18 May 2015

Chapt. 2 SENG 422 TA Progress Report

(Project Design Course Notes 1)

TA: Philip B. Alipour

2.1. Software Architecture concerns: Choice of design pattern based on the Performance, Data Type and Entropy of the LSCS system

- The following sketch notes and design will guide you through your choice of pattern by addressing key concerns in your architecture as the number of actors increase in population as well as tasks and hardware/software components:
    - **Software architectural concerns for the LSCS project, such as performance, service downtime and severed connection due to timely access issues, traversing spatial and time for data during sort in memory/DB, external service DB alternate switch when service, halt, delay or downtime, real vs. fictitious data report to parse, and who is the parser or part of a (sub)system as a collective solution.**
- What is your solution per concern as you apply a design pattern, which must satisfy **performance goals and criteria** (http://www.viewpoints-and-perspectives.info/home/perspectives/performance-and-scalability/) in your architecture?
- You may complement the current sketch or fill in the gaps concerning performance as you may append or revise according to your project plan.

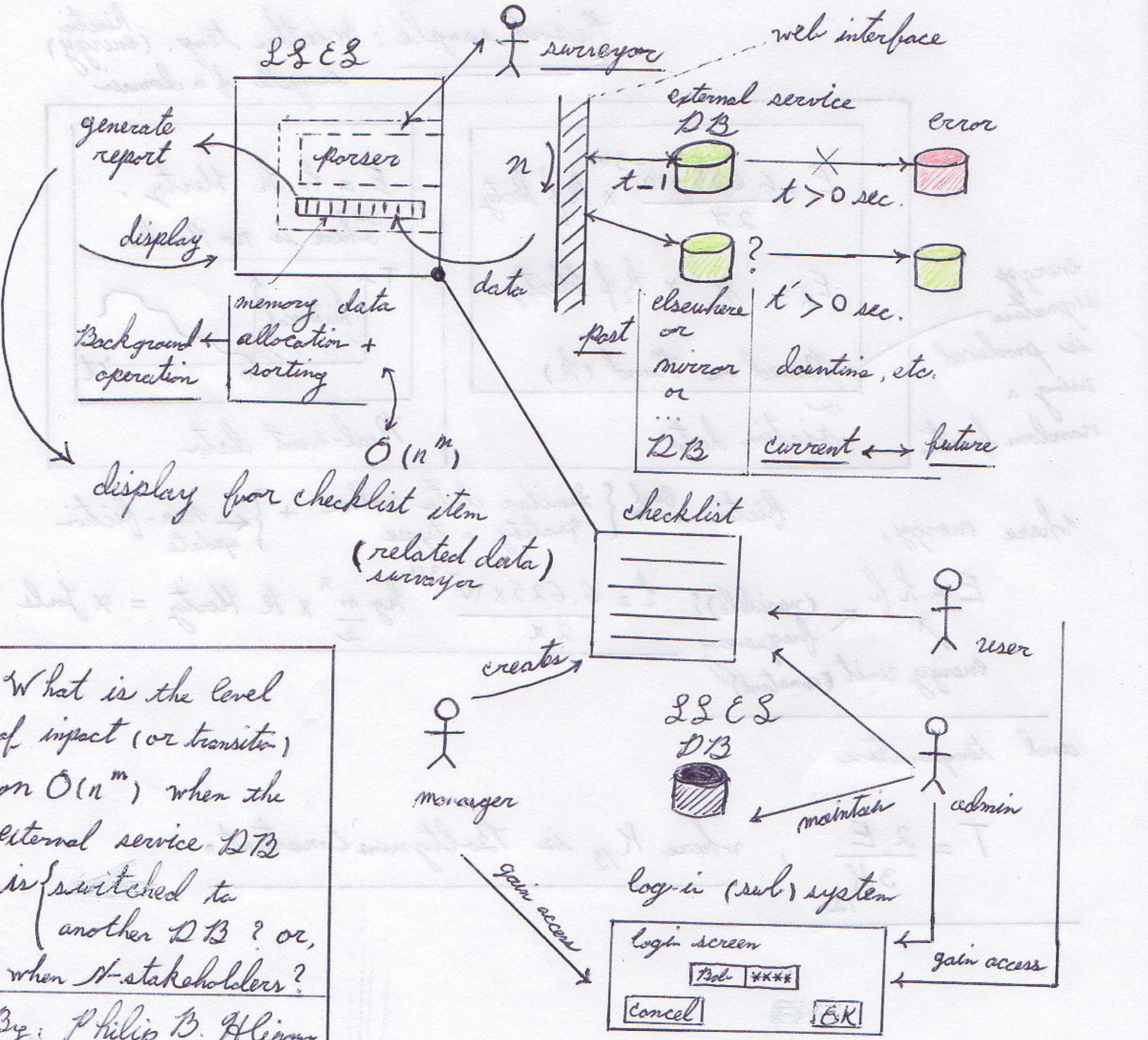
Choice of pattern based on Software Arch. Concerns: Key concerns

What is your solution?

① the performance (big-O notation):

$\tilde{\mathcal{L}}(n) = O(n^{m \geq 0})$. $\mathcal{L}$ as the time it takes to solve a problem of n steps. (algorithm)

② External service(s) downtime or severed connection due to e.g. HTTP 408 (too long response time) and HTTP 404 error: LSES as a client communicates with service but the page client looks for is non-existent (?)

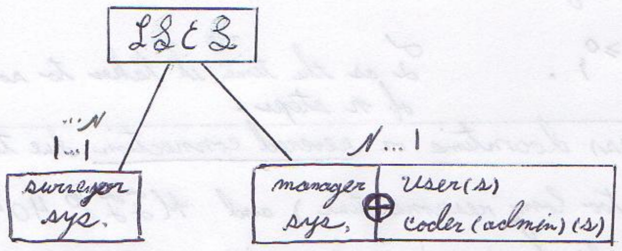


∴ What is the level of impact (or transition) on $O(n^m)$ when the external service DB is {switched to another DB ? or,
∴ when N-stakeholders?

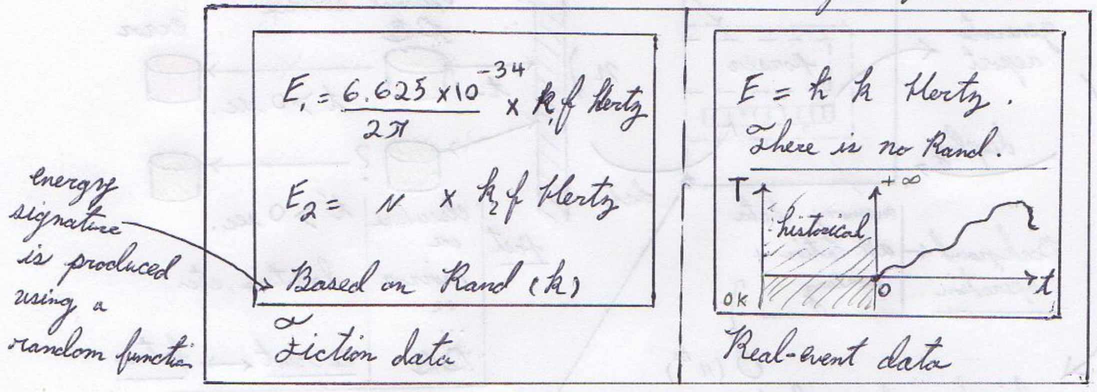
By: Philip B. Alipour
Date: 15 May, 2015

P.1

* The systems Logic and decomposition of system and relationships:



Report sample: Weather temp. (kinetic energy) sample of a domain


$E_1 = \frac{6.625 \times 10^{-34}}{2\pi} \times k_1 f$ Hertz
$E_2 = N \times k_2 f$ Hertz
Based on Rand (k)
Fiction data
energy signature is produced using a random function
$E = \hbar\, k$ Hertz.
There is no Rand.
T
$+\infty$
historical
0K
0
t
Real-event data

where energy, fiction $\xrightarrow{\text{feed}}$ $\left\{ \begin{array}{l} \text{Similar data volume} \\ \text{quality + type} \end{array} \right. + \left. \vphantom{\begin{array}{l} a \\ b \end{array}} \right\} \leftarrow$ non-fiction update

$E = \hbar f$ (variable); $\hbar = \frac{6.625 \times 10^{-34}}{2\pi} \, kg \frac{m^2}{s} \times k$ Hertz $= x$ Jouls

energy unit (constant) frequency

---

and temperature

$$T = \frac{2E}{3K_B}$$, where K_B is Boltzmann Constant.

P. 2

③ Where is the parser in the system? Do you need one or more parsers in the system? Why?

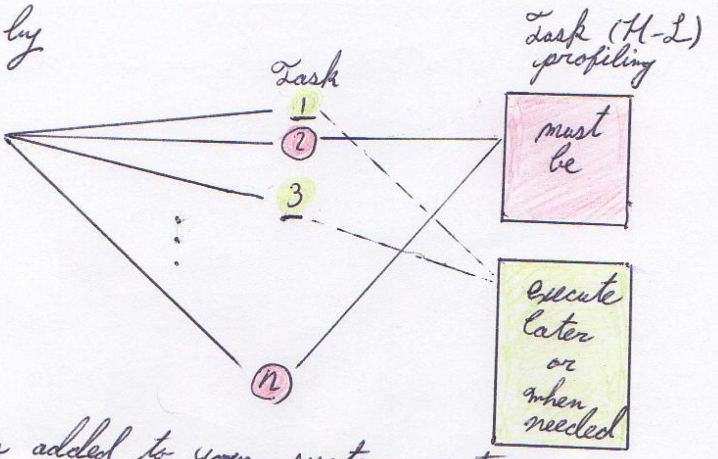
∴ From ①-③, which design pattern is appropriate?

Improve performance by

1- Task prioritization

and

2- Entropy measure
(info measure)



once a <u>component</u> or <u>actor</u> is added to your system, entropy increases, or

$H \longrightarrow \infty$. So, we need to reduce uncertainty (3rd law of Thermodynamics)

$H = (k_B \operatorname{Log} W) \longrightarrow 0$, where W is the probability of <u>components</u> (or <u>data atoms</u>) configured in your system i.e. configurable elements in your design to fall into a specific <u>system state</u> $\left\{ \begin{array}{l} \text{worst case (Bad performance)} \\ \text{Best case (perfect performance)} \end{array} \right\}$ uncertainty cycle

and

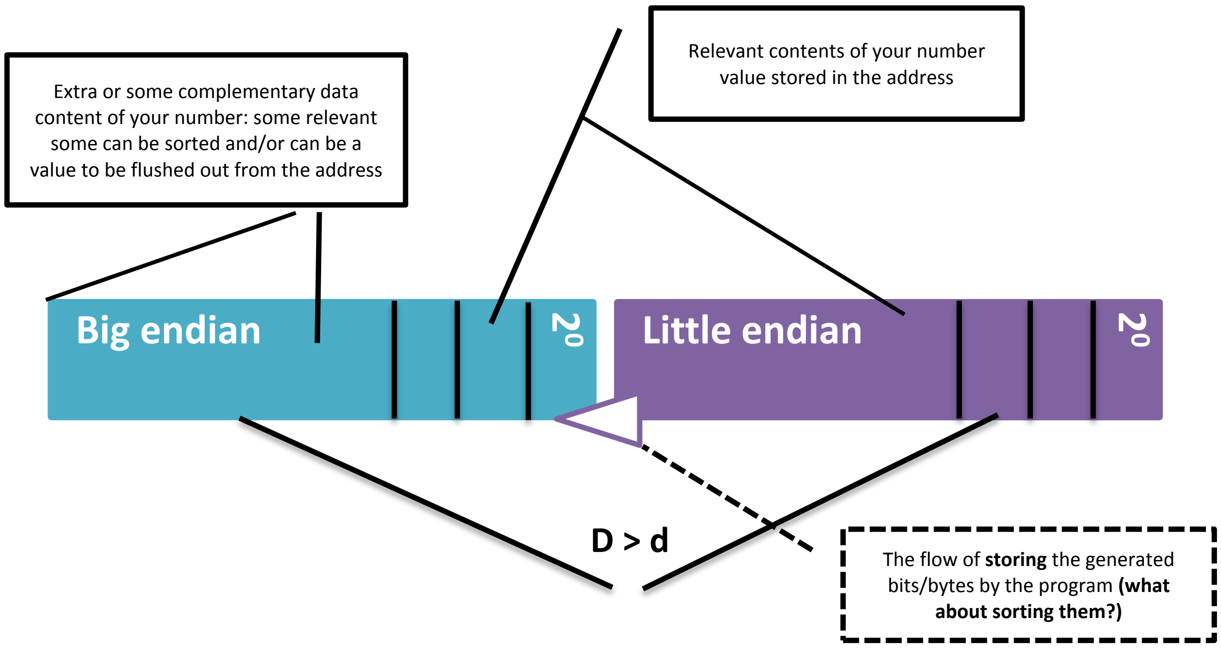
... Other factors of concern to contemplate. p.3

2.2. Software Architecture concerns: Hardware/software challenges (data atom access_update as CRUD database/memory for configurable components in design)

- Some system components such as the DB or on a low level memory, require a way in partitioning its space and manage/sort data relative to what is actually being stored/read/written to/from its address (to deal with the big O algorithm complexity issue of a looping algorithm which is refreshed when an item is created/updated on a checklist by an actor).
- **CRUD** is simply explained at http://en.wikipedia.org/wiki/Create,_read,_update_and_delete
- To make your code the most efficient when CRUDing, you need to know the difference between the times/steps **and the level of impact information could have on the overall system performance and code** (a concern http://en.wikipedia.org/wiki/Separation_of_concerns) e.g.,

T(n)= O(n^2) and **O(n).**

The aim is to achieve **O(n), and in a perfect world, O(n-{1, 2, 3, …})** when the code executes. For example, as depicted below, if an operation is performed from right-to-left (in the eclipse program for instance, the memory map is displayed and partitioned from left-to-right), the **little endian** distance is shorter than the **big endian.**


Relevant contents of your number value stored in the address
Extra or some complementary data content of your number: some relevant some can be sorted and/or can be a value to be flushed out from the address
Big endian
2^0
Little endian
2^0
D > d
The flow of storing the generated bits/bytes by the program (what about sorting them?)

If the relevant contents representing numbers (integer data type) on a list is stored in the targeted address, yet with extra bits (the way **space** is partitioned in your code as you define it), then it inevitably stores the content line-by-line for the ascending numbers with much greater **distances** (accessing the big endian addresses) **D.** This will impede or slow down the level of code execution i.e., greater algorithmic complexity or higher **O**'s.

The aim is to gain minimum distances of d's rather than **D > d** which pertains to bigger **O**'s (something to avoid or address upon).

And the relation between distance **d** or **D** to processing speed or performance is frequency **f** of the input/output of data to/from registers, such that

$\mathbf{d*f= p_s}$ or processing speed, where
$\mathbf{f=1/T(n)}$ and is measured in Hertz (or cycles per second).

- Another issue that could be addressed is changing the flow and order of your **For loops** as well as **nested loops** based on a condition which could be instead of satisfying long iterations (longer distances) satisfy shorter ones if possible.
- The proper solution to achieve an optimized code would be typing up a, e.g. a sorting algorithm like Bubble Sort (http://en.wikipedia.org/wiki/Bubble_sort) to address $\mathbf{O(n^2)}$ and satisfy $\mathbf{O(n)}$ levels of execution.
- Another solution is to use **lossless data compression** or for graphics, a **lossy one** as far as vital details of a picture (e.g. from a sequence of frames) is not lost after decompressing your data. This requires clever compression means as one converts characters (of ASCII), or pixels in terms of integers from a vast combination of **RGB = 256|R*256|G*256|B = 16.7 million color combinations** from a data compression/decompression table (my MSc thesis covers this where you can access it under **publications** section of my website!).
- The timing can be generated and measured for a successful loop series by the main source file where all LSCS subsystems depend on, and not a single numbers/characters source code where you are supposed to generate and sort as a solution (decomposition of source code).

Some notes to consider for your implementation:

- Depending on what type of machine your architecture is applicable (scalability and dependability attributes of the architecture), being a 64 bit machine, 32 bit, etc. (quite dependent on the little vs. big endianness hardware architecture) a question can be raised on say two's complement, which is a lengthy mathematical discussion to cover, however, the short version can be explained as hereby exemplified: https://www.youtube.com/watch?v=SXAr35BiqK8 and https://www.youtube.com/watch?v=Hof95YlLQk0 (this link also discusses about the complementary results in different machines as well as overflow (focus is on operation as well as size. In any case, size is always an issue, either in a buffer, stack or a long number stored by a register: "In a computer, the amount by which a calculated value is greater in magnitude than that which a given register or storage location can store or represent" http://en.wikipedia.org/wiki/Arithmetic_overflow)) as a useful concept.

- Also, operands during operations will change the register content when written in e.g., assembly or C. For instance, what is the difference between in the order and result of execution by the operators in the following expression? (What is prioritized in the operation in this example?):

  (2+3)*5 against **2+3*5**

- The purpose of **push** and **pop** is also of importance during operations. For instance, in the operand case above, which value will be pushed first and popped out in the queue of operation over a stack, or an array of two or three registers? For instance, software code execution per hardware-to-hardware component communication is always a challenge in order to obtain better performances (efficiency as the ratio between the useful output for a total number of inputs (including data quality and quantity)).

Good luck,

Philip

===

Philip B. Alipour,
Ph.D. Candidate in Electrical, Computer Engineering and Quantum Physics,
Dept. of Electrical and Computer Engineering, University of Victoria, V8W 3P6, Canada,
Office: ELW Room # A358,
Email: phibal12@uvic.ca or philipbaback_orbsix@msn.com
Homepage: http://web.uvic.ca/~phibal12/