XJCO3011: Web Services and Web Data

Session #4 –

Trying to Bridge the Sematic Gap

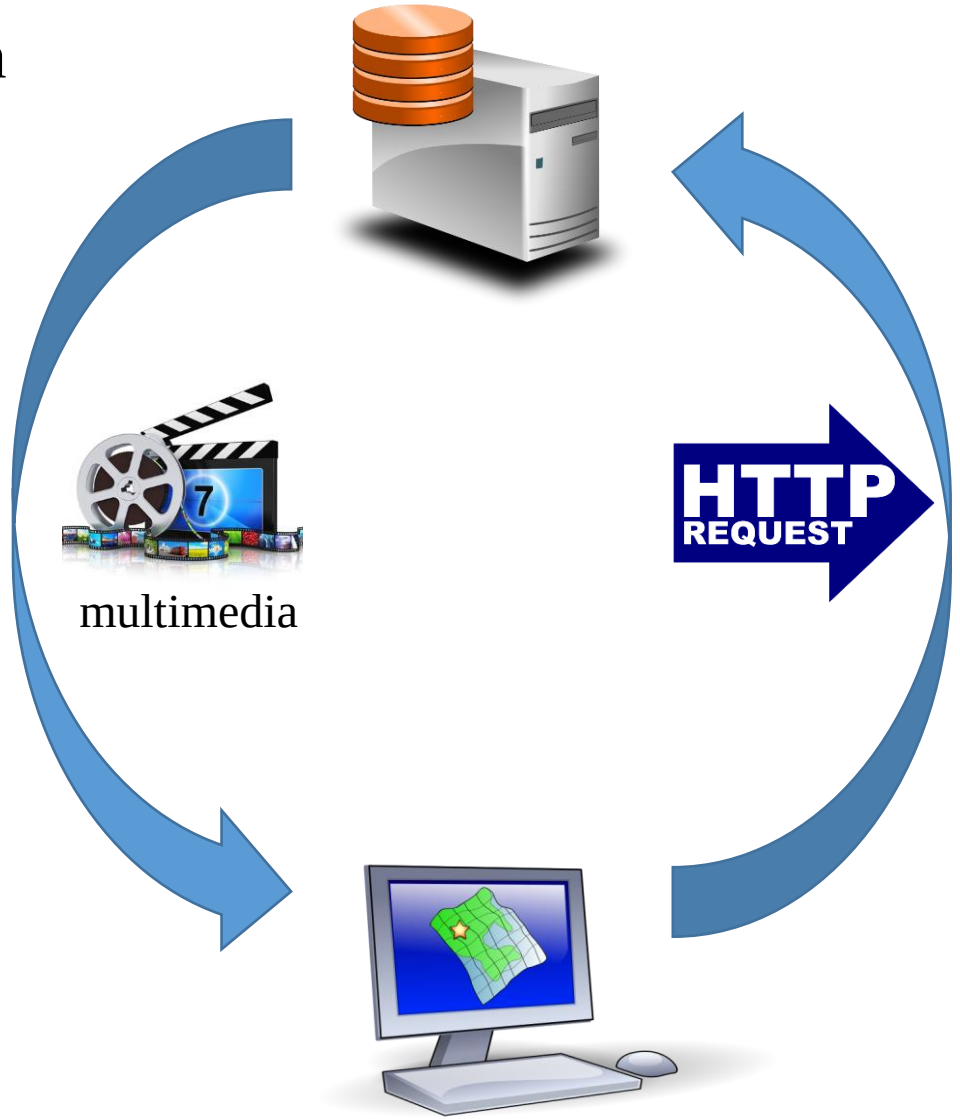
Instructor: Dr. Guilin Zhao

Spring 2023

SOUTHWEST JIAOTONG UNIVERSITY • UNIVERSITY OF LEEDS
西南交通大学－利兹学院
SWJTU-LEEDS JOINT SCHOOL

The Problem

- Representational State Transfer (REST) works when clients:
  1. request representations of resources,
  2. parse the representation (the multimedia),
  3. recognise control structurers that leads to further resources, e.g. HTML <a> tag,
  4. decide on a new resource to be requested, and
  5. change their current state when the new resource is received.
- Steps 1, 2, 3, and 5 are straightforward and can be easily implemented in fully autonomous code.
- Step 4 however is difficult and requires the client to have knowledge of the application domain. This is what we called the *Semantic Gap*.



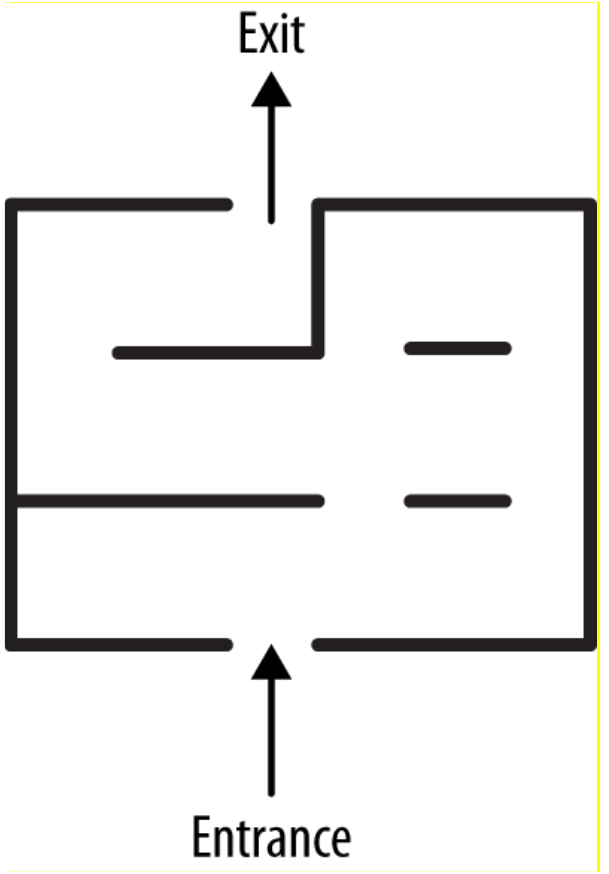
Request, Parse, Recognize, Decide

Semantic Gap

- It refers to the difference or mismatch between the way computers process and understand digital data and the way humans perceive and interpret that same data.
- Humans are able to understand and interpret information based on their senses, such as sight, hearing, touch, taste, and smell.
- In contrast, computers rely on algorithms and mathematical calculations to process and interpret digital data.
- As a result, there is often a gap between the way humans perceive and interpret digital data, and the way that computers do.
- This gap can cause problems in areas such as image recognition, natural language processing, and audio signal processing, where computers may struggle to accurately interpret data in the same way that humans do.

Bridging the Semantic Gap with a Domain-Specific Design

- We will design a web API for representing mazes
- The server's job will be to invent mazes like this one and present them to clients. But why this example?
- Because any complex problem (e.g., modifying complex insurance policies; selecting products from a catalog and paying for them) can be represented as a hypermedia maze that the client must navigate.
- These problems have commonalities:
  - ✓ The problem is too complex to be understood all at once, so it's split up into steps.
  - ✓ Every client begins the process at the same first step.
  - ✓ At each step in the process, the server presents the client with a number of possible next steps.
  - ✓ At each step, the client decides what next step to take.
  - ✓ The client knows what counts as success and when to stop.



The Maze+XML Media Type

- How can we represent the shape of a maze in a format that's easy for a computer to understand?
- There's a personal standard called Maze+XML, for representing mazes in a machine-readable format.
  - ✓ Maze+XML : the technology for migrating maze games to the web. It uses XML (Extensible Markup Language) to describe the maze so that web browsers can read, display, and manipulate it. With Maze+XML, developers can create complex maze games and easily modify and adjust game rules and elements (i.e., providing developers with a flexible and scalable way to create and deploy maze games).
- The media type of a Maze+XML document is application/vnd.amundsen.maze+xml.
  - ✓ application/vnd.amundsen.maze+xml is a MIME type for a Maze+XML file. MIME types are standards used to indicate the format and type of files transmitted over the internet. In this case, application/vnd.amundsen.maze+xml indicates that the file is a Maze+XML file, defined and supported by the Amundsen company. If a program supports this MIME type, it can read and process the Maze+XML file and generate and manipulate maze games based on the information described in the file.
  - ✓ By defining specific MIME types, developers can make their applications more accurate in identifying and processing different types of files. Therefore, application/vnd.amundsen.maze+xml is an identifier with a specific meaning and purpose, used to represent the type and format of a Maze+XML file.）

The Maze+XML Media Type (Cont'd)

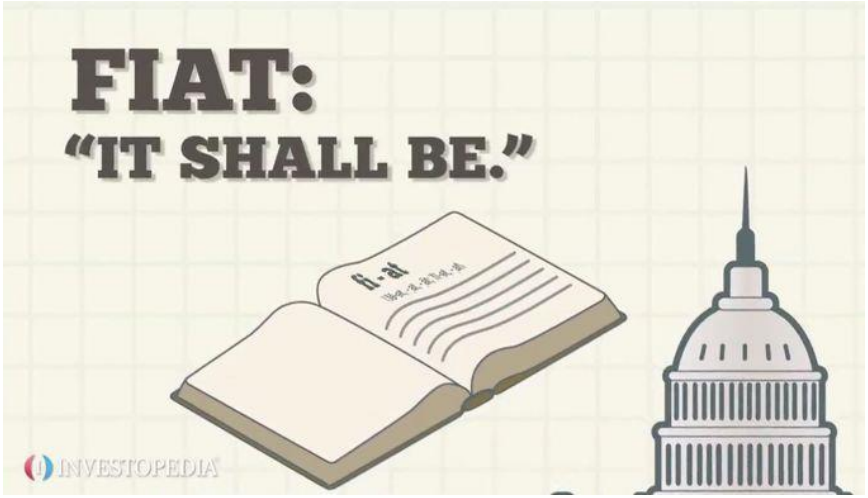
- This is one way a domain-specific design meets the semantic challenge: **by defining a document format that represents the problem** (such as the layout of a maze), **and by registering a media type for that format**, so that a client knows right away when it has encountered an instance of the problem.
- In general, it is not recommended to create new domain-specific media types. It is usually less work to add application semantics to a generic hypermedia format.
- If you set out to do a domain-specific design, you will probably end up with a fiat standard that doesn't take advantage of the work done by your predecessors.
- But a domain-specific design is the average developer's first instinct when designing an API.

The Structured Syntax Suffix

- A suffix is an augmentation to a media type to specify the underlying structure (e.g. syntax) of that media type.
- For example, MAZE+XML means that this media type uses XML syntax but it has its own defined semantics.
- Structured syntax suffix types are registered and maintained by IANA (Internet Assigned Numbers Authority)
- The currently registered suffixes are: +xml, +json, +ber, +der, +fastinfoset, +wbxml, +zip, and +cbor

Fiat Standards

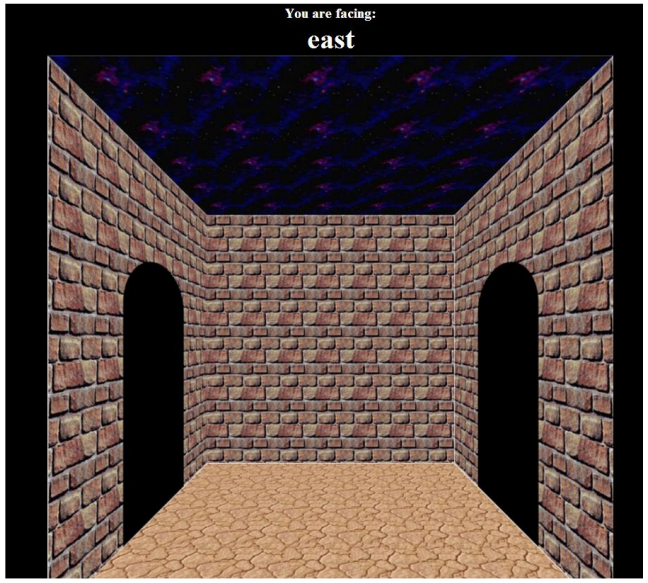
- Fiat standards are not really standards; they are behaviours. No one agreed to them. They're just a description of the way somebody does things.
- The behaviour may be documented, but the core assumption of a standard—that other people ought to do things the same way—is missing.
- Pretty much every API today is a fiat standard, a one-off design associated with a specific company. That's why we talk about the "Twitter API," the "Facebook API," and the "Google+ API."
- Even under ideal circumstances, your API will be a fiat standard, since your business requirements will be slightly different from everyone else's. But ideally a fiat standard would be just a light gloss over a number of other standards.

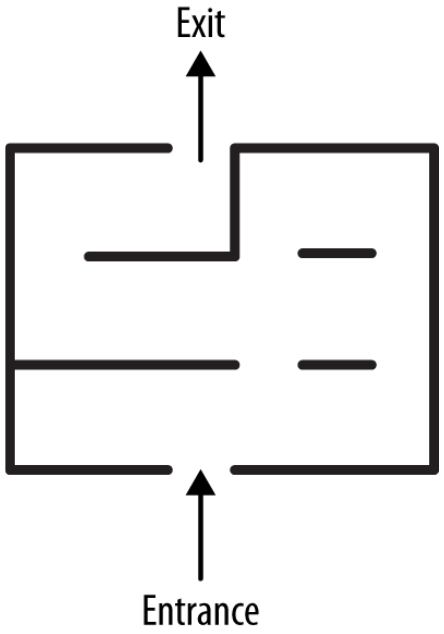


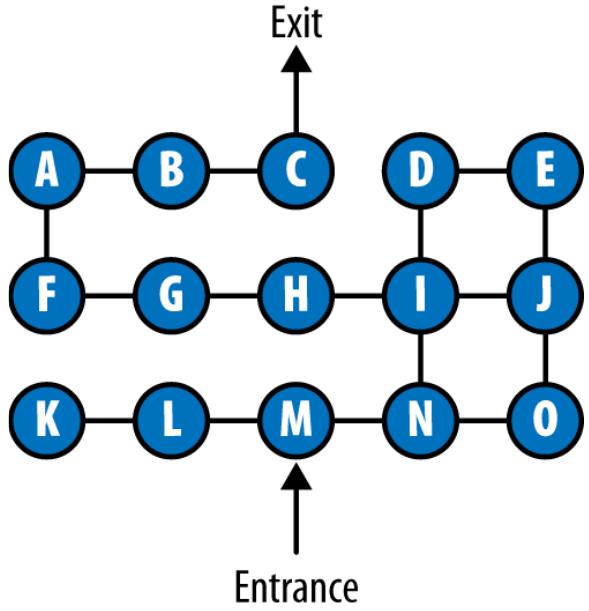
- Read more： page xxii – xxiii in RESTful Web APIs by Leonard Richardson and Mike Amundsen

How Maze+XML Works

- Upon entering the maze, you'd see something like the figure, a wall in front of you and the entrance behind you. But you'd have two choices: go left or go right. You'd have no way of knowing which direction would take you to the exit.
- The Maze+XML format simulates this rat's-eye-view by representing a maze as a network of "cells" that connect to each other. I've chopped the maze into a grid and created a cell for each grid square.
- Maze+XML cells connect to each other in the cardinal directions: north, south, east, and west.



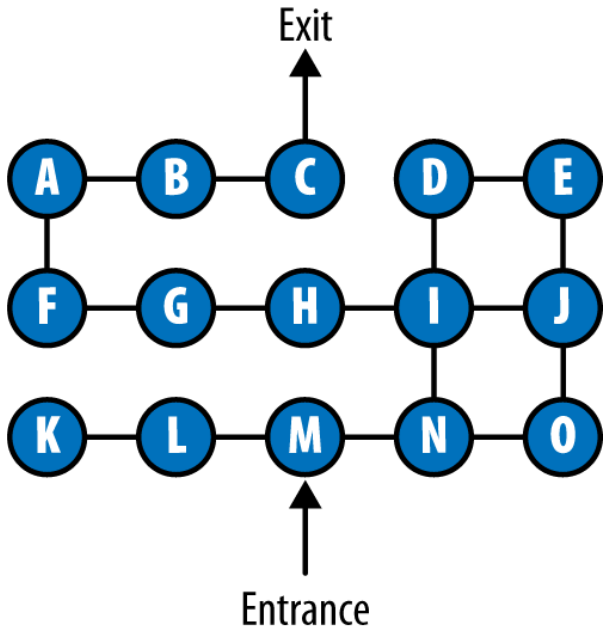




The Representation

- Each cell in a Maze+XML maze is an HTTP resource with its own URL.
- If you send a GET request to the first cell in this maze, you'll get **a representation** that looks like the one below.
- This representation includes a human-readable name of the cell, "The Entrance Hallway," like you'd see in an old, text-based adventure game.
- The representation also includes <link> tags that connect this cell to its neighbours. From cell M, you have a choice of going west to cell L or east to cell N.

```
<maze version="1.0">
 <cell href="/cells/M" rel="current">
  <title>The Entrance Hallway</title>
  <link rel="east" href="/cells/N"/>
  <link rel="west" href="/cells/L"/>
 </cell>
</maze>
```



Link Relations

- This representation shows off a powerful hypermedia tool called *the link relation*
- By themselves, rel="east" and rel="west" don't mean anything. A computer doesn't know the words "east" and "west."
- But the Maze+XML standard defines meanings for "east" and "west" and developers can program those definitions into their clients
  - ✓ east: refers to a resource to the east of the current resource. When used in the Maze+XML media type, the associated URI points to a neighboring cell resource to the east in the active maze.
  - ✓ west: refers to a resource to the west of the current resource. When used in the Maze+XML media type, the associated URI points to a neighboring cell resource to thewest in the active maze.
- In Maze+XML, following a link marked with the link relation east will move your client east through some abstract geographical space.
- This is how Maze+XML meets the semantic challenge: **by defining link relations that convey its application semantics.**

Link Relations (Continued)

- A link relation is a string associated with a hypermedia control (e.g., <link> tags).
- It explains the change in application state (for safe requests) or resource state (for unsafe requests) that will happen if the client triggers the control.
- Link relations are managed by the Internet Assigned Numbers Authority (IANA).
- Currently, it contains about 60 link relations that have been deemed to be generally useful and not tied to a particular data format
- The simplest examples are the next and previous relations, for navigating a list.
- RFC 5988 (which is a document that defines the Web Linking protocol for the World Wide Web) defines two kinds of link relations: registered relation types and extension relation types.
  - ✓ Registered link relations look like the ones you see in the IANA registry: short strings like next and previous.
  - ✓ Extension relations look like URLs. If you own mydoma.in, you can name a link relation http://mydoma.in/whatever and define it to mean anything you want.

Example: HTML Link Relations

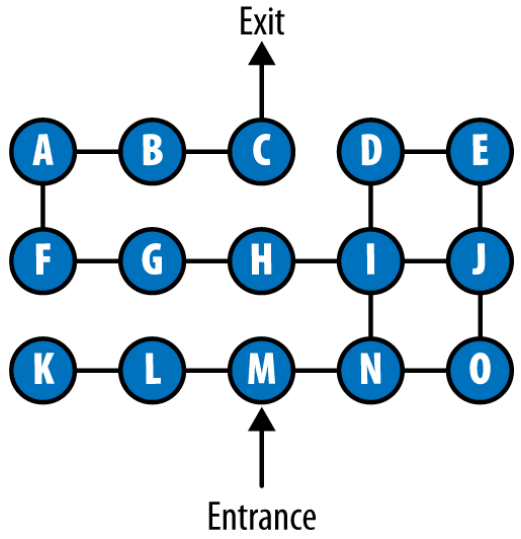
```
<a rel="nofollow" href="http://www.functravel.com/">Cheap Flights</a>
```

Value	Description
alternate	Provides a link to an alternate representation of the document (i.e. print page, translated or mirror)
author	Provides a link to the author of the document
bookmark	Permanent URL used for bookmarking
external	Indicates that the referenced document is not part of the same site as the current document
help	Provides a link to a help document
license	Provides a link to copyright information for the document
next	Provides a link to the next document in the series
nofollow	Links to an unendorsed document, like a paid link. ("nofollow" is used by Google, to specify that the Google search spider should not follow that link)
noreferrer	Requires that the browser should not send an HTTP referer header if the user follows the hyperlink
noopener	Requires that any browsing context created by following the hyperlink must not have an opener browsing context
prev	The previous document in a selection
search	Links to a search tool for the document
tag	A tag (keyword) for the current document

Follow a Link to Change Application State

- A client can “go east” from cell M by following the appropriate link (i.e. by sending a GET request to the URL labelled with rel="east").
- When the client does this, it will get a second Maze+XML representation, looking something like this:

```
<maze version="1.0">
 <cell href="/cells/N">
  <title>Foyer of Horrors</title>
  <link rel="north" href="/cells/I"/>
  <link rel="west" href="/cells/M"/>
  <link rel="east" href="/cells/O"/>
 </cell>
</maze>
```



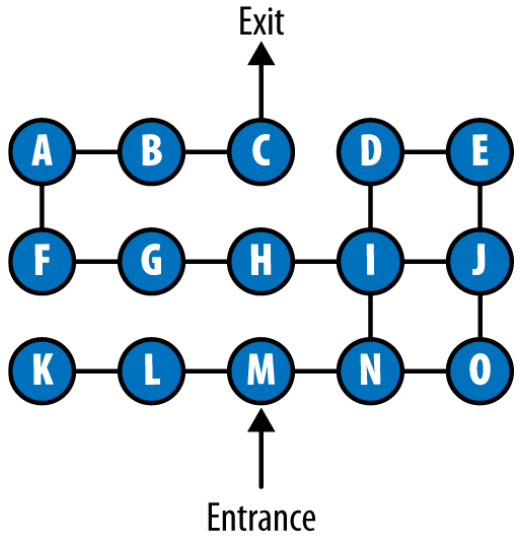

```
<maze version="1.0">
 <cell href="/cells/M" rel="current">
  <title>The Entrance Hallway</title>
  <link rel="east" href="/cells/N"/>
  <link rel="west" href="/cells/L"/>
 </cell>
</maze>
```

- This is the Maze+XML representation of cell N on the map. It links back to cell M (using the link relation west), as well as to cells I (north) and O (east).
- The client’s application state has changed. To borrow a term from the HTML standard, *the client was “visiting” cell M, and now it’s “visiting” cell N.*
- The client now has three new options, represented by the links in the representation of cell N.

Finally

- By following the right links (north, west, west, west, north, east, east, and finally east), a client can make its way from cell N to cell C. That cell includes the exit to the maze, indicated here by a <link> tag with the link relation exit:

```
<maze version="1.0">
 <cell href="/cells/C">
  <title>The End of the Tunnel</title>
  <link rel="west" href="/cells/B"/>
  <link rel="exit" href="/success.txt"/>
 </cell>
</maze>
```



- Here's what the Maze+XML standard says about exit:
  - ✓ exit: refers to a resource that represents the exit or end of the current client activity or process. When used in the Maze+XML media type, the associated URI points to the final exit resource of the active maze.
- Maze+XML provides no guidance as to what should appear at the other end of an exit link. It's a "resource," which means it can be anything at all. In this implementation, we've chosen to link to a textual congratulatory message (success.txt).

The Collection of Mazes

- Cell C leads out of the maze, because its representation includes a special link with rel="exit". But cell M, the entrance to the maze, doesn't include anything to distinguish it from the other fourteen cells.
- There's no rel="entrance". Cell M's title is "The Entrance Hallway," but that phrase doesn't mean anything to a computer. How do we bridge the semantic gap? How is the client supposed to know where to start the maze?
- The Maze+XML standard solves this problem with a collection: a list of mazes. If you send a GET request to the root URL of the maze API, you might get a Maze+XML representation that looks like this.
- A collection in Maze+XML is a <collection> tag that includes some <link> tags with the link relation maze.

```
<maze version="1.0">
 <collection>
  <link rel="maze" title="A Beginner's Maze" href="/beginner">
  <link rel="maze" title="For Experts Only" href="/expert-maze/start">
 </collection>
</maze>
```

To Get Started

Send a GET request to a URL labelled with the relation maze (e.g. /beginner), and you'll get a representation that looks like this:

```
<maze version="1.0">
 <item>
  <title>A Beginner's Maze</title>
  <link rel="start" href="/cells/C"/>
 </item>
</maze>
```

This is a high-level representation of the maze as seen from the outside. It's got a link with the relation start which points to cell C.

Navigating

Starting with no information but this URL, you can do everything it's possible to do with a Maze+XML API:

- Start off by GETting a representation of the collection of mazes. You know how to parse the representation, because you read the Maze+XML specification and programmed this knowledge into your client.
- Your client also knows that the link relation maze indicates an individual maze. This gives it a URL it can use in a second GET request. Sending that GET request gives you the representation of an individual maze.
- Your client knows how to parse the representation of an individual maze (because you programmed that knowledge into it), and it knows that the link relation 'start' indicates an entrance into the maze. You can make a third GET request to enter the maze.
- Your client knows how to parse the representation of a cell. It knows what east, west, north, and south mean, so it can translate movement through an abstract maze into a series of HTTP GET requests.
- Your client knows what exit means, so it knows when it's completed a maze.

There's more to the Maze+XML standard, but you've now seen the basics. A collection links to a maze, which links to a cell. From one cell you can follow links to other cells. Eventually you'll find a cell with an exit link leading out of the maze.

A Maze+XML Server

- A maze+xml server can store maze data in any format, including simple JSON documents.
- Here's the JSON document representing the "beginner" maze
- This is not a representation of the maze in the REST sense, because it will never be sent over HTTP.
- It's the raw data used to generate the Maze+XML document that *is* sent over HTTP.

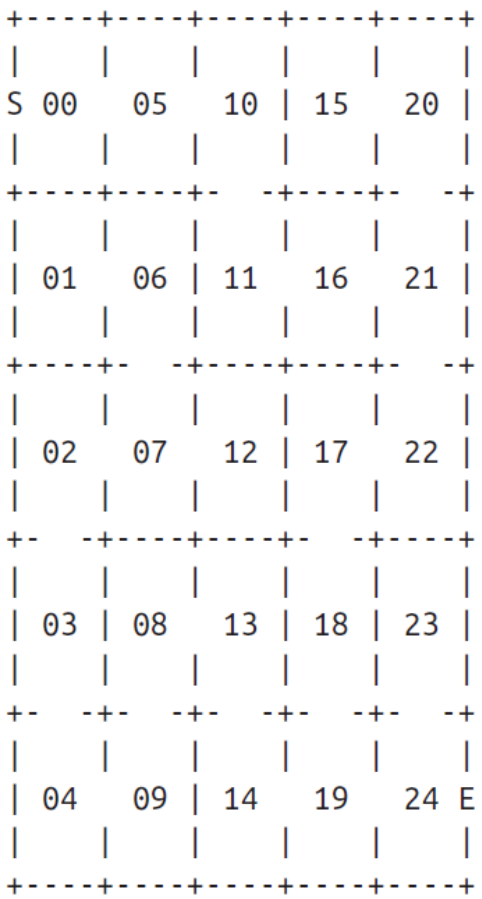

```
{
  "_id" : "five-by-five",
  "title" : "A Beginner's Maze",
  "cells" : {
      "cell0":{"title":"Entrance Hallway", "doors":[1,1,1,0]},
      "cell1":{"title":"Hall of Knives", "doors":[1,1,1,0]},
      "cell2":{"title":"Library", "doors":[1,1,0,0]},
      "cell3":{"title":"Trophy Room", "doors":[0,1,0,1]},
      "cell4":{"title":"Pantry", "doors":[0,1,1,0]},
      "cell5":{"title":"Kitchen", "doors":[1,0,1,0]},
      "cell6":{"title":"Cloak Room", "doors":[1,0,0,1]},
      "cell7":{"title":"Master Bedroom", "doors":[0,0,1,0]},
      "cell8":{"title":"Fruit Closet", "doors":[1,1,0,0]},
      "cell9":{"title":"Den of Forks", "doors":[0,0,1,1]},
      "cell10":{"title":"Nursery", "doors":[1,0,0,1]},
      "cell11":{"title":"Laundry Room", "doors":[0,1,1,0]},
      "cell12":{"title":"Smoking Room", "doors":[1,0,1,1]},
      "cell13":{"title":"Dining Room", "doors":[1,0,0,1]},
      "cell14":{"title":"Sitting Room", "doors":[0,1,1,0]},
      "cell15":{"title":"Standing Room", "doors":[1,1,1,0]},
      "cell16":{"title":"Hobby Room", "doors":[1,0,1,0]},
      "cell17":{"title":"Observatory", "doors":[1,1,0,0]},
      "cell18":{"title":"Hot House", "doors":[0,1,0,1]},
      "cell19":{"title":"Guest Room", "doors":[0,0,1,0]},
      "cell20":{"title":"Servant's Quarters", "doors":[1,0,0,1]},
      "cell21":{"title":"Garage", "doors":[0,0,0,1]},
      "cell22":{"title":"Tool Room", "doors":[0,0,1,1]},
      "cell23":{"title":"Banquet Hall", "doors":[1,1,0,1]},
      "cell24":{"title":"Spoon Storage", "doors":[0,0,1,1]}
    }
}
```

The Clients
Client 1: A Human Directed Maze Game

- The obvious use for the Maze+XML API is a game to be played by a human being
- We tend to think of an "API client" as an automated client.
- But human-driven clients like this have a big part to play in the modern API ecosystem.
- It's very common for a mobile application, driven by a human, to communicate with a server through a web API. Best of all, with a human in the loop, the semantic gap is no problem.

Client 2: The Mapmaker

- The first client relied on a human being making the decisions about where to go.
- But there are algorithms for automatically solving mazes, and there's no reason we can't write an automated client to go along with the manually operated one.
- This client does something a little different. It is called the Mapmaker, and it's a client for *mapping* a maze



The server doesn't define mazes in this graphical format; they're stored as JSON documents and served as XML documents. The Mapmaker builds up this graphical view of the maze by automatic exploration.

Meeting the Semantic Challenge

- For the designer of a domain-specific API, bridging the semantic gap is a two-step process:
  1. Write down your application semantics in a human-readable specification (like the Maze+XML standard).
  2. Register one or more IANA media types for your design, (like *application/vnd.amundsen.maze+xml*. In the registration, associate the media types with the human-readable document you wrote.
- Your client developers can reverse the process to bridge the semantic gap in the other direction:
  1. Look up an unknown media type in the IANA registry.
  2. Read the human-readable specification to learn how to deal with documents of the unknown media type.

There's no magic shortcut. To get working client code, your users will have to read your human-readable document and do some work. We can't get rid of the semantic gap completely, because computers aren't as smart as humans.

That's it Folks

Further Reading

Chapter 5: RESTful Web APIs by Leonard Richardson and Mike Amundsen