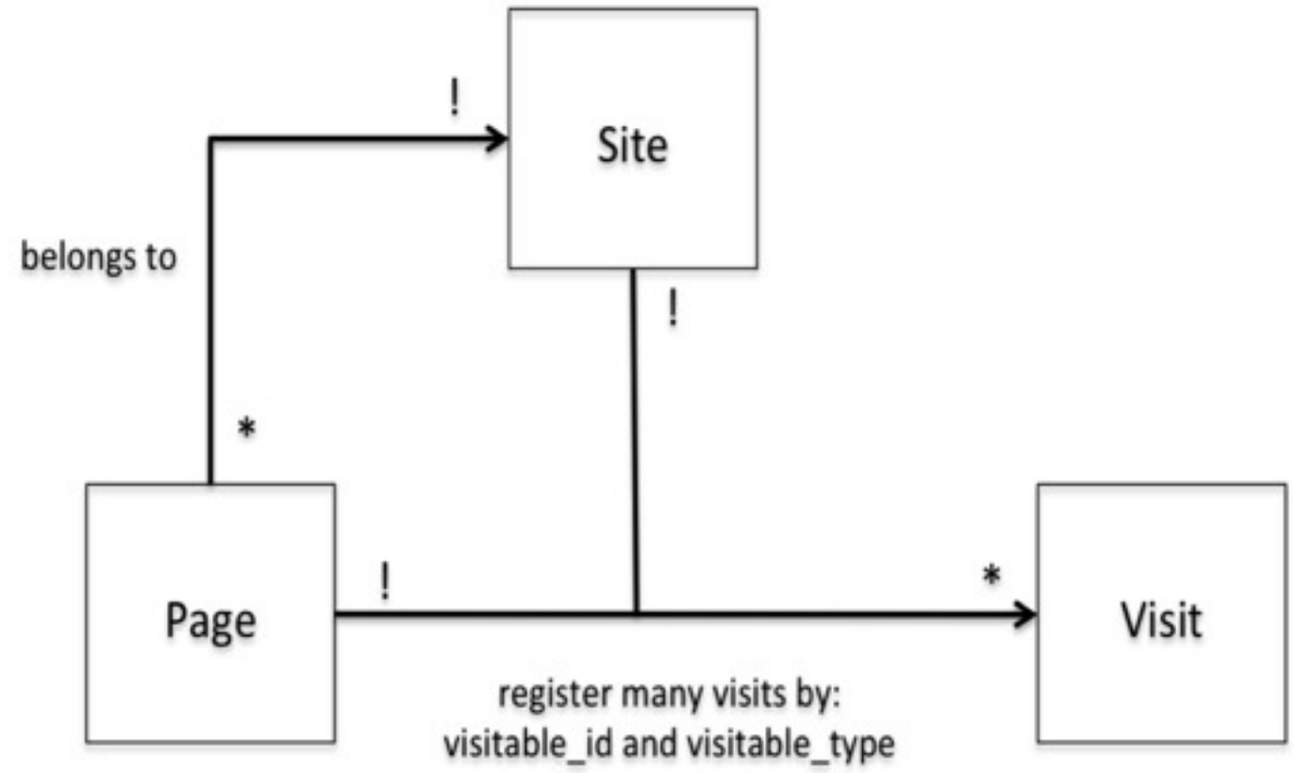




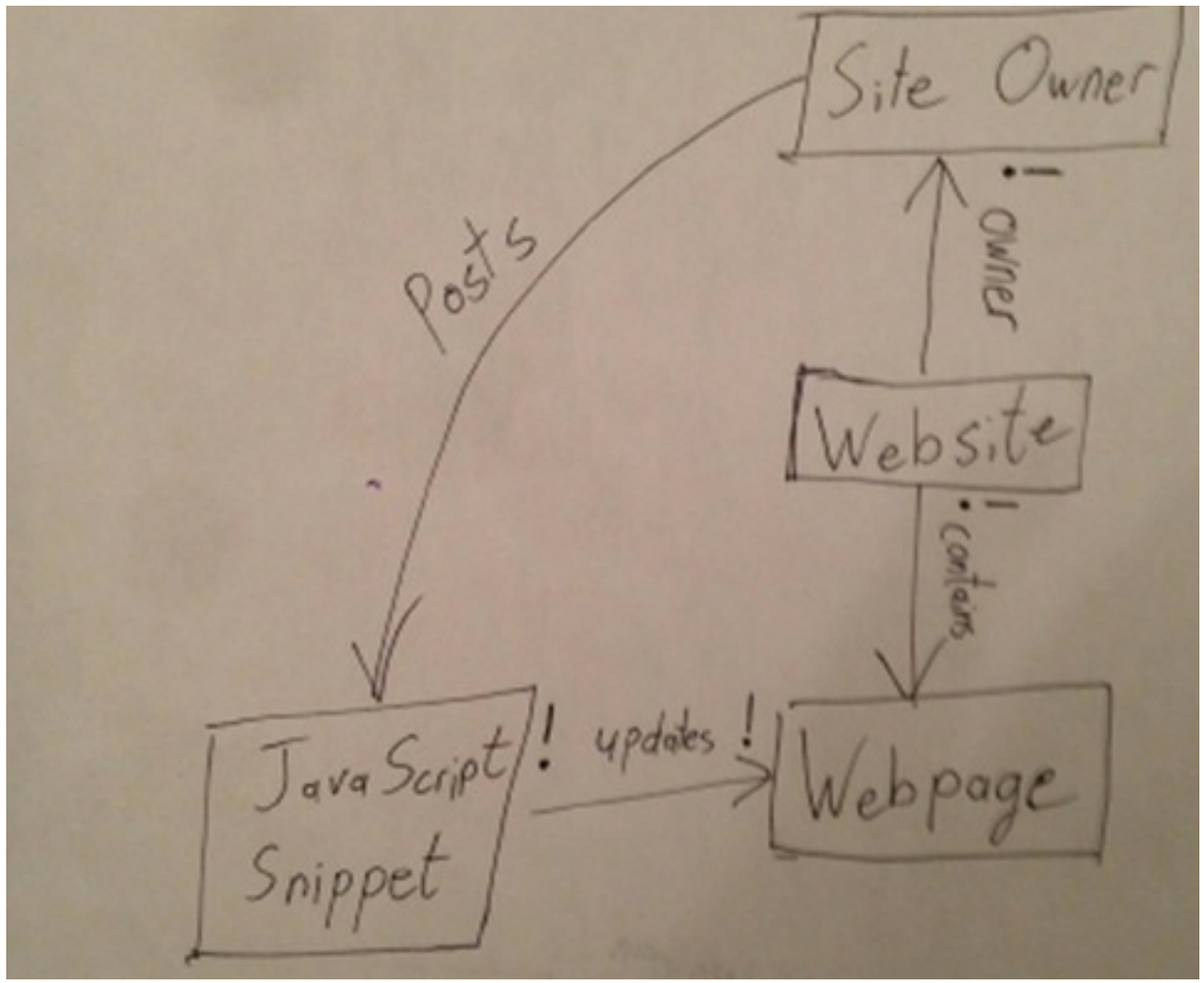
should have sets, not slots:
conversion to model classes will fail



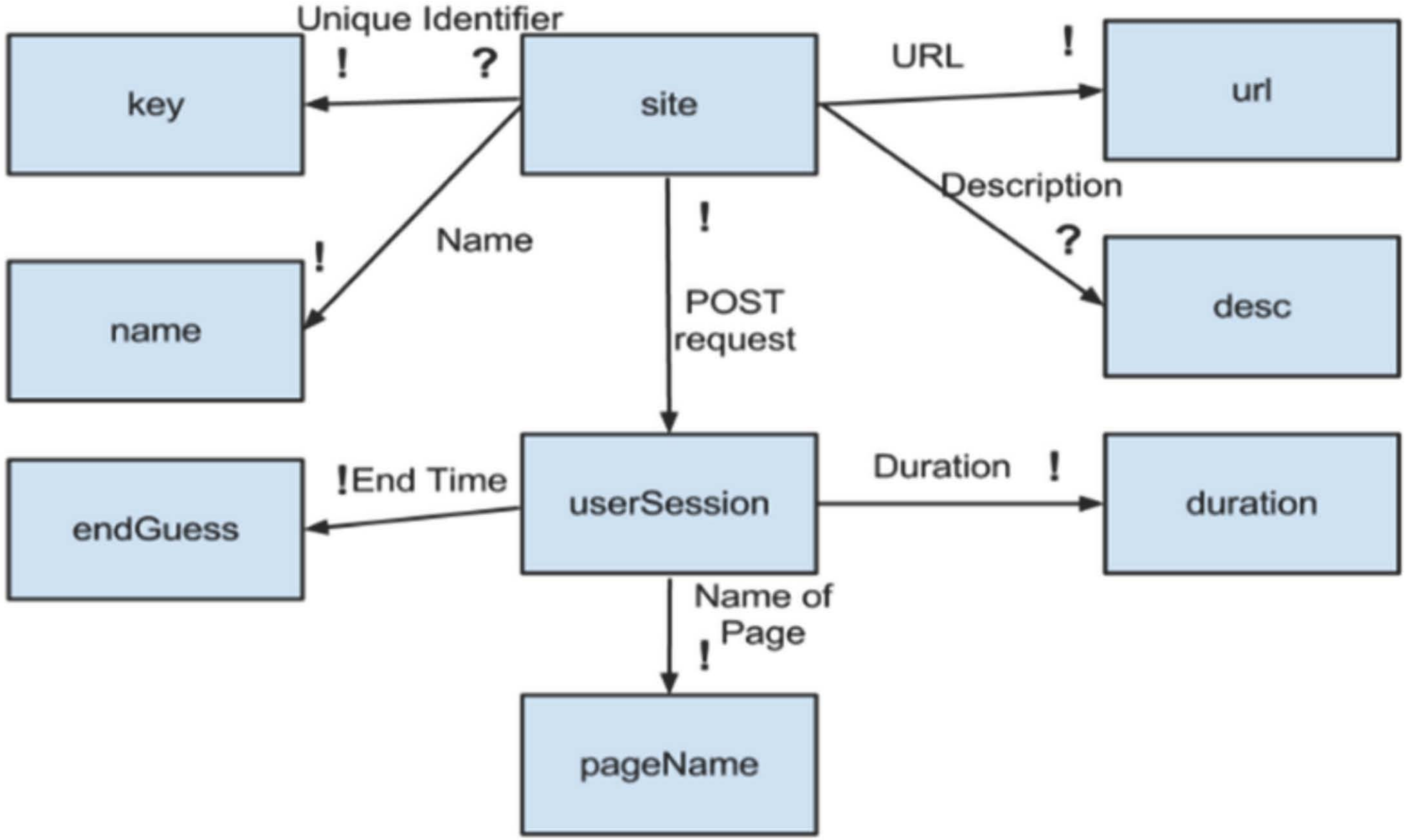
implementation details



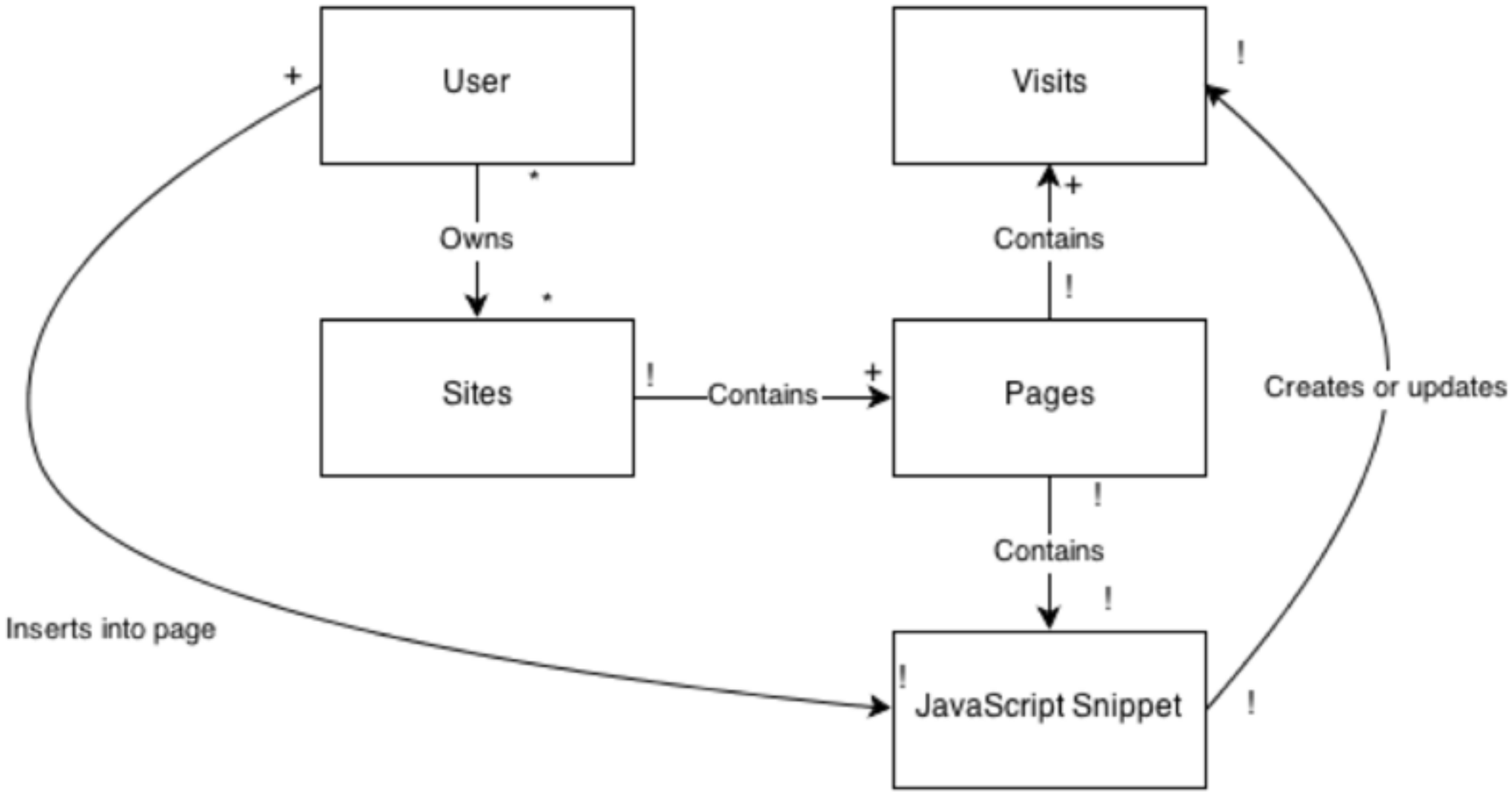
funky syntax



remember: OM describes data model; gets transformed into Rails model classes



POST relation?



cool, but not a data model

what goes in the OM?

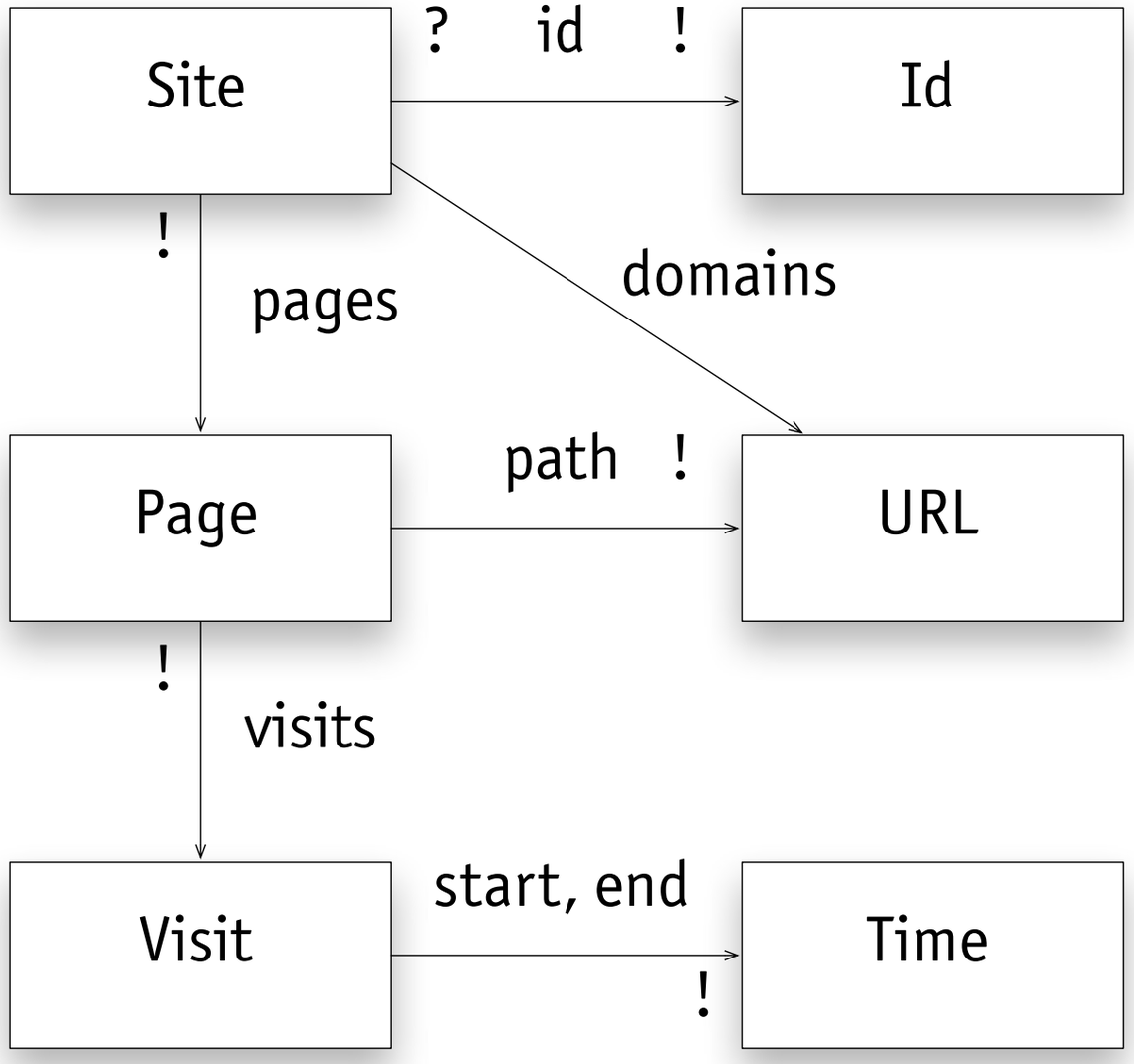
exclude

› non-state elements (eg, actions by user)
› small details (eg, profile attributes with no effect on function)

include

› major concepts that are part of the state
› state components needed to describe key functions
› external names and ids that play a behavioral role

an OM to discuss



challenges

Challenges

Problem: privacy. If there are multiple users of the web analytics service, how will we prevent them from seeing each others analytics?
Options available:
--Create a User resource which contains zero or more sites, such that each site belongs to exactly one user. This adds complexity and makes it more difficult for a user to begin using the service. It also doesn't exactly achieve privacy, because other users could simply url-hack to find this user's data
--Ignore this problem. This option was chosen, with the philosophy that this web analytics app will be a semi-public service. Users can choose to insert the javascript snippets, but by doing so they understand that the analytics information for their websites will be public knowledge. This could be useful, for example for aspiring bloggers or small businesses wanting to increase awareness of their site.

Problem: How will the duration of a visit be calculated?
Options available:
--Javascript snippet makes one call to the server, at the end of a visit, which includes the visit duration. The javascript snippet is simpler. The duration format may have to be converted on the server.
--Javascript snippet makes two calls to the server, once at the beginning of a visit and again at the end. The server records the start and end time of the visit. The duration can be calculated from these times. This provides consistency in all measurements—otherwise durations may be influenced by the speed of the browser that was used, for example. We chose this option.

Problem: How will visits be represented?
Options available
--Each Page keeps track of the number of visits and their duration, without keeping a separate object/database row for each visit. This reduces the size of the database significantly. However, it leaves little room for expansion later.
--Each visit is a new entry in a database of visits, containing the start and end time of the visit. This provides the same functionality as above, but is much friendlier to future development. If we wanted to add a feature to graph visits over time, or give the user the option to view only visits in the last n days, or even tell the user which day of the week or hour of the day their site is most commonly visited, we have all the data necessary to implement those features. So, despite being more expensive in terms of database size, we went with this option in order to avoid throwing away data which may be useful in the future.

Problem: How will users obtain the javascript snippet?
Options available:
--There is one javascript snippet which will work no matter what webpage it is pasted into. The snippet provides the server with information about the URI of that website. This forces the name of the Site and Pages in the web analytics service to be the same as the hostname and pathname of the webpages on the web. We already decided we wanted to decouple these. This does allow the set of pages being tracked to be dynamic—if a call comes in from a page or site that doesn't yet exist in the service, it would be created. But given our privacy decision, think it's better that users must create their sites/pages in the system before they can be tracked.
--When each page in each site in the web analytics service is created, it produces a specialized javascript snippet which already contains the correct page and site id numbers in the XMLHttpRequests. By calling the site and page by id number rather than name, this decouples the name of the site and page in the web analytics service, and the javascript used to register a visit to that page, which is what we wanted.

Registering Visits: To track visit duration on pages via Javascript, I considered the following options:

1. Send two AJAX calls to the server: the first when a client first loads the page, and the second when the page is unloading. Here the server handles finding the visit duration.
2. Send one AJAX call to the server at the end when the page is unloading, and pass the visit duration as a parameter of the request.

To reduce the effects of network problems, I decided to implement Option 2. The Javascript snippet gets a timestamp when the page is loaded, and upon its unloading, it sends the AJAX call. It also sends the visit duration to the server as the difference in start and end times. The downside to this implementation is that (1) the server must parse this millisecond time difference and save it in the desired format and (2) clients can easily modify the Javascript snippet to change/skew the duration values for their tracked sites.

However, I chose this option because Option 1 relies on the fact that the network connection remains open between the times that the client loads the page and then unloads it. If the connection is dropped, there is the possibility that a visit never "ends," and so the server must handle these special cases and not include them in calculations for visit duration. Also, once the server receives a request upon the loading of a page, it must send the client some sort of state information (such as the visit id) so that the server can record the ending of the corresponding visit. The benefits of this option, however, are that the server exclusively handles the calculation of visit duration, preventing clients from tampering with it. For the purposes of this project, I placed more emphasis on the application's ability to work even in spite of network disconnections in the middle of visits, and so I chose Option 2.

3. Challenges

Fake hits: As currently implemented, if one knows the site and page id of a page, he can simply type the url: '/sites/:site_id/pages/:pages_id/visit?duration=*some_number*' and fake a visit to the page. To combat this, there should either be a way to authenticate that the monitored page was actually opened, or the url for a page visit should not include page info; rather, the parameters of the page to update should be hidden in the body of the message.

- **Don't store visit objects.** We decided not to store information for each visit in our database for several reasons. We thought this might not scale well with our product, as we will have too many visit objects for each website and many websites as well. Particularly when the only essential information we were interested from these visit objects was just the time duration. The rest we calculated from the total amount of visits and total time spent in site; which can both be stored as numbers in the site object.
- **Communicate with controller when page closes.** Decided to only send the HTTP POST request to our server when a visit is over and the page is closed successfully. This minimizes the client-server communication, reduces load on client and server; and as a result we thus do not consider visits from crashed browsers, which we think as fine as this does not happen too often and may very well not be of interest for our customers.
- **POST Request.** Other client-side decisions involved deciding to use a POST request rather than a GET request. Simply, the request that we were doing is intended to update some data on our server, so it was more aligned with using a POST. Also, we decided to compute the visit duration on the client using the Date module from javascript; and we sent the duration as microseconds, just as it came. This in turned implied that we decided to parse this information at the server and changing it to seconds.

6.170 Software Studio
Spring 2013