SYMPOSIUM ON SIMULATION IN ARCHITECTURE + URBAN DESIGN
SimAUD 2020

A Three-Tier Architecture Visual-Programming Platform for Building-Lifecycle Data Management

Mahmoud M. Abdelrahman[1], Sicheng Zhan[1], Adrian Chong[1]

[1]**Department of Building, School of Design and Environment, The National University of Singapore, 4 Architecture Drive, Singapore 117566.**

ABSTRACT

In this paper, we present a platform that integrates three main aspects in the building industry: 1) Building data from both IoT devices and Building Management System (BMS), 2) Building Energy modeling and simulation engine, and 3)Data analysis and optimization libraries. All of which are combined in a three-tier architecture cloud platform. The platform aims to provide useful representations of the data to different stakeholders. We defined three main types of stakeholders. Each stakeholder uses the platform differently: (1) End-use programmers, who use visual programming and textual programming interfaces to perform computational tasks, (2) Dashboard viewers who are interested in viewing insightful real-time data about performance in the form of charts and diagrams, and (3) Data feedback inputters such as occupants to give feedback or fill questionnaires. The three-tier architecture enables the spatial and physical separation of the databases, the computational engines, and the user-interfaces. This separation resulted in some advantages such as flexibility, scalability (horizontal and vertical), reusability, and latency reduction. Currently, the platform is in the final stage of the alpha release development and will be released as open-source platform. The following stage includes community testing and user experience enhancement.

Author Keywords

three-tier architecture; n-tier architecture; Building Energy Modelling; BEM; building lifecycle; actors; presentation layer; application layer; data layer

ACM Classification Keywords

I.6.1 SIMULATION AND MODELING: Computing methodologies Model development and analysis - Software and its engineering Visual languages

1 INTRODUCTION

The building sector has witnessed immense development recently in the way by which building systems are managed [45]. This development aimed at alleviating the significant environmental impact of this sector (30% of the world energy consumption and a third of the associated CO2 emissions [20]). Decreasing this impact could be achieved by better controlling the resources [5]; providing sustainable, and more efficient solutions [46]; developing a better understanding of different deterministic and stochastic aspects of the built environment [10, 36, 43, 11]. In addition, making better decisions based on mining ground-truth data (black-box approach)[31], physics-based simulation models (white-box approach)[41, 31] or both of them (gray-box approach)[46].

The emergence of the Internet of Things (IoT) devices has enabled a massive amount of data, which posed some challenges. The data collected during the last two decades exceeded that which has ever been collected in history[34]. Two significant challenges, other than the availability of data, are to be addressed. On the one hand, managing this big data: transferring, storing, preprocessing, wrangling and mining, optimization, and control in a robust cyber-infrastructure. This lead to cloud computing or ubiquitous computing, that is, computing data in place, without paying much effort in transferring data to local storage/processing machines using scalable storage and computational power on demand maintained by professionals [28, 7, 14]. On the other hand, delivering useful information to different stakeholders based on their use is yet another challenge.

”Data is not information; data must be presented in a usable form before it becomes information” [35, p.134]. Raw data from the built environment varies in its degree of usefulness to different stakeholders (actors). ”Usability” is different for different stakeholders. For instance, A Building Energy Modeller (BEM) requires data such as Green Building eXtensible Markup Language (gbXML) [19], schedules of operations, set-points, number of occupants,etc.. At the same time, a Facility Manager (FM) would be more interested in reducing maintenance costs by fault detection and diagnostics (FDD)[47] algorithms and dashboards. This difference in uses requires users with proper domain knowledge alongside with programming or procedural thinking skills for automating, prototyping, analyzing, building work-flows using the big data and IoT sensors. At the same time, they do not need to be professional programmers, but rather End-use programmers.

SimAUD 2020 May 25-27, Online

The term *End-use programmer* (or Novice developer) first introduced by [32] is used to describe non-professionals without programming and code structures knowledge [**?**]. Most programs today are written by novice programmers [37] who use spreadsheets, writing add-ons/plugins to support their work or add functionality to the software, running MATLAB simulations [26], writing python codes, or IPython notebooks [39]. The main difference between the end-user developer and the professional developer is the goal of the development. The former writes a program focusing on getting tasks done without paying attention to debugging, unit-testing, or re-usability. However, the latter develops, maintain, debug, and test a robust software for others to use[25]. End-use development mostly involves modifying or extending a software artifact (known as programming by example, or programming by demonstration). For example, *Extended annotation or parameterization*: whereas a user can define a new functionality by selecting a readily developed set of functions by other users and stored in shared repositories [24]. To make this task handy to end-use programmers, a proper programming notation should be utilized.

Programming notation is the way by which a programming language is used to represent the state of the world[17]). The selection of the proper programming notation (e.g., textual, diagrammatic, or gesture-based) to describe the problem is a trade-off between several evaluation criteria[17]. This trade-off should lead non-specialist to find the programming environment easy-to-use and easy-to-understand.

A large body of studies have addressed some difficulties related to the everyday programming tasks such as, but not limited to, code navigation [13], and hidden dependencies. A study conducted by Bragdon et al. [9] that uses visual programming 2d spatially connected blocks of codes where it is possible to track each function in the form of bubble showed that visual programming significantly improve code understanding with less code navigation time.

Visual Programming Language (VPL), also known as box-and-wire programming or data-flow programming utilizes user-friendly graphical elements to display the flow of data between different components (Figure 1). To date, there has been little agreement on the superiority of visual programming languages (VPLs) over the textual programming languages (TPLs). Green et.al. [18] and Moher et.al. [30] adopted this stance. However, [29] argued that VPLs could be significantly useful in motivating beginners to learn to program as they are informative in the sense of spatial reasoning and ideas generation. However, he argued that those advantages are not generalized and should not be isolated from the task being studied. Also, TPL is more productive in terms of large-scale and complex software development tasks. There is no conclusive evidence of the superiority of one over the other. Thus, a hybrid VPL/TPL is proven to be suitable amongst users. Many software enables this type of scripting which gained wide popularity such as Rhinoceros VPL Grasshopper[6, 27] whose user-community extended its limits by contributing to developing TPL based functions amongst others for building and urban simulation

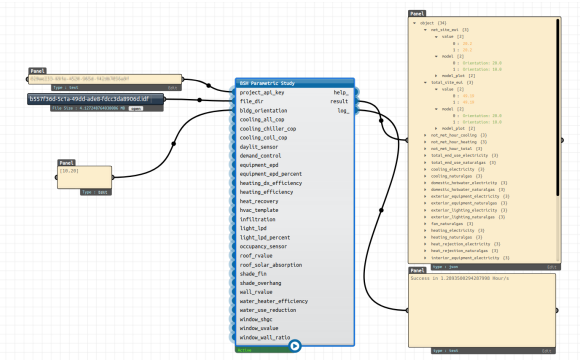

Figure 1. A screenshot from our platform indicating a parametric simulation process on the cloud using (BuildSimHub). The figure is an example of Visual programming Language (also known as box-and-wire programming notation, and data-flow programming) where blocks of code functions are encapsulated in the boxes (known as components), while data flows from one or more component output variable to other component input variable. A component can hold as many inputs as needed, while the code behind each component follows a specific schema. Some components are used only as input components such as "file, numerical, panel" and some components work as output visualizations such as "panel, 3d visualization canvas, plotting components .."

[38, 33, 22], daylight analysis [21, 23], Machine learning[2, 1, 3].

This research addresses two primary problems:

1. Managing buildings' big data requires flexibility and scalability.

2. Different users use data differently.

We introduce a Three-tier architecture web platform as a resolution to these problems. The objectives of this research are: (1) Making IoT data from the built environment useful to different stakeholders, (2) An Open platform for developing and sharing different models amongst end-use programmers. (3) Provide flexibility and scalability by separating the platform into three distributed layers. The rest of the paper is structured as follows: In section 2, we explain the three-tier architecture. The components of the three-tier architecture are explained in details in the subsequent sections 3 (Actors), 4 (Presentation Layer), 4.2 (Application Layer), and 6 (Data Layer).

2 THREE-TIER ARCHITECTURE

Three-tier architecture is a client-server software architecture pattern that has several potentials in distribution applications [8]. The key aspect of this architecture is that it enables the reusability of program building blocks (also referred to as components) such as sharing software components at run time, replicating the same component, controlled adaptation of code, or independent adaptation of components [42, 44]. This pattern consists of three functionally separated layers [40] (Figure 2), those are: **1) Presentation layer** (the client-side), **2) Application layer** (also known as business logic, controllers and process management layer), and **3) Data layer** (also known as data storage). The presentation layer constitutes the user-interface side of the application. All human-computer interactions (HCI) happen on this layer. On

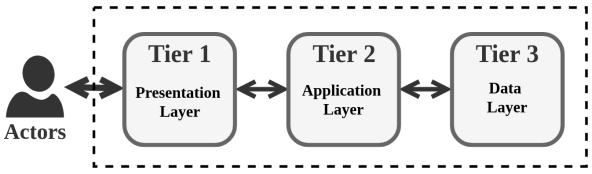

Figure 2. Three-tier architecture structure of the platform

the other side, the Business logic layer consists of the core engine where component functions that carry out the workload by receiving requests from the presentation layer get the relevant data from the Data layer and process this data then, sends back the response to the presentation layer. Finally, the Data layer consists of data querying (acquire from a data source), storage (stored in the relational database on a local server), and accessing (feed to the user). Each of these three tires is discussed in detail in the subsequent sections. A summarized workflow is illustrated in figure 3.

Advantages of using three-tier architecture [16]:

1. Reusability: The use of a modular component-function system enables many users/developers to contribute to the development. This contribution is shared in a public repository to be reusable by other users with small or no change
2. Scalability (both horizontally and vertically) is another essential feature of using three-tier architecture because of using distributed servers as well as because of its separation nature. Horizontal Scalability is achieved by adding more nodes of the same types where required. While the Vertical Scalability is achieved by adding more resources to a host node on demand. On the other hand, separating different tiers allows scaling each one independently depending on the needs at any given time.
3. Flexibility: As server nodes that are used in the application layer could be developed, configured and tested separately without affecting any of the other layers, and then it could be added to the system on demand. Readily PaaS (Platform as a Service) is used to maintain consistency and Scalability.
4. Latency reduction: As the nodes are distributed on data centers that are spatially close to the end-user as possible. This is basically handled by adopting open PaaS (Platform as a Service) where the infrastructure complexity of servers are maintained and optimized by professionals.
5. Anytime data processing regardless of the user's connection speed. The connection between the application layer and the data layer is separated from the presentation layer, reducing the bandwidth load on the user's side.

3 ACTORS

We use the term "Actor" to refer to different users/stakeholders within the built environment. "Usability" is the critical factor in obtaining "information" from data. Data obtained from the built environment does not hold equal usefulness for all stakeholders. Furthermore, in its raw format, IoT data consists of many problems, including noise, missing data, non-valid data, ill-labeled data points, duplicates, and non-standardized IoT representation. These problems should be dealt with before delivering this data to non-expert users. We define some actors or stakeholders who are involved in the building's life cycle energy modeling and data analysis and their corresponding data usage based on a systematic review conducted by Zou et.al [48] of 10 years publications till 2017 and using Natural language processing (NLP) [4].

Then, we identified three main data-related actions, namely, **End-use programming**, **Data monitoring**, and **Data input**. Different stakeholders are clustered based on these three activities (Figure 4). Then, these activities are reflected in the presentation layer *(explained in section 4)* as three major components: 1) Hybrid VPL/TPL interface, 2) Interactive, shareable, embedded dashboards, and 3) Shareable, embedded forms. The interface of the platform is shown in figure 5.

4 PRESENTATION LAYER

The presentation layer acts as the interface for the users. Each new project has a unique global unique identification (GUID), which is stored on the data layer and could be called using its GUID. Three main functions used in the presentation layer: (1) Visual/Textual programming interface, (2) dashboards, and (3) forms are used.

4.1 VPL/TPL interface

Both the visual and the textual programming interface use two main languages: Python and JavaScript. However, the selection of which is based on the complexity of the task and the response time required. For example: if the component function requires a real-time reaction, then it is more suitable to use a JavaScript component/script. We refer to the component in this case as *"Shallow Function"*. While if the function requires heavy calculation such as Machine Learning (ML), or Energy Simulation (ES), then Python-based components/scripts are used and are referred to as *"Deep Functions"*. Examples are shown in figure 6.

Each component must follow specific abstraction schema that defines its structure and relationships as well as its types. Mainly, it consists of inputs, the function body, and outputs. However, other information must be provided when defining a new component e.g., The `dataflowType`: either shallow or deep, the component `type`: numeric, panel, optionList, ListItem, Plot, Generic, etc., the `category` where this component falls into, and other supplementary features such as the color, documentation, and license.

Each component has a unique GUID on a project level. This GUID is used to trigger the component and call its inputs and outputs using the TPL panels. For example: the deep component (Figure 6) has a GUID : `'c56ad1cf-47ea-4fe3-805b-84d104ecff8b'` could be triggered using the Python API . This allows the end-use programmer to extend the functionality of the components by running blocks of codes either before running the component (i.e. doing preprocessing to the inputs) or after running the component (i.e. doing post-processing to

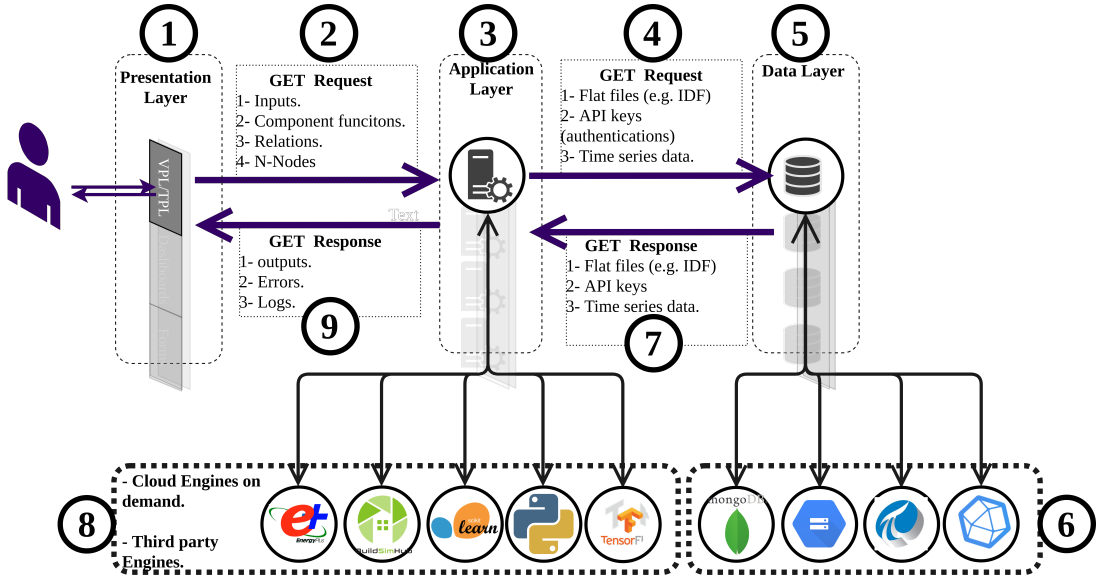

Figure 3. This figure shows the flow of data within the three tiers: (1) The actor interact with the presentation layer three ways: programming, viewing dashboards, or giving feedback; (2)The user interaction results in sending requests to the application layer to be processed; (3)The application layer starts processing inputs and relevant functions; (4) If there is any required data from the databases, a QUERY is sent to the data layer; (5) then, the data layer respond with the corresponding data from different databases (6,7); (8) The application layer then runs the required components using the distributed engines; (9) The output of step (7) is responded back to the presentation layer along with the logs/errors in JSON format. higher resolution image can be downloaded here

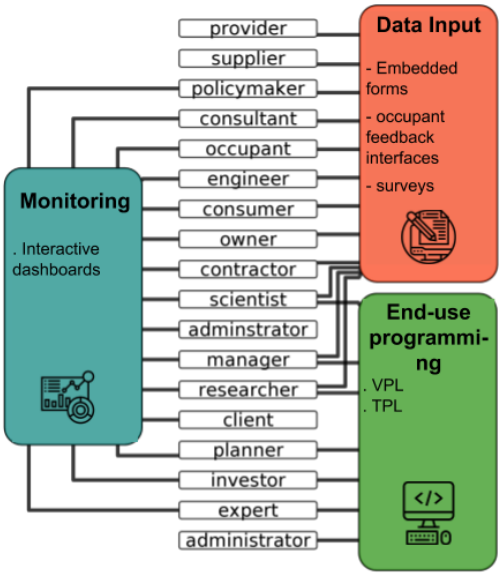

Figure 4. Actors are functionally clustered into three categories: Data monitoring, Data input, and End-use programming. Those categories constitute the Presentation layer.

the outputs) using the functions `before(callback)` and `after(callback)` as shown in listing 1 and figure 8.

```
import buildFit as bf
import json

component = bf.getComponent.by_guid("c56ad1cf-47ea
    -4fe3-805b-84d104ecff8b")
if component.inputs[0] == None:
    compoent.inputs[0] = "http://google.com"
```

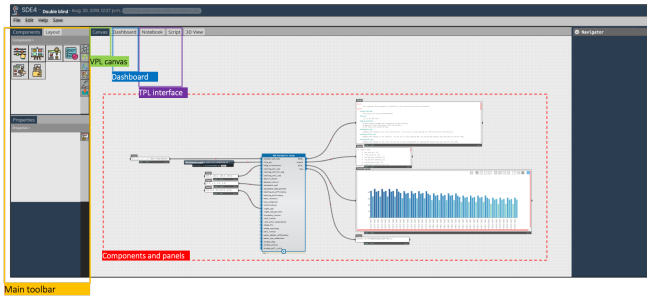

Figure 5. The main interface functions – a higher resolution image can be viewed here.

```

def preprocessInputs(input1):
    '''This function is applied to the input
    before running the component'''
    return input1

def postprocessOutputs(output1):
    '''This function is applied to the output
    after running the component'''
    return json.parse(output1)

inp1 = component.before(preprocessInputs,
    component.inputs[0])
out1 = component.after(postprocessOutputs,
    component.outputs[0])
```

Listing 1. Triggering a VPL component using TPL Pyton interface

4.2 Dashboard and forms

The dashboard and forms are located in the same tab called 'Dashboard'. It is an information management tool directed to stakeholders who are not involved directly in the development of the project i.e., to track performance, metrics, and

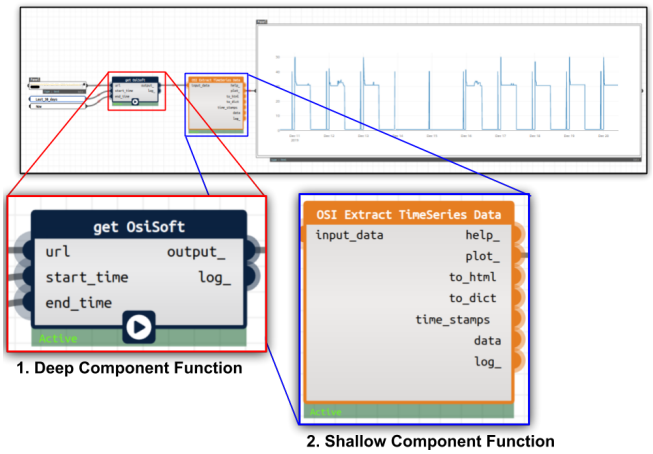

Figure 6. Two examples of Shallow and Deep functions. 1. The deep function is Python-based, and hosted on the application layer, the "play button" on the bottom of the component must be clicked to run the deep function. 2. The shallow function is JavaScript based and is hosted on the front-end. However, all the shallow functions are stored in the data layer and follow specific schema. The shallow functions are used to implement real-time actions, such as arithmetic operations, and JavaScript Object Notation (JSON) parsing. In this figure, the deep function on the left uses OSISoft Python API to request time-series data from the data layer and outputs it as a JSON object. Then, the shallow component on the right parses the JSON object and converts it into different formats, including plotting.

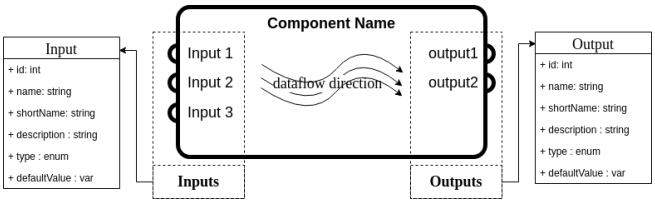

Figure 7. The data flows within components from left to the right. Each input of the component either carries a default value or receive value from other component's output, this value is stored in the input until the component runs, then the input variables are processed by calling the corresponding function. After the function is called, the return output of the function is stored back into the output of the component unless this output is connected to other component's input, it runs the other component automatically.

other key data points; or to get feedback and other user inputs (questionnaires or thermal comfort feedback). The project owner does the design of the dashboard, using the input components (e.g., numeric slider, optionList, listView, RadioButtons, and Panels) and the output components (e.g., Panels, Plots, 3D views, 2D plans, images, videos etc.). Then, the dashboard could be shared with public or specific people to view or interact. It can also be embedded within other websites.

5 APPLICATION LAYER

The application layer receives the deep function requests from the presentation layer in the form of a JSON object following specific schema. On the one hand, If there are data-related processes, it sends a request to the data layer (explained in detail in section 6. On the other hand, if there are no database-related processes, the JSON object consisted of the inputs and the component function callback. The function then starts to operate in three cases:

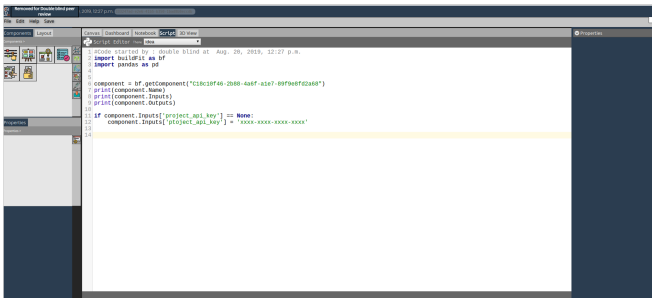

Figure 8. TPL interface using Python script. The python script enables reading, writing inputs and outputs of different components, as well as adding pre/post processing functions to the inputs/outputs – a higher resolution image can be viewed here

1. It runs the function directly from the node.
2. It starts other cloud engines to run the functions.
3. It uses third party engines (e.g. BuildSimHub) to perform the operation.

Consequently, the function response back to the presentation layer as a JSON object with the outputs (if any), Errors, and logs. The response JSON object also follows a predefined schema as shown in figure 7. The whole process is illustrated in figure 3 (steps: 1, 2, 3, 8, 9).

6 DATA LAYER

There are four major types of data involved in this platform: metadata of the users and projects, metadata of the buildings, building models, and building operational data. The data layer has three main functions: **Data querying**: to acquire data from multiple data sources, **Data storage**: to store different types of data, and **Data feed**: to respond to the requests from the application layer.

The metadata and models are either input by the users through the presentation layer or assigned by the application layer. These data are mainly text. Once passed to the data layer, they are static and stored in the relational databases (Figure 9). As for the building operational data, the time series data comes from servers of different Building Management Systems. BMS in different buildings are various, in terms of data structure, sampling rate, communication protocol, etc., making data querying a troublesome task. The platform dealt with this by deploying the PI system from OSIsoft, which queries data from different types of servers, such as BACnet (Building Automation and Control networks) and OPC (Open Platform Communications), and stores the compressed data on the local server.

Data exchange between the application layer and the data layer is done through RESTful API and mainly in JSON format. For example, if a user wants to see the energy consumption trend of a building, he/she will select the data point and define the time period in VPL canvas. The deep functions in the application layer will get the information and accordingly send the request. The API will retrieve data from the server and send it back as a JSON file, which will then be plotted in the canvas.

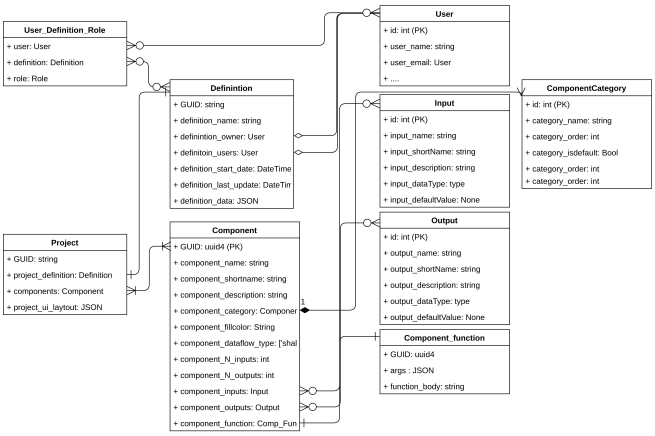

Figure 9. Relational database: A higher resolution image can be downloaded here.

7 USE CASE SCENARIO

In this section, we introduce a use case based on Cockburn's template [12] to illustrate how the platform works. The use case is a parametric energy simulation of a small office building from ANSI/ASHRAE/IES Standard 90.1 commercial reference buildings [15] shown in figure 10.

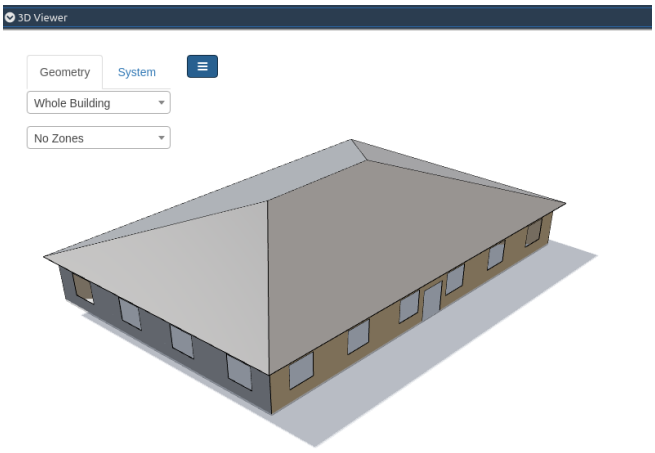

Figure 10. One of the reference buildings developed by the U.S. Department of Energy (DOE). The building represents a small office with 5,500 square feet area and one floor. The image is a screenshot taken from the embedded 3d viewer in our platform, and provided by BuildSimHub (in the application layer). an animated GIF of the 3d-viewer could be viewed here

USE CASE:1 Conduct a Parametric Energy Simulation

- -

CHARACTERISTIC INFORMATION

Context of use: EnergyModeller conducts a parametric building energy simulation of a small office building using the three-tier architecture platform.

Scope: Platform

Level: Summary

Preconditions: The user must have a valid authentication to the platform and a supported web browser (e.g. Google Chrome).

Actor(s): EnergyModeller, BuildSimHub (cloud simulation engine).

Trigger: Firstly, the EnergyModeller should start a new project. Secondly, IDF file of the EnergyPlus model of version greater than 8.0.

Description:

1. EnergyModeller starts a new definition – (Tier1 - presentation layer).
2. In the presentation layer, the EnergyModeller uploads the IDF file to the data layer (Tier 3) which is a scalable google cloud bucket using a file upload component (Figure 11). This step takes place through the application layer (Tier2) which contains a Google Cloud Storage application programming interface (API) and other file validation and security checks – an animated GIF image could be viewed here.
3. The IDF file link and global unique id (GUID) are retrieved.
4. EnergyModeller loads a "buildSimHub Parametric Study" component (Figure 12).
5. After running the component, a request is sent to the simulation engine in the application layer including the inputs (IDF file, project api key, and the simulation parameters).
6. The output of the simulation comes in a form of JavaScript Object Notation (JSON) as well as a chart (Figure 13) – a full view of the project can be downloaded here.

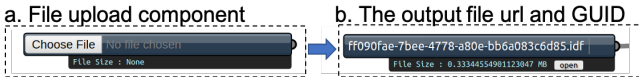

Figure 11. File upload component in the presentation layer: a) The user is prompted to select a file. The file is uploaded to a google cloud bucket; b) Then, the file could be triggered by its global unique id

8 CONCLUSION

In this paper, we explained an approach for applying Client/Server based architecture called 3-tier architecture (summarized in figure 3). This approach is used for managing data from the built environment using a cloud-based user-friendly visual programming interface. This approach depends on separating the client-side (called the presentation layer) from the back-end engines (the application layer) and the databases (the data layer). This separation eased the distribution of computation power over many nodes without compromising the efficiency of the other layers. Furthermore, it overcame some problems related to latency and connection speed. Moreover, flexibility and scalability are two key features of this type of architecture.

Currently, the platform still in the final stages of developing and debugging the alpha release. The platform will be released as an open source and contributions are welcome from the community. Contribution instructions will be available on the following GitHub repository:

https://github.com/ideas-lab-nus/paper-SIMAUD2020-three-tier-architecture-platform

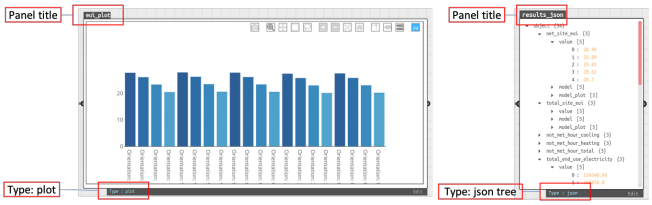

Figure 13. Display results as a chart and/or JSON tree in the presentation layer.

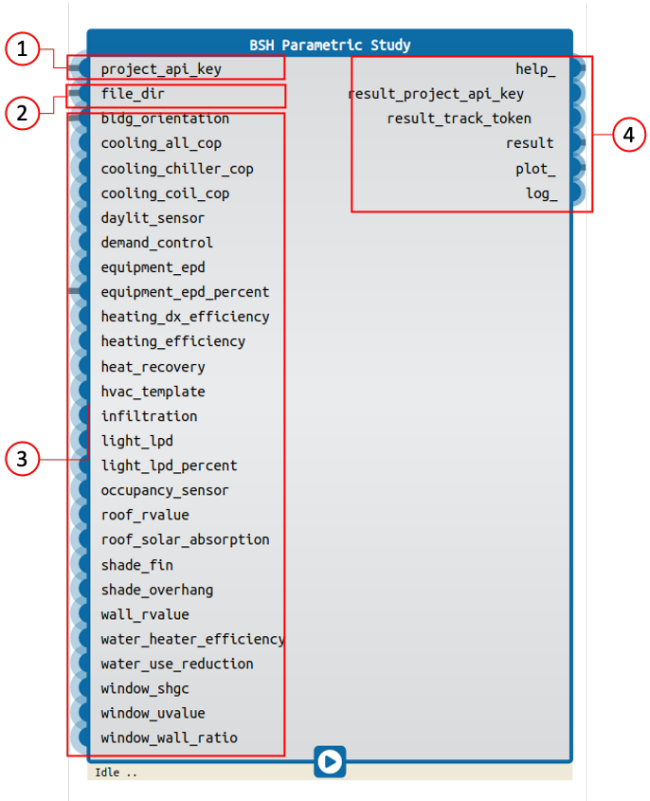

Figure 12. Parametric simulation study: this component runs a cloud simulation engine in the application layer. The inputs are: (1) The project_api_key, i.e. a unique id for each simulation project, and is used to track the project and run different simulation models' progress; (2) file_dir: which is the IDF file stored in the data layer (tier 3) in the form of a flat file; (3) The simulation parameters. After running the component, the outputs (4) come in the form of JavaScript Object Notation (JSON) object and a plot which can be displayed in a panel (Figure 13) and embedded in a website.

Future development includes enabling real-time teamwork, version controls, besides continuous testing and user-experience improvements.

ACKNOWLEDGMENTS

This paper is a part of a project funded by the Ng Teng Fong Charitable Foundation (NTFCF) research funding.

REFERENCES

1. AbdelRahman, M. GH_CPython: CPython plugin for grasshopper, 2017.
2. Abdelrahman, M. M. Enhancing Computational Design with Python high performance scientific libraries : Integration of Grasshopper and CPython language. 2–3.
3. Abdelrahman, M. M., and Toutou, A. M. Y. [ANT]: A Machine Learning Approach for Building Performance Simulation: Methods and Development. *The Academic Research Community publication 3*, 1 (2019), 205.
4. Abdelrahman, M. M., Zhan, S., and Chong, A. Building Life-Cycle Usability Data Segmentation: A NLP-based review. *Unpublished work* (2020).
5. Allcott, H., and Mullainathan, S. Behavior and energy policy, 3 2010.
6. Bachman, D. *Grasshopper: Visual Scripting for Rhinoceros 3D*. 2017.
7. Bhardwaj, S., Jain, L., and Jain, S. Cloud Computing : a Study of Infrastructure As a Service (Iaas). *International Journal of Engineering 2*, 1 (2010), 60–63.
8. Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Connallen, J., and Houston, K. A. Object-oriented analysis and design with applications, third edition. *ACM SIGSOFT Software Engineering Notes 33*, 5 (2008), 29.
9. Bragdon, A., Zeleznik, R., Reiss, S. P., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., and Laviola, J. J. Code bubbles: A working set-based interface for code understanding and maintenance. In *Conference on Human Factors in Computing Systems - Proceedings*, vol. 4 (2010), 2503–2512.
10. Brohus, H. ., Frier, C. ., Heiselberg, P., and Haghighat, F. Quantification of Uncertainty in Predicting Building Energy Consumption: a stochastic approach. *Energy and Buildings 55* (2012), 127–140.
11. Chong, A., Xu, W., and Lam, K. P. Uncertainty analysis in building energy simulation: A practical approach. *14th International Conference of IBPSA - Building Simulation 2015, BS 2015, Conference Proceedings* (2015), 2796–2803.
12. Cockburn, A., and Cockburn, A. Use Case Template Basic Use Case Template. Tech. rep., 1998.
13. De Line, R., Czerwinski, M., Meyers, B., Venolia, G., Drucker, S., and Robertson, G. Code Thumbnails: Using spatial memory to navigate source code. In *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2006* (2006), 11–18.
14. Dillon, T., Wu, C., and Chang, E. Cloud Computing: Issues and Challenges.
15. Field, K., Deru, M., and Studer, D. Using DOE commercial reference buildings for simulation studies.
16. Furht, B., and Escalante, A. Handbook of Cloud Computing. Tech. rep., 2010.
17. Green, T. R. G. Instructions and descriptions. In *Proceedings of the working conference on Advanced visual interfaces* (2004), 21–28.

18. Green, T. R. G., Petre, M., and Bellamy, R. K. E. Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against the 'Match-Mismatch' Conjecture. *Proceedings of the Fourth Annual Workshop on Empirical Studies of Programmers*, January (1991), 121–146.

19. Green Building XML (gbXML) Schema. gbXML Green Building - Current Schema, 2019.

20. Iea. Tracking Clean Energy Progress 2016: IEA Input to the Clean Energy Ministerial. *International Energy Agency (IEA) Directorat* (2013), 148.

21. Jakubiec, J. A., and Reinhart, C. F. DIVA 2.0: Integrating daylight and thermal simulations using rhinoceros 3D, DAYSIM and EnergyPlus. In *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association* (2011), 2202–2209.

22. Koltsova, A., Zurich, E., Schmitt, G., Schneider1, C., Koltsova2, A., Schmitt3, G., and Strasse, W. P. Components for parametric urban design in Grasshopper from street network to building geometry. Components for Parametric Urban Design in Grasshopper. From Street Network to Building Geometry. Tech. rep., 2011.

23. Lagios, K., Niemasz, J., and Reinhart F, C. Animated Building Performance Simulation (Abps) – Linking Rhinoceros / Grasshopper With Radiance / Daysim. *conference proceedings of SimBuild 2010*, August (2010), 7.

24. Lieberman, H., Paternò, F., Klann, M., and Wulf, V. End-User Development: An Emerging Paradigm. In *End User Development*. Springer Netherlands, 10 2006, 1–8.

25. Lieberman, H., Paternò, F., and Wulf, V. End-user development. Tech. rep.

26. Mathworks. MATLAB - Mathworks - MATLAB & Simulink, 2016.

27. McNeel, R. Grasshopper generative modeling for Rhino. *Computer software (2011b), http://www. grasshopper3d. com* (2010).

28. Mell, P., and Grance, T. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. Tech. rep., 2011.

29. MENZIES, T. EVALUATION ISSUES FOR VISUAL PROGRAMMING LANGUAGES. 5 2002, 93–101.

30. Moher, T. G., Mak, D. K. H., Blumenthal, B. B., and Levanthal, L. M. Comparing the comprehensibility of textual and graphical programs, 1993.

31. Molina-Solana, M., Ros, M., Ruiz, M. D., Gómez-Romero, J., and Martin-Bautista, M. J. Data science for building energy management: A review, 2017.

32. Nardi, B. *A small matter of programming: perspectives on end user computing*. 1993.

33. Peronato, G., Kämpf, J. H., Rey, E., and Andersen, M. Integrating urban energy simulation in a parametric environment: a Grasshopper interface for CitySim. Tech. rep.

34. Ramaswamy, S., and Tripathi, R. Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications 3* (2015), 164–173.

35. Reen, T. R. G. G., and Etre, M. P. Usability Analysis of Visual Programming Environments : A 'Cognitive Dimensions' Framework. Tech. rep., 1996.

36. Reviews, N. S. R., Energy, S., and 2012, u. Stochastic techniques used for optimization in solar systems: A review. *Elsevier*.

37. Rothermel, G., Shaw, M., and Wiedenbeck, S. The state of the art in end-user software engineering. *ACM Comput. Surv 43* (2011), 44.

38. Roudsari, M., Pak, M., Lyon, A. S. I. c. h. i., France, u., and 2013, u. Ladybug: a parametric environmental plugin for grasshopper to help designers create an environmentally-conscious design. *ibpsa.org*.

39. Scaffidi, C., Shaw, M., and Myers, B. Estimating the Numbers of End Users and End User Programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, 207–214.

40. Schuldt, H. Multi-tier Architecture. In *Encyclopedia of Database Systems*. Springer New York, 2017, 1–3.

41. Tardioli, G., Kerrigan, R., Oates, M., James, O., Procedia, D. F. E., and 2015, u. Data driven approaches for prediction of building energy consumption at urban level. *Elsevier*.

42. Taylor, P. The Three-Tier Architecture: An Object-Oriented Perspective. 1998, 79–94.

43. US Department of Energy. Technology for Building Systems Integration and Optimization – Landscape Report.

44. Wijegunaratne, I., and Fernandez, G. *Distributed Applications Engineering*. Practitioner Series. Springer London, London, 1998.

45. Wong, J., Li, H., construction, S. W. A. i., and 2005, u. Intelligent building research: a review. *Elsevier*.

46. Zhang, Z., Chong, A., Pan, Y., Zhang, C., and Lam, K. P. Whole building energy model for HVAC optimal control: A practical framework based on deep reinforcement learning. *Energy and Buildings 199* (2019), 472–490.

47. Zibion, D., Singh, D., Braun, M., and Yalcinkaya, D. Development of a BIM-enabled Software Tool for Facility Management using In-teractive Floor Plans, Graph-based Data Management and Granular Information.

48. Zou, P. X., Xu, X., Sanjayan, J., and Wang, J. Review of 10 years research on building energy performance gap: Life-cycle and stakeholder perspectives, 11 2018.