(12) INNOVATION PATENT (11) Application No. AU 2019100212 A4
(19) AUSTRALIAN PATENT OFFICE

(54) Title
SYSTEM AND METHOD FOR DEVELOPING MODULARIZED APPLICATION

(51) International Patent Classification(s)
G06F 8/36 (2018.01) ***G06F 8/20*** (2018.01)

(21) Application No: **2019100212** (22) Date of Filing: **2019.02.27**

(30) Priority Data

(31) Number	(32) Date	(33) Country
2018900658	**2018.02.28**	**AU**

(45) Publication Date: **2019.04.04**
(45) Publication Journal Date: **2019.04.04**
(45) Granted Journal Date: **2019.04.04**

(71) Applicant(s)
Michael Changcheng Chen

(72) Inventor(s)
Chen, Michael Changcheng

(74) Agent / Attorney
Michael Changcheng Chen, 21 Larool Cres, Castle Hill, NSW, 2154, AU

ABSTRACT

A system and method of developing modularized application is disclosed. In one preferred embodiment, a modularized application 12' is implemented as a directory-hierarchy comprises a plurality of directories, with each directory 20 containing module-implementation definition 22 for implementing a module 14'. Directory relationship 26, such as parent-child, is used in defining modules association for inter-module communication. In one preferred embodiment, when executing a modularized application, modules' module-implementation definitions 22A 22B are used by application runtime, for running hardware processes 24A 24B performing the modules' intended tasks, and for conducting inter-module communication 28 between hardware processes 24A 24B of associated modules 14A" 14B".

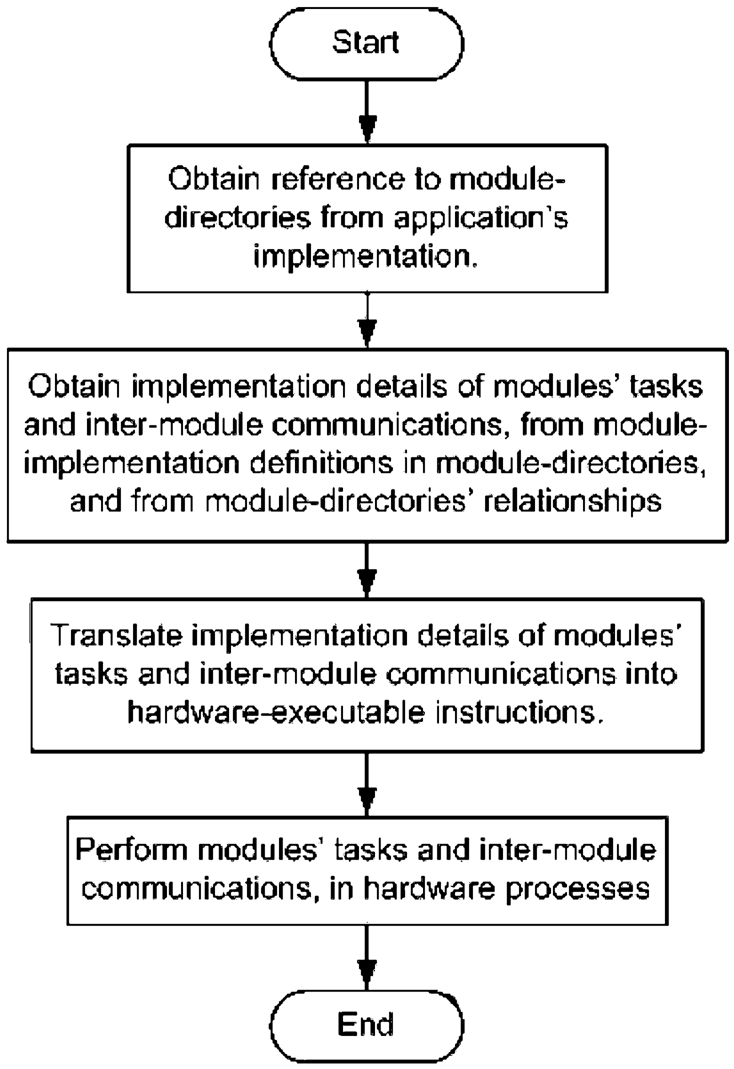

Start
Obtain reference to module-directories from application's implementation.
Obtain implementation details of modules' tasks and inter-module communications, from module-implementation definitions in module-directories, and from module-directories' relationships
Translate implementation details of modules' tasks and inter-module communications into hardware-executable instructions.
Perform modules' tasks and inter-module communications, in hardware processes
End

FIG. 3D

2019100212 27 Feb 2019

AUSTRALIA Patents Act 1990

INNOVATION SPECIFICATION

SYSTEM AND METHOD FOR DEVELOPING MODULARIZED APPLICATION

The invention is described in the following statement

2019100212 27 Feb 2019

SYSTEM AND METHOD FOR DEVELOPING MODULARIZED APPLICATION

FIELD OF THE INVENTION

[0001] The present invention relates to software application development. In particular, the present invention relates to system and method for implementing and executing modularized applications.

BACKGROUND OF THE INVENTION

[0002] Using software component in application development promotes higher-level software reuse, it brings the benefits to the development such as faster development cycles, lower development costs, and higher software quality.

[0003] Application module, or simply "module", is high-level software component, functionally for "performing a task". This contrasts with low-level software component, such as classes in a Java library, which is for "performing a function (of a task)".

[0004] One character of application module is that, being high-level software component, it exposes its attributes or parameters through the application-building environment (i.e. an IDE), where a developer can change the application’s setting or behaviour by configuring these parameters, rather than by programming the source code.

[0005] Modularized application, or simply “application”, is a type of component-based application that uses application modules as its application-building blocks. Modularized applications are developed by assembling modules and configuring the modules' parameters. Such development is easier and has less software defects, as it does not involve low-level programming.

[0006] Application runtime, or simply “runtime”, is a program that provides resources and supports modularized application's execution. Through an API referred as

2019100212 27 Feb 2019

"runtime service", application's implementation is translated into computer executable instructions, by application runtime, which then calls computer's operating system (referred as the "hardware") to execute the instructions.

[0007] Fig 1 is a functional diagram of modularized application architecture. As shown in Fig 1, modularized application 12 can have one or more modules 14, with each module 14 performs a task in the application. Application 12 can also have inter-module communications 18 between associated modules 14. Runtime 10 executes modularized application 12 through providing runtime service 16, which translates application's implementation (not shown) into hardware-executable instructions and executes the instruction 8 on computer's hardware 6.

[0008] Development of modularised application has three phases: design, implementation, and execution. Design phase is where the required tasks are identified from an application's functional requirements, and modules for these tasks are selected. Inter-module communications, such as data and commands exchange, are also identified and specified.

[0009] Implementation phase is where the application's selected modules are assembled into a "runtime executable" program (referred as the application's "implementation"). In this phase, each module is configured for its intended task in the application, inter-module communications are also established and configured.

[0010] Application's execution phase is where the application's implementation is executed, as computer hardware processes that perform the application's designed functions, by a runtime.

Prior Arts

[0011] Microsoft SSIS (SQL Server Integration Services) is specialised program (an "application runtime") that runs a kind of modularized application, called "SSIS package", for data warehousing ETL tasks. SQL Server Data Tool (SSDT) is an IDE (Integrated Development Environment) for developing SSIS packages.

2019100212 27 Feb 2019

[0012] Fig 2A shows in SSDT, modules 14 are used in assembling a data-processing flow of a SSIS package, each module has an assigned task. Data-flow link 18 ("inter-module communication") associates related modules according to the required data-processing sequence. Fig 2A also indicates, in SSDT, pre-built modules can be added into or be removed from the data-processing flow, allowing easy customisation to a package.

[0013] Oracle JCAPS is type of program called "middleware", which is primarily used in the field of System Integration. JCAPS interface is another example of modularized application. JCAPS Enterprise Designer is the IDE for building JAPS interfaces. Developed JCAPS interfaces (i.e. the "modularized applications") can be executed in a runtime environment called "logical host" (i.e. the "application runtime").

[0014] Fig 2B shows in Enterprise Designer, multiple modules 14 are linked ("associated") in building an interface.

SUMMARY OF THE INVENTION

Technical Problem

[0015] One drawback in these prior arts is that the format of the pre-built modules is proprietary, meaning modules from different vendors cannot be used in the same application's development. For example, JCAPS modules cannot be used in a SSIS package, and vice versa.

[0016] Another drawback of these prior arts is that a developed application can only be executed in vendor-specific runtimes. For example, a SSIS package must be run in SSIS, it cannot be run in JCAPS' Logical host, or in any other third-party developed runtime.

[0017] Yet another drawback of these prior arts is that a developed application cannot be customised without using vendor-specific development tools. Tools such as SSDT and Enterprise Designer are specialised tools, they can be expensive and

2019100212 27 Feb 2019

not available in every computer. Furthermore, these tools can be complex to install and to learn.

Solution to Problem

[0018] In accordance with the present invention, a system and method for developing modularized application, comprise an abstraction layer that separates a module's implementation from the runtime, so compatible module and runtime can be independently implemented.

[0019] The system and method of the present invention use file-system directory in defining module in application's implementation; and use directory relationship in defining module-associations for inter-module communications.

[0020] Because directory is a file-system feature universally available on every computer, the present invention allows building and customising applications through simple file-system operations that are familiar to most computer users. Not only it makes developing and managing applications easier and more convenient, it gives users more choices in their application development, by not restricting them to only use proprietary modules, runtimes, and application development tools.

Objects and Advantages

[0021] Accordingly, several objects and advantages of the present invention are:

[0022] To have an open format for defining modularized application's implementation, so compatible modules can interoperate and can be mixed in developing applications in such format, and such developed applications can be run in compatible runtimes;

[0023] To enable end-users to conveniently build and customise modularized applications, using tools and features that are available in every computer, thus to avoid the cost and restrictions from using proprietary tools for application development.

2019100212 27 Feb 2019

[0024] Further objects and advantages will become apparent from a consideration of the ensuring description and drawings.

BRIEF DESCRIPTION OF DRAWINGS

[0025] The invention may best be understood by reference to the following description taken in conjunction with the accompanying drawings in which:

[0026] FIG. 1 is a functional diagram of modularized application architecture.

[0027] FIG. 2A is a screenshot of a prior art, Microsoft SQL Server Data Tool (SSDT), showing “data flow tasks” of a SSIS package.

[0028] FIG. 2B is a screenshot of a prior art, Oracle JCAPS Enterprise Designer, showing a “connectivity map” of a JCAPS schema.

[0029] FIG. 3A is a schematic illustrates the three phases of modularized application development, of one preferred embodiment.

[0030] FIG. 3B-3C are schematics of modularized applications’ design and their corresponding implementation, of one preferred embodiment.

[0031] FIG. 3D is a flowchart of the runtime process for running an application.

[0032] FIG. 4 is a schematic of an example application in its development phases.

DETAILED DESCRIPTION OF THE INVENTION

[0033] One object of the invention is to have an "open" format for defining application’s implementation, so it can be run by any compatible runtimes.

[0034] Fig 3A shows a modularized application under development, in its design 12, implementation 12’, and execution 12” phases, in accordance to an embodiment of the present invention. It shows in Fig 3A, file-system directories are used for constructing and representing modules in an application’s implementation 12'.

2019100212 27 Feb 2019

[0035] As Fig 3A shows, the use of file-system directory in developing modularized application is twofold: 1) directory 20A (same as 20B) is used to contain a module's implementation details, which describe to a runtime how to perform the module's task, and 2) directory relationships 26 are used to associate modules, which are presented by the related directories in the application, for inter-module communications.

[0036] Effectively in a such implementation, an application is a directory-hierarchy, with each directory in the hierarchy represents a module in the application. Directory used for representing a module is referred herein as "module-directory".

[0037] Since directory is a common file-system feature on all computers, using module-directories in defining application's implementation allows such implementation (format-wise) universally portable and accessible. The application in such "format" (that is, a hierarchy of directories) can then be easily copied, moved, like operating on a normal group of directories.

Module-Implementation Definition

[0038] Module-implementation definition refers to a specific content in module-directory. The purpose of such content is to provide "sufficient information about a module" to a runtime, so the later can retrieve the module's implementation details, and convert such information into hardware executable instructions that perform the module's intended task.

[0039] From software design perspective, module-implementation definition is an abstraction layer (i.e. an "interface") between a runtime's and a module's underlying implementation. That is, it is a "contract" between a runtime and a module, about how the runtime can access the module's implementation, and how such implementation can be utilised to perform the module's intended task. It enables the runtime and the module to work together, but at the same time, it "hides" each party's implementation details from the other.

2019100212 27 Feb 2019

[0040] Module-implementation definition (the “contract”) can be complex and explicitly expressed (e.g. as a web service described in WSDL), or it can also be simply implied. The actual module implementation can be in a form directly accessible, but it can also be in a form that needs to be indirectly derived.

[0041] For example, an implied contract can be 1) for the runtime, is “to run every executable file placed in a module-directory”, and 2) for the module’s implementation, is “to place executables for performing a task in the module-directory”. If such an application executes, both parties will exercise the contract, resulting the runtime running the executables in the directory and performing the module’s task.

[0042] In this case, the module’s implementation (the “executable”) is included in the files that reside in the module-directory and it’s directly accessible. There can also be cases where a reference is used to refer to such implementation at a remote location (e.g. by using an URL in the module-implementation definition).

[0043] Nonetheless, it can be said the content of a module-directory conclusively describes how to perform a module's task to a runtime. In other words, a module-directory includes “everything about the module a runtime needs to know”. Such conclusiveness means a modularized application can be built or customised by ways of normal directory operations (e.g. copy-and-paste, drag-and-drop, etc.), without the need for any specialized tools.

Modules Association and Inter-module Communication

[0044] Defining inter-module communications is also part of application’s implementation. In the present invention, this is done in two steps:

[0045] First, file-system directory relationship is used to associate modules having inter-module communication. As shown in Fig 3A, the “parent-child” relationship 26 of directories 20A and 20B indicates module 14A’ and 14B’ in the application implementation are associated for the designed inter-module communication 18.

[0046] Second, details of inter-module communication are defined or specified in the module-implementation definition. Such details can include, but not limited to, the module's role in communication (e.g. client or server), the direction (e.g. one-way or two-ways), types of data or commands that are exchanged, etc, and they can be expressed implicitly or explicitly. Such information is used when runtime conducts the communication during the application's execution.

[0047] As illustrated in Fig 3B and 3C, inter-module communications 18 from an application's design 12 can be flexibly setup and easily changed in the application's implementation 12', by ways of arranging module-directories relationships 26 and configuring module-implementation definition 22 that resides in module-directories 20, corresponding to the application's design.

[0048] Note there are many ways of defining directory relationship, other than nesting directories as parent-child as illustrated in Fig 3A. For example, operating systems like Unix and Linux use "symbolic link" as reference to another directory in the file-system. Windows has similar feature, called a "shortcut", for referencing a directory. Another technique of relating two directories is through lookup tables, where related directories' paths can be linked as table entries. Such lookup table can be placed inside module-implementation instructions or can be placed in somewhere the runtime can read.

Application Runtime Process

[0049] An application enters its execution phase when its implementation is executed by a runtime. Fig 3A shows an application 12" in its execution phase, where module-implementation definition is translated by runtime into hardware executable instructions 22A' 22B', which are in turn executed in computer hardware as processes (or threads) 24A 24B that perform the modules' intended tasks.

[0050] Fig 3D is a flowchart of runtime process executing an application. It's highlighted in the flowchart how module-directories (as in the application's implementation) are used to derive detailed module-implementation information,

2019100212 27 Feb 2019

which are subsequently used for running module's intended tasks and communications, in computer hardware processes.

[0051] As being illustrated, the present invention has many advantages over prior arts. By using file-system directories in application implementation, it allows application to be developed and customized through normal file-system directory operations, which are familiar to normal users and available on every computer. By using module-implementation definition as an abstraction layer between modules and runtime, it allows modules and runtime to be implemented independently, possibly by different vendors, using different technologies, thus giving end-user more choices.

[0052] These and other advantages of the present invention become more apparent though the following example, in which we demonstrate how an application is developed, in each of the development phases. We also give details about how a runtime can be implemented to run the developed application, so a person skilled in the art can follow the example and carry out the invention.

[0053] Without any specific preference, .NET technology and related terminology are used in illustrating the example application development.

Example Application Development - The Design

[0054] Assuming we are developing a hypothetical data-processing application with the following requirement, as in Table 1.

TABLE 1: Example application requirement.

From a give path, retrieve all 'admission records' from a set of input files, and write the retrieved records to an output file. The records in the input files are encoded in HL7.

2019100212 27 Feb 2019

[0055] To provide the background to people who's not familiar with the subject: HL7 is a messaging standard used in the Healthcare system integration industry. HL7 messages are text-based records used for transferring patient information between integrated health information systems. Admissions HL7 messages indicate patient's hospital admission status, and can be identified by checking a HL7 message element at address location 'EVN-1', for matching value 'A01'.

[0056] As part of the design, it is identified that from the requirement, the application needs to perform three different tasks, in the following order:

- The "Reading" task, which reads input files and parse the file content for retrieving HL7 records, output retrieved records;
- The "Filtering" task, which filters the retrieved HL7 records from the input and selects only messages with data element at "EVN-1" having value "A01", output selected records; and
- The "Writing" task, which writes selected records from the input to a file of a given path.

[0057] Fig 4 shows, as in the application's design 12, three modules 14R,14F and14W are selected for performing these tasks. It shows in the design 12, between modules 14R-14F and 14F-14W, there are data-exchange 18A, 18B for exchanging input and output data, as part of the data-processing requirement.

Example Application Development - The Implementation

[0058] Fig 4 shows the application's implementation 12' that is corresponding to its design 12. It shows three module-directories 20R, 20F and 20W are used to construct the identified modules 14R, 14F and 14W from the design 12.

[0059] In this application development example, the "module-implementation definition" is a single XML file referred as the "step-config" file, resides in each module-directory. Each step-config file has the same XML structure that contains the configurations of the module. These files are listed as the following:

2019100212 27 Feb 2019

TABLE 2: step-config file content for the “Reading” module.

```
<StepConfig>
  <Handler>HL7FileReader</Handler>
  <HandlerAssembly>handlers.dll</HandlerAssembly>
  <Parameters>
    <Parameter>
      <Name>source-file-name-pattern</Name>
      <Value>*.hl7</Value>
    </Parameter>
    <Parameter>
      <Name>source-path</Name>
      <Value>D:\DATA\Reader-in</Value>
    </Parameter>
  </Parameters>
</StepConfig>
```

TABLE 3: step-config file content for the “Filtering” module.

```
<StepConfig>
  <Handler>HL7Filter</Handler>
  <HandlerAssembly>handlers.dll</HandlerAssembly>
  <Parameters>
    <Parameter>
      <Name>filtering-rule</Name>
      <Value>EVN-1=A01</Value>
    </Parameter>
  </Parameters>
</StepConfig>
```

2019100212 27 Feb 2019

TABLE 4: step-config file content for the “Writing” module.

```
<StepConfig>
  <Handler>HL7FileWriter</Handler>
  <HandlerAssembly>handlers.dll</HandlerAssembly>
  <Parameters>
    <Parameter>
      <Name>destination-path</Name>
      <Value>D:\DATA\Out</Value>
    </Parameter>
  </Parameters>
</StepConfig>
```

[0060] Being the “module-implementation definition”, the step-config files are to provide modules’ details to the runtime, so the runtime can perform the modules’ intended tasks. Because this is an illustrative mock example, also for the sake of simplicity, we assume the meaning of each configuration item is understood and agreed by the application and runtime developers. The following configuration item interpretation is based on such assumption.

[0061] Referring to Table 2, in our example, it specifies to the runtime how to perform the “reading” task. More specifically, the “Handler” configuration is set as “HL7FileReader” referring to a .NET class by name. The next configuration item “HandlerAssembly” refers to a .NET DLL file (called an “assembly” in .NET terms). Together these two configuration values enable the runtime to locate and load the named class in the specified DLL, for performing the HL7 file-reading task.

[0062] Table 2 also shows that, in the step-config file, the “parameter” XML element allows extra parameters to be set and passed to the runtime, for performing the module’s task. The “source-file-name-pattern” parameter specifies the reader handler only to process files with “*.hl7” naming pattern; and the “source-path”

2019100212 27 Feb 2019

parameter specifies the reader handler to only scan for data files in the "D:\Data\Reader-in" file location.

[0063] Similarly, configurations in Table 3 and Table 4 are pseudo instructions following the same pattern, that is, these step-config files contain reference to the task-implementing .NET classes, and parameters required for running each of the task. These are the details required for the runtime to construct the modules' task-handlers and perform the "filtering" and "writing" tasks.

[0064] It shall be noted, as in the example, the module's task handler can be changed through modifying configuration elements in the step-config file. This is an advantage of the present invention, because for a given task, it's possible to make the module-implementation refer to another .NET class, from another DLL, perhaps from a different vendor, or even use a different technology other than .NET. Such flexibility give the user choices when implementing and executing the modules.

Implementing Data-exchange as Inter-Module Communication

[0065] As shown in Fig 4, module-directories 20R, 20F, 20W are nested in each other, forming a directory-hierarchy. The parent-child directory-hierarchy in the implementation 12' corresponds to the order of the data-processing modules 14R, 14F, 14W in design 12.

[0066] It's intended that the parent-child module-directory relationship 26A, as in Fig 4, is used to indicate the two related modules 14R', 14F' are involved in exchanging input and output data. Furthermore, it's implied the data-exchange direction is from the "parent" module 14R' to the "child" module 14F'.

[0067] In this example, it's assumed the handlers are programmed to specifically handle HL7 data, this implies all records processed by the application are HL7 records, therefore in this example, it is not necessary to specify the data-types in the data-exchange details in the step-config files, as it's implied.

Application Development Example - The Execution

2019100212 27 Feb 2019

[0068] In the application implementation described above, with information contained in step-config files, and module associations that can be derived from the directory hierarchy, it includes the required information for a runtime to perform these modules' tasks and inter-module communications on a targeted computer hardware.

[0069] Fig 4 shows the application 12" in its execution phase. It shows running modules 14R", 14F", 14W" as hardware processes (or threads, or similar) 24R, 24F, 24W which are created by the runtime (not shown), these processes 24R, 24F, 24W are executing hardware instructions 22R', 22F', 22W' that are translated from setting values retrieved from the step-config files 22R, 22F,22W.

[0070] Specifically, for example, the runtime retrieves the task-implementing .NET class "HL7FileReder" in DLL "handlers.dll", as specified in the step-config file in the "Reading" module implementation 14R', for creating the "reader handler" hardware process 24R. Similarly, the "filter handler" process 24F and the "writer handler" process 24W can also be created. Dynamically locating an executable implementation (e.g. a compiled .NET class) by name from an assembly (EXE or DLL file) is a well-known technique called "reflection". Such technique is familiar to those skilled in the art, i.e. software developers.

[0071] Fig 4 shows inter-module communication 18A in the design 12 is performed in the application's hardware execution 12" as inter-module communication execution 28A: the output from the "reader" module (execution) 14R" is passed to the "filter" module (execution) 14F" as its input. This is done by the runtime associating the two hardware processes 24R and 24F, based on the module-directory relationship 26A in application implementation 12', and conducting the required data-processing (i.e. data-exchange) between the processes, based on the communication's implementation details from the module-implementation definitions 22R and 22F.

[0072] Similarly, inter-module communication 28B is also performed between the "filter" module (execution) 14F" and the "writer" module 14W".

[0073] As illustrated in the above example application development, using the system and method of the present invention, the application's design 12 is developed into an implementation 12', and through the application's execution 12", the designed modules' tasks and inter-module communications are performed corresponding to application's requirement.

Conclusion, Ramification, and Scope

[0074] Accordingly, the advantages of the present invention become apparent. As a novel application development system and method, it allows implementing a modularized application physically as a directory hierarchy with module-directories.

[0075] As directory is a common feature to all computer file-systems, and directory operations are familiar to most computer users, the system and method of the present invention have many advantages over prior arts, in regard to developing and managing applications, including being able to easily and conveniently customize an application without the need for any vendor-specific tools.

[0076] Through using module-implementation definition, an abstract layer between a module and a runtime, both the module and the runtime can be independently implemented, it potentially allows an application to be developed using modules from different implementations, from different vendors and using different technologies, at the same time, and allow applications can be run by compatible runtimes.

[0077] Although the description above contains many specificities, these should not be construed as limiting the scope of the invention but as merely providing illustrations of some of the presently preferred embodiments of this invention. For example, there can be embodiments where the modules of an application are geographically distributed, and the directories can be related though many kinds of technologies such as the Web Services.

[0078] Thus the scope of the invention should be determined by the appended claims and their legal equivalents, rather than by the examples given.

2019100212 27 Feb 2019

Reference Signs List

[0079] 6 - operating system (hardware)

[0080] 8 - hardware instruction (execution)

[0081] 10 - runtime

[0082] 12 - modularized application (design of)

[0083] 12’ - modularized application (implementation of)

[0084] 12” - modularized application (execution of)

[0085] 14, 14A,14B, 14R, 14F, 14W - module (design of)

[0086] 14’, 14A’,14B’, 14R’, 14F’, 14W’ - module (implementation of)

[0087] 14”, 14A”,14B”, 14R”, 14F”, 14W” - module (execution of)

[0088] 16 - runtime service

[0089] 18, 18A, 18B - inter-module communication (implementation of)

[0090] 20, 20A, 20B, 20R, 20F, 20W - module-directory

[0091] 22, 22A, 22B, 22R, 22F, 22W - module-implementation definition

[0092] 22’, 22A’, 22B’, 22R’, 22F’, 22W’ - hardware instruction

[0093] 24, 24A, 24B, 24R, 24F, 24W - hardware process

[0094] 26, 26A, 26B - directory relationship

[0095] 28, 28A, 28B - inter-module communication (execution of)

Editorial Note
2019100212

There is only One page of Claim

2019100212 27 Feb 2019

CLAIMS

1. A modularized application implementation, comprising: 1) a first directory, wherein said first directory contains a first module-implementation definition, whereby said first module-implementation definition defines implementation of a first application module; 2) a second directory, wherein said second directory contains a second module-implementation definition, whereby said second module-implementation definition defines implementation of a second application module; and 3) a directory relationship between said first directory and said second directory, whereby said directory relationship associates said first application module with said second application module for inter-module communication between said first application module and said second application module.

2. The modularized application implementation of Claim 1 wherein said inter-module communication is data-exchange.

3. The modularized application implementation of Claim 1 wherein said first module-implementation definition includes a computer-executable.

4. The modularized application implementation of Claim 1 wherein said module-implementation definition includes a reference to a computer-executable.

5. A method of implementing an application module for performing a task, and for performing an inter-module communication to another application module, comprising steps of: 1) providing a directory, wherein said directory contains a module-implementation definition; 2) including a first execution instruction in said module-implementation definition, whereby said application module performs said task when said first execution instruction is executed; 3) including a second execution instruction in said module-implementation definition, whereby said application module performs said inter-module communication to said other module when said second execution instruction is executed.

2019100212 27 Feb 2019

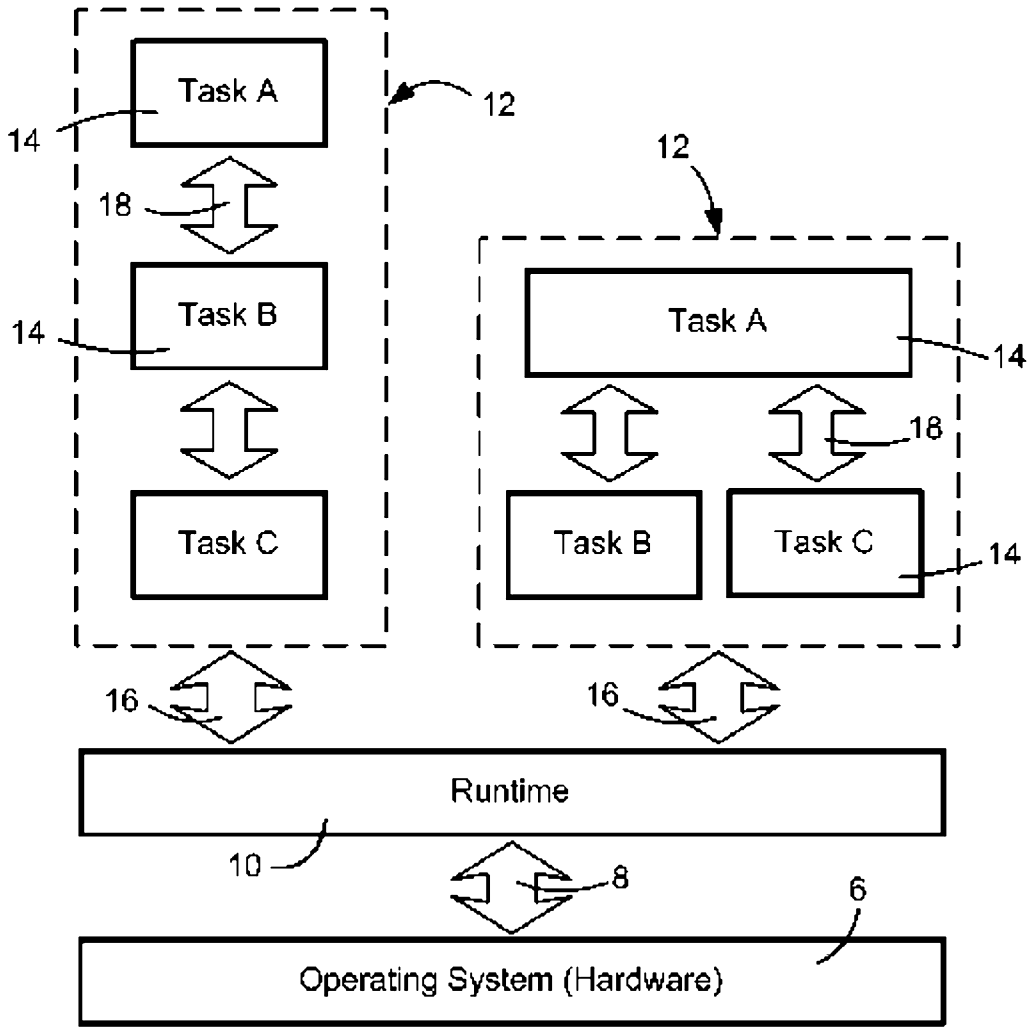


FIG. 1

2019100212 27 Feb 2019

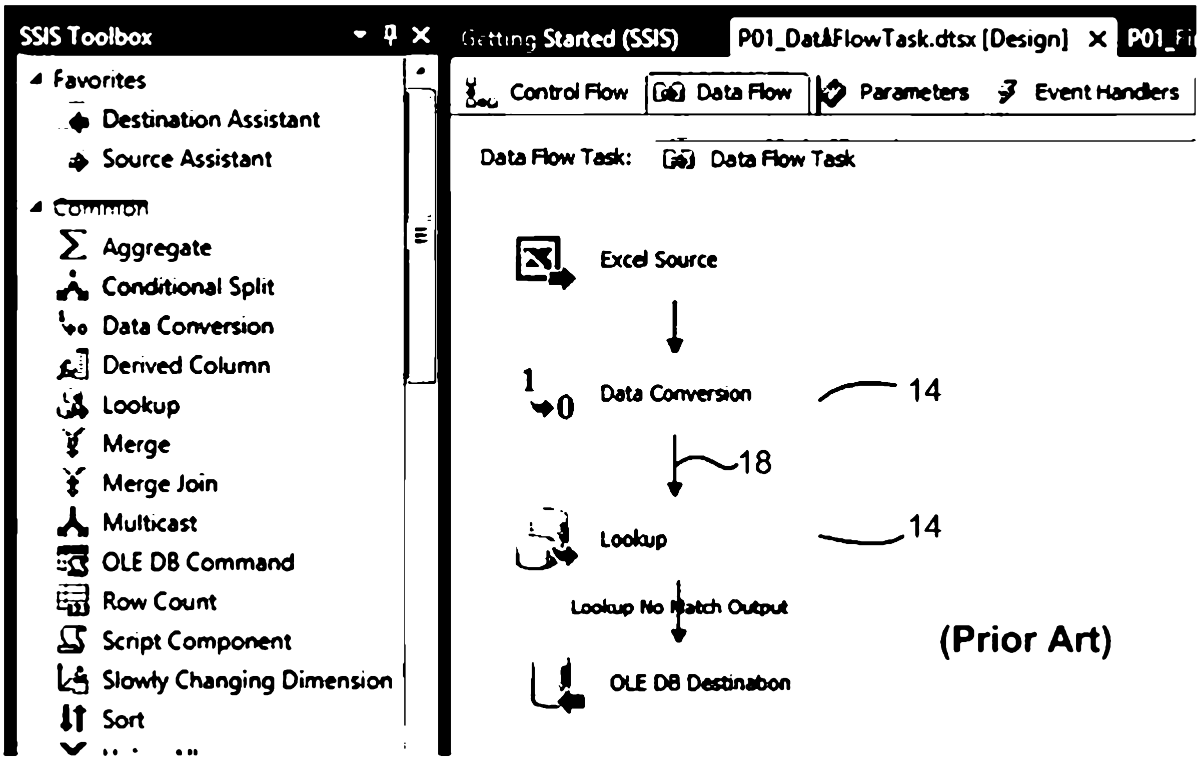


FIG. 2A

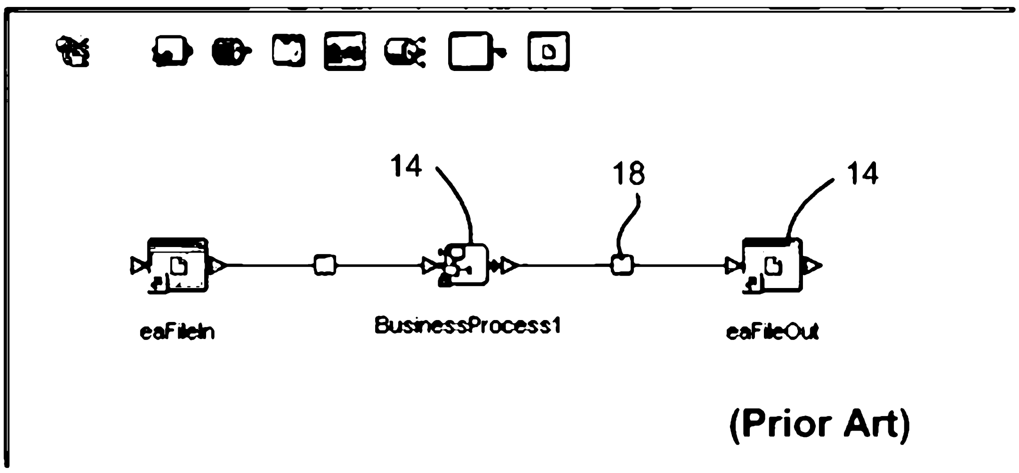
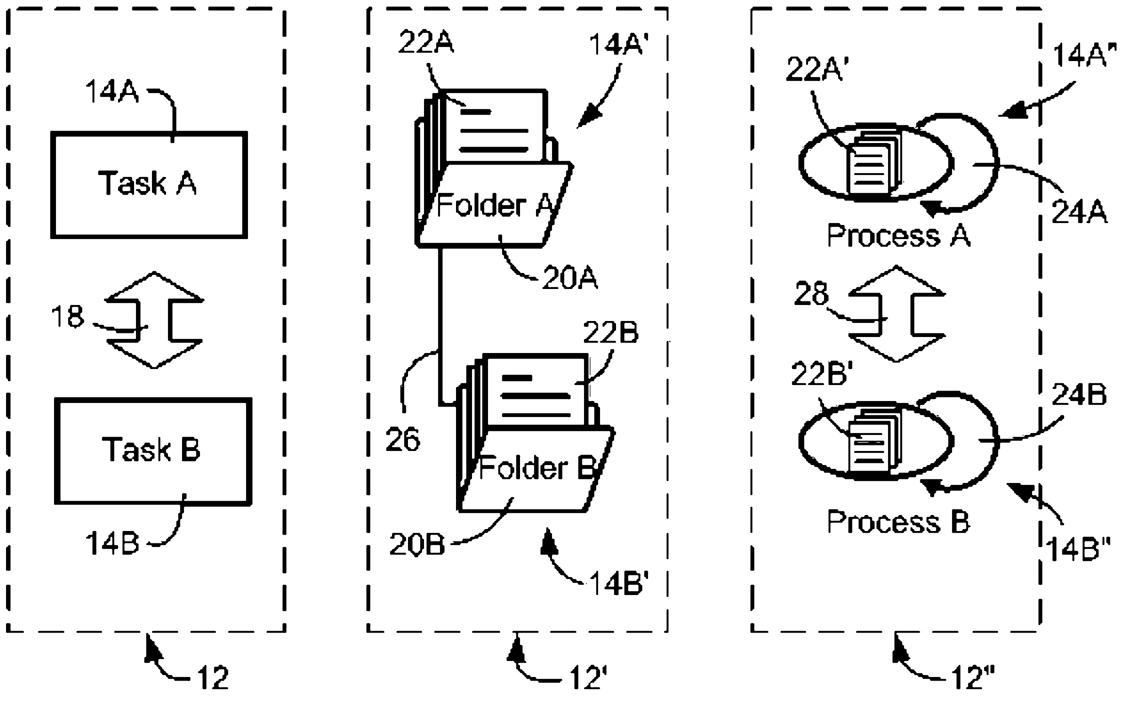
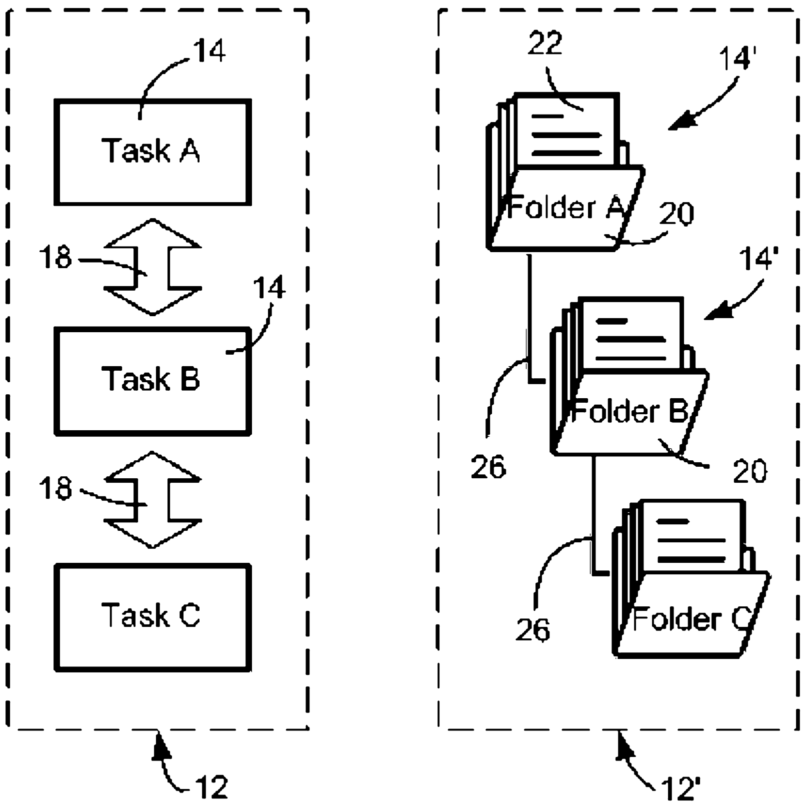


FIG. 2B



14A
Task A
18
Task B
14B
12
22A
14A'
Folder A
20A
22B
26
Folder B
20B
14B'
12'
22A'
14A"
24A
Process A
28
22B'
24B
Process B
14B"
12"

FIG. 3A



14
Task A
18
14
Task B
18
Task C
12
22
14'
Folder A
20
14'
26
Folder B
20
26
Folder C
12'

FIG. 3B

2019100212 27 Feb 2019

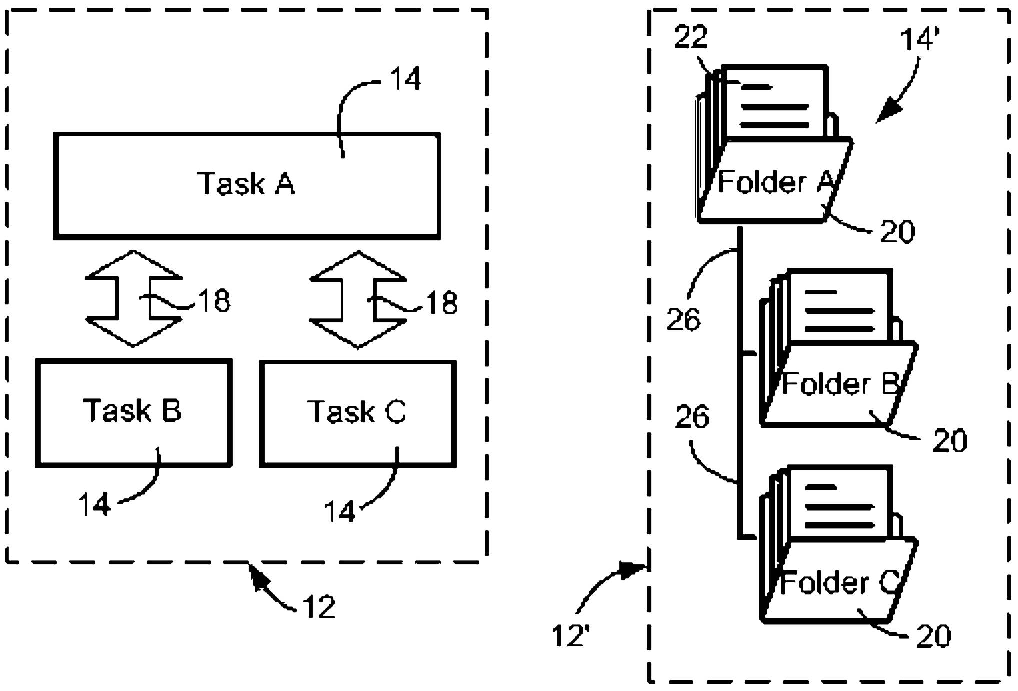


FIG. 3C

2019100212 27 Feb 2019

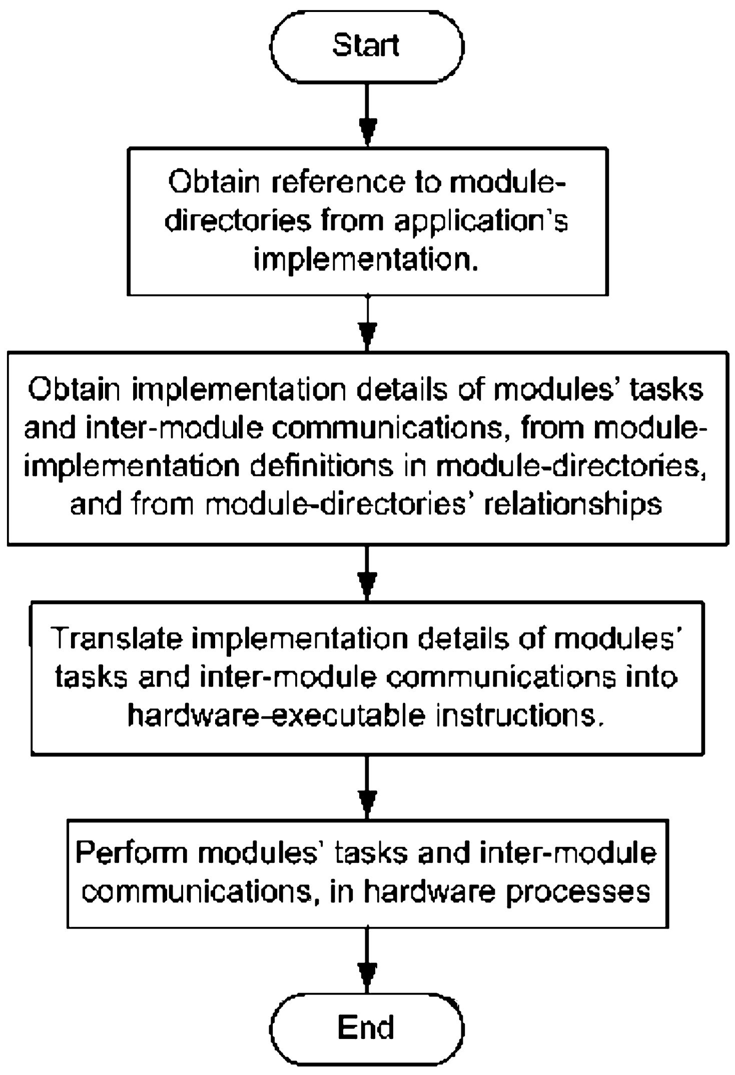

Start
Obtain reference to module-directories from application's implementation.
Obtain implementation details of modules' tasks and inter-module communications, from module-implementation definitions in module-directories, and from module-directories' relationships
Translate implementation details of modules' tasks and inter-module communications into hardware-executable instructions.
Perform modules' tasks and inter-module communications, in hardware processes
End

FIG. 3D

2019100212 27 Feb 2019

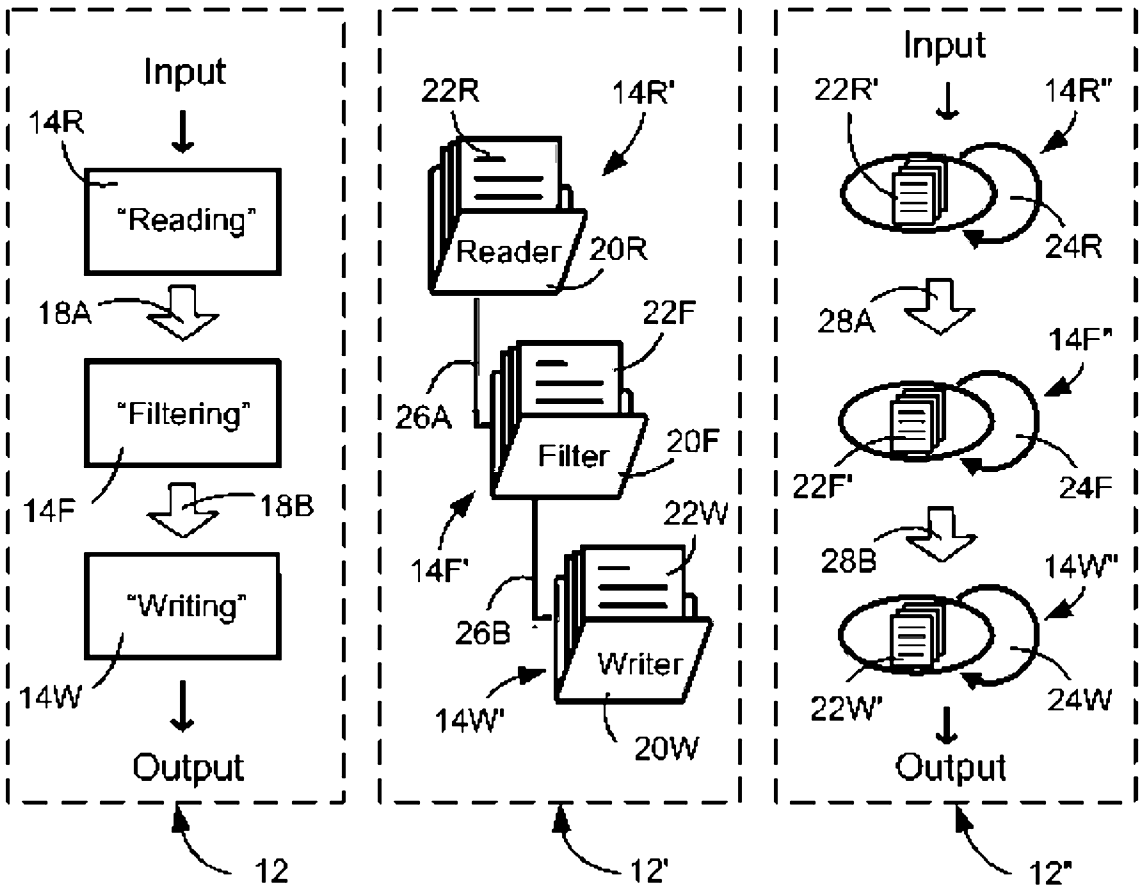


FIG. 4