(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 August 2009 (13.08.2009)

PCT

(10) International Publication Number
WO 2009/099691 A2

(51) International Patent Classification:
G06F 9/06 (2006.01) *G06F 15/17* (2006.01)
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/US2009/030211

(22) International Filing Date:
6 January 2009 (06.01.2009)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
12/027,990 7 February 2008 (07.02.2008) US

(71) Applicant *(for all designated States except US)*: **MICROSOFT CORPORATION** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **KIMMERLY, Randy S.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **ANDERSON, Christopher L.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **SZYPERSKI, Clemens A.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **MOORE, Anthony J.**; One Microsoft Way, Redmond, Washington 98052-6399 (US). **BOX, Donald F.**; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO,

[Continued on next page]

(54) Title: ITERATIVE COMPONENT BINDING

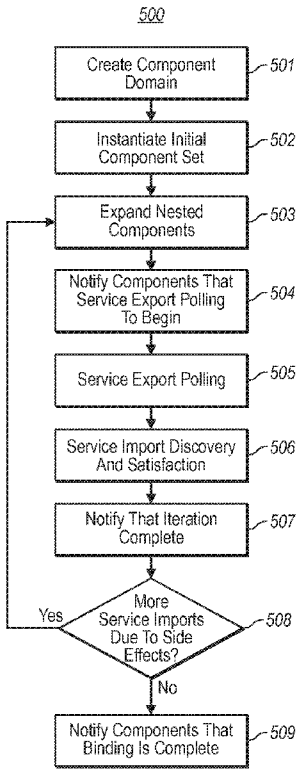

FIG. 5

(57) Abstract: Component domains used to define a binding between various components associated with the component domain. The component domain tracks the various services to be imported by each component, and tracks the various services exported by each component. At runtime, rather than at compile time, the various interrelations between the components are bound using the service import and export data. Accordingly, depending on the components available and their characteristics, the bindings may differ. This permits applications to be much more flexible. In one implementation, the binding is performed iteratively. For example, if after one iteration of binding service imports to service exports, some components may expand the services that they export, or the services that they desire to import.

WO 2009/099691 A2

NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

ITERATIVE COMPONENT BINDING

BACKGROUND

[0001] Object-oriented programming structures software as a collection of object instances or "components," each patterned after a class that defines the methods and properties of the component. When referring to programming, "componentization" refers to the defining of the components such that they are made available to a program. Of course, components of a program are functionally interconnected in order to accomplish the larger purposes of the program. "Composition" refers to the defining of how objects in the program are related together.

[0002] Typically, the interrelation between objects is accomplished using code or declarations that define very specifically the relationship between components of a program. Great progress on the art of object-oriented programming has been made using such explicit composition of the interconnection between coded objects. In conventional explicit composition, for example, code that defines the composition of the program (i.e., the interrelation between components) is compiled after authoring the program using source code. The executable file thus from its birth represents the various components of the program and the various interrelations between the program components.

BRIEF SUMMARY

[0003] Embodiments described herein relate to the use of component domains to define a binding between various components associated with the component domain. The component domain tracks the various services to be imported by each component, and tracks the various services exported by each component. At runtime, rather than at compile time, the various interrelations between the components are bound using the service import and export data. Accordingly, depending on the components available and their characteristics, the bindings may differ. This permits applications to be much more flexible. In one embodiment, the binding is performed iteratively. For example, if after one iteration of binding service imports to service exports, some component may expand the services that they export, or the services that they desire to import.

[0004] This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] In order to describe the manner in which the above-recited and other advantages and features of embodiments described herein can be obtained, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments which are illustrated in the appended drawings. Understanding that these drawings depict only example embodiments of the broader principles and are not therefore to be considered to be limiting of the scope of the invention, the embodiments will be described and explained with additional specificity and detail through the use of the accompanying drawings in which:

[0006] Figure 1 illustrates an example computing system that may operate a component domain;

[0007] Figure 2 illustrates an example architecture of a component domain with its associated logic and components contained within the component domain;

[0008] Figure 3 illustrates an example of the various components in the component domain after being bound;

[0009] Figure 4 illustrates a process flow associated with nested component expansion;

[0010] Figure 5 illustrates a flowchart of a method for iteratively binding the components in the component domain;

[0011] Figure 6 illustrates a flowchart of a method for expanding nested components after instantiating a component having nested component; and

[0012] Figure 7 illustrates a flowchart of a method for satisfying a service import with a service export.

DETAILED DESCRIPTION

[0013] In accordance with embodiments described herein, component domains are used to create a binding between various components in the component domain. The component domain tracks the various services to be imported by each component, and tracks the various services exported by each component. At

runtime, rather than at compile time, the various interrelations between the components are bound using the service import and export data. Accordingly, depending on the components available and their characteristics, the bindings may differ. This permits applications to be much more flexible. In one implementation, the binding is performed iteratively. For example, if after one iteration of binding service imports to service exports, some component may expand the services that they export, or the services that they desire to import.

[0014] First, after some introductory discussion regarding computing systems in general, a basic computing system will be described with respect to Figure 1. Then, various embodiments of a component domain with component domain logic and various embodiments of its operation will be shown and described with respect to Figures 2 through 7.

[0015] Computing systems are now increasingly taking a wide variety of forms. Computing systems may, for example, be handheld devices, appliances, laptop computers, desktop computers, mainframes, distributed computing systems, or even devices that have not been conventionally considered a computing system. In this description and in the claims, the term "computing system" is defined broadly as including any device or system (or combination thereof) that includes at least one processor, and a memory capable of having thereon computer-executable instructions that may be executed by the processor. The memory may take any form and may depend on the nature and form of the computing system. A computing system may be distributed over a network environment and may include multiple constituent computing systems.

[0016] Referring to Figure 1, in its most basic configuration, a computing system 100 typically includes at least one processing unit 102 and memory 104. The memory 104 may be physical system memory, which may be volatile, non-volatile, or some combination of the two. The term "memory" may also be used herein to refer to non-volatile mass storage such as physical storage media. If the computing system is distributed, the processing, memory and/or storage capability may be distributed as well. As used herein, the term "module" or "component" can refer to software objects or routines that execute on the computing system. The different

components, modules, engines, and services described herein may be implemented as objects or processes that execute on the computing system (e.g., as separate threads).

[0017] In the description that follows, embodiments are described with reference to acts that are performed by one or more computing systems. If such acts are implemented in software, one or more processors of the associated computing system that performs the act direct the operation of the computing system in response to having executed computer-executable instructions. An example of such an operation involves the manipulation of data. The computer-executable instructions (and the manipulated data) may be stored in the memory 104 of the computing system 100.

[0018] Computing system 100 may also contain communication channels 108 that allow the computing system 100 to communicate with other computing systems over, for example, network 110. Communication channels 108 are examples of communications media. Communications media typically embody computer-readable instructions, data structures, program modules, or other data in a modulated data signal such as a carrier wave or other transport mechanism and include any information-delivery media. By way of example, and not limitation, communications media include wired media, such as wired networks and direct-wired connections, and wireless media such as acoustic, radio, infrared, and other wireless media. The term computer-readable media as used herein includes both storage media and communications media.

[0019] In Figure 1, the computing system 100 is shown with a component domain 112 within the non-volatile portion of the memory 104. As will be described in further detail with respect to Figure 2 and subsequent figures, the component domain 112 is used as a logical container into which to instantiate various components. The component domain logic 114 then dynamically binds the various components, rather than having the components bound at compile time.

[0020] Embodiments within the scope of the present invention also include computer-readable media for carrying or having computer-executable instructions or data structures stored thereon. Such computer-readable media can be any

available media that can be accessed by a general purpose or special purpose computer. By way of example, and not limitation, such computer-readable media can comprise physical storage and/or memory media such as RAM, ROM, EEPROM, CD-ROM or other optical disk storage, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to carry or store desired program code means in the form of computer-executable instructions or data structures and which can be accessed by a general purpose or special purpose computer. When information is transferred or provided over a network or another communications connection (either hardwired, wireless, or a combination of hardwired or wireless) to a computer, the computer properly views the connection as a computer-readable medium. Thus, any such connection is properly termed a computer-readable medium. Combinations of the above should also be included within the scope of computer-readable media.

[0021] Computer-executable instructions comprise, for example, instructions and data which cause a general purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions. Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described herein. Rather, the specific features and acts described herein are disclosed as example forms of implementing the claims.

[0022] Figure 2 illustrates a component domain 210 and an associated component domain manager labeled as "component domain logic" 220. The component domain 210 of Figure 2 may be, for example, the component domain 112 of Figure 1. The component domain 210 may be any logical container that may contain component instances. The flexibility in the component domain 210 allows the component domain to contain a variety of component instances of a variety of classes. The types of components included within the component domain 210 will depend on the application, and the building blocks used by that application. Accordingly, the principles of the present invention are not limited to the types of components included within the component domain 210, nor the number of

components within the component domain. Nevertheless, for illustrative purposes, six components 211, 212, 213, 214, 215 and 216 are illustrated as being contained within the component domain 210.

[0023] The component domain 210 may be any logical container that is capable of literally or logically containing other components. For example, the component domain 210 may be an instance of a container class capable of receiving various component instances, with the collection of component instances being a data structure. The component domain 210 may also be an object that does not directly contain each component instance, but perhaps has information regarding associated components. Even in that case, the components may be considered as logically within the component domain 210. For instance, in one embodiment, the component domain includes only a group of values, perhaps a group of name-value pairs. The component domain receives a value from components that export the service, and the component that imports that service receives the value. Still, even in that context, the components may be considered to be logically inside or within the component domain. In this description and in the claims, when components are referred to as being "within" or "inside" the component domain, that means that the components are either contained within the component domain, or are somehow logically associated with the component domain in a manner that the component domain logic 220 is capable of communicating with and binding the various components associated with the component domain.

[0024] The component domain logic 220 is illustrated as including various mechanisms including, for example, an association mechanism 222, a service inventory mechanism 223, a nested component expansion mechanism 224, a matching mechanism 225 having associated matching rules 226, an abstraction mechanism 227, and a cataloging mechanism 228 having an associated catalog 229. These various mechanisms 222 through 229 are shown as separate mechanisms. However, this is for purposes of clarity in explaining the functionality of the component domain logic 220. It is not used to represent that they are actually separate mechanisms. Some of the mechanisms may be combined into a single component. In fact, all of the component domain logic may be monolithic. On the

other extreme, each mechanism may be composed of multiple components. A single component may also include logic that contributes to multiple mechanisms. The operation of the various component domain logic 220 mechanisms will be described shortly.

[0025] First, however, this description refers back to the various constituent components instantiated within the component domain 210. In the illustrated example, the constituent components include components 211 through 216. Each component is an instance of a particular class. To symbolically illustrate this principle, each component is illustrated as a particular shape. For instance, in the example, the component 211 is illustrated as a square. Components 212 and 214 are illustrated as triangles indicating that they are each instances of the same class, which is not the same class as the component 211. The component 213 is illustrated as a circle symbolizing that it is an instance of a class that is different than components 211 and 212. The component 215 is illustrated as a top-heavy parallelogram symbolizing that it is an instance of a class that is different than components 211 through 214. Lastly, the component 216 is illustrated as a bottom-heavy parallelogram symbolizing it is an instance of a class that is different than components 211 through 215.

[0026] Each component includes potentially one or more service imports, and potentially one or more service exports. A "service import" with respect to a component is an indication that the component is requesting a service. A "service export" with respect to a component is an indication that a component is offering to provide a service. Service imports and exports are each symbolically represented in Figure 2 by an arm extending from the corresponding component. Service imports are represented by an arm that has a cup form at its end. Service exports are represented by an arm that has a ball form at its end. In the configuration of Figure 2, all of the components 211-216 are instantiated within the component domain 210. However, none of the components are connected together. The manner in which the components are to be connected is furthermore not explicitly specified in advance of runtime. Instead, the service imports and exports are

specified. The various services' imports and exports are for various services represented abstractly using letters A through F in Figure 2.

[0027] For instance, component 211 has one service export for service B as represented by arm a; and two service imports, one for service A as represented by arm i, and yet another for service A as represented by arm j. Component 212 has two service exports, one for service A as represented by arm b, and one for service E as represented by arm c. Component 212 has no service imports. Component 213 has one service export for service F as represented by arm d; and two service imports, one for service B as represented by arm k, and another for service C as represented by arm m. Component 214 is of the same class as component 212. In one embodiment, the type of service imports and service exports are defined by the class. Accordingly, component 214 is like component 212, in that it has no service imports, and two service exports, one for service A as represented by arm f, and one for service E as represented by arm e. Component 215 has no service imports and one service import, which is for service D as represented by arm g. Component 216 has one service import for service D as represented by arm n, and one service export for service C as represented by arm h.

[0028] The component domain 210 shows the various components 211 through 216 prior to the composition of the components. "Composition" refers to the defining of how the various components are to be interconnected. Ultimately, the components in the component domain 210 will have their interconnections defined. However, this was not done prior to running the application represented by the various components in the component domain 210. Rather, the component domain logic 220 will define the connections between the various components at runtime.

[0029] Figure 3 illustrates the various components 211 through 216 after composition. In one particular matching operation, the service exports for a particular service are to be coupled with a service import for the same particular service. In Figure 3, the service export for service A of the component 212 is connected to the service import for service A of the component 211 (as represented by the arm b meeting with the arm j). This represents that component 212 provides service A to component 211 using the relevant interface. The same symbolism

applies to the other connections between service imports and exports illustrated in Figure 3 with respect to their respective service. For example, the following arm pairs are shown connected, arms f and i for service A, arms a and k for service B, arms h and m for service C, and arms g and n for service D. The service export for service E of the component 212 (represented by arm c) is not connected to any service import, representing that this service export is latent. The same symbolism applies to the arms d and e.

[0030] Although the connections between components are not defined ahead of time, the service exports and imports for the components are indicated. It does not matter where or how the service exports and imports are specified. However, one possible location may be in the class definition itself. For example, consider the following class definition for a class called "Consumer" written in C# code:

```
class Consumer
{
        [Import ("Consumer Report")]
        IReport MyReport {get; set;}
        .
        .
        .
}
```

[0031] In square brackets, the class Consumer declares that it imports a service called "Consumer Report." The content of square brackets does not affect the functionality of the compiled object in C#. Rather, the contents are written to the compiled class as metadata that accompanies a compiled type or member that describes it in a way that is not part of its default execution, but can be inspected and acted on from the outside. In this example, when an object of a class is to be instantiated, the class definition, either in its pre-compiled or post-compiled form, may be referred to in order to identify the service import for a particular object following that class.

[0032] In one example matching between service imports and service exports, the service name is to exactly match, although other matching rules are conceivable. The following represents a class definition represented in C# in which the class exports a service called "Consumer Report":

```
class ReportEnvelope
{
        [Export ("Consumer Report")]
        IReport Content {get; set}
        .
        .
        .
}
```

[0033] Here, the metadata expression in the brackets declares that the class ReportEnvelope exports a "Consumer Report" service. For instance, when an object of the ReportEnvelope class is connected with the object of the Consumer class, the Consumer object may get reports from the ReportEnvelope object.

[0034] In this example, the class itself, whether compiled or pre-compiled, may be referred to in order to determine the service exports and service imports for a particular object. In another embodiment, the object may be configured to offer an interface that the component domain logic may use to ask the various objects for the service imports and export identifications.

[0035] The various mechanisms 222-229 of the component domain manager 220 will now be described in detail. Once again, these mechanisms are divided by function, and do not represent any actual object boundaries in the component domain manager 220. As will be mentioned below, some of the various mechanisms 222-229 may occasionally use an instantiation mechanism that is often provided by the underlying runtime provided in the system. The instantiation mechanism has the capability to use a class definition to instantiate a component that follows that class.

[0036] An association module 222 is capable of associating instantiated components with the component domain 210. For instance, in Figure 2, components 211 through 216 are shown associated within the component domain 210, symbolizing that they are not just instantiated, but they are instantiated in such a way that they are associated with the component domain 210. The components associated with the component domain 210 will be subjected to a particular binding processing at runtime. The association of the components with the component

domain may be done at the time of instantiation of each component, or sometime thereafter.

[0037] A service inventory mechanism 223 indexes services exported and imported by the components associated with the component domain. The service inventory mechanism 223 may have a mechanism for discovering those services. For example, in one embodiment, when a component of a particular class is instantiated, the service inventory mechanism 223 may access a class definition to evaluate the service(s) imported or exported. In another embodiment, the components may be drafted so as to respond to discovery requests. For example, the components may offer an Application Program Interface that serves to notify querying entities of the service(s) that it exports and imports. Alternatively, the component may be drafted to write its service imports and exports to a particular location immediately upon instantiation.

[0038] For instance, referring to Figure 2, the service inventory mechanism may conceptually maintain the following Table 1:

Component Identifier	Services Exported	Services Imported
211	B	A, A
212	A, E	
213	F	B, C
214	A, E	
215	D	
216	C	D

Table 1

The service inventory mechanism 223 need not represent information as a table, but a table is a useful illustration to show the reader the type of information that may be maintained. Furthermore, the service inventory mechanism 223 may not know all of the services exported or imported by a particular component.

[0039] A nested component expansion mechanism 224 evaluates whether a component in the component domain has nested components therein. For example, Figure 4 represents that one of the components (i.e., component 215) of Figure 2 includes a hierarchy of nested components prior to expansion represented by arrow 410. In particular, the component 215 in Figure 4 includes two child components

401 and 402 prior to expansion. One of those child components 401 includes their own child components 411 and 412 prior to expansion.

[0040] The nested component expansion mechanism 224 is configured to discover nested components, and expand them such that services exported and imported by nested components are also made visible to the binding operation. For example, after expansion 410, the various nested components 401, 402, 411 and 412 are shown with their various services exported (service A and E in the case of component 411) shown as visible for binding. Additionally, the various services imported (service A for component 401, service B for component 402, and service C for component 412) are shown as visible for binding. Thus, after expansion, these nested components are recognized by the service inventory mechanism 223.

[0041] A matching mechanism 225 actually responds to a request to bind the components. In order to do that, the matching mechanism 225 follows a set of matching rules 226. The matching mechanism 225 follows the matching rules 226 in order to match up and bind service imports to service exports. For example, in Figure 3, the various components in the component domain 210 are shown bound together.

[0042] The matching rules may be a default implicit set of matching rules, or there may be other matching rules that override the default matching rules. For instance, perhaps, by default, matching of a service import to a service export is made by finding an exact service name match. For instance, in the class definition examples set forth above, the class "Consumer" has a service import called "Consumer Report." On the other hand, the class "ReportEnvelope" has a service export by the same exact name "Consumer Report." In accordance with the exact service name matching rules, therefore, a service export of an instance of ReportEnvelope may be bound to a service import of an instance of Consumer.

[0043] However, other service matching rules may be applied. For instance, case may be ignored in some matching rules. In other rules, there may be clusters of service names that may be deemed to match each other. For instance, there might be a service name in Italian, a service name in French, and so forth, which may each be deemed to match. In accordance with another set of matching rules, if a

service import or export is not specified by name, the matching mechanism 224 may actually structurally evaluate member names, types and parameters of the service import, against member names, types and parameters of the service export.

[0044] The service matching mechanism 225 may also take steps to not just logically match components, but may also at least contribute to the actual functional binding of the service import and export. In one embodiment, once the service matching mechanism 225 finds a match, the service matching mechanism 225 may acquire the requested service from the service exporting component, and provide that service to the service importing component.

[0045] In another embodiment, the service matching mechanism 225 may provide a handle data structure to the service importing component. The service importing mechanism may then use that handle to complete the binding with the service exporting mechanism. The service matching mechanism may optionally allow the service importing mechanism to accept or reject the match. For instance, the service importing component may notify the service matching mechanism if the match is acceptable, following which the actual binding is to occur. If the service is not deemed acceptable, perhaps the service matching mechanism finds a backup match, and so forth, until an acceptable match is found and accepted by the service importing component.

[0046] For example, the handle data structure might include a factory that the service importing component might use to perform the binding operation between its import and the export proffered by the matching component. The handle might also include metadata (called hereinafter "selection metadata") that contains information that the service importing component might use to determine whether or not to trigger the factory to perform the binding. In one embodiment, the service importing component might not use the selection metadata, but may instead use the factory to automatically accept the binding. However, the service importing component may alternatively use the selection metadata to evaluate whether the proposed export will be acceptable to the service importing component. If the service importing component elects to accept the binding, then the service

importing component may use the factory to trigger the system to complete the binding.

[0047] In other cases, there might be multiple possible matches of service exports to a particular service import. In that case, the handle might include a factory and selection metadata for each potential service export. The service importing component may then use the metadata to select none or perhaps a subset of the service exports for importing. The service importing component may then use the corresponding factories of the selected service exports to complete the binding with each of the selected service exports.

[0048] For example, suppose a Consumer component is to import a service named "Consumer Report" as in the example above. If there were multiple components that offered the service named "Consumer Report," the Consumer component may be given factories and selection metadata for each potential service export match. The selection metadata might include, for example, 1) the date that the Consumer Report was generated, and 2) an identification of a product class that the Consumer Report is regarding. If the Consumer component was interested in only one service export named "Consumer Report," the Consumer component might use the selection metadata to identify a Consumer Report service export that was generated recently, and which indicates a product class that was most relevant to the Consumer component.

[0049] In some cases, there may be no components within the component domain that offer a service export that matches a particular service import. In such a case, the binding may still be potentially performed. There will now be described a number of ways of handling the case where a bind cannot be initially performed for a particular service import.

[0050] In one case, the service import may have associated with it an "importance" property or the like, which specifies how urgent it is to the functionality of the overall application that the service import be satisfied. If the importance is high, then a service export should be found somewhere. However, if the importance is low, then perhaps the application can get by without binding the service import at

all. There may be a variety of intermediary importance levels, which may govern to what extent the binding operation is to go to try to find a service export.

[0051] In one embodiment, if a service export for a particular service import is not found within the component domain, an abstraction mechanism 227 may perhaps find another component from another component domain. Therefore, there may be many component domains running on a single computing system, or that may be accessible over a network by the computing system. In that case, the abstraction mechanism 227 may consult the service inventory mechanism (or perhaps the cataloging mechanism 228) for those other component domains until a component is found that offers the service export corresponding to the service import to be bound. In one embodiment, the abstraction mechanism 227 may simulate itself as a component that offers the service export. The service import may be thus bound to the abstraction mechanism 227 itself. When the service importing component makes a call to the abstraction mechanism 227 for the service, the abstraction mechanism 227 may interface with that external component to perform the requested service.

[0052] In one embodiment, the component domain may be associated with a number of related components domains. The abstraction mechanism 227 may search out all of those related component domains for the appropriate service exports. In other embodiments, the abstraction mechanism 227 may search out only some of the related component domains. For instance, if the component domain 210 was a node in a hierarchical tree of component domains, perhaps the abstraction mechanism 227 may search only the component domains representing nodes in the ancestral line, rather than searching cousin component domains. The abstraction component 227 may thus abstract service exports for a number of different service exports, while performing the underlying processing necessary to interface with the various external components to thereby perform the service.

[0053] Alternatively, or in addition, if a service import cannot be satisfied by an already instantiated component within the component domain, a cataloging mechanism 228 may use a catalog 229 to find components that are not presently instantiated that offer a particular service export. For instance, the catalog 229 may

contain a variety of class definitions, each having metadata that describes the services exported by objects of that class. Based on the evaluation of the associated metadata, the cataloging mechanism 228 may find a particular class definition that offers the needed service export. The cataloging mechanism 228 may then select the external component of that class to be instantiated within the component domain 210. Alternatively, the component importing the service may be provided with the metadata that describes the service exported by the cataloged class, so that the importing component can make its own decision regarding whether the service exported will be satisfactory. If it is satisfactory, the component importing the service may cause a component of the class to be instantiated. For instance, the component importing the service may also have been provided with the ability to access a factory that allows the component exporting the service to be instantiated.

[0054] Having described the various functionality of the component domain logic 220 of Figure 2, various processes will now be described with respect to Figures 5 through 7.

[0055] Figure 5 illustrates a flowchart of a method 500 for composing components of a program. After creating a component domain (act 501), an initial set of components are instantiated and associated with the component domain (act 502). For example, referring to Figure 2, the component domain 210 may be created with some or all of the associated component domain logic 220. The initial set of components in the component domain may be as few as one, but may be many more without limit. In the example of Figure 2, there are six components 211 through 216 in the component domain. However, the number of components in the initial set of components is not important to the broader principles of the present invention. Each component in the initial set of components, and any subsequent component that should be instantiated in the component domain, is associated with the component domain such that services imported, if any, and service exported, if any, by any of the components are made visible to the component domain.

[0056] As part of this instantiation process, nested components are expanded (act 503). Figure 6 illustrates a flowchart of a method 600 for instantiating components. The component itself is first instantiated (act 601). If the component does not

contain any nested components (No in decision block 602), then no expansion occurs. On the other hand, if there are nested components (Yes in decision block 602), those nested components are expanded (act 603) such that the nested components are also visible to the component domain to the extent that services imported, if any, by the nested component are made visible to the component domain, and such that services exported, if any, by the nested component are made visible to the component domain. An example of this expansion was illustrated with respect to Figure 4. This expansion may be performed whenever a component having one or more nested components is instantiated into the component domain. Alternatively or in addition, the components may be expanded at a predetermined time each time a binding is iterated. For instance, in Figure 3, the expansion of nested components occurs at the beginning of each iteration.

[0057] Returning to Figure 5, once all of the components have been expanded, the component domain logic optionally notifies at least some of the initial set of components that the component domain logic is about to poll the components for services exported (act 504). The component domains may respond by taking any action it is programmed to take in preparation for being polled regarding its service exports. For instance, the component might programmatically instantiate another component into the component domain. The component might otherwise put its state in appropriate readiness for receiving a service export request.

[0058] Next, the component domain logic polls each component in the component domain for service exports offered by the components (act 505). From this list, service export data may be formulated. Next, the component domain logic may poll each component for service imports, and may satisfy each of the service imports with the list of service exports (act 506). As explained in further detail with respect to Figure 7, if a service export is not found for satisfying a service import, the component domain logic may search the catalog for classes that provide the service, and then if appropriate cause a component of the cataloged class to be instantiated.

[0059] If there are multiple service exports that can satisfy a particular service import, and multiple exports are acceptable to the service import, then all of the

service exports are made available to the service import as a collection of service exports. There is also a case in which there might be more service exports that can satisfy a service import than the service import wants. For example, perhaps there are multiple service exports that might satisfy a service import that has indicated that it wants only one or no more than one service export. Alternatively, perhaps there are 10 service exports available for a service import that has designated that it wants no more than 4 service exports. In that case conflict resolution is used to identify how to handle these cases.

[0060] One optional conflict resolution step is to simply have the composition process fail if there are more service exports than the service import wants. In that case, an exception report may give some guidance as to the reason for the failure to complete the composition process. Another option would be to have rules for designating the acceptability of a particular match, and have the more acceptable service export(s) matched with the service import such that the appropriate number of service exports are bound with the service import. If there were no matching service exports, then the service import might also specify whether or not an empty match would be acceptable. This might depend on the importance of functionality enabled by the importation of the service.

[0061] For example, referring to Figure 7, upon searching the components within the component domain (act 701), it is determined whether or not there exists a service export within the component domain that satisfies a particular service import (decision block 702). If a service component that provides the service export is found (Yes in decision block 702), then the service import is satisfied using the service export of the component found within the component domain (act 703).

[0062] On the other hand, if a service component that satisfies a particular service import is not found within the component domain (No in decision block 702), then a component is found outside of the component domain that exports the appropriate service (act 704). This may be accomplished by, for example, consulting a catalog listing component classes and their services exported.

[0063] Depending on whether the component is to be instantiated within the component domain ("Instantiation" in decision block 705), or abstracted by the component domain logic ("Abstraction" in decision block 705), different action may then be taken.

[0064] If instantiation of the component is to occur, the found component is to be instantiated in the component domain (act 706), and the service exported by the newly instantiated component is bound with the service import of the pre-existing component in the component domain (act 707). If abstraction is to occur, the abstraction mechanism of the component domain logic may abstract the service export offered by the found component (act 708).

[0065] Returning to Figure 5, once the initial iteration of binding is completed, the various components are notified that the binding iteration is completed (act 507). At this stage, the components might make visible new service imports or exports in response to the initial binding.

[0066] In this embodiment, the expansion of nested components are performed first, the polling for service exports is done next, and the polling for service imports and satisfaction of each service import is done next. However, so long as these steps are performed in a predetermined order, and the various components have an understanding of that order, the components may take appropriate action based on where in the predetermined order the components are. For example, in the above example, the components were to first expand nested components prior to service export polling. Furthermore, the services were in a position to get themselves ready for polling by, for example, instantiating other components programmatically. Finally, once the initial binding was complete, the components knew that the iteration might be repeated, and so they had the opportunity to expose new service imports and exports for the next iteration.

[0067] Returning to Figure 5, it is then determined whether or not the binding operation from the last iteration caused any side effects that would require another iteration of the binding operation (decision block 508). For example, in the binding operation, new components may have been instantiated into the component domain. For instance, perhaps components were found in the catalog and instantiated into

the component domain. Those newly instantiated components may themselves have service imports. Furthermore, when a component has a service bound, it may discover that it can offer new service exports, and may need to expose new service imports.

[0068] If reiteration is required (Yes in decision block 508), the component domain logic once again expands any nested components of any newly instantiated components (act 503) (if not expanded already during initial instantiation), and then notifies each of the components that service export polling is about to begin (act 504). The determination that reiteration is to be performed may be performed well after the initial set of components is already bound. For example, a new version of a program may become available requiring a rebinding operation for each of the components.

[0069] If reiteration is not required (No in decision block 508), then the components are notified that the binding operation is complete (act 509). This composition of components was performed at runtime, not at compile time, and was based on a loose identification of a service import and export. This allows for new components to be added, and for old components to be modified, and added to the system after the time of shipment. For instance, a catalog may be updated when new components become available for a particular program. When the program next runs, the components offering a particular service may be acquired from a remote location such as, for example, over a network from a program vendor.

[0070] In some cases, an iteration might be terminated. For instance, in the process of attempting to satisfy the imports with the exports (in act 506), it may be discovered that some imports cannot be acceptably satisfied using the current set of available components. In that case, the process may terminate. The computing system may automatically, or perhaps with user help or perhaps following some policy rules, may then change the set of components by removing one or more components and/or adding one or more components to the component domain. Then, the composition process may again be reinitiated. At the time of the reinitiation of the composition process, if some imports are already bound to some exports, then the rebinding might optionally honor that binding by not attempting to

rebind to another exports. Alternatively, the composition process might start afresh without giving deference to existing bindings. In between these two extremes, there are a variety of different levels of deference that can be given to existing bindings.

[0071] The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

CLAIMS

What is claimed is:

1. A computer program product comprising one or more computer-readable media (104) having thereon computer-executable instructions that, when executed by one or more processors (102) of a computing system (100), cause the computing system to perform a method for composing a plurality of components of a program, the method comprising:

an act of creating (501) a component domain (112, 210);

an act of instantiating (502, 601) an initial set of one or more components (211) such that they are associated with the component domain, and such that services (A, B, C, D, E, F) imported, if any, by each of the components in the initial set are made visible to the component domain, and such that services exported, if any, by each of the components of the initial set are also made visible to the component domain;

an act of the component domain identifying (602) any component of the initial set of one or more components that includes nested components (401, 402, 411, 412);

for each component of the initial set of components that includes one or more nested components, an act of expanding (503, 603) the component that includes one or more nested components such that the nested components are also visible to the component domain to the extent that services imported, if any, by the nested component are made visible to the component domain, and such that services exported, if any, by the nested component are made visible to the component domain;

an act of the component domain polling (505) the initial set of components for services exported; and

an act of the component domain polling (506) the initial set of components for services imported and satisfying at least one of the services imported of at least one of the components of the initial set of components using one of the services exported by another of the initial set of components, wherein the acts of expanding for each component that includes one or more nested components, the act of the

component domain polling for services exported, and the act of the component domain polling for services imported and satisfying at least one of the services imported occur in a predetermined order.

2. A computer program product in accordance with Claim 1, wherein the predetermined order is first the act of expanding for each component that includes one or more nested components, second the act of the component domain polling for services exported, and third the act of the component domain polling for services imported and satisfying at least one of the services imported.

3. A computer program product in accordance with Claim 2, the method further comprising:

prior to the act of polling for services exported, an act of notifying at least some of the initial set of components that polling for services exported is about to begin.

4. A computer program product in accordance with Claim 3, the method further comprising:

after the act of polling for services imported and satisfying at least one of the services imported, and act of notifying at least some of the initial set of components that polling for services imported and satisfying at least one of the services imported has now completed for now.

5. A computer program product in accordance with Claim 2, the method further comprising:

after the act of polling for services imported and satisfying at least one of the services imported, an act of notifying at least some of the initial set of components that polling for services imported and satisfying at least one of the services imported has now completed for now.

6. A computer program product in accordance with Claim 2, wherein the collective acts including the act of expanding for each component that includes one or more nested components, the act of the component domain polling for services exported, and third the act of the component domain polling for services imported and satisfying at least one of the services imported are performed in an initial iteration of the collective acts of polling, the method further comprising:

after the act of polling for services imported and satisfying at least one of the services imported in the initial iteration of the collective acts, an act of determining if there might be further services imports that were added as a result of the act of satisfying at least one of the service imports; and

an act of performing a subsequent iteration of the collective acts.

7. A computer program product in accordance with Claim 6, wherein the collective acts are iteratively performed for a plurality of iterations until all service imports are satisfied.

8. The computer program product in accordance with Claim 6, wherein the additional service imports were added as a result of additional components being associated with the component domain in order to satisfy a service import of the initial set of components, an additional service import being a service import of at least one of the additional components.

9. The computer program product in accordance with Claim 6, wherein the additional service import was added by one of the initial set of components after having had another of its service imports satisfied.

10. The computer program product in accordance with Claim 1, wherein the one or more computer-readable media are physical memory and/or storage media.

11. A computer program product comprising one or more computer-readable media (104) having thereon computer-executable instructions that, when executed by one or more processors (102) of a computing system (100), cause the computing system to perform a method (500) for composing a plurality of components (211-216) of a program, the method comprising:

an act of creating (501) a component domain (112, 210) that is capable of identifying services (A, B, C, D, E, F) exported and services imported by each component instantiated within the component domain;

an act of instantiating (502) an initial set of components in the component domain; and

an act of iteratively performing the following for at least two iterations:

an act of performing (503, 603) a nested component (401, 402, 411, 412) expansion (410) of each of the components in the initial set of components, wherein the nested component expansion allows the component domain to identify services exported and imported by the nested components;

an act of the component domain identifying (505) the services exported, if any, by each of the components in the initial set of components including the nested components;

after identifying the services exported, an act of the component domain identifying the services imported (506), if any, by each of the components in the initial set of components;

an act of satisfying (506) at least one of the services imported; and

an act of determining (508) whether there are yet other services imported that are yet to be satisfied after the act of satisfying, and if so, continuing the act of iterating with the next iteration with respect to a possibly expanded set of components in the component domain, and if not, ending the act of iterating.

12. The computer program product in accordance with Claim 11, wherein the one or more computer-readable media are physical memory and/or storage media.

13. The computer program product in accordance with Claim 12, wherein for at least one of the iterations, the act of satisfying at least one of the services imported comprises:

an act of satisfying at least one of the services imported with a service exported by another of the components in the component domain.

14. The computer program product in accordance with Claim 12, wherein for at least one of the iterations, the act of satisfying at least one of the services imported comprises:

an act of searching the instantiated components within the component domain;

an act of determining that none of the instantiated components in the component domain export the service;

an act of finding a component that is not yet instantiated inside the component domain that exports the service;

an act of instantiating the found component in the component domain; and

an act of binding the service exported by the instantiation of the found component with the service imported by the component that was already instantiated within the component domain.

15. The computer program product in accordance with Claim 14, wherein the act of finding comprises:

an act of consulting a catalog listing components and their services exported.

16. A computer program product in accordance with Claim 11, wherein the nested components expanded includes at least two levels of nested components.

17. A computer program product in accordance with Claim 11, further comprising:

for at least one of the iterations, an act of notifying the components in the component domain before identifying the services exported by the components, and after the act of satisfying at least one of the services exported.

18. A computer program product in accordance with Claim 11, further comprising:

an act of notifying the components in the component domain after the act of iterating is complete.

19. A computer program product comprising one or more computer-readable media (104) having thereon computer-executable instructions that, when executed by one or more processors (102) of a computing system (100) , cause the computing system to perform a method (500) for composing a plurality of components (211-216) of a program, the method comprising:

an act of creating (501) a component domain (112, 21) that is capable of identifying services (A, B, C, D, E, F) exported and services imported by each component instantiated within the component domain;

an act of instantiating (502) an initial set of components in the component domain;

an act of iteratively performing the following for at least two iterations:

an act of performing (503, 603) a nested component (401, 402, 411, 412) expansion (410) of each of the components in the initial set of components, wherein the nested component expansion allows the component domain to identify services exported and imported by the nested components;

an act of notifying (504) the components in the component domain before identifying (505) the services exported by the components,

an act of the component domain identifying (505) the services exported, if any, by each of the components in the initial set of components including the nested components;

after identifying (505) the services exported, an act of the component domain identifying the services imported (506), if any, by each of the components in the initial set of components;

an act of satisfying (506) at least one of the services imported; and

an act of notifying (507) the components in the component domain that the act of satisfying is complete;

an act of determining (508) whether there are yet other services imported that are yet to be satisfied after the act of satisfying, and if so, continuing the act of iterating with the next iteration with respect to a possibly expanded set of components in the component domain, and if not, ending the act of iterating; and

an act of notifying (509) the components in the component domain that the act of iterating is complete.

20. The computer program product in accordance with Claim 19, wherein the one or more computer-readable media are physical memory and/or storage media.

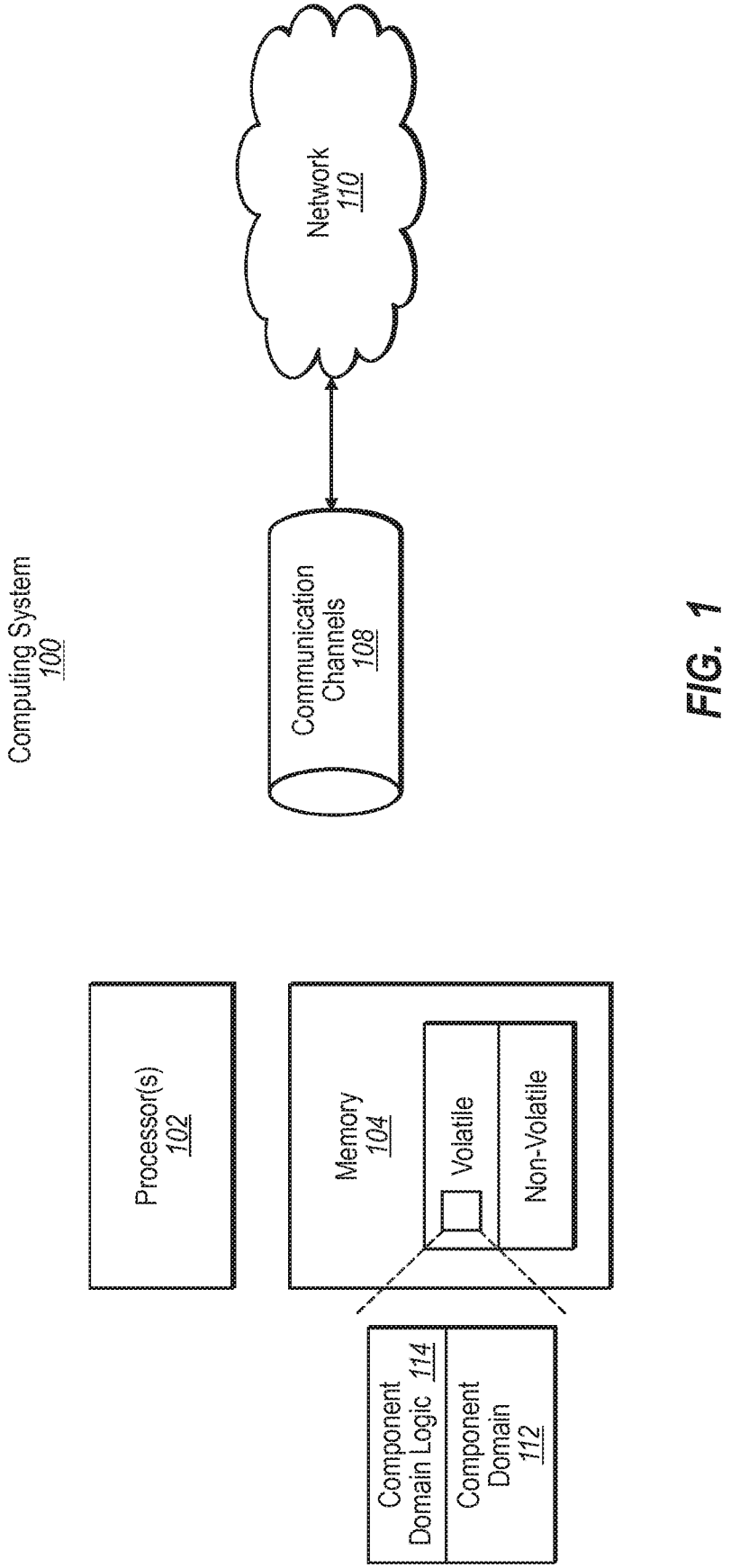
Computing System
100
Network
110
Communication
Channels
108
Processor(s)
102
Memory
104
Volatile
Non-Volatile
Component
Domain Logic
114
Component
Domain
112

FIG. 1

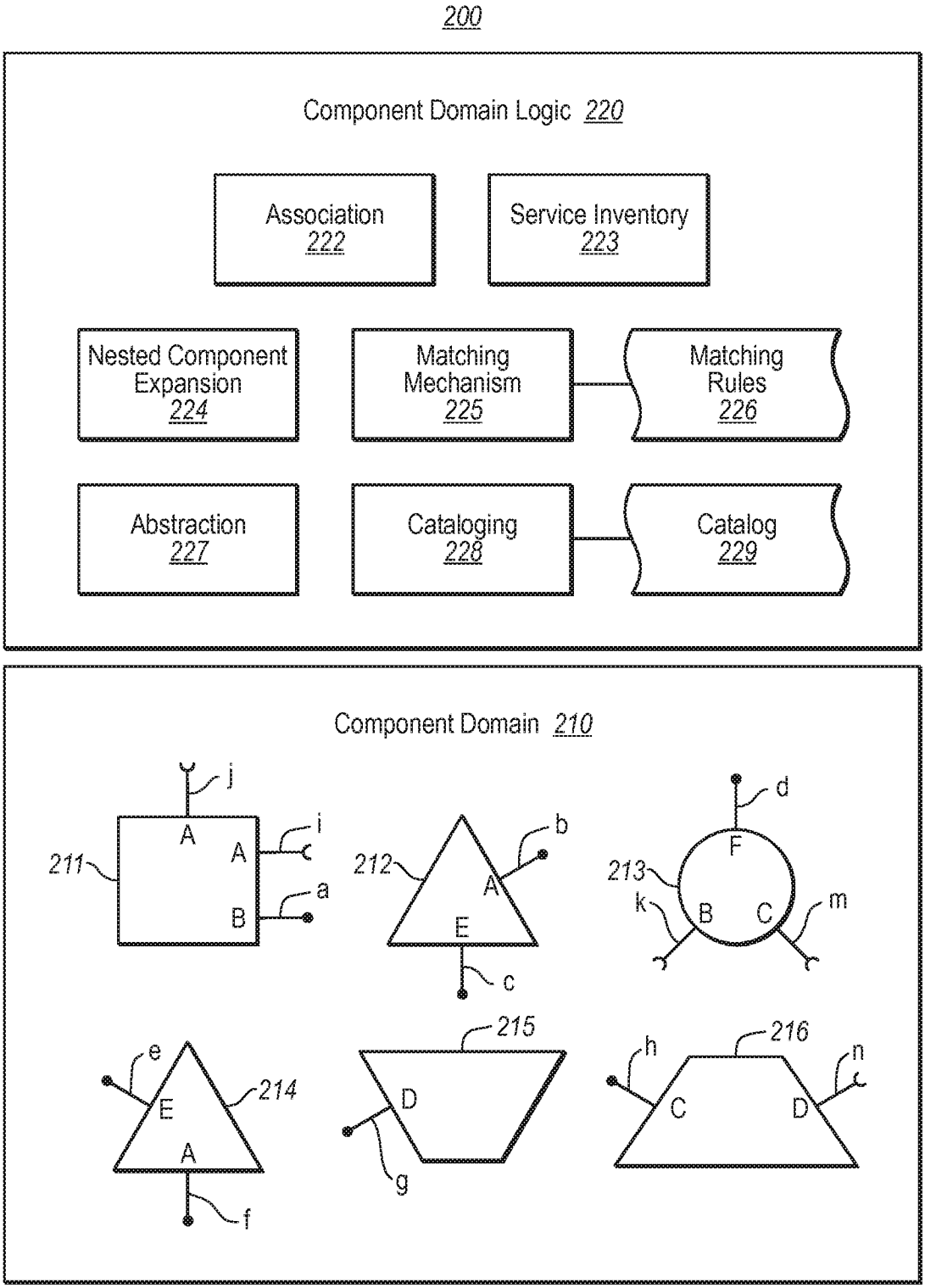
200
Component Domain Logic 220
Association 222
Service Inventory 223
Nested Component Expansion 224
Matching Mechanism 225
Matching Rules 226
Abstraction 227
Cataloging 228
Catalog 229
Component Domain 210
211
212
213
214
215
216
A
B
E
F
C
D
a
b
c
d
e
f
g
h
i
j
k
m
n

FIG. 2

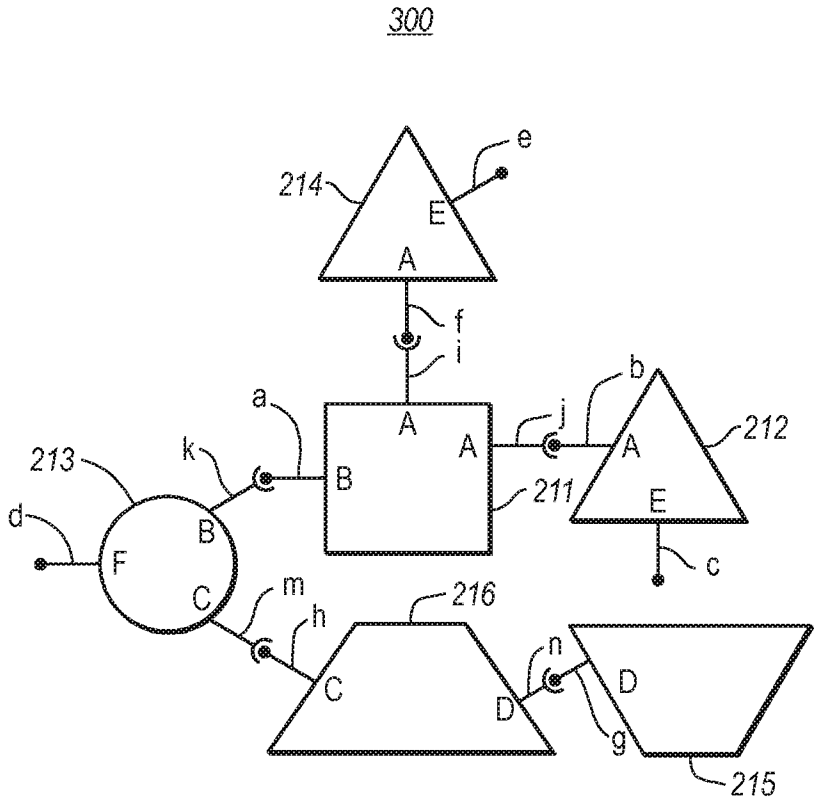

FIG. 3

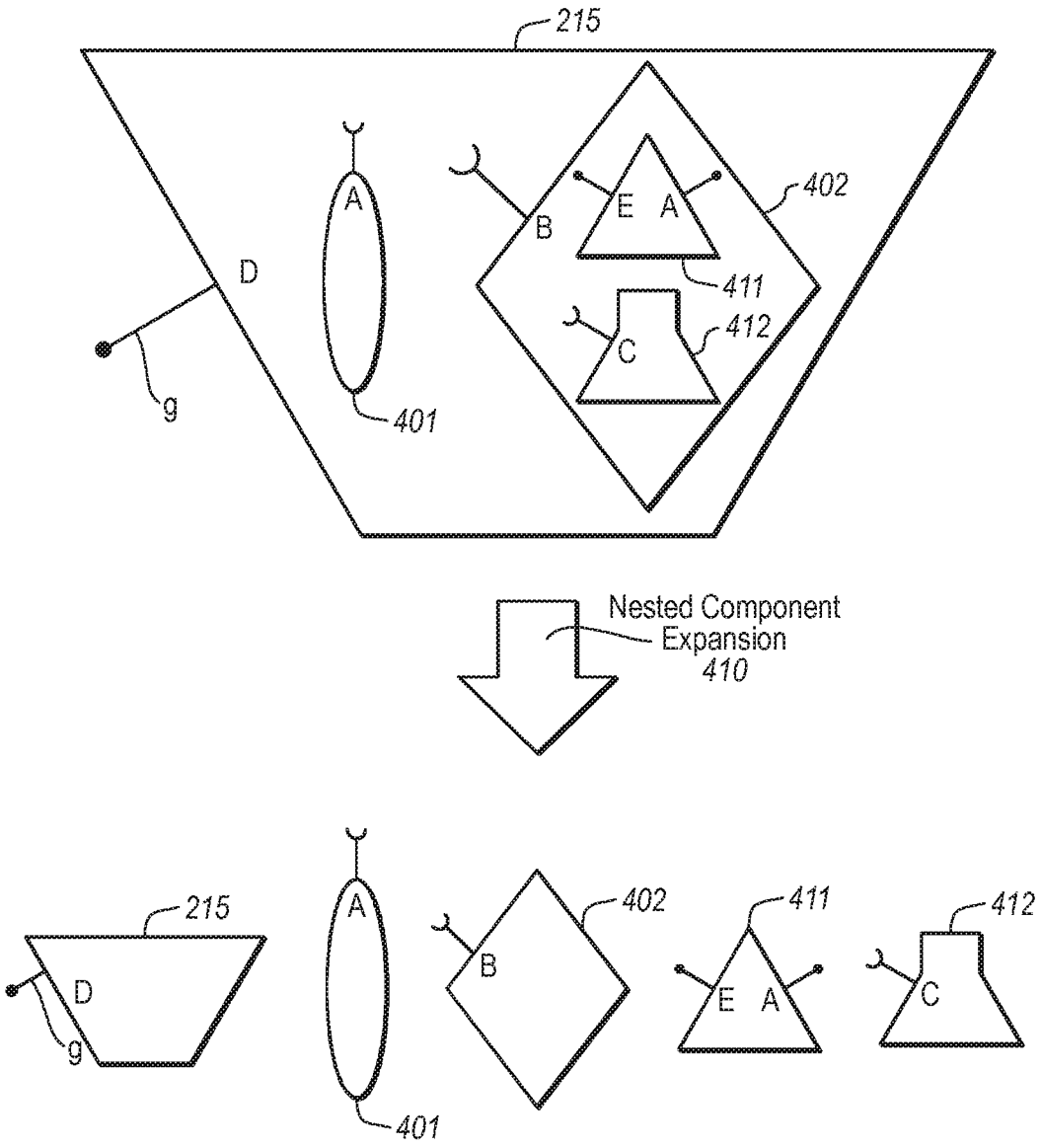
215
A
B
E
A
402
D
411
412
C
g
401
Nested Component
Expansion
410
215
D
g
A
401
B
402
411
E
A
412
C

FIG. 4

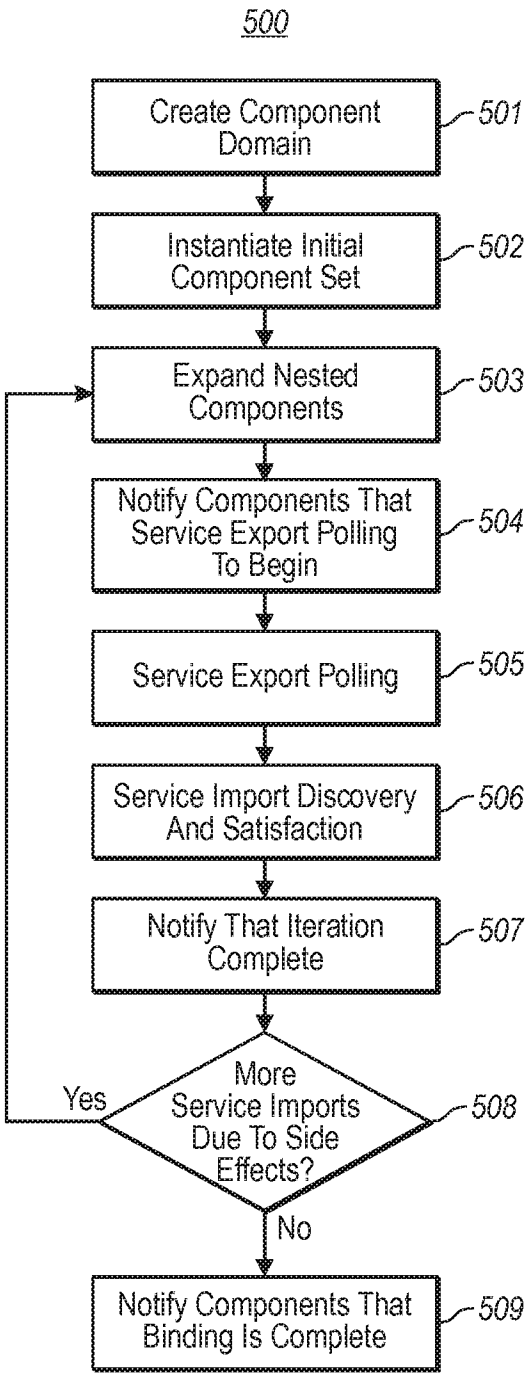
500
Create Component Domain
501
Instantiate Initial Component Set
502
Expand Nested Components
503
Notify Components That Service Export Polling To Begin
504
Service Export Polling
505
Service Import Discovery And Satisfaction
506
Notify That Iteration Complete
507
More Service Imports Due To Side Effects?
508
Yes
No
Notify Components That Binding Is Complete
509

FIG. 5

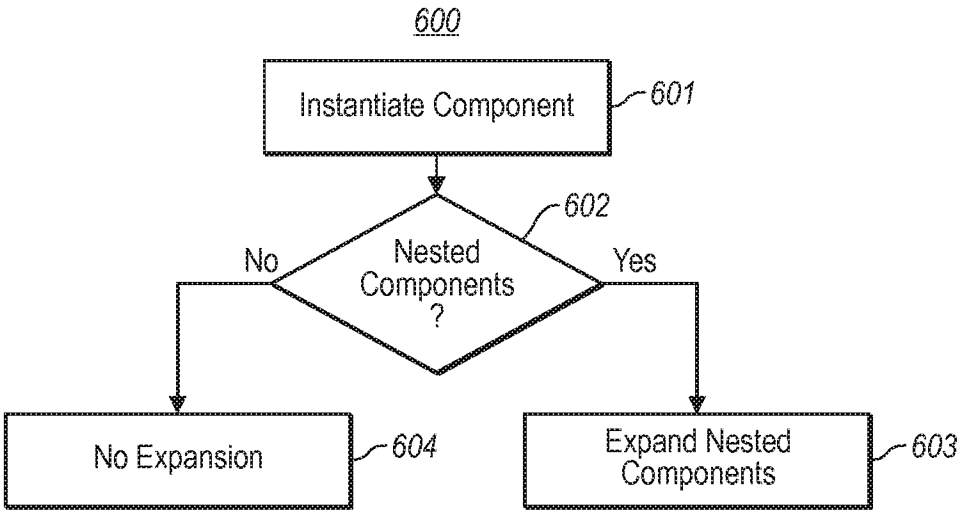
600
Instantiate Component
601
602
Nested Components ?
No
Yes
No Expansion
604
Expand Nested Components
603

FIG. 6

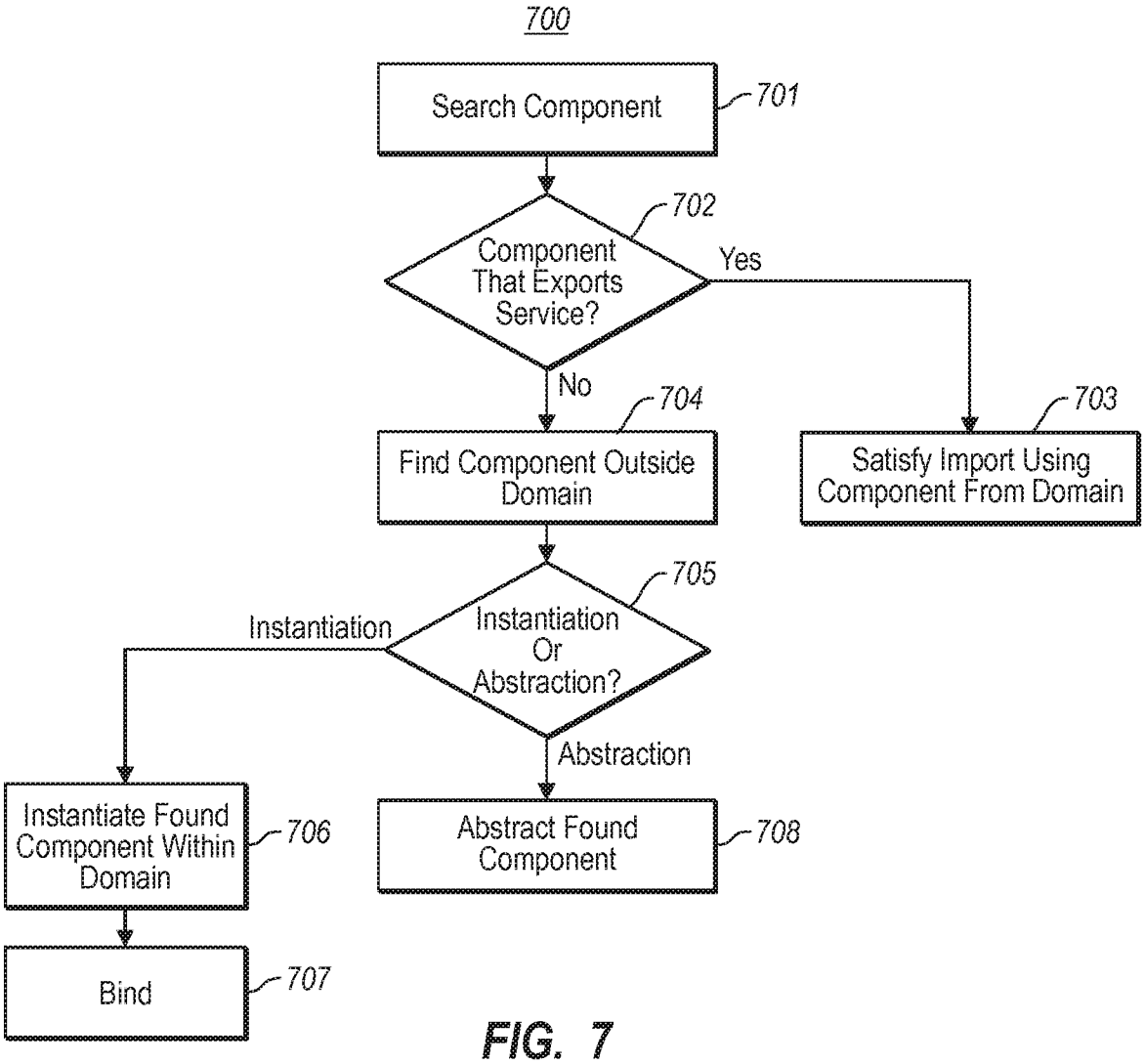
700
Search Component
701
702
Component That Exports Service?
Yes
No
704
703
Find Component Outside Domain
Satisfy Import Using Component From Domain
705
Instantiation Or Abstraction?
Instantiation
Abstraction
Instantiate Found Component Within Domain
706
Abstract Found Component
708
Bind
707

FIG. 7