Journal of Convergence Information Technology
Volume 5, Number 4, June 2010

PSE Park: Framework for Problem Solving Environments

[1,2]Hiromichi Kobashi, [2]Shigeo Kawata, [3]Yasuhiko Manabe, [4]Masami Matsumoto
[5]Hitohide Usami, [2]Daisuke Barada
[1] Fujitsu Laboratories Limited
[2]Graduate School of Engineering, Utsunomiya University
[3] Numazu National College of Technology
[4] Yonago National College of Technology
[5] Tamagawa K-12 & University
doi: 10.4156/jcit.vol5.issue4.23

Abstract

In this paper, we introduce a new framework called PSE Park for constructing a Problem Solving Environment (PSE); it enables us to construct PSEs easily. PSE Park outputs PSEs depending on user's demand/input. In this sense, PSE Park is a kind of PSE for PSE, and helps users to construct PSEs. PSE Park consists of four engines: PIPE server, core, registration engine, and console. A PSE designed and constructed in PSE Park consists of several cores, which are functions of a PSE. The PIPE server manages the cores on the basis of the core map, which expresses the flow of the cores for a specific PSE. The output of each core is retrieved and merged by the PIPE server. All outputs of the cores are saved and easily reused. The cores are independent of programming languages because each core is executed individually as a process in PSE Park. They are registered by using the registration engine, and users access the engines via the console. All data including the core itself, definitions related to the core, the core map, results, and so on are stored in a distributed key-value store on the cloud computing environment. PSE Park retrieves the data by using a key name that can identify individual data uniquely. We applied PSE Park to develop the job execution PSE and the PSE for partial differential equation (PDE)-based problems. The job execution PSE helps Finite Difference Time Domain (FDTD) simulation execution. This PSE outputs the simulation results of the electric field. PDE-based PSE supports some simulation steps. Seven cores were used to construct this example PSE. By using this PSE, users can execute a PDE-based simulation and obtain a detailed document about the simulation. We believe that the concept of PSE Park, i.e., a framework for PSE development, presents a meaningful new direction for problem solving environments.

Keywords: *Computer-assisted simulation, PSE, PDE, FDTD*

1. Introduction

In scientific computing and e-Life (lifestyles revolving around technology), computers, sensors, networks and so on have become essential elements. In the past few decades, computer power and algorithm power have been extraordinarily developed, although programming power has stayed relatively constant. Problem solving environments (PSEs) were conceptualized in the 1970s and were originally explored to enhance programming power. A PSE is defined as "a system that provides all the computational facilities necessary to solve a target class of problems. It uses the language of the target class and users need not have specialized knowledge of the underlying hardware or software" (E. Gallopoulo et al. [1], E. Houstus et al. [2]).

Conventional PSEs, which can be called first-generation PSEs, output computer programs, especially for partial differential equation (PDE)-based problems. First-generation PSEs include PSILAB (Y. Umetani et al. [3], C. Konno et al. [4]), ELLPACK (J. Rice et al. [5]), and NCAS (C. Boonmee et al. [6][7], T. Teramoto et al. [8], S. Kawata et al. [9][10]). These PSEs are libraries or program generation systems for scientific simulations. PSILAB and ELLPACK are black-box type program generation systems, and NCAS is a white-box type of program generation system. The research areas of second-generation PSEs have extended toward the lower layers of computing environments, including Grid and Cloud computing support. In Grid computing [11] environments, which are similar to cloud computing environments, many PSEs have been created and developed because of the complexity of the environment. Cactus Code (G. Allen et al. [12]) is a PSE for Grid computing to simulate the relativity theory, and IRIS Explore and SCIRun are data flow management PSEs (C. E. Goodyer et al. [13]). NAREGI-PSE is a portal to a Grid computing environment (S. Kawata et al. [14], H. Kanazawa et al. [15]). A PSE for Particle Image Velocimetry (PIV) is the PIV Virtual Laboratory (Y. Kadooka et al. [16]).

At present, PSE researchers are investigating a variety of fields, e.g., Cloud/Grid computing support, education support, CAE usage support, document generation support, and so on. In the near future, PSEs may be distributed to surround people to support e-Science and e-Life.

However, developing PSEs is difficult and requires intensive work. It comes from the complexity of the underlying computing environment or the lack of computing skills of domain scientists. Therefore, a PSE for PSEs, i.e., a meta PSE, is important and necessary for the future e-Science and e-Life.

For third-generation PSEs, we believe that a meta PSE system or framework enabling us to construct PSEs easily will be required. In the near future, many users may start simulations for their hobbies, for a better lifestyle, etc.; one user may have his/her own PSEs as his/her simulation systems. At that time, the PSE for PSE framework is needed because it is not that straightforward for scientists to develop PSEs on their own without any help from a framework.

We developed a framework called PSE Park as a PSE for PSE. Users who want to solve problems can access PSE Park, construct their PSEs, and execute simulations for their purposes.

We describe PSE Park in Section 2. We constructed examples PSE for Finite Difference Time Domain (FDTD) and PDE-based problems on PSE Park, and Section 3 presents adaptations of them. In Section 4, we conclude this paper.

2. PSE Park

PSE Park is a PSE construction framework that makes it easy to construct PSEs.

For easy construction of PSE, flexibility and extensibility are required. In scientific simulation research areas, discretization methods such as finite difference methods are still being improved, and many researchers want to try the new methods quickly. When a user wants to try another method or to develop a new one, the framework should meet it. The architecture of PSE Park is presented in Figure 1.

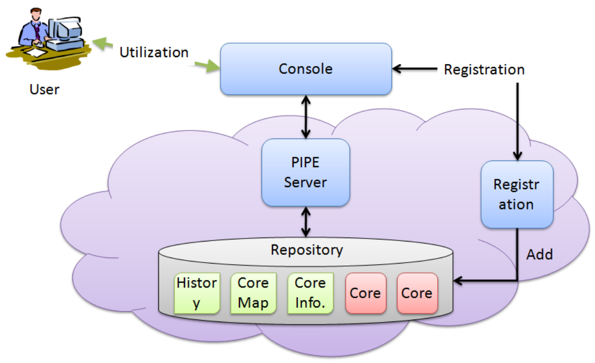


Figure 1. PSE Park Architecture

PSE Park provides functions called “core” to construct module-based PSEs for flexibility and extensibility, and each function is supported by a core. The core may a part of one PSE or a PSE itself. A PSE constructed by PSE Park consists of several cores. Users who construct PSEs connect the cores for their PSEs. The PIPE server in PSE Park handles the cores and constructs PSEs. PSE Park provides several templates, each called a core map, which expresses the PSEs of each target domain. Users access the console in PSE Park and construct/use a PSE or registered cores and a core map.

Any user may use and register cores in PSE Park. We assume three user types: developers, experts, and entry-level users (Figure 2). Developers provide cores to PSE Park and contribute to further improving and expanding cores registered to PSE Park or PSE Park itself. Experts provide and register

PSE Park core maps. Their contribution is the same as that of conventional PSE developers. They construct typical PSEs in each specific domain by using developer-provided cores. Entry-level users can run simulations easily by using the PSEs that are constructed and registered by experts.

We expect that the data sharing layer and infrastructure shown in Figure 2 are provided by cloud computing environment. They are called "Platform as a Service" (PaaS) and "Infrastructure as a Service" (IaaS), respectively in Cloud computing. The machines on which PSE Park runs are provided by IaaS. The repository in PSE Park is shared by using PaaS between machines in PSE Park. We do not describe PaaS Layer tool in this paper because they are a function of Cloud computing, not of PSE Park.

In PSE Park, a user selects cores and designs a PSE by connecting the cores. PSE Park outputs a PSE for the user's specific purpose. Therefore, PSE Park is a kind of PSE for PSEs. We describe each engine of PSE Park below.

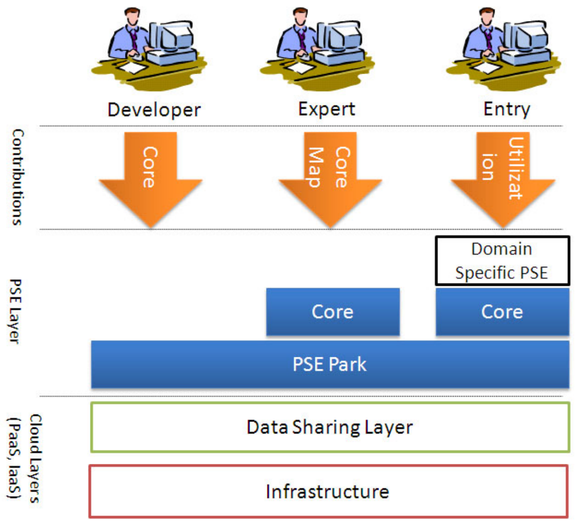


Figure 2. Types of PSE Park Users

2.1. PIPE Server

The PIPE server is the main engine of PSE Park. It manages and handles cores and constructs PSEs by connecting cores.

Users write core maps and pass them to the PIPE server. In a core map, the relations of cores are described, and on this basis, the PIPE server constructs a PSE. A core map is the similar as Taverna [17].

The difference between a usual module based framework and PIPE Server is the handling of intermediate results. When a core is executed, it refers the all variables generated by former cores. In usual module based frameworks, a module refers the previous core. This means that in PSE Park, PIPE server manages the history of global variables and the program sequence, and a core is a function. If a core returns some values, then these are retrieved as global variables by the PIPE server and passed to the next core. The same name parameters are overwritten. This is the unique characteristic of PSE Park.

Moreover, cores are connected loosely by PIPE server. This means that PIPE server keeps core's independence and flexibility. PIPE server executes cores as processes and does not compile at the execution time. Cores are independent of programming language, and users can implement cores by using a language that users know. This makes it easy for developers to develop cores and for any user to register cores into PSE Park. This is another unique characteristic of PSE Park.

By using these characteristics of PIPE server, PSE Park realizes the flexible and extensible framework. An example of a simple core map expression is shown in Figure 3 and 4. The core map in Figure 3 expresses a PSE, which has a *start* core, *stop* core, and another four cores in this example case. Arrows indicate process sequence. The PIPE Server executes the cores between the *start* core and the *stop* core. In this core map, the PIPE server executes core *A* first, and the output of core *A* is passed to core *B* as its input. If a core has multiple inputs, such as core *D*, then the PIPE server merges the outputs of core *B* and *C*. This merged output becomes the input for executing core *D*. The PIPE server stores each output, and the output can be easily reused.

The core map in Figure 4 has a list of information of core. The exchange data format of core map expression is in JSON [18]. As the core information, the core name, the core identifier, the next core identifier and parameters should be indicated. There are primitive core identifiers, “0” and “-1”. The *start* core is identified by “0” and “-1” means the *stop* core. In this example case, the core *A* is the *start* core and the core *D* indicates the *stop* core.

An execution sequence is determined by a tree, the root of which is the *stop* core. First, the execution starts recursively from the branch of the *start* core to its parent node. If a parent node has children nodes, the execution is moved to the next deepest node of the non-executed branch.

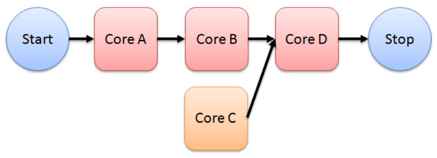


Figure 3. Core map example

```
[
{“name”:”CoreA”, “core_id”:0, “to”:1, “params”:{}},
{“name”:”CoreB”, “core_id”:1, “to”:3, “params”:{}},
{“name”:”CoreC”, “core_id”:2, “to”:3, “params”:{}},
{“name”:”CoreD”, “core_id”:3, “to”:-1, “params”:{}}
]
```

Figure 4. Core map expression

2.2. Core

A core is executed as a function of PSE. Cores collaborate with each other through the PIPE server. With regard to cores, PSE Park requires input/output definitions and an explanation of each core. These are stored in the repository. The PIPE server uses these definitions to check whether a core is connectable to the next or previous core. The PIPE server connects the output of the previous core to the input of the next core. If any unknown input parameters are found, the PIPE server asks the user which parameter should be used. An example input/output definition is shown in Figure 5.

We assume that Figure 5 corresponds to the input/output definition of cores *A* and *B* in Figure 3. These definitions are defined by a core developer. In this case, “computation target” is defined in both the input and output. PIPE sever understands that this parameter is connectable. In contrast, “equation” in the definition of the input of core *B* is not defined in the previous core *A*. The PIPE server then gives a warning about the lack of definition. If “eq” in the output of core *A* is the “equation” for the input of core *B*, the user should indicate the relation in the console of PSE Park. The “equation” in core *B* is overwritten and is passed to core *D*. The “init_condition” in the output of core *A* is not used in core *B*. If the “init_condition” is defined as the input of core *D* (Figure 3), the PIPE server can deliver it from the output of the core *A*, even the parameter is not output by core *B* and *C*. The exchange data format among cores is serialized by JSON.

Type;

string "'' , list [] , map {}

ex. 1) Output of core *A*;
"computation_target" : ""
"eq" : []
"init_condition": {}

ex. 2) Input of core *B*;
"computation_target": ""
"equation": []

ex. 3) Output of core *B;*
"equation": []

ex. 4) Input of core *D*
"init_condition": {}

Figure 5. Input/output definitions of core

2.3. Registration Engine

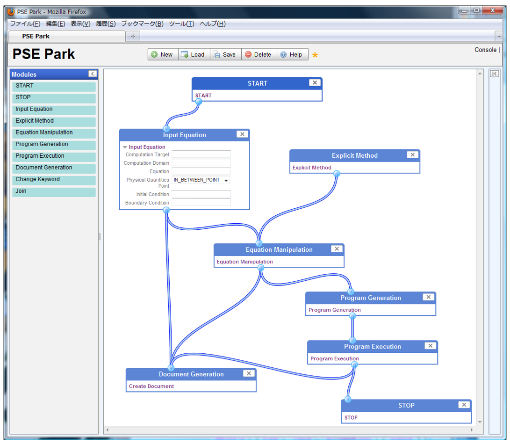
Cores are registered into PSE Park by the registration engine. For the registration of a core, the core itself and the definition of its input/output are required. Core maps can also be registered with the registration engine. After the registration, a user can use the cores and the core maps.

The registration engine saves the execution history of each core, and the history and the core map registered can be reused easily. Core and core map management are handled by the registration engine.

Functions for cores and core maps, such as "search", "remove," and "modify," are provided by this engine.

2.4. Console

The console provides an interface to the PSE Park functions. Users can create core maps, namely PSEs, by themselves or reuse previously constructed PSEs, which are registered core maps. The cores and core maps are registered from the console (Figure 6).



Physical Quantities Point: IN_BETWEEN_POINT

Figure 6. Snapshot of Console

2.5. Implementation

We show an example of engine allocation. In this example situation, we assume the use of two machines (Figure 7) for simplicity.

The console, PIPE server, registration, and repository of PSE Park can be installed on different machines because they communicate with each other through UNIX sockets. However, to remove the difficulty of allocation, we recommend deploying them in one machine, as in this situation.

User accesses are distributed by a load balancer, and users can use PSE Park via the consoles. Whichever console the user accesses, they can use every core and core map in PSE Park, because all the data are stored in the repository. For sharing of the repository data, we use a key-value data store, "x4u", developed by Fujitsu Laboratories Limited. x4u is similar to Dynamo (G. DeCandia et al. [19]), which is the typical distributed key-value data store. There are many types of data in PSE Park. Therefore it is difficult to normalize data for Relational Data Base. Key-value data store is suitable for PSE Park.

In the repository, there are three name space, "core", "core map" and "result". In "core" name space, core, input definition, output definition and core's readme are stored. The core name is used for key of core. The others are indicated by the core name plus "__INPUT__", "__OUTPUT__" or "__README__". Core map name can be set by the expert user uniquely and stored in "core map" name space. Results generated by a core are assigned a unique key by execution time.

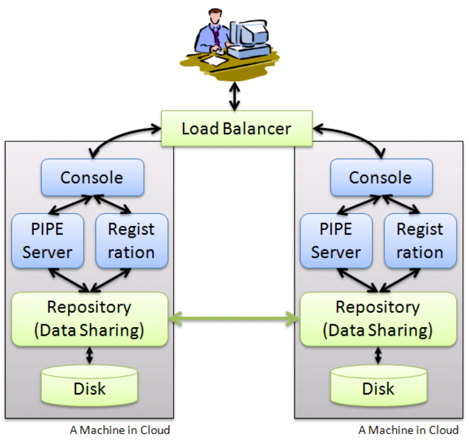


Figure 7. Allocation of Engines

3. Adaptation

We describe two examples of PSE Park applications. The first example is a job execution PSE for simulations. In this example a target problem is a simulation by FDTD method. The second is a PSE for PDE-based problem solving, and the PSE generates programs to solve PDE-based problems.

3.1. Job execution PSE

In this subsection, we show a PSE generation example for a simulation job execution by PSE Park. The job execution PSE supports simulation execution, and in this subsection a FDTD program is employed to demonstrate a viability of PSE Park.

FDTD is a simulation technique for electrodynamics field modeling. This technique is widely used to deal with electromagnetic phenomena. For example, microwaves like antennas and wireless communications or visible light can be simulated by this technique.

3.1.1. Core Map for FDTD simulation

This core map consists of three cores, *input parameters*, *run simulation* and *print result* except *start* and *stop* core. (Figure 8)

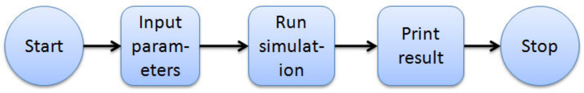


Figure 8. Core map for PSE in FDTD

In the *input parameters* core, users input parameters to run this simulation. The parameters are the simulation time step, the interval step for a data output and the magnetic permeability. Based on these parameters, a FDTD simulation is executed in the *run simulation* core. This core outputs simulation results of the electric and magnetic fields.

The *print result* core requires parameter values: simulation results, size of simulation area, file names and delimiter. The file name information includes the path to the file stored. The delimiter specifies the character to split each simulated values in results. In many cases, commas or tabs are used. The file name and delimiters do not contain the previous core's output. Therefore, they should be specified in the core map as shown in Figure 9.

{"name":"Print result", "core_id":2, "to":-1, "params":{
***"file_name": "result.tsv", "delimiter": " \t"** }}*

Figure 9. Example of parameters setting

If a user wants to create a GNU Plot [20] file, he/she should replace the *print result* core to the "Print GNU Plot" (see Figure 10). This core requires only one parameter of "file_name" and outputs two files of a result file and a GNU Plot file. The GNU Plot file is an executable file. The user can execute this file and obtain the graph of results.

PSE Park provides the flexibility in the job execution PSE; users can easily change the core function on the core map.

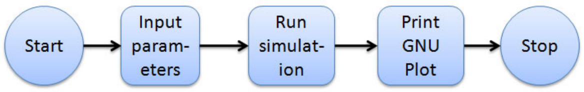


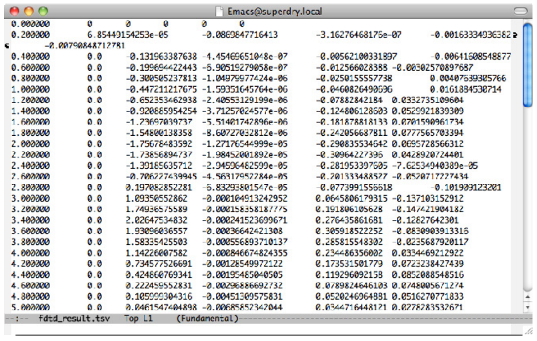
Figure 10. Print GNU Plot core is employed core

3.1.2. Result

We show results of PSEs of Figure 8 and Figure 10. Figure 11 shows the result of core map in Figure 8. In this PSE, we applied the tab as delimiter to visualize the graph easily. User can retrieve this result via Console.

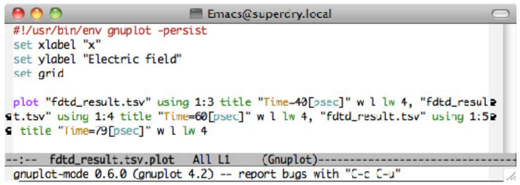
Figure 12 shows the results of the case of the *print GNU Plot* core. Figure 12 (a) is the executable GNU Plot file. When a user executes this file, Figure 12 (b) is obtained. In this simulation, a Gaussian pulse is input at $x = 0$ and it propagates to the right direction. The boundary condition is the Differential-based absorbing boundary condition. We observe the propagation of the pulse.

The results demonstrates that the PSE for the job execution is successfully generated by PSE Park, and that the job execution PSE generated is flexible, so that one can easily change the Core function for the user's purpose. Therefore, this example shown in this section 3.1 presents a viability of PSE Park.



```
0.000000        0       0       0       0       0
0.200000        6.85449154253e-05       -0.0869847716413        -3.16276468176e-07      -0.00163334936382
        -0.00790848712781
0.400000        0.0     -0.131963387638 -4.45469651048e-07      -0.00562100331897       -0.00641608548877
0.600000        0.0     -0.199694422443 -6.90519279058e-07      -0.012566028388 -0.00302570897687
0.800000        0.0     -0.300505237813 -1.04979977424e-06      -0.025015555738         0.00407639305766
1.000000        0.0     -0.447211217675 -1.59351645764e-06      -0.0460826490696        0.0161884530714
1.200000        0.0     -0.652353462938 -2.40553129199e-06      -0.07882842184  0.0332735109604
1.400000        0.0     -0.920885954254 -3.71257024577e-06      -0.124806128603 0.0529921839309
1.600000        0.0     -1.23697039737  -5.51401742896e-06      -0.181878818133 0.0701590961734
1.800000        0.0     -1.54800138358  -8.60727032812e-06      -0.242056687811 0.0777565703394
2.000000        0.0     -1.75678483592  -1.27176544999e-05      -0.290835534642 0.0695728566312
2.200000        0.0     -1.73856894737  -1.98452001892e-05      -0.30964227396  0.0428920724401
2.400000        0.0     -1.39185635712  -2.94596482599e-05      -0.281953397605 -7.62534940389e-05
2.600000        0.0     -0.706227439945 -4.56317952284e-05      -0.201333488527 -0.0520717727434
2.800000        0.0     0.197082852281  -6.83293801547e-05      -0.0773991555618        -0.101909123201
3.000000        0.0     1.09350552862   -0.000104913242952      0.0645806179315 -0.137103152912
3.200000        0.0     1.74936575589   -0.000158358187775      0.191806105628  -0.147421904182
3.400000        0.0     2.02647534832   -0.000241523699671      0.276435861681  -0.12827642301
3.600000        0.0     1.93096036557   -0.00036642421308       0.305918522252  -0.0830903913316
3.800000        0.0     1.58335425503   -0.000556893710137      0.285815548302  -0.0235687920117
4.000000        0.0     1.14226007582   -0.000846674824355      0.234486356002  0.0334469212922
4.200000        0.0     0.734577526691  -0.00128549972122       0.17353150179   0.0723238427439
4.400000        0.0     0.424860769341  -0.00195485040505       0.119296092158  0.0852088548516
4.600000        0.0     0.222459552831  -0.00296886692732       0.0789824646103 0.0748005671274
4.800000        0.0     0.105999304316  -0.00451309575831       0.0520246964881 0.0516270771833
5.000000        0.0     0.0461547404898 -0.00685852347044       0.0344716448121 0.0278283532671
--:--  fdtd_result.tsv   Top L1     (Fundamental)----------------------------------
```

Figure 11. Tab separated result of FDTD

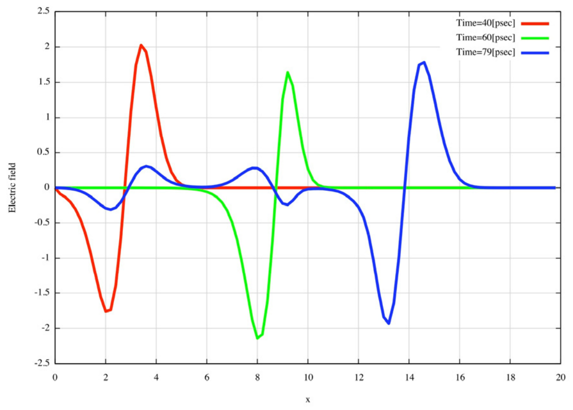


```
#!/usr/bin/env gnuplot -persist
set xlabel "x"
set ylabel "Electric field"
set grid

plot "fdtd_result.tsv" using 1:3 title "Time-40[psec]" w l lw 4, "fdtd_resul
t.tsv" using 1:4 title "Time=60[psec]" w l lw 4, "fdtd_result.tsv" using 1:5
 title "Time=79[psec]" w l lw 4

--:--  fdtd_result.tsv.plot   All L1     (Gnuplot)--------------------------------
gnuplot-mode 0.6.0 (gnuplot 4.2) -- report bugs with "C-c C-u"
```

(a) GNU Plot executable file



(b) Plotted result

Figure 12. Visualized Result of GNU Plot Core

3.2. PDE-based PSE

There are five steps in simulation: “problem discovery,” “program designing/writing,” “program execution,” “aggregate execution result,” and “discussion.” If an engineer or researcher wants to run simulations of his/her target problem, he/she should study not only the problem itself but also how to use computers, how to write a program, or how to execute the program.

This PSE supports the simulation step “program designing/writing.” Users can easily create a program on PSE Park, and then they can simulate and solve the target problem on the resultant PSE.

The target of this example PSE is PDE-based problems. This PSE generates a simulation program from “equations”, “boundary conditions”, “simulation domain”, “initial conditions”, “discretization method" and so on. This PSE is a white-box system, which shows intermediate results, and a document of the target problem is automatically created. [21]

3.2.1. Core Map of PDE-based PSE

We have developed a PSE for PDE-based problems by using seven cores. This PSE is focused on a two-dimensional heat conduction phenomenon. The core map of this PSE is shown in Figure 13.

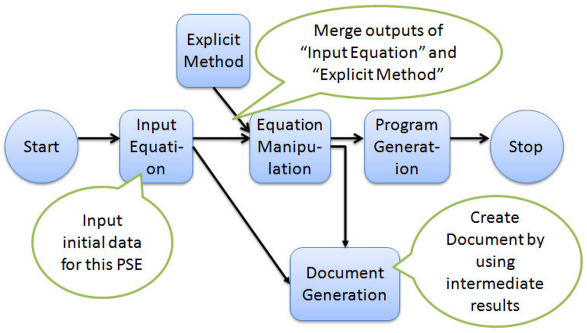

Explicit Method
Merge outputs of "Input Equation" and "Explicit Method"
Start
Input Equati-on
Equation Manipu-lation
Program Generat-ion
Stop
Input initial data for this PSE
Document Generation
Create Document by using intermediate results

Figure 13. Core map for PSE in PDE-based Problems

In this example application case, a PSE for PDE-based problems is created by PSE Park. The generated PSE has input of problem data, which are equations, and initial and boundary conditions, and a discretization method and outputs a computing program. In this specific example, a two-dimensional diffusion problem is solved, and a program is automatically and successfully generated as shown below.

3.2.2. Input Equation Core

The *input equation* core passes the inputs defined by users to the *equation manipulation* core. The parameters, which users set in this PSE, are "computation target in equation," "computation domain of simulation," "equation to be solved," "physical quantities point ("in between point" or "on the point")," "initial condition of simulation," and "boundary condition of simulation."

The initial conditions of this result are as follows.

Equation:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{1}$$

Computation Domains:

$$0.0 \le x \le 1.0 \text{ and } 0.0 \le y \le 1.0 \tag{2}$$

Initial Values:

$$0.3 < x < 0.5 \wedge 0.3 < y < 0.5 : u = 10.0 \tag{3}$$

$$0.7 < x < 0.9 \wedge 0.7 < y < 0.9 : u = 8.0 \tag{4}$$

Boundary Condition:
Periodic boundary in both x and y axes

Figure 14 shows the output of the *input equation* core.

```
{
"computation_target": "u",
"computation_domains": ["0.0 =< x =< 1.0","0.0 =< y =< 1.0",],
"equations": ["ut = uxx + uyy"],
"physical_quantities_points": ["IN_BETWEEN_POINT", "IN_BETWEEN_POINT"],
"initial_conditions": ["0.3 < x < 0.5 and 0.3 < y < 0.5: u = 10.0", "0.7 < x < 0.9 and 0.7 < y < 0.9:
u = 8.0"],
"boundary_conditions": ["PERIODIC_BOUNDARY", "PERIODIC_BOUNDARY"]
}
```

Figure 14. Output of Input Equation Core

In "computation_target" and "equations," *t*, *x*, *y,* and *z* are the independent variables of time and space, respectively. Here *tt*, *xx*, *yy*, and *zz* mean the second-order derivatives of the variables, respectively. In "computation_domains", the range of computation is required. The output "physical_quantities_points" means where the physical quantities are defined. The output "initial_conditions" requires the initial values and their ranges. In "boundary_conditions," boundary conditions are specified, and in this example "PERIODIC_BOUNDARY" is used.

3.2.3. Explicit Method Core

In this PSE, we adapted an explicit method as the difference method. This *explicit method* core is a kind of difference method core that generates discretized equations for each physical quantity on specified points. In the *explicit method* core, the discretized equations are as follows.

$$\frac{\partial u}{\partial i} = \frac{u_{i+1} + u_i}{\Delta i^2} \tag{5}$$

$$\frac{\partial^2 u}{\partial i^2} = \frac{u_{i+2} - 2u_{i+1} + u_i}{\Delta i^2} \tag{6}$$

$$\frac{\partial u}{\partial i} = \frac{u_{i+1} - u_{i-1}}{2\Delta i^2} \tag{7}$$

$$\frac{\partial^2 u}{\partial i^2} = \frac{u_{i+2} - 2u_i + u_{i-2}}{4\Delta i^2} \tag{8}$$

$$\frac{\partial u}{\partial t} = \frac{u^{j+1} + u^j}{\Delta j} \tag{9}$$

The PSE performs discretization for the basic PDEs on the basis of the discretization method specified. If a user wants to use the successive Crank Nicolson method as the difference method, i.e., one of the implicit methods, the user should exchange this *explicit method* core to the *SOR method* core.

3.2.4. Equation Manipulation Core

The *equation manipulation* core receives outputs from both the *input equation* core and the *explicit method* core. In fact, the PIPE server merges the outputs of the two cores and passes them to the *equation manipulation* core. The input of the *equation manipulation* core is in Figure 15.

In the *equation manipulation* core, the discretized equations are manipulated to obtain new forms for the target dependent variables.

Figure 16 shows the expression of an equation as a tree. In the *equation manipulation* core, the obtained equation is converted to a tree. The tree has "=" as its root. Operators such as "+", "-", "*", and "/" are set as inner nodes. Values that are located on both sides of the operator are set as children of the operator node. Sub-trees of the tree in Figure 16 are replaced by the trees that are indicated in the *explicit method* core. In this example, we use Eq. (9) for the sub-tree of Figure 16 (a) and Eq. (6) for the sub-tree of Figure 16 (b) and (c).

```
{
"computation_target": "u",
"computation_domains": ["0.0 =< x =< 1.0","0.0 =< y =< 1.0",],
"equations": ["ut = uxx + uyy"],
"physical_quantities_points": ["IN_BETWEEN_POINT", "IN_BETWEEN_POINT"],
"initial_conditions": ["0.3 < x < 0.5 and 0.3 < y < 0.5: u = 10.0", "0.7 < x < 0.9 and 0.7 < y < 0.9: u
= 8.0"],
 "boundary_conditions": ["PERIODIC_BOUNDARY", "PERIODIC_BOUNDARY"]
"diff_methods": [
        "(u[i+1] - u[i]) / di",
    "(u[i+2] - 2 * u[i+1] + u[i]) / (di * di)",
    "(u[i+1] - u[i-1]) / (2 * di)",
    "(u[i+2] - 2 * u[i] + u[i-2]) / (4 * di * di)",
    "u[n+1] - u[n] / dn"]
}
```

Figure 15. Input of Equation Manipulation Core

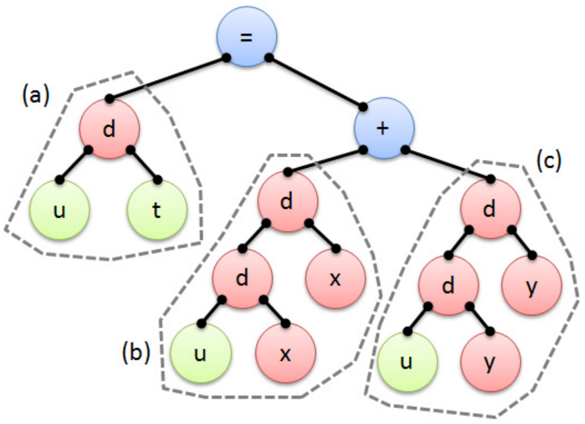


Figure 16. A tree expressing an example equation

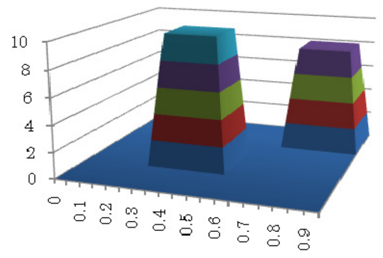
3.2.5. Document Creation Core

This core enables to create a document from output of cores. PSE Park keeps all results generated by cores. In addition, PSEs created by PSE Park in these examples contain all the information required to produce the document for the computer program generated by the PSEs. For example, problem itself, equations employed, discretization method, program structure, variables, constants and so on are stored in the PSEs. The capability of the automatic documentation is remarkable, and important for users to understand the problem and the program. The *document creation* core generates a LaTeX [22] format text file from cores' results.

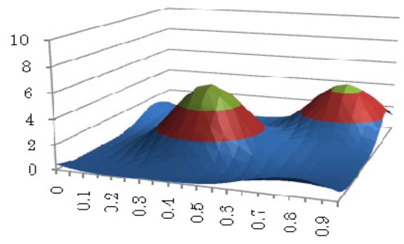
3.2.6. Program Generation Core

This core generates the simulation program from the output of the cores. Currently, this core is specialized for the only PDE-based PSE and output Python [23] program.

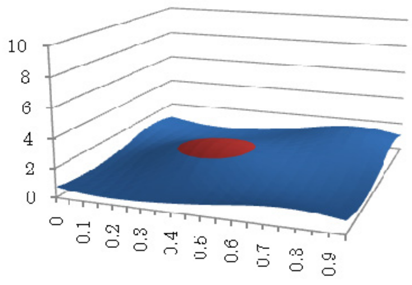
The initial values for the program are retrieved from the outputs of *input equation* core by PIPE server. The manipulated equation generated by *equation manipulation* core is translated into Python. The output of the simulation is printed into stdout.



(a) step = 0



(b) step = 50



(c) step = 1200

Figure 17. Simulation results for a diffusion problem. The computation program is automatically generated by the PSE, which is created on PSE Park.

3.2.7. Results

We show the simulation execution results in Figure 17. The simulation program was generated by the PSE for PDE-based problems, and that PSE was generated by PSE Park. This simulation was executed in one local machine.

We obtained two peaks of heat quantity in this simulation of a two-dimensional diffusion problem. Figure 18 shows a part of results of *document creation* core. There are two outputs in Figure 18 (a). The former is the converted result of "equation" of *input equation* core. This is the original equation that user input. The latter is the *equation manipulation* core one. This is the result of equation manipulation with explicit method (Figure 18 (a)). When it is compiled, the equation is visualized as shown in Figure 18 (b).

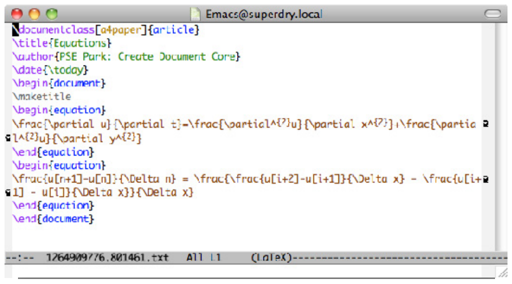
3.3. Evaluation

We constructed PSEs by using PSE Park. If PSE Park is not available, users should construct the PSE by themselves. In this case, users have to prepare functions for the PSE and study how to handle processes and data. The functions are provided by cores, and users can choose the cores that they want to use. The PIPE server in PSE Park handles processes and data access instead of users. These are the advantages of using PSE Park.

The first example of the job execution shows the flexibility of the PSE generated. The second example PSE for the PDEs-based problem solving provides a good example for the first generation of PSEs [3-10]. The examples employed in this section demonstrates that PSE Park provides a new PSE world, that is PSE for PSE or meta PSE, which we discussed in the Introduction and Section 2 in this paper.

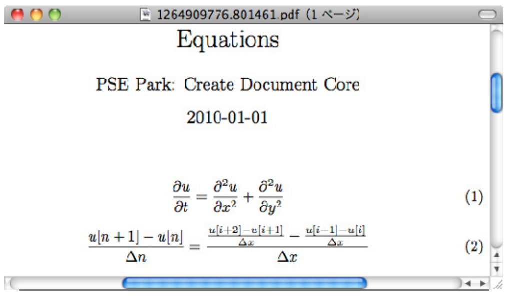
4. Conclusion

In this paper, we presented our development of PSE Park. PSE Park is a PSE construction framework that is a meta PSE. By using PSE Park, a user can construct PSEs easily and perform simulations for his/her target problems. PSE Park supports PSE developers and engineers, as well as entry-level users of computer simulations. PSE Park may satisfy the majority of user requirements for problem solving: program generation, easy modification of programs, PSE development, automatic documentation supply, and simulation execution support in Cloud. The concept of PSE Park, i.e., a framework for PSE development, presents an important new direction for problem solving environments.



```
\documentclass[a4paper]{article}
\title{Equations}
\author{PSE Park: Create Document Core}
\date{\today}
\begin{document}
\maketitle
\begin{equation}
\frac{\partial u}{\partial t}=\frac{\partial^{2}u}{\partial x^{2}}+\frac{\partia
l^{2}u}{\partial y^{2}}
\end{equation}
\begin{equation}
\frac{u[n+1]-u[n]}{\Delta n} = \frac{\frac{u[i+2]-u[i+1]}{\Delta x} - \frac{u[i+
1] - u[i]}{\Delta x}}{\Delta x}
\end{equation}
\end{document}
--:--  1264909776.801461.txt   All L1     (LaTeX)-----------------------------------
```

(a) Raw LaTeX



Equations

PSE Park: Create Document Core

2010-01-01

$$\frac{\partial u}{\partial t}=\frac{\partial^{2}u}{\partial x^{2}}+\frac{\partial^{2}u}{\partial y^{2}} \quad (1)$$

$$\frac{u[n+1]-u[n]}{\Delta n} = \frac{\frac{u[i+2]-u[i+1]}{\Delta x} - \frac{u[i-1]-u[i]}{\Delta x}}{\Delta x} \quad (2)$$

(b) Compiled LaTeX

Figure 18. A part of a document created by the *document generation* core

5. Acknowledgments

This work was supported partly by JSPS and MEXT in Japan. The authors would like to extend their appreciation to Prof. Y. Tago and colleagues of the PSE Research Group in Japan.

6. References

[1] E. Gallopoulo, E. Houstis and J. Rice, “Computer as thinker/doer: problem-solving environments for computational science”, Computational Science & Engineering, IEEE Volume 1, Issue 2, Summer 1994, pp. 11 - 23

[2] E. Houstis, E. Gallopoulos, R. Bramley and J. Rice, “Problem-Solving Environments for Computational Science”, Computing in Science and Engineering, vol. 4, no. 3, July-Sept. 1997, pp. 18-21

[3] Y. Umetani, M. Tsuji, K. Iwasawa, H. Hirayama , "DEQSOL- A Numerical Simulation Language for Vector/Parallel Processors", Proc. of IFIP WG 2.5 Working Conference on Problem Solving Environments for Scientific Computing, 1985, pp. 147-164

[4] C. Kon'no, M. Saji, N. Sagawa and Y. Umetani, “Advanced implicit solution function of DEQSOL and its evaluation”, Proceedings of 1986 ACM Fall joint computer conference, pp. 1026 – 1033

[5] J. Rice, R. Boisvert, Solving elliptic problems using ELLPACK, Springer-Verlag New York, Inc. New York, NY, USA, 1985

[6] C. Boonmee and S. Kawata, “Computer-Assisted Simulation Environment for Partial-Differential-Equation Problem: 1. Data Structure and Steering of Problem Solving Process”, Transactions of JSCES, Paper No. 19980001, 1998

[7] C. Boonmee and S. Kawata, “Computer-Assisted Simulation Environment for Partial-Differential-Equation Problem: 2. Visualization and Steering of Problem Solving Process”, Transactions of JSCES, Paper No. 19980002, 1998

[8] T. Teramoto, T. Nakamura, S. Kawata, S. Machide, K. Hayasaka, H. Nonaka, E. Sasaki and Y. Sanada, “A Distributed Problem Solving Environment (PSE) for Partial Differential Equation Based Problems”, Transactions of JSCES, Paper No.20010018, 2001

[9] S. Kawata, H. Usami, Y. Hayase Y. Miyahara, M. Yamada, M. Fujisaki, Y. Numata, S. Nakamura, N. Ohi, M. Matsumoto, T. Teramoto, M. Inaba, R. Kitamuki, H. Fuju, Y. Senda, Y. Tago and Y. Umetani, “A problem-solving environment (PSE) for distributed computing” , Int. J. High Performance Computing and Networking, Vol. 1, No. 4, 2004, pp. 223-230

[10] S. Kawata, M. Inaba, H. Fuju, H. Sugiura, Y. Saitoh and T. Kikuchi, “Computer-Assisted Liaison among Modules in a Distributed Problem Solving Environment (PSE) for Partial Differential Equation Based Problems”, Transaction of JSCES, Paper No.20050029, 2005

[11] I. Foster and C. Kesselman, The Grid 2: Blueprint for a New Computing Infrastructure, The Elsevier Series in Grid Computing, 2004

[12] G. Allen, W. Benger, T. Goodale, H. C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, J. Shalf, “The Cactus Code: A Problem Solving Environment for the Grid”, High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium, 2000, pp. 253-260

[13] C. E. Goodyer1 and M. Berzins, Solving Computationally Intensive Engineering Problems on the Grid using Problem Solving Environments Software Environments and Tools. Springer, 2006

[14] S. Kawata, H. Usami, Y. Hayase, Y. Miyahara, M. Yamada, M. Fujisaki, Y. Numata, S. Nakamura, N. Ohi, M. Matsumoto , T. Teramoto, M. Inaba, R. Kitamuki, H. Fuju, Y. Senda, Y. Tago and Y. Umetani, “A problem-solving environment (PSE)”, Int. J. High Performance Computing and Networking, Vol. 1, No. 4, 2004, pp223-230

[15] H. Kanazawa, M. Yamada, Y. Miyahara, Y. Hayase, S. Kawata, and H. Usami, “Problem Solving Environment based on Grid Services: NAREGI-PSE”, Proceedings of the First International Conference on e-Science and Grid Computing, 2005

[16] Y. Kadooka, H. Kobashi, J. W. Choi, Y. H. Lee and Y. Tago, “PIV Virtual Laboratory using Grid Technology”, Proceedings of PSFVIP, 2003, F4027
[17] http://taverna.sourceforge.net/
[18] http://www.json.org/
[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, “Dynamo: Amazon’s Highly Available Key-value Store”, SOSP’07, pp. 205 – 220
[20] http://www.gnuplot.info/
[21] M. Inaba, H. Fuju, R. Kitamuki, S. Kawata, T. Kikuchi: “Computer-Assisted Documentation in a Problem Solving Environment (PSE) for Partial Differential Equation Based Problems”, , Transactions of JSCES, Paper No. 20040025, 2004
[22] http://www.latex-project.org/
[23] http://www.python.org/.