J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

A Flexible and Expandable Representation Framework for Computational Science Data☆

Jaesung Kim[1], Sunil Ahn[2]*, Jeongchoel Lee[3], and Jongsuk Ruth Lee[4]

ABSTRACT

EDISON is a web-based platform that provides easy and convenient use of simulation software on high-performance computers. One of the most important roles of a computational science platform, such as EDISON, is to post-process and represent the simulation results data so that the user can easily understand the data. We interviewed EDISON users and collected requirements for post-processing and represent of simulation results, which included i) flexible data representation, ii) supporting various data representation components, and iii) flexible and easy development of view template. In previous studies, it was difficult to develop or contribute data representation components, and the view templates were not able to be shared or recycled. This causes a problem that makes it difficult to create ecosystems for the representation tool development of numerous simulation software. EDISON-VIEW is a framework for post-processing and representing simulation results produced from the EDISON platform. This paper proposes various methods used in the design and development of the EDISON-VIEW framework to solve the above requirements and problems. We have verified its usefulness by applying it to simulation software in various fields such as material, computational fluid dynamics, computational structural dynamics, and computational chemistry

☞ keyword: Computational Science Data, Visualization, Representation, Simulation, EDISON

1. Introduction

EDISON is a platform that helps students learn computational science research methodologies by supporting high-performance computer and simulation software. [1] EDISON has over 400 simulation softwares in 7 special fields and is being used by tens of thousands of users. [2] One of the most important roles of a computational science platform, such as EDISON, is to post-process the simulation data and represent the post-processed data so that the user can easily understand the data. [3] The simulation data is a set of raw files, and it is often difficult to intuitively understand the data. Therefore, instead of representing the raw file as it is, it is necessary to refine it through post-processing and then represent it. Post-processing means extracting the desired information from the simulation data, processing the data into an understandable form, and visualizing the post-processed data.

The EDISON platform provides the service to extract metadata and refine the data to the desired form through post-processing the data through a virtualized container environment for each type of simulation data. [4] In addition, we studied how to represent the post-processed data. We collected requirements related to the representation of the simulation data through interviews with simulation software developers participating in the EDISON project. The results of the requirements are as follows. First, the developers wanted to represent various types of simulation data. The simulation data vary depending on the simulation software. In addition to basic data such as number, string, array, and image, three-dimensional data, such as molecular structure in computational chemistry and polygonal data in structural dynamics, are produced. The developers wanted this data to be visible within the EDISON platform. Thus, the EDISON platform required flexibility in data representation to support multiple data formats. Second, the developers wanted to freely represent the data without IT knowledge such as HTML. On a web-based platform, constructing a view for representing the data requires knowledge such as HTML. But for simulation software developers, learning this knowledge is a

1,2,3,4 Center for Computational Science Platform, Korea Institute of Science and Technology Information, Daejeon, 34141, Korea
* Corresponding author: Sunil Ahn (siahn@kisti.re.kr)
[Received 12 February 2020, Reviewed 26 February 2020(R2 1 April 2020), Accepted 16 April 2020]
☆ This research was supported by KISTI Program (No. K-20-L02-C05-S01), the EDISON Program through the National Research Foundation of Korea (NRF) (No. NRF-2011-0020576).
☆ A preliminary version of this paper was presented at APIC-IST 2019.

time-consuming job. They want to freely layout the data. For example, a software developer in computational chemistry wants to organize the information derived from post-processing as follows. The developer requests that the material properties information be displayed in the form of a table on the left side of the screen, the structural information should be visualized and placed on the right side of the screen, and the numerical values related to the structure should be placed below. Another field, computational fluid dynamics, a developer requires that the simulation input values be represented in the form of a table at the top of the screen and that the analysis results be placed at the bottom of the screen in three-dimensional representation. They also demanded an environment where users can share and reuse the configured data screens. Third, the developers requested the use of various web representation tools that are already developed. Instead of developing new representation tools, they wanted to reduce the development costs and increase the productivity by recycling good open-source tools that were already developed. Currently, there are well-developed tools such as Jsmol[5], Nglviewer[6], and 3Dmol[7] for represent ing molecular structures, Paraview glance[8] for representing polygonal data. The developers in computational chemistry demanded the use of Jsmol for molecule representation, and the developers in computational fluids dynamic and structual dynamics demanded the use of Paraview for polygonal structure representation. The software developers thought that importing the open-source representation tools, developed by web-developers all around the world, into the EDISON platform is essential for effective representation. Thus, the EDISON platform required a scalable environment for easily registering new representation tools.

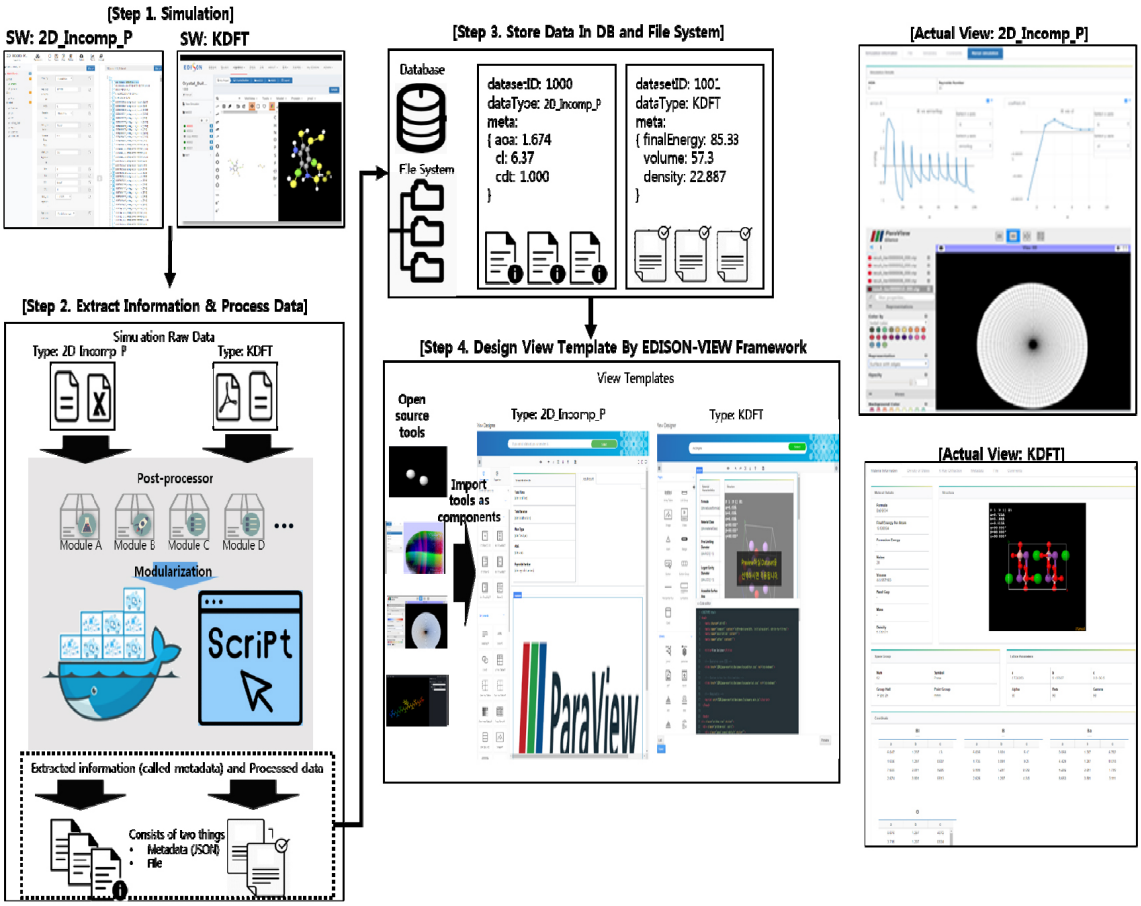
The proposed approach[9] in the existing studies is difficult to flexibly represent the data, and it is not easy to develop or contribute the data representation component. This causes a problem that makes it difficult to create an ecosystem for the development of representation tools for a large number of simulation software existing in the platform. In a computational science platform like EDISON, it is important that the environment and ecosystem that anyone can easily develop and share the representation tools as much as sharing software and its content. [10] We developed the EDISON-VIEW framework to represent the simulation data based on the requirements of the EDISON communities. This paper presents various methods used in the development of EDISON-VIEW to solve the above requirements and problems. The EDISON-VIEW provides a view editing service that can display various types of post-processed data on the view in a simple notation and provide various representation components that can represent the various data such as images, charts, morecular structures, polygonal structures, etc. Second, the EDISON-VIEW solves the representation flexibility problem by allowing the data to be placed freely using the drag-and-drop method. It also provides the ability to share and reuse through the free import and export environment of the configured data screens. Third, the EDISON-VIEW provides a way to solve representation tool scalability problem by simply importing various open-source web representation tools into the framework.

The composition of this paper is as follows. Section 2 introduces the related work, and Section 3 presents the method used in the EDISON-VIEW framework to solve the requirements of representation gathered from the EDISON communities. Section 4 presents examples of the EDISON-VIEW framework in each field, and Section 5 concludes this paper.

First, the EDISON-VIEW uses AngularJS's notation to easily and flexibly represent JSON-formatted metadata. AngularJS uses {{data}} notation to represent the data. When displaying 'reducedFormula' in the form of string and 'volume' in the form of number on the screen, users can represent them in the form of {{dm.reducedFormula}} and {{dm.volume}}. And if the users want to extract only specific data from array, it is possible to represent individual data in the same way as {{dm.lattice [0]}}, {{dm.lattice [1]}}, and {{dm.lattice [2]}}.

2. Related Work

The NanoHub[9] Project, which was started in 1995 at Purdue University with the aim of (1) developing and collecting computational science software related to the nanotechnology field, (2) providing an easy way to utilize it in an online environment, and (3) providing computational science education for nanoscience. It is HubZero[11] Project



(Figure 1) Example: The steps from simulation to data representation

that started with the success of the NanoHub project. The HubZero Project aims to develop and disseminate computational science software and platform that utilizes computing resources. Rappture[12] is a toolkit that makes it easy to create I/O interface of computational science software on the HubZero platform. The I/O interfaces of the Rappture are defined in XML format and expressed in GUI (Graphical User Interface). The mapping and conversion from I/O of the Rappture to I/O of actual software can be defined in various languages. The Rappture aims to make the I/O interface of computational science software easily, but it has the following disadvantages. First, the layout of I/O interfaces is fixed. Second, 10 components such as graph, molecule, and polygonal data are provided as basic components for I/O representation, but it is very difficult to add and develop new components. Third, mapping and conversion code for each software is not shared among users, resulting in poor reusability.

3. EDISON-VIEW Framework

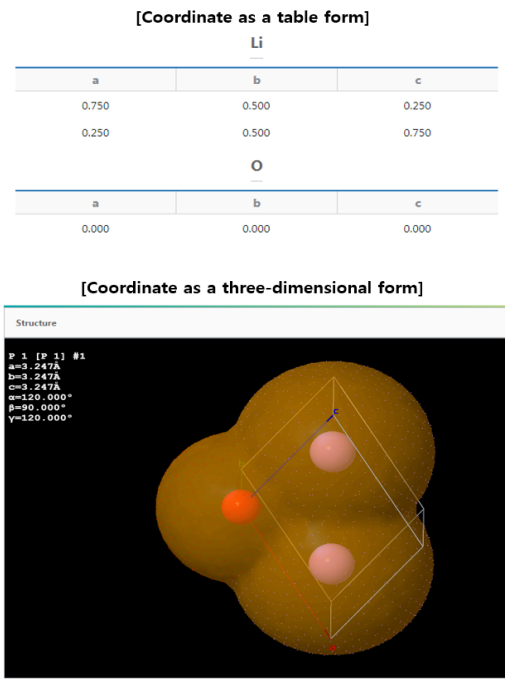
This section introduces the solutions used in the EDISON-VIEW framework to solve the EDISON community's requirements. The whole step of post-processing and representation of the simulation data of the EDISON platform is shown in Fig 1. The simulation data generated from each simulation software is processed into metadata in a JSON format and standardized files through post-processing modules for each type. The JSON metadata and standardized files are assigned a dataset id and stored in the database and

file system separately. With the EDISON-VIEW framework, users can then quickly and easily create view templates using the data stored in the platform. The created view templates are stored in the database for sharing and reusing. When the users search the actual dataset, the view is created by inputting the actual data value into the template.

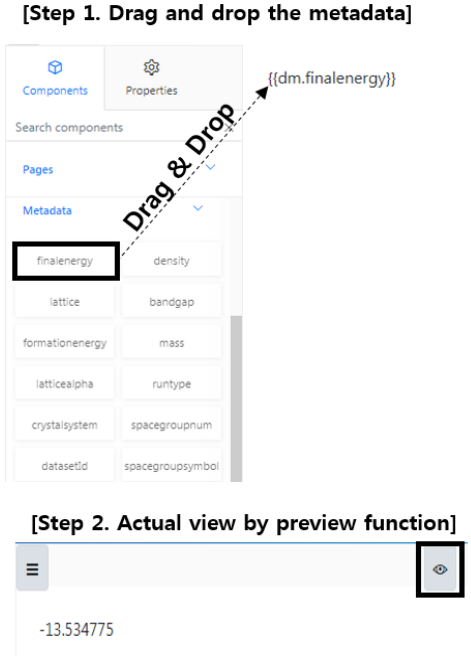
3.1 Flexible Data Representation

In the EDISON-VIEW framework, users can easily configure the data view template for each type of data derived from each simulation software. The simulation data that is extracted by post-processing varies depending on the software. The extracted simulation data is called metadata. For example, the metadata derived through 2D_Comp_P software in the field of computational fluid dynamics are {“flowType”: “laminar”, “aoa”: 1.674, “cl”: 6.38, “cdt”: 1.000}, and the metadata derived from VASP [13] software in the field of material is {“reduced Formula”: “Li2O”, “volume”: 57.3, “finalEnergy”: 85.33, “coordinate”: [{“value”: [0.750, 0.500, 0.250], “label”: “Li”},{“value”: [0.250, 0.500, 0.750], “label”: “Li”},{“value”: [0.000, 0.000 , 0.000], “label”: “O”}], “lattice”: [3.247, 3.247, 3.247]}. These metadata come in many forms, including string, number, array, and so on. Different representation methods must be supported because the same data can be represented in different forms. When representing the data of ‘coordinate’ as mentioned above example, as can be seen in Fig 2, some users may want to display individual data one by one, or they may want to display the data in three-dimensional form. To represent these various types of data, the EDISON-VIEW provides two methods.

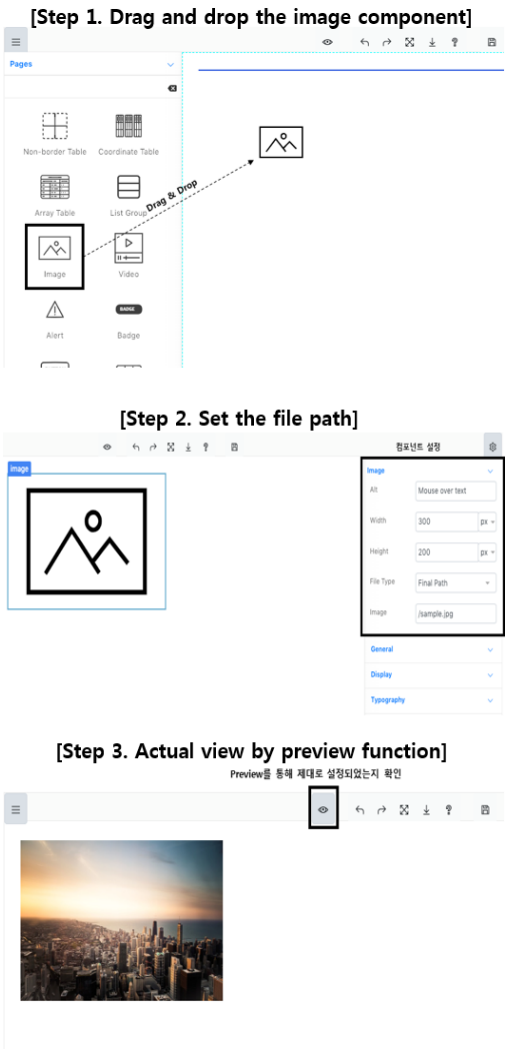
First, the EDISON-VIEW uses AngularJS’s notation to easily and flexibly represent JSON-formatted metadata. AngularJS uses {{data}} notation to represent the data. When displaying ‘reducedFormula’ in the form of string and ‘volume’ in the form of number on the screen, users can represent them in the form of {{dm.reducedFormula}} and {{dm.volume}}. And if the users want to extract only specific data from array, it is possible to represent individual data in the same way as {{dm.lattice [0]}}, {{dm.lattice [1]}}, and {{dm.lattice [2]}}.



(Figure 2) Example: Two types of representation forms for same data



(Figure 3) Example: Drag and drop the metadata



(Figure 4) Example: Drag and drop the component

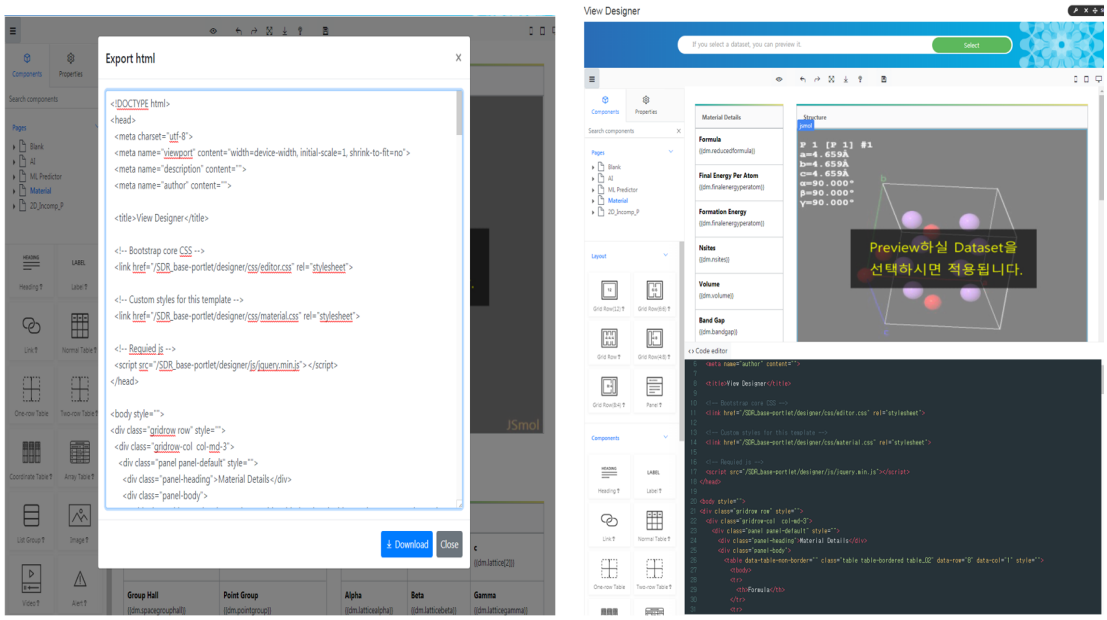
Second, EDISON-VIEW provides a variety of web-based representation components. Rather than simply showing string and number, it provides representation components to represent the data in various forms such as images, charts, and three-dimensional representations. The representation components provided by the EDISON-VIEW and the types of data formats that can be represented are shown in Table 1. The user represents the simulation result by matching extracted files to each component. The user can set the file path as the input value of the component. For example, as shown in Fig 4, the user use an image component and specify the path to the file system where the image file is stored. In addition to the above components, basic HTML building components such as grid, panel, table, link, and label are also provided.

The users can use components to customize the view templates for each data type. The EDISON-VIEW provides the preview function that previews the actual view using temporary simulation data for the configured view template. Fig 3 is an example of placing the metadata by draggging & dropping and using the preview button to check the actual view with the real data. Fig 4 shows an example where the user drag and drop the image component, set the path to the image file, and use the preview button to check the actual view with the real image data. The preview allows the user to check the view where the data is actually displayed and to interactively modify the data. As shown in Fig 5, the created view template can be exported and downloaded as an HTML file, and it is also possible to import the HTML file to the EDISON-VIEW as a view template and edit the view template and provide an environment that can be shared and reused.

3.2 Recycling Web-based Representation Components

There is a limitation in representing simulation data using simple components such as text, table, image, and chart. Users can understand data more easily by using various representation components. Recently, with the development of web technology, most representation tools are being developed using web technology and open-source tools for representing various simulation data such as Jsmol, Paraview, and Nglviewer have also been developed. We take advantage of ways to reuse existing web representation tools developed by third parties to support various types of simulation data.

The EDISON-VIEW framework allows modularization of web-based representation tools developed by third parties, allowing each tool to be added, removed and updated. This provides the scalability to embrace a variety of web-based tools. The registration of a new component consists of defining two parts: i) define the representation component, ii) define the input of representation component and execution



[Export the view template as a html file] [Import the html file as a view template]

(Figure 5) Export and import functions

```
componentsGroup.push('Library/jsmol');
Vvveb.ComponentsGroup['Library'] = componentsGroup;

Vvveb.Components.extend("_base", "Library/jsmol", {
    name: "jsmol",
    attributes: ["data-component-jsmol"],
    image: "icons/custom/icon_05_jsmol.svg",
    dragHtml: '<img src="' + Vvveb.baseUrl + 'icons/chart.svg">',
    html: '<div data-component-jsmol class="jsmol" style="width:400px; height:400px;" \
    data-config=\'{\
        "width" : "100%",\
        "height" : "100%",\
        "color" : "#FFF",\
        "addSelectionOptions" : false,\
        "use" : "HTML5",\
        "j2sPath" : "/SDR_base-portlet/js/jsmol/j2s",\
        "script" : "set antialiasDisplay; load ",\
        "disableJ2SLoadMonitor" : true,\
        "disableInitialConsole" : true,\
        "allowJavaScript" : true\
    }\' data-file-type="{{finalPath}}" data-file-path="/POSCAR" data-file-list="{{fileList}}">\
        <img width="100%" height="100%" src="/SDR_base-portlet/images/designer/jsmol_preSample.jpg" style="pointer-events:none;"/>\
    </div>',
});
```

1. Identify the component
2. Select icon of the component
3. Define the default file path and the default size of component
Additional parts for Jsmol

(Figure 6) Registration of jsmol component

code.

The first part is the part that identifies a component. In this step, the name of the component and the representative icon are registered. The HTML attributes for inputting the file path are also defined in this part. As shown in Fig 6, 7, and 8, most of the information is formalized in this part, so all you need to do is type the name and path of icon to identify the component.

The second part is the main part that writes the executable code. In this part, as can be seen in Fig 9, we import the open-source libraries for execution, map the file path of the data to the input of the component, and write the execution code. In this way, open-source representation tools were imported as components of the EDISON-VIEW

```
componentsGroup.push('Library/paraview');
Vvveb.ComponentsGroup['Library'] = componentsGroup;

Vvveb.Components.extend("_base", "Library/paraview", {
    name: "paraview",
    attributes: ["data-component-paraview"],
    image: "icons/custom/icon_06_paraview.svg",
    dragHtml: '<img src="' + Vvveb.baseUrl + 'icons/chart.svg">',
    html: '<div data-component-paraview class="paraview" style="width:1000px; height:800px;" \
           data-browser="Chrome|Opera|Firefox|Safari"\
           data-file-type="{{finalPath}}" data-file-path="/output.vtp" data-file-list="{{fileList}}">\
               <img width="100%" height="100%" src="/SDR_base-portlet/images/designer/paraviewSample.jpg" style="pointer-events:none;"/>\
           </div>',
```

1. Identify the component
2. Select icon of the component
3. Define the default file path and the default size of component

(Figure 7) Registration of paraview component (1/2)

```
componentsGroup.push("Library/rlt2chart");
Vvveb.ComponentsGroup['Library'] = componentsGroup;

Vvveb.Components.extend("_base", "Library/rlt2chart", {
    name: "rlt2chart",
    attributes: ["data-component-rlt2chart"],
    image: "icons/custom/icon_10_xrd.chart.svg",
    dragHtml: '<img src="' + Vvveb.baseUrl + 'icons/chart.svg">',
    html: '<div data-component-rlt2chart class="rlt2chart" style="width:800px; height:500px; background-color: #eeeeee;" \
           data-file-type="{{finalPath}}" data-file-path="/error.rlt" data-file-list="{{fileList}}">RLT File To 2D Chart</div>',
```

1. Identify the component
2. Select icon of the component
3. Define the default file path and the default size of component

(Figure 8) Registration of rlt2chart component

```
var body = Vvveb.Builder.frameBody;

if ($(".paraview-script", body).length == 0)
{
    $(body).append("<script class='paraview-script' src='/SDR_base-portlet/js/glance/glance.js' type='text/javascript'></script>");
    $(body).append("<script class='paraview-script' src='/SDR_base-portlet/js/glance/glance-external-ITKReader.js' type='text/javascript'></script>");
    $(body).append("<script class='paraview-script' src='/SDR_base-portlet/js/glance/glance-external-Workbox.js' type='text/javascript'></script>");
    $(body).append(
        "<script class='paraview-script' type='text/javascript'>\
        $(document).ready(function() {\
            $('.paraview').each(function (index) {\
                var self = $(this).attr('id','paraview_'+index);\
                if(!validate(self)) return true;\
                var container = document.querySelector('#paraview_'+index);\
                var viewer = Glance.createViewer(container);\
                var path = this.dataset.fileType + this.dataset.filePath;\
                var fileName = this.dataset.filePath;\
                $(container).children().children().eq(1).children().css('height',$( container ).height()-$(container).children().children().eq(0).height());\
                if(fileName.indexOf('*') > -1) {\
                    var index = path.indexOf('SDR_baseportlet_path');\
                    var oldPath = path.substring(0, index+21);\
                    var newPath = path.substring(index+21);\
                    var multiPath = $(this).attr('data-file-list');\
                    var multiDatasetPath = multiPath + newPath;\
                    $.ajax({\
                        url: multiDatasetPath,\
                        type: 'GET',\
                        dataType: 'json',\
                        success: function(data){ \
                            for(var i in data){\
                                var fileName = data[i].fileName;\
                                var filePath = oldPath + data[i].filePath;\
                                viewer.loadURL(fileName, filePath);\
                            }\
                        },\
                        error: function() { alert('File Path Error!!!');}\
                    });}\
                else{\
                    viewer.loadURL(this.dataset.filePath, path);\
                }\
            });\
        });\
        </script>"
    );
}
```

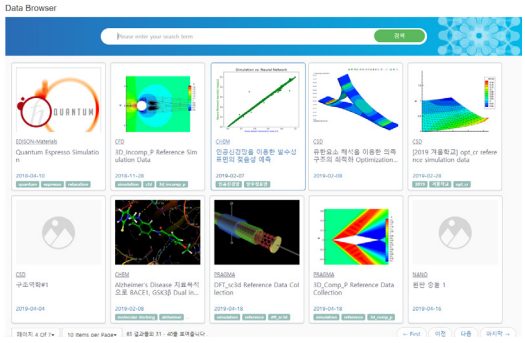
1. Import the paraview library
2. Map file path to the input of paraview
3. Write the execution code

(Figure 9) Registration of paraview component (2/2)

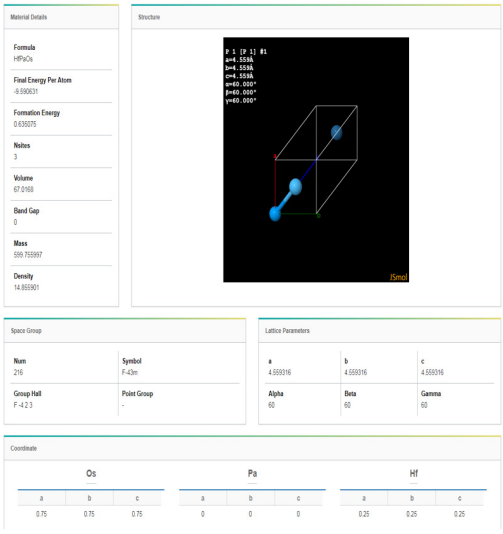
framework. Currently, we have provided 7 open-source representation components on the EDISON-VIEW framework. The code in these two-steps is written based on Javascript.

4. Use-cases of EDISON-VIEW

The EDISON-VIEW framework is applied to 13 simulation softwares in the field of computational fluid



(Figure 10) The collections of simulation data in EDISON platform



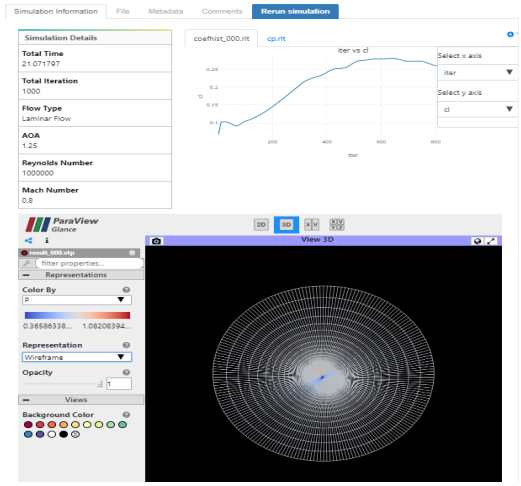
(Figure 11) Example: Data view for VASP

dynamics, computational structural dynamics, nano-physics, and materials. A total of about 30 simulation results were collected on the EDISON site(www.edison.re.kr), and the EDISON-VIEW framework was used to represent the data of each simulation result. Fig 10 shows a list of data collections of the simulation results. Fig 11, 12, and 13 show the result of various data view for softwares such as VASP, Quantum Espresso, and 2D_Incomp_P.

What's noteworthy about data representation in VASP software and Quantum Espresso[14] software in the field of materials is that the generated view templates are shared and expanded. The simulation results of VASP software are files

(Figure 12) Example: Data view for Quantum Espresso

such as POSCAR, WAVECAR, INCAR, and OUTCAR. The simulation results of Quantum Espresso software are files such as CIF, IN, and OUT. The simulation results of the two softwares are different, but the post-processed metadata is similar: formula, spacegroup, density, coordinate, pointgroup, lattice and volume. In Quantum Espresso, additional values such as energy cutoff, run type, and the number of kpoints are extracted. Since the information we want to show in the two software is similar, we just configure and register the view template for VASP software, then import the VASP view template for Quantum Espresso software and expand it by adding additional information such as energy cutoff, run type, and the number of kpoints as shown in Fig 14. And we drastically reduced the cost of representation using Jsmol component. Among the results of the VASP software, the POSCAR file is data representing the molecular structure information. Among the results of the Quantum Espresso software, the cif file is also data representing the molecular structure information. Rather than developing each representation component to represent the files in different formats, we import and use the Jsmol component that embraces both formats.



(Figure 13) Example: Data view for 2D_Incomp_P

(Figure 14) Example: View templates for VASP and Quantum Espresso software

(Table 1) Representation components

Component	Format	Field	Open-source Self-developed
Jsmol	POSCAR, cif, xyz, mol	Material, Chemical, Nano-physics	Open-source
Paraview	plt	Fluid, Structural dynamics	Open-source
pdf	pdf	General	Open-source
epub	epub	General	Open-source
xrd	json	Material	Self-developed
dos	json	Material	Self-developed
rlt2chart	rlt	Fluid dynamics	Self-developed
html	html	General	Self-developed
x3dom	x3d	Structural dynamics	Open-source
p3d	p3d	Fluid, Structural dynamics	Self-developed
ngl	pdb, cif, cube	Material, Chemical	Open-source
csv	csv	General	Open-source
text	txt	General	Self-developed
onedviewer	oneD	General	Self-developed
sc3dviewer	js (dedicated to dft_sc3d)	Nano-physics	Self-developed
atomtransistor	js (dedicated to web_io_siesta)	Nano-physics	Self-developed

5. Conclusion

One of the most important roles of the computational science platform is to post-process and represent the simulation data so that the user can easily understand the data. In this paper, we have analyzed issues such as the difficulty in representing simulation data, the lack of extensibility of data presentation components. We have provided a reference model called EDISON-VIEW that can solve these issues. We have provided flexible data representation using {{data}} notation, and a variety of web-based representation components. We have verified the usefulness of our framework by applying it to create view templates for various simulation software in the field of computational fluid dynamics, computational structural dynamics, and materials. We expect that the EDISON-VIEW will be used for

representation of various simulation data in various fields in the future.

References

[1] JL Yu, et al., “EDISON Platform: A Software Infrastructure for Application-Domain Neutral Computational Science Simulations”, Future Information Communication Technology and Applications, pp. 283-291, 2013. https://doi.org/10.1007/978-94-007-6516-0_31

[2] J. Ma, J. Seo, J. s. Ruth-Lee and M. j. Park, ″Implementation and Application of the EDISON platform’s integrated file management service,″ Journal of Internet Computing and Services, vol. 17, no. 6, pp. 71-80, 2016. https://doi.org/10.7472/jksii.2016.17.6.71

[3] N. On, N. Kim, K. Ru, H. Jang and J. R. Lee, ″An Analysis of the Factors Affecting User Satisfaction in Computational Science and Engineering Platforms: A Case Study of EDISON,″ Journal of Internet Computing and Services, vol. 20, no. 6, pp. 85-93, 2019. https://doi.org/10.7472/jksii.2019.20.6.85

[4] S. Ahn, J. Lee, J. Kim and J. R. Lee, “EDISON-DATA: A flexible and extensible platform for processing and analysis of computational science data,” Software: Practice and Experience, vol. 49, pp. 1509-1530, 2019. https://doi.org/10.1002/spe.2732

[5] Jmol: an open-source Java viewer for chemical structures in 3D. http://www.jmol.org/

[6] Rose, Alexander S, and Peter W Hildebrand. “NGL Viewer: a web application for molecular visualization.” Nucleic acids research, vol. 43, W576-9, 2015. https://doi.org/10.1093/nar/gkv402

[7] N. Rego and D. Koes, ″3Dmol.js: molecular visualization with WebGL,″, Bioinformatics, vol. 31, pp. 1322-1324, 2015. https://doi.org/10.1093/bioinformatics/btu829

[8] Ahrens, James, et al., “ParaView: An End-User Tool for Large Data Visualization, Visualization Handbook”, Elsevier, ISBN-13: 978-0123875822, 2005. https://paraview.org

[9] Gerhard Klimeck, Michael McLennan, Sean B. Brophy, George B. Adams III, Mark S. Lundstrom, ″nanoHUB.org: Advancing Education and Research in Nanotechnology,″ IEEE Computers in Engineering and Science (CISE), Vol. 10, pp. 17-23, 2008. https://doi.org/10.1109/MCSE.2008.120.

[10] Y. Kwon, I. Jeon, J. Ma, S. Lee, K. Cho and J. Seo, “A Study on Workbench-based Dynamic Service Design and Construction of Computational Science Engineering Platform,” Journal of Internet Computing and Services, Vol. 19, No. 3, pp. 57-66, 2018. http://dx.doi.org/10.7472/jksii.2018.19.3.57

[11] McLennan, Michael, and Rick Kennell. “HUBzero: a platform for dissemination and collaboration in computational science and engineering”, Computing in Science & Engineering, 2010. https://doi.org/10.1109/MCSE.2010.41

[12] MCLENNAN, M, “The rappture toolkit”, Article (CrossRef Link), 2004. https://nanohub.org/infrastructure/rappture/

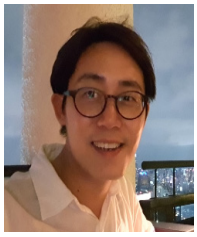
[13] G. Kresse and D. Joubert, Phys. Rev. 59, 1758, 1999. vasp.at

[14] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. Fabris, G. Fratesi, S. de Gironcoli, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, R. M. Wentzcovitch, J.Phys.:Condens.Matter 21, 395502, 2009. http://dx.doi.org/10.1088/0953-8984/21/39/395502

◐ 저 자 소 개 ◑

김 재 성(Jaesung Kim)
2013년 한양대학교 컴퓨터공학부(공학사)
2015년 한국과학기술원 전산학부(공학석사)
2016년~2020년 한국과학기술정보연구원 국가슈퍼컴퓨팅본부 기술원
2020년~현재 SK C&C 플랫폼1그룹 엔지니어
관심분야 : 인공지능, 클라우드
E-mail : kaka7537@gmail.com

안 선 일(Sunil Ahn)
1999년 서울대학교 전산학과(이학석사)
2010년 서울대학교 대학원 컴퓨터공학과(공학박사)
2004년~2005년 IITA 텔레매틱스 PM실
2005년~현재 KISTI 국가슈퍼컴퓨팅본부 책임연구원
관심분야 : 정보시스템, 분산컴퓨팅, 인공지능
E-mail : siahn@kisti.re.kr



이 정 철(Jeongcheol Lee)
2008년 충남대학교 컴퓨터공학과(공학사)
2014년 충남대학교 대학원 컴퓨터공학과(공학박사)
2015년~2017년 미국 UCLA 대학원 컴퓨터과학과 박사후연구원
2017년~현재 KISTI 국가슈퍼컴퓨팅본부 선임기술원
관심분야 : 인공지능, 계산과학, 사물인터넷, 무선네트워크
E-mail : jclee@kisti.re.kr

이 종 숙(Jong-Suk Ruth Lee)
2001년 University of Canterbury, Department of Computer Science and Software Engineering, New Zealand(공학박사)
2002년~현재 한국과학기술정보연구원(KISTI) 책임연구원, 계산과학플랫폼센터(센터장)
2005년~현재 한국과학기술연합대학원대학교(UST) 슈퍼컴퓨팅 전공(겸임 정교수)
관심분야 : 계산과학데이터플랫폼, 스마트러닝, 빅데이터, 분산병렬처리, 슈퍼컴퓨터
E-mail : jsruthlee@kisti.re.kr