Mike Hahn – hahnbytes@gmail.com

Jiphynet

Jiphynet is an open source tool used to build websites, and is implemented in Java. Many websites use a freemium business model, in which subscribers pay optional fees to access premium features. Website builders can use either of 2 languages: Java or Jiphytalk. Jiphytalk is based on Python and includes Java-like features. The sister websites of Jiphynet are Jiphygames, where you make your own board games, and Jiphacademy, which links tutors with students.

- Makes use of 2 new open source web programming languages: Jiphytalk and Jiphytags
- Jiphytags is similar to HTML
- For a website called "mysite", the URL is mysite.jiphynet.com
- Subscribers pay $20/year (gold) or $12.50/year (silver)
- Silver members can only join at most 2 websites at once, and can only switch websites once a week
- Gold members are members of all websites
- Calculation of website resources consumed:
  - Let W = (no. of image file megabytes served) x (no. of kilo-nodes created)
  - Nodes are 12-byte chunks of RAM
  - Summed up for all users of a given website in one month
  - Let W' = (no. of image file megabytes served) x (no. of tokens in Java source code)
  - Comments and the following chars. ignored: ( ) { } [ ] ;
  - Constants: numeric, string, character count as one token
- Let H = cost of web hosting
- Websites where W is less than twice the median M are free: H = 0
- H = for W between 2 and 4 times M = $10/mo.
- W between 4 and 8 times M = $15/mo.
- W between 8 and 16 times M = $25/mo.
- W between 16 and 32 times M = $40/mo.
- W greater than 32,768 times M = $300/mo.
- W greater than 100,000 times M: website speed is throttled
- Every paying website gets a rebate (part of subscription fee revenue = F) proportionate to HR where R = A / B and A is less than or equal to B
- All rebates add up to F
- A = W for non-members only and
- B = W for members only or vice versa
- Purpose of R = A / B: motivate developers to create balanced websites, where population sizes of members and non-members are as equal as possible
- Allow financial transactions between developers and end-users using credit cards/PayPal
- Don't charge transaction fees
- Transaction disputes (cheating):
  - Developer accused of cheating by 3 or more unique users in 90 days or less
  - User accused of cheating by 2 or more unique developers in 180 days or less
  - User/developer gets F rating for 2 years (allowed to appeal)
- Any Jiphytags web page can have a corresponding HTML web page which contains converted HTML code and a link to its Jiphytags page

factor = W / M	monthly fee = H	W / M	H	W / M	H
2	$10	64	$80	2048	$200
4	15	128	100	4096	225
8	25	256	125	8192	250
16	40	512	150	16,384	275
32	60	1024	175	>32,768	300

Revenue and Expenses

- Let Q = no. of freemium projects
- Assume Q = 20
- Let q = no. of freemium projects who pay hosting fees
- Assume q = Q x 30 percent = 6
- Let s = no. of silver members/project
- Assume s = 20
- Divide by 2, assuming each silver member belongs to 2 projects
    - Then no. of silver members = Qs / 2 = 20 x 20 / 2 = 200
    - Assume user conversion rate = 5 percent
    - Let U = no. of users
    - Then U = 200 / 5 percent = 4000
    - F = subscription fees total = 200 x 12.5 = $2500/year
    - Let H = web hosting fees total
    - Let h = avg. hosting fee per project per month
    - Assume h = $20
    - H = 12qh = 12 x 6 x 20 = $1440/year
    - Let N = net amt. paid to each project
    - Then N = (F - H) / q
    - N = (2500 - 1440) / 6
    - N = $177/year
    - Let N = (KF - H) / q
    - Assume N = 0
    - H = KF
    - K = H / F
    - K = 1440 / 2500 = 58 percent
    - Let V = revenue
    - V = F - H
    - V = 2500 - 1440 = $1060/year
- Assume Q = 100
- Assume q = Q x 30 percent = 30
- Assume s = 40
- Divide by 2, assuming each silver member belongs to 2 projects
    - Then no. of silver members = Qs / 2 = 100 x 40 / 2 = 2000
    - Assume user conversion rate = 5 percent
    - Let U = no. of users
    - Then U = 2000 / 5 percent = 40,000
    - F = subscription fees total = 2000 x 12.5 = $25,000/year
    - Assume h = $20/month
    - H = 12qh = 12 x 30 x 20 = $7200/year
    - Let N = (KF - H) / q
    - Assume N = 0
    - K = H / F = 7200 / 25,000 = 29 percent
    - V = F - H = 25,000 - 7200 = $17,800/year

- Let E = expenses
- E = Google AdWords cost + web hosting
- E = 3600 + 7500 = $11,100/year
- Let P = profit
- P = V - E
- P = 17,800 - 11,100 = $6700/year

Jiphytags

Jiphytags is a simplified markup language used to replace HTML. Arbitrary Jiphytags code can be embedded in the Jiphytalk echo statement. Jiphytags syntax, where asterisk (*) means repetition, is defined as follows:

- Tags:
  - [tag]
  - [tag: body]
  - [tag (fld val)*: body]
- Body:
  - text
  - [: text]*
  - [(fld val)*: text]*
- Call Jiphytalk code:
  - [expr: <expr>]
  - [exec: <stmt>...]
  - [jiph: <path>]

Monospace Mode

In monospace mode, all body text rendered to the screens of end-users is in a mono-spaced, typewriter-style font. Every character takes up 2 square cells: an upper cell and a lower cell. Superscripts and subscripts are handled by employing a vertical offset of one square cell. Header text is also mono-spaced, and each character takes up 2 oversized square cells.

Additional Formatting

The grid of characters can be subdivided into panels, which can themselves be subdivided into more panels, and so on. Any panel can contain zero or more text boxes, which may overlap each other. Vertical grid lines each take up one square cell per row of square cells. Horizontal grid lines are displayed in the same pixel row as underscore characters. Any row of square cells containing a horizontal grid line which is 2 pixels wide is taller by exactly one pixel. The following bracket characters: ( ) [ ] { } can be oriented vertically or horizontally, taking up a single column or row of at least 2 square cells, respectively. Widgets such as check boxes, radio buttons, and combo box arrows take up 4 square cells (2 by 2). Images, animations, and diagrams are contained in canvas objects, which can appear anywhere panels can appear.

Rich-Text Mode

In rich-text mode, a given header or paragraph of body text can consist of a single variable-width font. Paragraphs have before/after spacing, left/right indent, and line spacing (single, double, 1.5, etc.). Panels have margins on all 4 sides. In both rich-text and monospace modes, text is rendered to the HTML5 canvas object. Some features like form fields and submit buttons use hidden HTML.

Jiphacademy

Jiphacademy.com is a website which links tutors with students. Some tutors charge their students an hourly rate, and Jiphynet receives 10 percent of that revenue stream. Non-members can only make use of volunteer tutors. The tutors teach math, literacy, and coding. A web-based interactive whiteboard enables the tutor to interact with a single student. The tutor and student take turns interacting with the whiteboard. At the beginning of each turn, every move (mouse clicks and text entered during the previous turn) is replayed, and then further interaction takes place. Prefabricated lessons are prepared by volunteer curriculum-writers and displayed on the interactive whiteboard.

Implementation Steps

1. Read Murach's Java Servlets and JSP book
2. Implement Jabbler: web-based Scrabble game, user vs. robot
    - Jabbler is currently console-based Java Scrabble game
3. Implement web-based chess and backgammon:
    - Play against robot which makes random but legal moves
4. Write basic Jiphytags design specs
5. Implement Jiphytalk 0.1, console-based
    - Token parsing and building program tree has already been implemented
6. Finish Jiphytalk 1.0, console-based
7. Write advanced Jiphytags design specs
8. Implement JTAG-to-HTML converter
9. Implement monospace mode
10. Implement rich-text mode
11. Integrate Jiphytalk with Jiphytags (monospace/rich-text modes)
12. Implement JIPH-to-JS converter
13. Recruit GitHub open source coders/testers
14. Implement Jabbler: web-based, 2-player
15. Implement monospace mode, dual user
16. Implement rich-text mode, dual user
17. Implement Jiphygames
    1. Convert Jabbler from Java to Jiphytalk
    2. WYSIWYG board/piece editor
    3. Codeless prototyping system
    4. Beta test
    5. Implement Jiphytalk code editor
    6. Integrate with board editor/prototyping system
18. Implement Jiphynet
19. Design website
20. Launch website
21. Beta test Jiphynet
22. Accept credit card payments
23. Implement Jiphacademy
24. Hire Java programmer with expertise in making websites scalable

About Me

I am Mike Hahn, the founder of Jiphynet.com. I was previously employed at Brooklyn Computer Systems as a Delphi Programmer and a Technical Writer (I worked there between 1996 and 2013). At the end of 2014 I quit my job as a volunteer tutor at Fred Victor on Tuesday afternoons, where for 5 years I taught math, computers, and literacy. I'm now a volunteer math/computer tutor at West Neighbourhood House. My hobbies are reading quora.com questions/answers and the news at cbc.ca. About twice a year I get together with my sister Cathy who lives in Victoria. She comes here or I go out there usually in the summer. A few months prior to starting my Jiphynet project I used to lie on the couch a lot, not being very active. Now I'm busy most of the time. I visit my brother Dave once a month or so and I also visit my friends Main and Steph once or twice a month.

Contact Info

Mike Hahn
Founder, Jiphynet.com
515-2495 Dundas St. West
Toronto, ON M6P 1X4

Country: Canada
Phone: 416-533-4417
Email: hahnbytes (AT) gmail (DOT) com
Web: www.hahnbytes.com

Jiphygames

Jiphygames.com is a free website where you can play 2-player non-animated games: think board games and card games. You can also create your own games using Jiphytalk and Jiphytags. Jiphygames users can create home pages/bios written in Jiphytags, post in forums, view games in progress, participate in tournaments, and hold player rating values for each game they are involved with. An example of a player rating value is a chess rating, where a rating of 1700 or more would be held by a very skilled chess player.

Jiphytalk

Jiphytalk is a Python dialect in which all operators precede their operands, and parentheses are used for all grouping (except string literals, which are delimited with double quotes). Jiphytalk code is often accompanied by Jiphytags screen definition files. Jiphytags is similar to HTML, except open tags begin with an open square bracket and a keyword, and the closing tag is simply a close square bracket. Any text enclosed in a tag is preceded by a colon. File extensions include .JIPH (Jiphytalk) and .JTAG (Jiphytags).

Special Characters

- `()` grouping
- `-` used in identifiers
- `;` end of stmt.
- `:` dot operator
- " string delimiter
- \ escape char.
- # comment
- Extra:
  - _ used in identifiers
  - $ string prefix char.
  - * public switch
  - { } block comment

Version 0.1

- No inheritance
- No interfaces
- No IDE
- No rich text

Differences from Python

- Parentheses, not whitespace
- Integration with Jiphytags
- Operators come before their operands
- Information hiding (public/private)
- Single, not multiple inheritance
- Adds interfaces ("scool" defs.)
- Drops iterators and generators
- Adds lambdas
- Adds quote and list-compile functions, treating code as data

Grammar Notation

- Non-terminal symbol: <symbol>
- Optional text in brackets: [*text*]
- Repeats zero or more times: [*text*]…
- Repeats one or more times: <symbol>…
- Pipe separates alternatives: *opt1* | *opt2*
- Comments in *italics*

Keyboard Aid

This optional feature enables hyphens, open parentheses, and close parentheses to be entered by typing semicolons, commas, and periods, respectively. When enabled, keyboard aid can be temporarily suppressed by using the Ctrl key in conjunction with typing semicolons, commas, and periods (no character substitution takes place). By convention, hyphens are used to separate words in multi-word identifiers, but semicolons are easier to type than hyphens. Similarly, commas and periods are easier to type than parentheses. Typing semicolon converts previous hyphen to a semicolon, and previous semicolon to a hyphen (use the Ctrl key to override this behaviour). Typing semicolon after close parenthesis simply inserts semicolon. The close delim switch automatically inserts a closing parenthesis/double quote when the open delimiter is inserted.

Jiphytalk Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space, also any semicolon before a close parenthesis may be omitted):

<source file>:
- [<use>] [* <vars>] [<vars>] [do <block>] [<def>]… [<class>]… [do <block>]

<use>:
    use (<import-semi>...)

<import-semi>:
    <import-stmt> **;**

<import stmt>:
    import <module>
    import (<module>…)
    from <rel module> import <mod list>
    from <rel module> import all

<module>:
    <name>
    (<name> as <name>)
    (**:** <name>… [as <name>])

<mod list>:
    <id as>
    (<id as>…)

<id as>:
    <mod id>
    (<mod id> as <name>)

<mod id>:
    <mod name>
    <class name>
    <func name>
    <var name>

<rel module>:
    (**:** [<num>] [<name>]...)
    <name> *// ?*

<class>:
- ([*] <cls typ><name> [<base class>] [<does>] [* <vars>] [<vars>] <def>…)
- ([*] scool <name> [<does>] [<const list>] [<def hdr>]...)
- ([*] enum <name><elist>)

<cls typ>:
    class
    abclass

<does>:
    does (<scool name>...)

<scool name>:
<base class>:
    <name>
    (**:** <name><name>…)

<const list>:
    const (<const pair>...)

<const pair>:
    (<name><const expr>)

<def hdr>:
    (<defun><name> ([<parms>]) [<dec>])

<def>:
- ([*] <defun><name> ([<parms>]) [<vars>] [<dec>] do <block>)

<defun>:
    def
    abdef

<vars>:
    var (<id>...)

<parms>:
    <parm>... [(* <id>)] [(** <id>)]

<parm>:
    <id>
    (tuple <id>...)
    (<set op><id><expr>)
    (<set op> (tuple <id>...) <expr>)

<dec>:
    decor <call expr>...

<block>:
    ([<stmt-semi>]…)

<stmt-semi>:
    <stmt> **;**

<jump stmt>:
    <continue stmt>
    <break stmt>
    <return stmt>

<pjump stmt>:
    return <expr>
    ** <raise stmt>

<raise stmt>:
    raise [<expr> [from <expr>]]

<stmt>:
    <open stmt>
    <closed stmt>

<open stmt>:
    <if stmt>
    <while stmt>
    <for stmt>
    ** <try stmt>
    <pjump stmt>
    <pcall stmt>
    <asst stmt>
    <del stmt>

<closed stmt>:
    <jump stmt>
    <call stmt>
    <print stmt>
    <lstg tag>

<call expr>:
- (<name> [<arg list>])
- (**:** <obj expr> [<colon expr>]…
  (<method name> [<arg list>]))
- (call <expr> [<arg list>])

<call stmt>:
    (<name> [<arg list>])

<pcall stmt>:
- **:** <obj expr> [<colon expr>]…
  (<method name> [<arg list>])
- call <expr> [<arg list>]

<colon expr>:
    <name>
    (<name> [<arg list>])

<arg list>:
    [<expr>]... [(<set op><id><expr>)]...

<asst stmt>:
    <asst op><target expr><expr>
    <set op> (tuple <target expr>...) <expr>

<asst op>:
    set | addset | minusset | mpyset | divset |
    idivset | modset |
    shlset | shrset | shruset |
    andbset | xorbset | orbset |
    andset | xorset | orset |
    = | += | -= | *= | /= |
    //= | %= |
    <<= | >>= | >>>= |
    &= | ^= | '|=' |
    &&= | ^^= | '||='

<set op>:
    set | =

<target expr>:
    <name>
    (**:** <name> [<colon expr>]… <name>)
    (slice <arr><expr> [<expr>])
    (slice <arr><expr> all)

<arr>: *// string or array/lyst*
    <name>
    <expr>

<obj expr>:
    <name>
    <call stmt>

<if stmt>:
- if <expr> do <block> [elif <expr> do <block>]… [
  else <block>]

<while stmt>:
    while <expr> do <block>
    do <block> while <expr>

<for stmt>:
    for <name> in <expr> do <block>

<try stmt>:
- try <block> <except clause>… [else <block>]
  [finally <block>]
- try <block> finally <block>

<except clause>:
    except <name> [as <name>] do <block>

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<del stmt>:
    del <expr>

<paren stmt>:
    (<open stmt>)
    <closed stmt>

<qblock>:
    (quote [<paren stmt>]...)

<expr>:
    <keyword const>
    <literal>
    <name>
    (<unary op><expr>)
    (<bin op><expr><expr>)
    (<multi op><expr><expr>…)
    (<quest><expr><expr><expr>)
    <lambda>
    (quote <expr>...)
    <renum expr>
    <tuple expr>
    <lyst expr>
    <dict expr>
    <bitarray expr>
    <string expr>
    <bytezero expr>
    <bytes expr>
    <target expr>
    <obj expr>
    <cast>

<quest>:
    quest | ?

<unary op>:
    minus | notbitz | not |
    - | ~ | !

<bin op>:
    <arith op>
    <comparison op>
    <shift op>
    <bitwise op>
    <boolean op>

<arith op>:
    div | idiv | mod | mpy | add | minus |
    / | // | % | * | + | -

<comparison op>:
    ge | le | gt | lt | eq | ne | is | in |
    >= | <= | > | < | == | !=

<shift op>:
    shl | shr | shru |
    << | >> | >>>

Note: some operators delimited with single quotes for clarity (quotes omitted in source code)

<bitwise op>:
    andbitz | xorbitz | orbitz |
    & | ^ | '|'

<boolean op>:
    and | xor | or |
    && | ^^ | '||'

<multi op>:
    mpy | add | strdo | strcat |
    and | xor | andbitz | xorbitz |
    or | orbitz |
    * | + | % | + |
    && | ^^ | & | ^ |
    '||' | '|'

<const expr>:
    <literal>
    <keyword const>

<literal>:
    <num lit>
    <str lit>
    <bytes lit>

<tuple expr>:
    (tuple <expr>…)

<lyst expr>:
    (lyst [<expr>]…)

<dict expr>:
    (dict [<pair>]…)

<bitarray expr>:
    (bitarray <enum name> [<elist>])
    (bitarray <enum name><idpair>...)

<elist>:
    <id>...
    <intpair>...
    <chpair>...

<intpair>
    // integer constant
    <int const>
    (<int const><int const>)

<chpair>
    // one-char. string
    <char lit>
    (<char lit><char lit>)

<idpair>
    <idt>
    (<id><id>)

<pair>:
    // expr1 is a string
    (<expr1><expr2>)
    (<str lit><expr>)

<renum expr>
    (renumize <expr><ren id>...)
    (renumize <expr><ren int>...)
    (renumize <expr><ren ch>...)

<ren id>:
    (0 <id>)
    (1 <id>)
    (1 <id><id>)

<ren int>:
<ren ch>:
    // expr is <dec int> | <char lit>
    (0 <expr>)
    (0 <expr><expr>)
    (1 <expr>)
    (1 <expr><expr>)

<cast>:
    (cast <type><expr>)

<print stmt>:   *// built-in func*
    (print <expr>…)
    (println [<expr>]…)
    (echo <expr>…)

<lambda>:
    (lambda ([<id>]...) <expr>)
    (lambda ([<id>]...) do <block>)
    (lambda ([<id>]...) do <qblock>)
    // must pass qblock thru compile func

No white space allowed between tokens, for rest of Jiphytalk Grammar

<white space>:
    <white token>...

<white token>:
    <white char>
    <line-comment>
    <blk-comment>

<line-comment>:
    # [<char>]... <new-line>

<blk-comment>:
    { [<char>]... }

<white char>:
    <space> | <tab> | <new-line>

<name>:
- [<underscore>]… <letter> [<alnum>]…
  [<hyphen-alnum>]… [<underscore>]…

<hyphen-alnum>:
    <hyphen><alnum>…

<alnum>:
    <letter>
    <digit>

In plain English, names begin and end with zero or more underscores. In between is a letter followed by zero or more alphanumeric characters. Names may also contain hyphens, where each hyphen is preceded and succeeded by an alphanumeric character.

<num lit>:
    <dec int>
    <long int>
    <oct int>
    <hex int>
    <bin int>
    <float>

<dec int>:
    [<hyphen>] 0
    [<hyphen>] <any digit except 0> [<digit>]…

<long int>:
    <dec int> L

<float>:
    <dec int><fraction> [<exponent>]
    <dec int><exponent>

<fraction>:
    <dot> [<digit>]…

<exponent>:
    <e> [<sign>] <digit>…
<e>:
    e | E

<sign>:
    + | -

<keyword const>:
    none
    true
    false

<oct int>:
    0o <octal digit>…

<hex int>:
    0x <hex digit>…
    0X <hex digit>…

<bin int>:
    0b <zero or one>…
    0B <zero or one>…

<octal digit>:
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<hex digit>:
    <digit>
    A | B | C | D | E | F
    a | b | c | d | e | f

<string lit>:
    [$ <str prefix>] <short long>

<str prefix>:
    r | u | R | U

<short long>:
    " [<short item>]... "
    " " " [<long item>]... " " "

<short item>:
    <short char>
    <escaped str char>

<long item>:
    <long char>
    <escaped str char>

<short char>:
    any source char. except "\", newline, or
    end quote

<long char>:
    any source char. except "\"

<bytes lit>:
    $ <byte prefix><shortb longb>

<byte prefix>: *// any case/order*
    b | br

<shortb longb>:
    " [<shortb item>]... "
    " " " [<longb item>]... " " "

<shortb item>:
    <shortb char>
    <escaped char>

<longb item>:
    <longb char>
    <escaped char>

<shortb char>:
    any ASCII char. except "\", newline, or
    end quote

<longb char>:
    any ASCII char. except "\"

<escaped char>:
    \newline
        ignore "\", newline chars.
    \\ *backslash*
    \" *double quote*
    \} *close brace*
    \a *bell*
    \b *backspace*
    \f *formfeed*
    \n *new line*
    \r *carriage return*
    \t *tab*
    \v *vertical tab*
    \ooo *octal value = ooo*
    \xhh *hex value = hh*

<escaped str char>:
    <escaped char>
    \N{name} *Unicode char. = name*
    \uxxxx *hex value (16-bit) = xxxx*