Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2802225 A1 2011/10/20

(21) 2 802 225

(12) DEMANDE DE BREVET CANADIEN
CANADIAN PATENT APPLICATION

(13) A1

(86) Date de dépôt PCT/PCT Filing Date: 2011/03/25
(87) Date publication PCT/PCT Publication Date: 2011/10/20
(85) Entrée phase nationale/National Entry: 2012/09/17
(86) N° demande PCT/PCT Application No.: US 2011/030068
(87) N° publication PCT/PCT Publication No.: 2011/129989
(30) Priorité/Priority: 2010/04/15 (US12/760,565)

(51) Cl.Int./Int.Cl. *G06F 9/06* (2006.01),
G06F 9/44 (2006.01)

(71) Demandeur/Applicant:
MICROSOFT CORPORATION, US

(72) Inventeurs/Inventors:
BYKOV, EVGUENI N., US;
FINDIK, FERIT, US;
BENSON, RYAN S., US;
OTRYSHKO, VOLODYMYR V., US

(74) Agent: SMART & BIGGAR

(54) Titre : COMPOSITION DE PRESENTATION INDEPENDANTE D'UNE PLATEFORME
(54) Title: PLATFORM INDEPENDENT PRESENTATION COMPOSITION

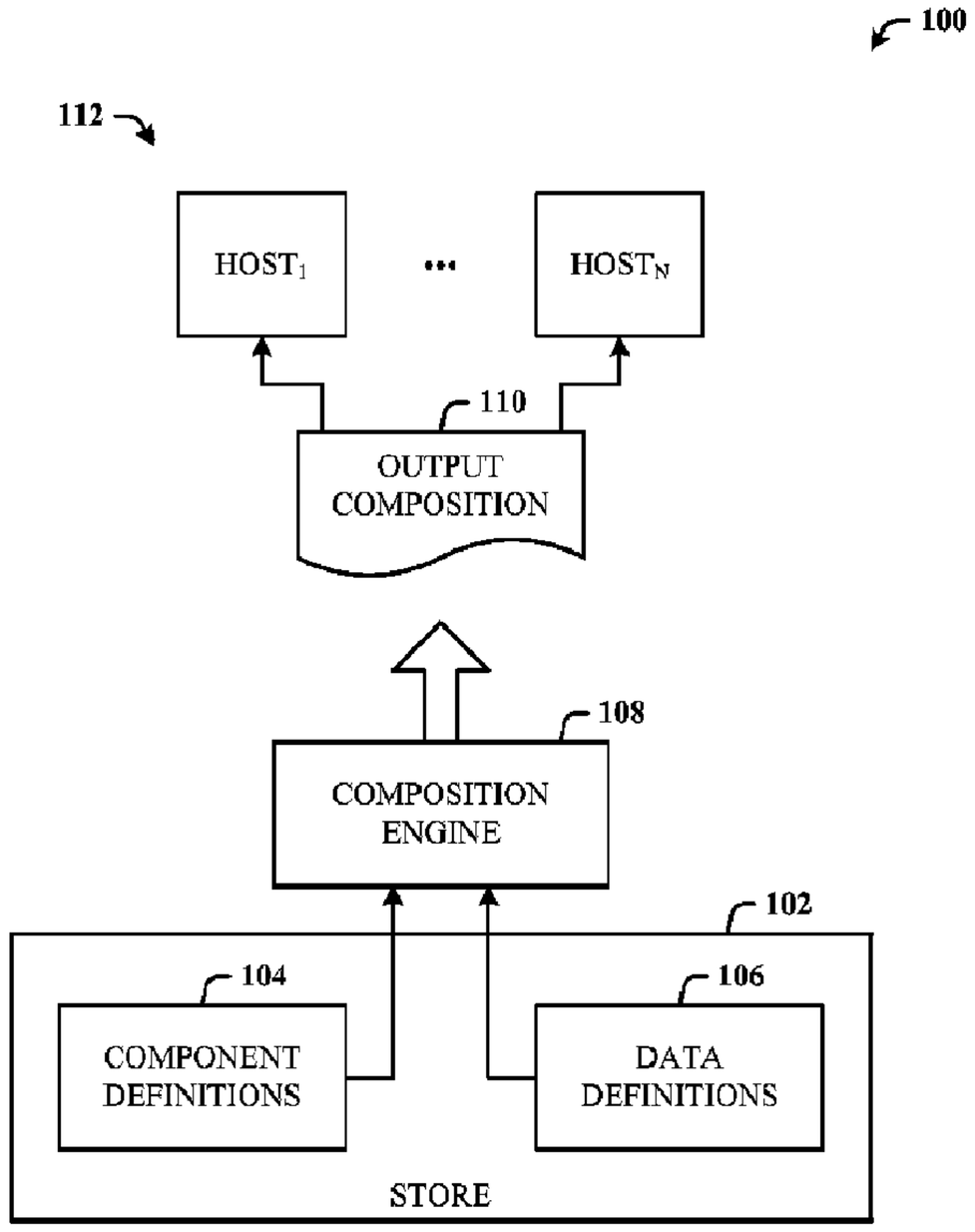

FIG. 1

(57) Abrégé/Abstract:
Architecture that includes a platform independent, configuration driven, presentation composition engine. The composition engine that allows dynamic generation of multiplatform user experience (UX) based on a data contract. By composition, the user can

CA 2802225 A1 2011/10/20

(21) 2 802 225

(13) A1

(57) Abrégé(suite)/Abstract(continued):
select the parts, interactions, and constraints between the interaction and parts, as well as the placement with respect to each other. The UX is dynamically composed from components that are targeted to particular data classes. At runtime, platform dependent component implementations are automatically selected by the engine based on the execution platform of the composition host. A user can create or customize the UX without writing code by composing from a wide variety of presentation widgets that access a wide variety of data sources that can work on many platforms. Compositions are targeted to both a data class and presentation type and can be either predefined or generated.

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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 October 2011 (20.10.2011)

PCT

(10) International Publication Number
WO 2011/129989 A3

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

(21) International Application Number:
PCT/US2011/030068

(22) International Filing Date:
25 March 2011 (25.03.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
12/760,565 15 April 2010 (15.04.2010) US

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

(72) Inventors: **BYKOV, Evgueni N.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **FINDIK, Ferit**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **BENSON, Ryan S.**; c/o Microsoft Corporation, LCA - International Patents, One Microsoft Way, Redmond, Washington 98052-6399 (US). **OTRYSHKO, Volodymyr V.**; c/o Microsoft Corporation, LCA - International Patents, 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, CL, 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, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, 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, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ,

[Continued on next page]

(54) Title: PLATFORM INDEPENDENT PRESENTATION COMPOSITION

(57) Abstract: Architecture that includes a platform independent, configuration driven, presentation composition engine. The composition engine that allows dynamic generation of multiplatform user experience (UX) based on a data contract. By composition, the user can select the parts, interactions, and constraints between the interaction and parts, as well as the placement with respect to each other. The UX is dynamically composed from components that are targeted to particular data classes. At runtime, platform dependent component implementations are automatically selected by the engine based on the execution platform of the composition host. A user can create or customize the UX without writing code by composing from a wide variety of presentation widgets that access a wide variety of data sources that can work on many platforms. Compositions are targeted to both a data class and presentation type and can be either predefined or generated.

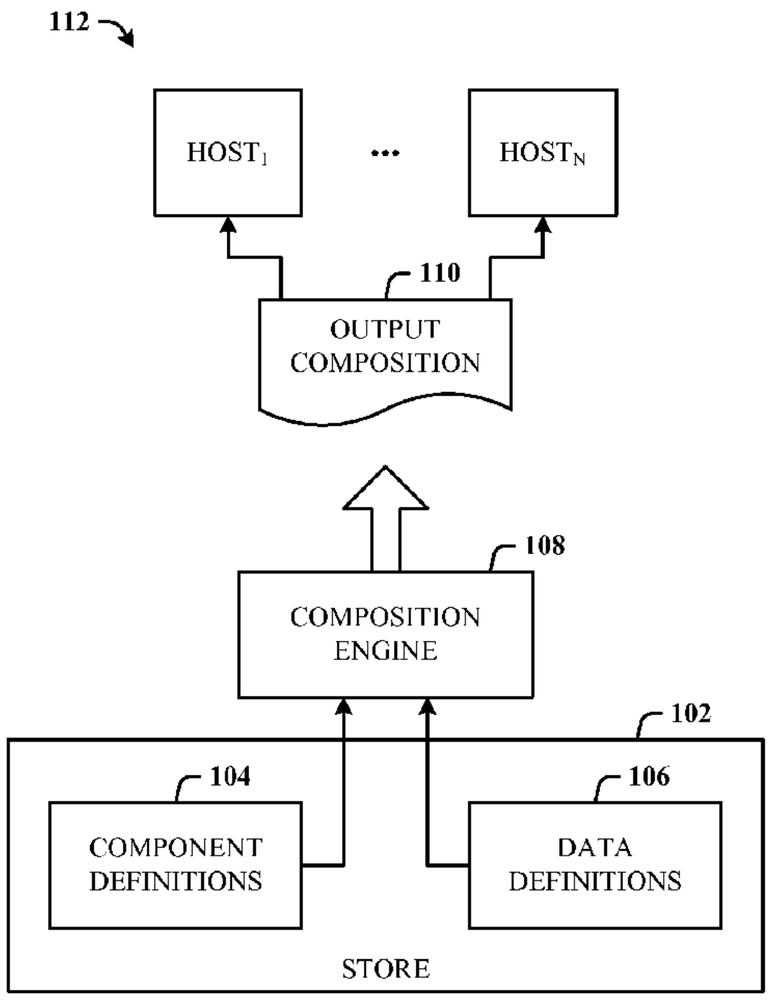

FIG. 1

WO 2011/129989 A3

WO ٢٠١١/١٢٩٩٨٩ A٣ ٢٠١١١٢٩٩٨٩٣

TM), European (AL, 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, RS, SE, SI, SK, SM, 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:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(88) Date of publication of the international search report:
19 January 2012

PLATFORM INDEPENDENT PRESENTATION COMPOSITION

BACKGROUND

[0001] The quality of a user experience (UX) is based on how well the UX is aligned with the user expectations. Having to deal with many data types, many data sources, and many UX platforms, designers have to make a choice from unattractive approaches that include writing presentation code for a specific persona that consumes specific data from data sources for a specific UX platform, or providing a broadly targeted UX that does not meet the needs of any single persona.

[0002] For example, existing UX composition systems such as HTML (hypertext markup language), XAML (extensible application markup language), and XSLT (extensible stylesheet language transformations) are designed such that the markup code be developed for a specific platform. If the developer wants the code to work on several platforms a custom logic is to be built in the code to handle the platform differences. Moreover, existing UX composition systems require specific presentation be explicitly defined for every data interface element. The functionality that allows dynamic generation of UX elements based on underlying data structures the elements represent is limited to nonexistent, especially if the data structures are complex and/or inheritable.

[0003] As a result of these limitations, the mass market (e.g., email) may have been served, but the smaller communities of users (e.g., the exchange administrator or the CRM (customer relationship management) service owner) are underserved.

SUMMARY

[0004] The following presents a simplified summary in order to provide a basic understanding of some novel embodiments described herein. This summary is not an extensive overview, and it is not intended to identify key/critical elements or to delineate the scope thereof. Its sole purpose is to present some concepts in a simplified form as a prelude to the more detailed description that is presented later.

[0005] The disclosed architecture includes a platform independent configuration-driven presentation composition engine. The composition engine allows dynamic generation of multiplatform user experience (UX) based on a data contract. By composition, the user can select the parts, interactions, and constraints between the interaction and parts, as well as the placement with respect to each other.

[0006] The UX is dynamically composed from components that are targeted to particular data classes. At runtime, platform dependent component implementations are

automatically selected by the engine based on the execution platform of the composition host.

[0007] The disclosed architecture allows a user to create or customize a UX without writing code by composing from multiple presentation widgets that can access many data sources that work on many platforms. Compositions are targeted to both a data class and presentation type and can be either predefined or generated.

[0008] To the accomplishment of the foregoing and related ends, certain illustrative aspects are described herein in connection with the following description and the annexed drawings. These aspects are indicative of the various ways in which the principles disclosed herein can be practiced and all aspects and equivalents thereof are intended to be within the scope of the claimed subject matter. Other advantages and novel features will become apparent from the following detailed description when considered in conjunction with the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] FIG. 1 illustrates a visualization system in accordance with the disclosed architecture.

[0010] FIG. 2 illustrates an alternative visualization system in accordance with the disclosed architecture.

[0011] FIG. 3 illustrates an exemplary composition composed by the composition engine.

[0012] FIG. 4 illustrates a parent component that includes data context and a visual base component of a composition system.

[0013] FIG. 5 illustrates a component definition.

[0014] FIG. 6 illustrates a component registry for finding or selecting components.

[0015] FIG. 7 illustrates a declarative diagram that represents the use of variables in the composition engine.

[0016] FIG. 8 illustrates a visualization method in accordance with the disclosed architecture.

[0017] FIG. 9 illustrates further aspects of the method of FIG. 8.

[0018] FIG. 10 illustrates an alternative visualization method.

[0019] FIG. 11 illustrates further aspects of the method of FIG. 10.

[0020] FIG. 12 illustrates a method of obtaining components in the composition engine.

[0021] FIG. 13 illustrates a more detailed method of obtaining components in the composition engine.

[0022] FIG. 14 illustrates a block diagram of a computing system that executes composition in accordance with the disclosed architecture.

DETAILED DESCRIPTION

[0023] The disclosed architecture is a presentation composition engine. The composition engine is a generic composition framework which is exposed as a set of services that allows the user to "glue" together (compose) different components, and the composition (output composition of the engine) of the component(s). By composition, the user can select the parts, interactions, and constraints between the interaction and parts, as well as the placement of the parts with respect to each other. The engine is a presentation-neutral framework for both UI (user interface) and non-UI components.

[0024] A component is the smallest reusable building block of UI declaration for the composition engine, and that is identifiable by name, and optionally, targeted to a data type. A component can be a base component (unit component) or a container component (composite component). Data context is an instance of target data for a component. In other words, the data context is a name/value pair set that represents data associated with a component. Data context entries support change notifications, and compositions can initiate changes and/or listen to the changes initiated by other compositions. The composition engine assembles components for a particular host as a user experience that is platform independent. The virtualization host is the execution environment of the composition for a particular platform (runtime).

[0025] Compositions are targeted to both a data class and presentation type and can be either predefined or generated. While there can be multiple components composed, the component chain ends at the concrete base component (e.g., a TextBox control, database query component, etc.).

[0026] The composition engine allows the dynamic generation of multiplatform UX (user experience) based on a data contract. The UX is dynamically composed from components that are targeted to a particular data classes. At runtime, platform dependent component implementations are automatically selected by the engine based on the execution platform of the composition host.

[0027] Reference is now made to the drawings, wherein like reference numerals are used to refer to like elements throughout. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding thereof. It may be evident, however, that the novel embodiments can be practiced without these specific details. In other instances, well known structures and

devices are shown in block diagram form in order to facilitate a description thereof. The intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the claimed subject matter.

[0028] FIG. 1 illustrates a visualization system 100 in accordance with the disclosed architecture. The system 100 includes a store 102 of store definitions that include component definitions 104 and data definitions 106 for components and data associated with a user experience. The component definitions 104 can include definitions for base component, container components, and compositions of base and container components. In this way, an existing composition of the components is readily available for dynamic selection and composition into the output component 110.

[0029] A composition engine 108 that automatically and declaratively composes an instance of an output component 110 based on a store definition. The output component is specific to a user experience of a visualization host of different hosts 112.

[0030] The output component 110 includes a base component, a container component, or a combination of base and container components. The output component 110 is composed based on a target data type of the user experience. The system 100 can further comprise a component registry via which a component is searched based on the target data type. The output component binds associated component properties to data context elements to link child components. The composition engine 108 includes global variables that enable data exchange between output components in unrelated data contexts.

[0031] FIG. 2 illustrates an alternative visualization system 200 in accordance with the disclosed architecture. The system 200 includes the entities of the system 100 of FIG. 1, as well as data context 202, a personalization (private) override 204, and component implementation 206. The output component 110 can be composed based on the data context 202 rather than an existing component definition. That is, based on the data, a customized component can be created and output purely based on the context data 202 (instance of data in the target UX). The composition engine 108 employs the personalization override 204 that is composed with a selected component definition to override a global variable with a private variable.

[0032] FIG. 3 illustrates an exemplary composition 300 composed by the composition engine. Here, the composition 300 is described in terms of based components (e.g., a StackPanel base component 302) and a container component 304. Here, the base component 302 includes two text box base components: a first text box base component

showing text "ABC" and a second text box base component showing text "DEF". The base component 302 also includes a button base component.

[0033] The base component is a concrete implementation targeted to specific platforms, the leaf node of the composition process, and can be visual or non-visual. Following is an example of a base component definition (in terms of component type rather than component type).

```
<ComponentType ID="EventQuery">
  <Parameters>
    <Parameter Name="Scope" Type="String"  />
    <Parameter Name="Output" Type="IEnumerable" />
   </Parameters>
</ComponentType>
<ComponentImplementation TypeId="EventQuery">
  <SupportedPlatforms>
    <Platform>WPF</Platform>
  </SupportedPlatforms>
  <Unit>
    <MefFactory>

<ContractName>Company.EnterpriseManagement.EventQuery</ContractName>
    </MefFactory>
    <Properties>
      <Property Name="QueryVerb" Direction="In">Events</Property>
      <Property Name="Scope"
Direction="In">$Parameter/Scope$</Property>
      <Property Name="Output"
Direction="Out">$Parameter/Output$</Property>
    </Properties>
  </Unit>
</ComponentImplementation>
```

The MEF (managed extensibility framework) factory calls an MEF runtime to pull corresponding types from registered UI assemblies. Note that MEF is just one way example implementation; other Factory implementations can be employed as well.

[0034] The container component (also, composite component) is the container for the base components (also, unit components), does not have a custom implementation, and is platform independent. Following is an example of a composite component definition.

```
<ComponentImplementation TypeId="SampleComponent">
  <SupportedPlatforms>
    <Platform>All</Platform>
</SupportedPlatforms>
<Composite>
  <Variables>
    <Variable Id="abc" Type="String" />
  </Variables>
  <Component TypeId="SampleContainerComponent">
    <Parameter Id="Child1">
      <Component TypeId="SampleComponent1">
        <Parameter Id="Bla">$Variable/abc$</Parameter>
      </Component>
    </Parameter>
```

```
                <Parameter Id="Child2">
                  <Component TypeId="SampleComponent2">
                    <Parameter Id="Bla">$Variable/abc$</Parameter>
                  </Component>
                </Parameter>
              </Component>
            </Composite>
          </ComponentImplementation>
```

[0035] In FIG. 3, the container component 304 combines the base components, as illustrated in the following code, where PropertyA is “ABC” and PropertyB is “DEF”.

```
<Component TypeId="StackPanel">
   <Parameter Name="Child">
<Component TypeId="Edit">
          <Target>$Target/propertyA$</Target>
</Component>
<Component TypeId="Edit">
          <Target>$Target/propertyB$</Target>
</Component>
<Component TypeId="Button">
</Component>
   </Parameter>
```

[0036] Each component (base or container) is backed with a data context (Data Context) instance. Data Context is a key(string)-value(object) pair collection which supports property changed notification and error set notification. The component can bind its properties to data context elements to link child components together.

[0037] FIG. 4 illustrates a parent component 400 that includes data context 402(as DataContext elements) and a visual base component 404 (e.g., StackPanel base component 302 of FIG. 3) of a composition system. Here, the data context 402 includes three properties: PropertyA, PropertyB and PropertyC, with corresponding values “ABC”, “DEF”, and “XYZ”. The visual base component 404 includes bindings to PropertyA and PropertyB via corresponding 2-way databinds, while multiple data components 406 of a view model 408 are bound to PropertyB and PropertyC. In other words, the parent component 400 is composed that includes bindings of properties to child components (the text box base components) and bindings of the properties to the data components 406.

[0038] FIG. 5 illustrates an component definition 500. The component definition 500 comprises two parts: a type declaration 502 and an implementation definition 504. Each component has a name, and optional target type attributes that can be used to look up a relevant component. An example type declaration (in component terms) is the following:

```
  <ComponentType ID="SampleComponent" Target="String"
Accessibility="Internal">
```

[0039] A component can also include a Parameters subnode which defines the datashape (e.g., string) expected to be passed, as illustrated below (in component terms):

```
<ComponentType ID="AnotherComposition">
  <Parameters>
    <Parameter Name="Parameter1" Type="String" />
    <Parameter Name="SelectedText" Type="String" BindingDirection="Both"
/>
  </Parameters>
```

[0040] FIG. 6 illustrates a component registry 600 for finding or selecting components. The registry 600 maintains a list of all defined components and component compositions. For example, the base component (e.g., StackPanel/Button/Edit of FIG. 3) can be looked up (searched) based on TypeId and TargetType (target data type). The output is then the base component that corresponds to the TargetType, or TypeId and TargetType.

[0041] To set a property on an component, a "Parameter" node is used, as shown in the following sample code:

```
<Component Id="EventView">
<Parameter Id="Scope">Microsoft.SystemCenter.SqlDB </Parameter>
```

[0042] In this case, the property "Scope" of component "EventView" is set to text "Company.SystemCenter.SqlDB". Oftentimes, however, parameters are not static, but are bound to other elements. In general, reference is in the form `$<protocol>/<protocol-specific string>`.

[0043] For example, two components are bound to a variable "abc":

```
  <ComponentImplementation TypeId="SampleComponent">
      <SupportedPlatforms>
        <Platform>All</Platform>
      </SupportedPlatforms>
      <Composite>
        <Variables>
          <Variable Id="abc" Type="String" />
        </Variables>
        <Component TypeId="SampleComponent1">
          <Parameter Id="A">$Variable/abc$</Parameter>
        </Component>
        <Component TypeId="SampleComponent2">
          <Parameter Id="A">$Variable/abc$</Parameter>
        </Component>
      </Composite>
    </ComponentImplementation>
```

[0044] Following is an example list of reference protocols in a Parameter node:

```
$Parameter/<propertyName>$ Parameter passed to component
$Variable/<propertyName>$  Variable declared in component
$Target/<propertyName>$    Property of a target instance passed to
component
$Target$                   Target instance
```

[0045] Global variables can be used to enable data exchange between compositions (base components and/or container components) in non-related data contexts. First, a global variable is declared:

```
<GlobalVariable ID="GlobalSelectedItem" Type="String"/>
```

The variable can be referenced in base component and container components using `$GlobalVariable/<variable name>$`.

[0046] Any given component (e.g., base, container) can override the variable with a private implementation so that component children see a private copy of the variable, illustrated in the following example code:

```
<Variables>
    <GlobalVariableOverride GlobalVariableId="GlobalSelectedItem" />
  </Variables>
```

[0047] FIG. 7 illustrates a declarative diagram 700 that represents the use of variables in the composition engine. At 702, a global variable "A" is declared. At 704, a parameter name "Blah" is passed to an component. At 706, a local copy of the global variable is created. At 708 and 710, local copies of the variable are used rather than the global variable.

[0048] Included herein is a set of flow charts representative of exemplary methodologies for performing novel aspects of the disclosed architecture. While, for purposes of simplicity of explanation, the one or more methodologies shown herein, for example, in the form of a flow chart or flow diagram, are shown and described as a series of acts, it is to be understood and appreciated that the methodologies are not limited by the order of acts, as some acts may, in accordance therewith, occur in a different order and/or concurrently with other acts from that shown and described herein. For example, those skilled in the art will understand and appreciate that a methodology could alternatively be represented as a series of interrelated states or events, such as in a state diagram. Moreover, not all acts illustrated in a methodology may be required for a novel implementation.

[0049] FIG. 8 illustrates a visualization method in accordance with the disclosed architecture. At 800, a request is received for a component to be employed in an execution environment. At 802, a component definition associated with the component is searched. At 804, one or more data definitions are selected for a found component definition. At 806, the one or more data definitions are automatically composed with the

component definition to output the component in the execution environment at environment runtime.

[0050] FIG. 9 illustrates further aspects of the method of FIG. 8. At 900, the component definition is searched based on a data type of the requested component when the component definition is not found. At 902, a custom component is created based on absence of the component requested. At 904, a global variable is applied to the component to enable data exchange between unrelated data contexts. At 906, a global variable is overridden with a private variable to impose the private variable on child components of the component. At 908, a container component is created when the requested component is not found. At 910, the container component is loaded with base components associate data type properties. At 912, the container component is output as the component.

[0051] FIG. 10 illustrates an alternative visualization method. At 1000, a request for a component based is received on a component execution environment. At 1002, a component definition associated with the component is searched. At 1004, one or more data definitions for the component definition are selected if the component definition is found. At 1006, a custom component is created based on a data type related to the requested component when the component definition is not found. At 1008, a global variable is applied to the component or custom to enable data exchange between unrelated data contexts. At 1010, the one or more data definitions are automatically composed with the component definition to output the component in the execution environment at environment runtime.

[0052] FIG. 11 illustrates further aspects of the method of FIG. 10. At 1100, the global variable is overridden with a private variable to impose the private variable on child components of the component. At 1102, a container component is created when the requested component is not found. At 1104, the container component is loaded with base components associate data type properties. At 1106, the container component is output as the component. At 1108, data to be passed to the component is defined via a parameter node. At 1110, a parent component is composed that includes bindings of properties to child components and bindings of the properties to data components.

[0053] FIG. 12 illustrates a method of obtaining components in the composition engine. At 1200, a component is obtained (e.g., via a “get component” call). At 1202, the target data type is obtained. At 1204, a check is made to determine of a component exists for the target data type. If not, flow is to 1206 to create a component container. The component

can have one or more associated data properties. At 1208, property types are obtained for the component. At 1210, a call "get component" is made for the property types. At 1212, the component is added to the container. Flow is back to 1208 to continue until completed. After all data properties and types have been applied to the container, flow is to 1214 to return the results. At 1204, if the check determines that a component exists for the target data type, flow is to 1216 to select the component, and then return the results, at 1216.

[0054] FIG. 13 illustrates a more detailed method of obtaining components in the composition engine. At 1300, function parameters such as target type and data type are received. At 1302, a check is made for a component defined for the target type and name. If so, flow is to 1304 to verify the interface. Alternatively, if no component exists for the target type and names, flow is to 1306 such that for every property using the type system lookup, is a component defined for the data type. If yes, flow is to 1304 to verify the interface. Access to the type system 1308 is provided to make the check and verify the interface. Once the interface is verified, flow is to 1310 to check for the component type. If a unit component, flow is to 1312 to create an instance of the unit component for the correct platform and pass all declared parameters to the unit component. At 1314, the loader can load an assembly for the UX composition system (e.g., XAML, MEF, etc).

[0055] If the component type, at 1310, is a composite component, flow is to 1316 to walk the child nodes in a configuration (e.g., written in XML) setting the values on the components of the composition. This includes receiving parameter values from parameter node(s) 1318, parameters from component node(s) 1320, and a parameter set from child nodes, as provided from build data from parameters at 1322. The typeID is sent from the component node 1320 to build data 1322. At 1324, the component referenced by the target as data and name, as the name for lookup, based on name and target information received from the component node 1320.

[0056] One or more components can reside within a process and/or thread of execution, and a component can be localized on one computer and/or distributed between two or more computers. The word "exemplary" may be used herein to mean serving as an example, instance, or illustration. Any aspect or design described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other aspects or designs.

[0057] Referring now to FIG. 14, there is illustrated a block diagram of a computing system 1400 that executes composition in accordance with the disclosed architecture. In

order to provide additional context for various aspects thereof, FIG. 14 and the following description are intended to provide a brief, general description of the suitable computing system 1400 in which the various aspects can be implemented. While the description above is in the general context of computer-executable instructions that can run on one or more computers, those skilled in the art will recognize that a novel embodiment also can be implemented in combination with other program modules and/or as a combination of hardware and software.

[0058] The computing system 1400 for implementing various aspects includes the computer 1402 having processing unit(s) 1404, a computer-readable storage such as a system memory 1406, and a system bus 1408. The processing unit(s) 1404 can be any of various commercially available processors such as single-processor, multi-processor, single-core units and multi-core units. Moreover, those skilled in the art will appreciate that the novel methods can be practiced with other computer system configurations, including minicomputers, mainframe computers, as well as personal computers (e.g., desktop, laptop, etc.), hand-held computing devices, microprocessor-based or programmable consumer electronics, and the like, each of which can be operatively coupled to one or more associated devices.

[0059] The system memory 1406 can include computer-readable storage (physical storage media) such as a volatile (VOL) memory 1410 (e.g., random access memory (RAM)) and non-volatile memory (NON-VOL) 1412 (e.g., ROM, EPROM, EEPROM, etc.). A basic input/output system (BIOS) can be stored in the non-volatile memory 1412, and includes the basic routines that facilitate the communication of data and signals between components within the computer 1402, such as during startup. The volatile memory 1410 can also include a high-speed RAM such as static RAM for caching data.

[0060] The system bus 1408 provides an interface for system components including, but not limited to, the system memory 1406 to the processing unit(s) 1404. The system bus 1408 can be any of several types of bus structure that can further interconnect to a memory bus (with or without a memory controller), and a peripheral bus (e.g., PCI, PCIe, AGP, LPC, etc.), using any of a variety of commercially available bus architectures.

[0061] The computer 1402 further includes machine readable storage subsystem(s) 1414 and storage interface(s) 1416 for interfacing the storage subsystem(s) 1414 to the system bus 1408 and other desired computer components. The storage subsystem(s) 1414 (physical storage media) can include one or more of a hard disk drive (HDD), a magnetic floppy disk drive (FDD), and/or optical disk storage drive (e.g., a CD-ROM drive DVD

drive), for example. The storage interface(s) 1416 can include interface technologies such as EIDE, ATA, SATA, and IEEE 1394, for example.

[0062] One or more programs and data can be stored in the memory subsystem 1406, a machine readable and removable memory subsystem 1418 (e.g., flash drive form factor technology), and/or the storage subsystem(s) 1414 (e.g., optical, magnetic, solid state), including an operating system 1420, one or more application programs 1422, other program modules 1424, and program data 1426.

[0063] The one or more application programs 1422, other program modules 1424, and program data 1426 can include the entities and components of the system 100 of FIG. 1, the entities and components of the system 200 of FIG. 2, the composition 300 of FIG. 3, the parent component 400 of FIG. 4, the component definition 500 of FIG. 5, the registry 600 of FIG. 6, the diagram 700 of FIG. 7, and the methods represented by the flowcharts of Figures 8-13, for example.

[0064] Generally, programs include routines, methods, data structures, other software components, etc., that perform particular tasks or implement particular abstract data types. All or portions of the operating system 1420, applications 1422, modules 1424, and/or data 1426 can also be cached in memory such as the volatile memory 1410, for example. It is to be appreciated that the disclosed architecture can be implemented with various commercially available operating systems or combinations of operating systems (e.g., as virtual machines).

[0065] The storage subsystem(s) 1414 and memory subsystems (1406 and 1418) serve as computer readable media for volatile and non-volatile storage of data, data structures, computer-executable instructions, and so forth. Such instructions, when executed by a computer or other machine, can cause the computer or other machine to perform one or more acts of a method. The instructions to perform the acts can be stored on one medium, or could be stored across multiple media, so that the instructions appear collectively on the one or more computer-readable storage media, regardless of whether all of the instructions are on the same media.

[0066] Computer readable media can be any available media that can be accessed by the computer 1402 and includes volatile and non-volatile internal and/or external media that is removable or non-removable. For the computer 1402, the media accommodate the storage of data in any suitable digital format. It should be appreciated by those skilled in the art that other types of computer readable media can be employed such as zip drives, magnetic

tape, flash memory cards, flash drives, cartridges, and the like, for storing computer executable instructions for performing the novel methods of the disclosed architecture.

[0067] A user can interact with the computer 1402, programs, and data using external user input devices 1428 such as a keyboard and a mouse. Other external user input devices 1428 can include a microphone, an IR (infrared) remote control, a joystick, a game pad, camera recognition systems, a stylus pen, touch screen, gesture systems (e.g., eye movement, head movement, etc.), and/or the like. The user can interact with the computer 1402, programs, and data using onboard user input devices 1430 such a touchpad, microphone, keyboard, etc., where the computer 1402 is a portable computer, for example. These and other input devices are connected to the processing unit(s) 1404 through input/output (I/O) device interface(s) 1432 via the system bus 1408, but can be connected by other interfaces such as a parallel port, IEEE 1394 serial port, a game port, a USB port, an IR interface, etc. The I/O device interface(s) 1432 also facilitate the use of output peripherals 1434 such as printers, audio devices, camera devices, and so on, such as a sound card and/or onboard audio processing capability.

[0068] One or more graphics interface(s) 1436 (also commonly referred to as a graphics processing unit (GPU)) provide graphics and video signals between the computer 1402 and external display(s) 1438 (e.g., LCD, plasma) and/or onboard displays 1440 (e.g., for portable computer). The graphics interface(s) 1436 can also be manufactured as part of the computer system board.

[0069] The computer 1402 can operate in a networked environment (e.g., IP-based) using logical connections via a wired/wireless communications subsystem 1442 to one or more networks and/or other computers. The other computers can include workstations, servers, routers, personal computers, microprocessor-based entertainment appliances, peer devices or other common network nodes, and typically include many or all of the elements described relative to the computer 1402. The logical connections can include wired/wireless connectivity to a local area network (LAN), a wide area network (WAN), hotspot, and so on. LAN and WAN networking environments are commonplace in offices and companies and facilitate enterprise-wide computer networks, such as intranets, all of which may connect to a global communications network such as the Internet.

[0070] When used in a networking environment the computer 1402 connects to the network via a wired/wireless communication subsystem 1442 (e.g., a network interface adapter, onboard transceiver subsystem, etc.) to communicate with wired/wireless networks, wired/wireless printers, wired/wireless input devices 1444, and so on. The

computer 1402 can include a modem or other means for establishing communications over the network. In a networked environment, programs and data relative to the computer 1402 can be stored in the remote memory/storage device, as is associated with a distributed system. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers can be used.

[0071] The computer 1402 is operable to communicate with wired/wireless devices or entities using the radio technologies such as the IEEE 802.xx family of standards, such as wireless devices operatively disposed in wireless communication (e.g., IEEE 802.11 over-the-air modulation techniques) with, for example, a printer, scanner, desktop and/or portable computer, personal digital assistant (PDA), communications satellite, any piece of equipment or location associated with a wirelessly detectable tag (e.g., a kiosk, news stand, restroom), and telephone. This includes at least Wi-Fi (or Wireless Fidelity) for hotspots, WiMax, and Bluetooth™ wireless technologies. Thus, the communications can be a predefined structure as with a conventional network or simply an ad hoc communication between at least two devices. Wi-Fi networks use radio technologies called IEEE 802.11x (a, b, g, etc.) to provide secure, reliable, fast wireless connectivity. A Wi-Fi network can be used to connect computers to each other, to the Internet, and to wire networks (which use IEEE 802.3-related media and functions).

[0072] The illustrated and described aspects can be practiced in distributed computing environments where certain tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules can be located in local and/or remote storage and/or memory system.

[0073] What has been described above includes examples of the disclosed architecture. It is, of course, not possible to describe every conceivable combination of components and/or methodologies, but one of ordinary skill in the art may recognize that many further combinations and permutations are possible. Accordingly, the novel architecture is intended to embrace all such alterations, modifications and variations that fall within the spirit and scope of the appended claims. Furthermore, to the extent that the term "includes" is used in either the detailed description or the claims, such term is intended to be inclusive in a manner similar to the term "comprising" as "comprising" is interpreted when employed as a transitional word in a claim.

CLAIMS

1. A computer-implemented visualization system having computer readable media that store executable instructions executed by a processor, comprising:

a store of store definitions that include component definitions and data definitions for components and data associated with a user experience; and

a composition engine that automatically and declaratively composes an instance of an output component based on a store definition, the output component specific to a user experience of a visualization host.

2. The system of claim 1, wherein the output component includes a base component, a container component, or a combination of base and container components.

3. The system of claim 1, wherein the output component is composed based on a target data type of the user experience.

4. The system of claim 3, further comprising a component registry via which an component is searched based on the target data type.

5. The system of claim 1, wherein the output component is composed based on a data context.

6. The system of claim 1, wherein the output component binds associated component properties to data context elements to link child components.

7. The system of claim 1, wherein the composition engine includes global variables that enable data exchange between output components in unrelated data contexts.

8. The system of claim 1, wherein the composition engine employs a personalization override that is composed with a selected component definition to override a global variable with a private variable.

9. A computer-implemented visualization method executable via a processor and memory, comprising:

receiving a request for an component to be employed in an execution environment;

searching for an component definition associated with the component;

selecting one or more data definitions for a found component definition; and

automatically composing the one or more data definitions with the component definition to output the component in the execution environment at environment runtime.

10. The method of claim 9, further comprising searching for the component definition based on a data type of the requested component when the component definition is not found.

11. The method of claim 9, further comprising creating a custom component based on absence of the component requested.

12. The method of claim 9, further comprising applying a global variable to the component to enable data exchange between unrelated data contexts.

13. The method of claim 9, further comprising overriding a global variable with a private variable to impose the private variable on child components of the component.

14. The method of claim 9, further comprising:

creating a container component when the requested component is not found;

loading the container component with base components associate data type properties; and

outputting the container component as the component.

15. The method of claim 9, further comprising composing a parent component that includes bindings of properties to child components and bindings of the properties to data components.

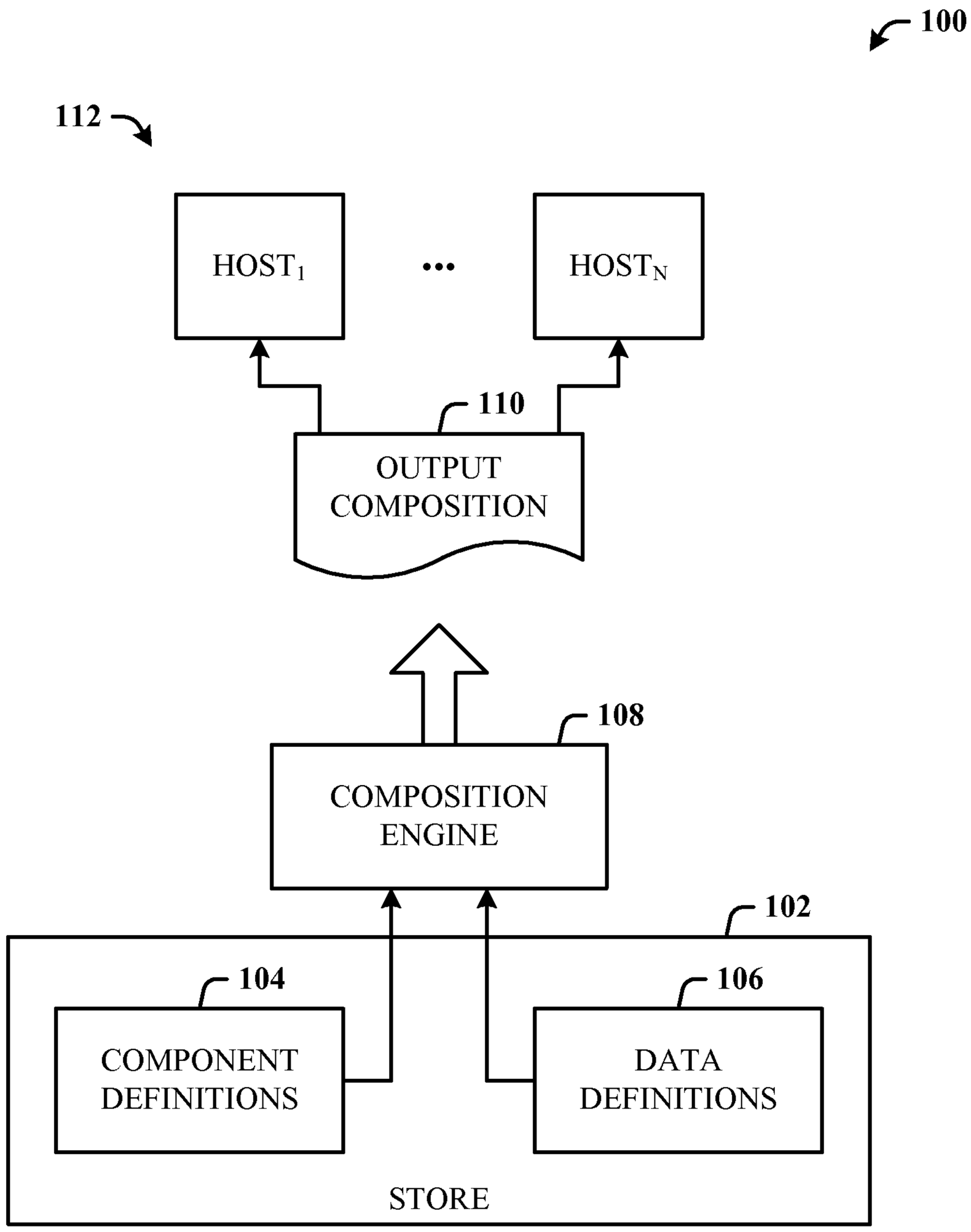
100
112
HOST1
HOSTN
110
OUTPUT COMPOSITION
108
COMPOSITION ENGINE
102
104
COMPONENT DEFINITIONS
106
DATA DEFINITIONS
STORE

FIG. 1

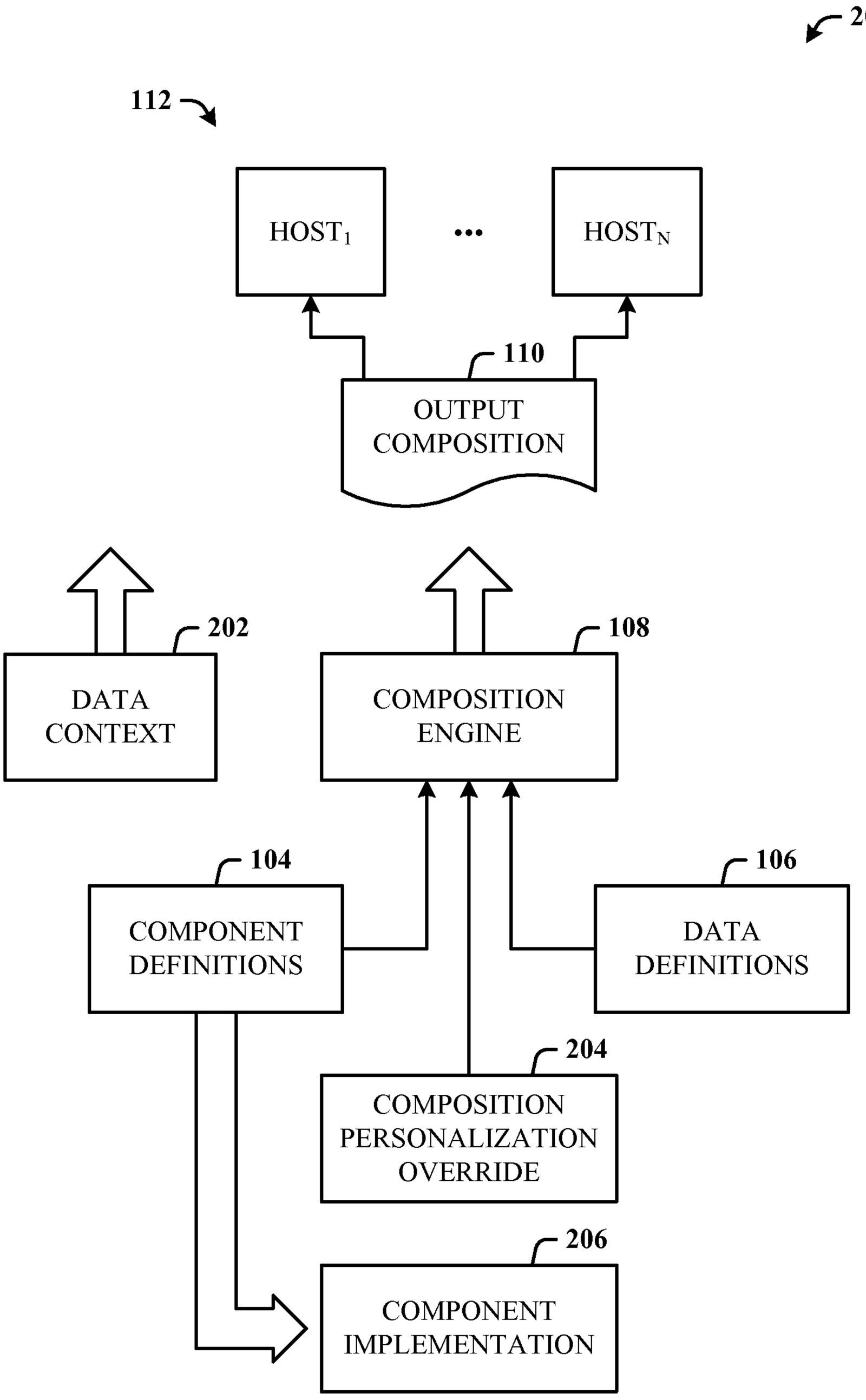
200
112
HOST1
HOSTN
110
OUTPUT COMPOSITION
202
DATA CONTEXT
108
COMPOSITION ENGINE
104
COMPONENT DEFINITIONS
106
DATA DEFINITIONS
204
COMPOSITION PERSONALIZATION OVERRIDE
206
COMPONENT IMPLEMENTATION

FIG. 2

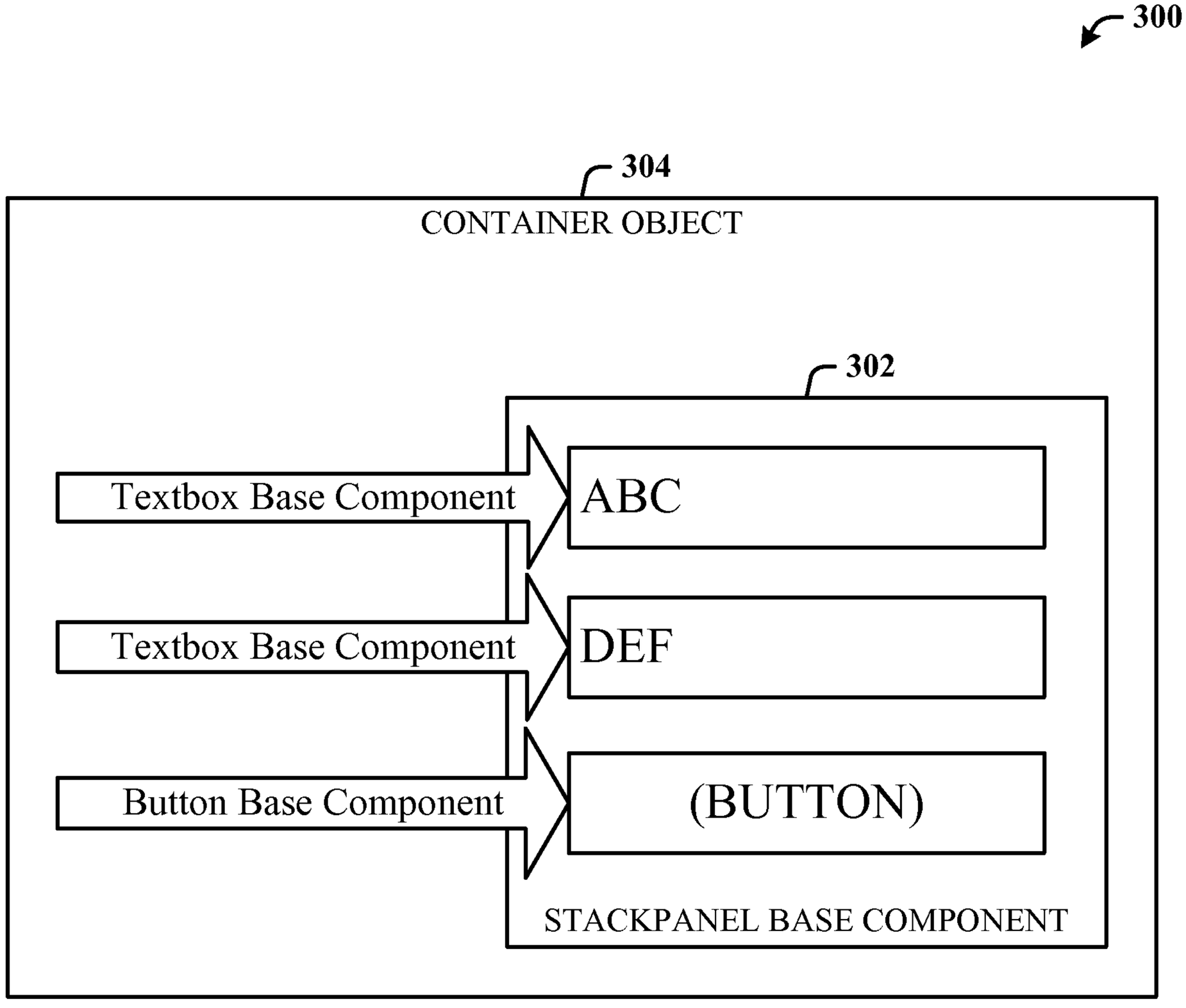
300
304
CONTAINER OBJECT
302
Textbox Base Component
ABC
Textbox Base Component
DEF
Button Base Component
(BUTTON)
STACKPANEL BASE COMPONENT

FIG. 3

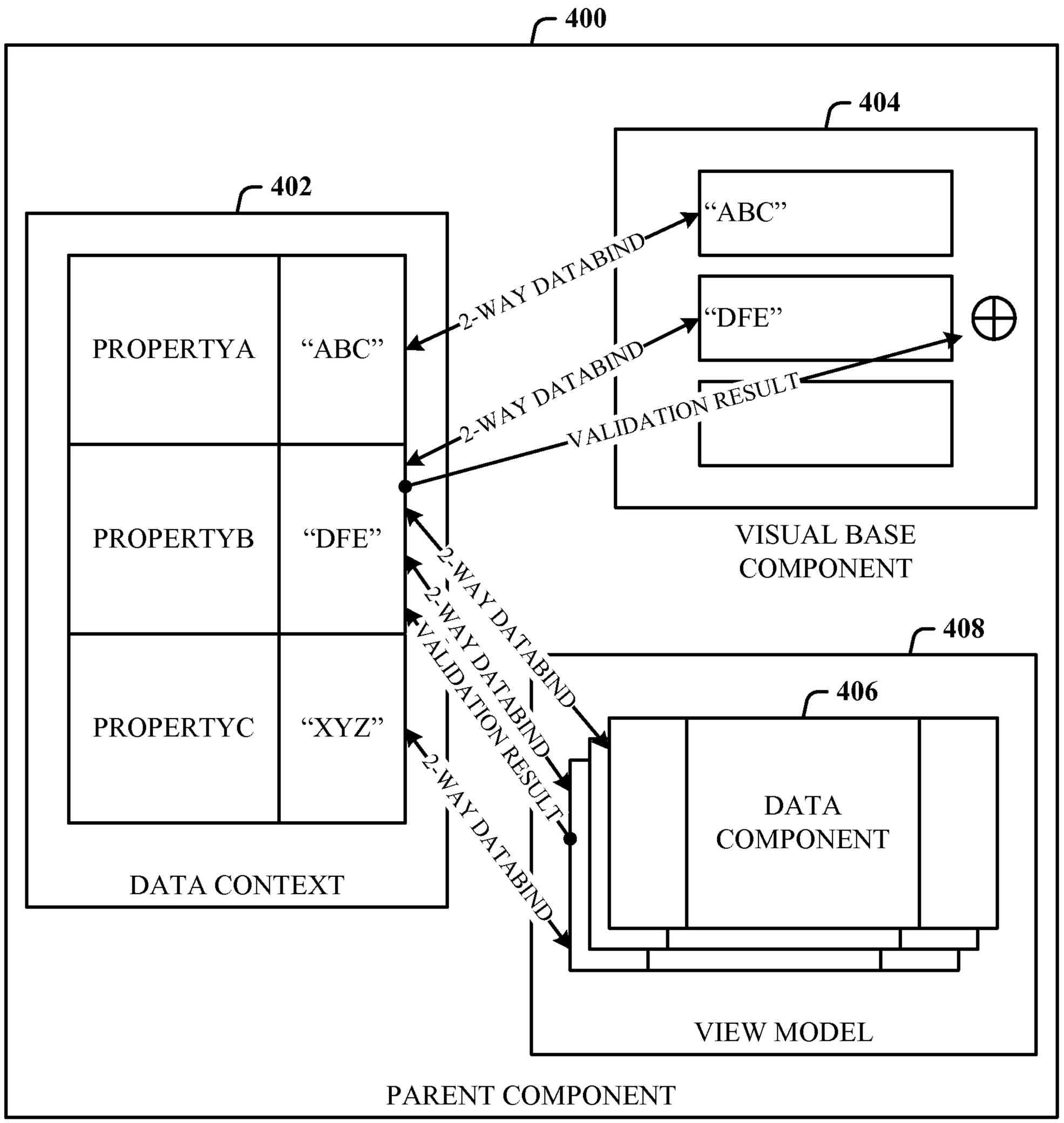
400
404
402
“ABC”
“DFE”
PROPERTYA
“ABC”
2-WAY DATABIND
2-WAY DATABIND
VALIDATION RESULT
PROPERTYB
“DFE”
VISUAL BASE COMPONENT
2-WAY DATABIND
2-WAY DATABIND
VALIDATION RESULT
2-WAY DATABIND
408
406
PROPERTYC
“XYZ”
DATA COMPONENT
DATA CONTEXT
VIEW MODEL
PARENT COMPONENT

FIG. 4

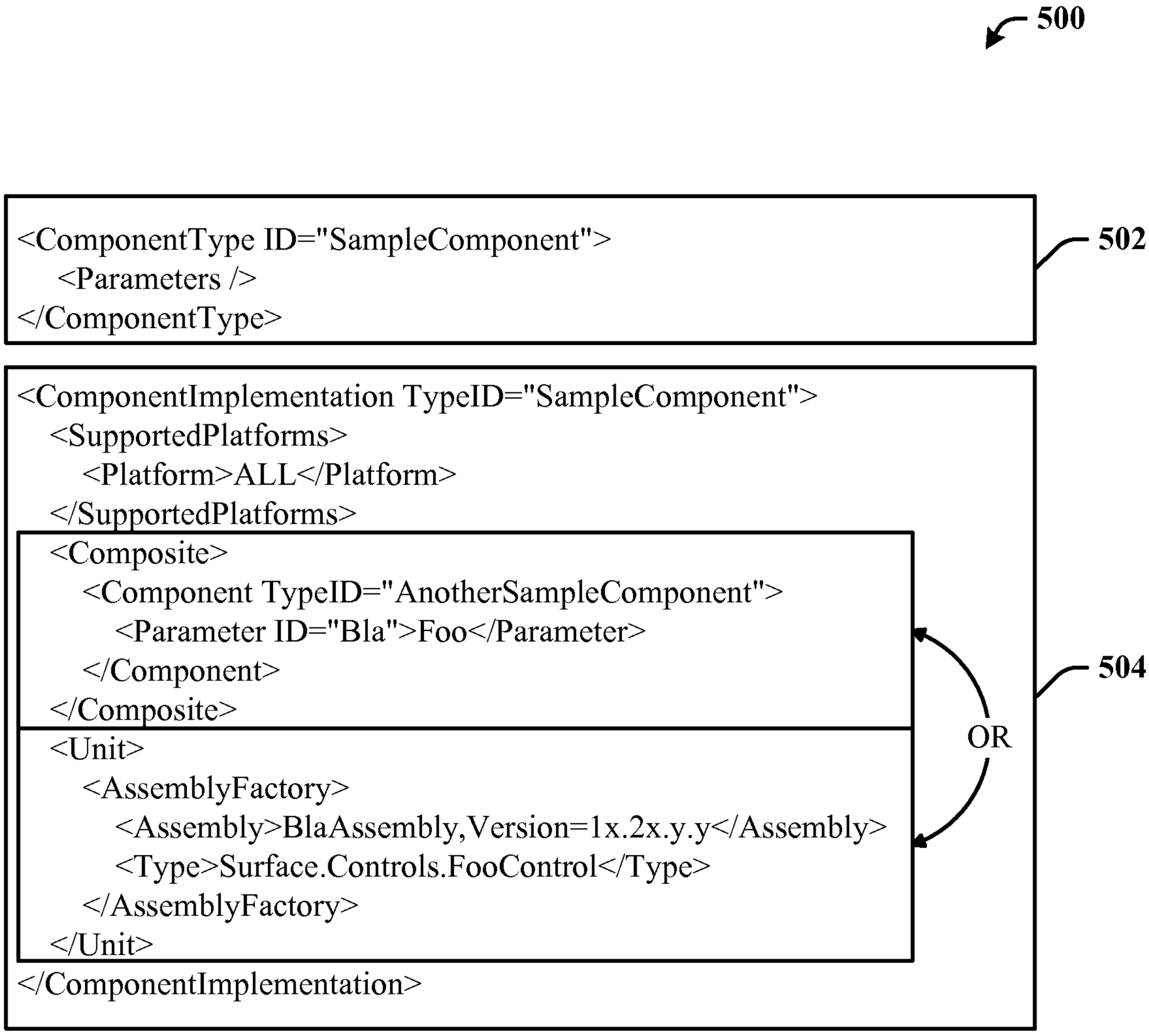
500
<ComponentType ID="SampleComponent">
<Parameters />
</ComponentType>
502
<ComponentImplementation TypeID="SampleComponent">
<SupportedPlatforms>
<Platform>ALL</Platform>
</SupportedPlatforms>
<Composite>
<Component TypeID="AnotherSampleComponent">
<Parameter ID="Bla">Foo</Parameter>
</Component>
</Composite>
OR
<Unit>
<AssemblyFactory>
<Assembly>BlaAssembly,Version=1x.2x.y.y</Assembly>
<Type>Surface.Controls.FooControl</Type>
</AssemblyFactory>
</Unit>
</ComponentImplementation>
504

FIG. 5

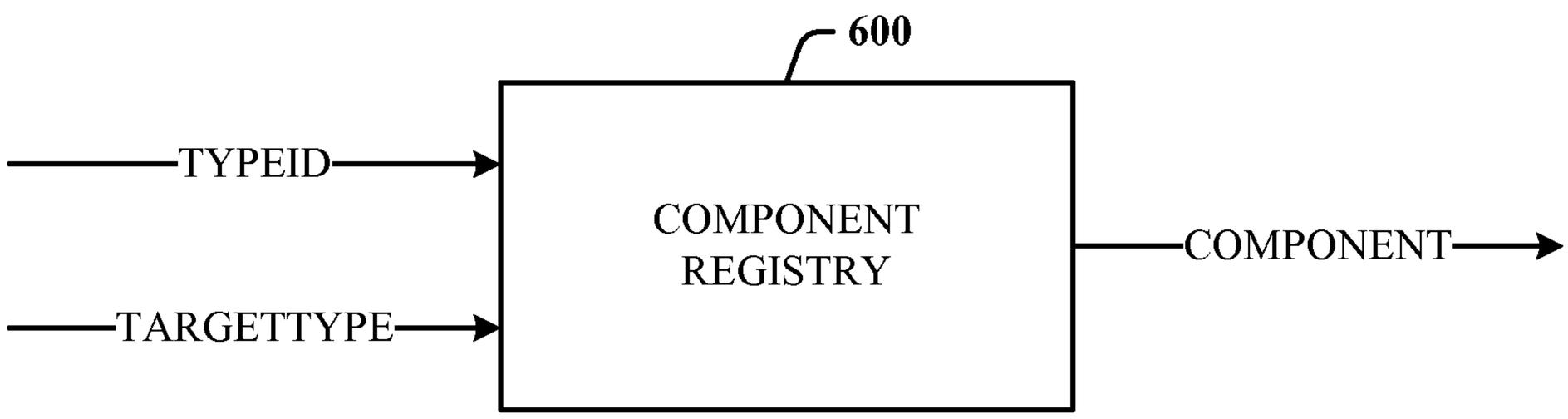

FIG. 6

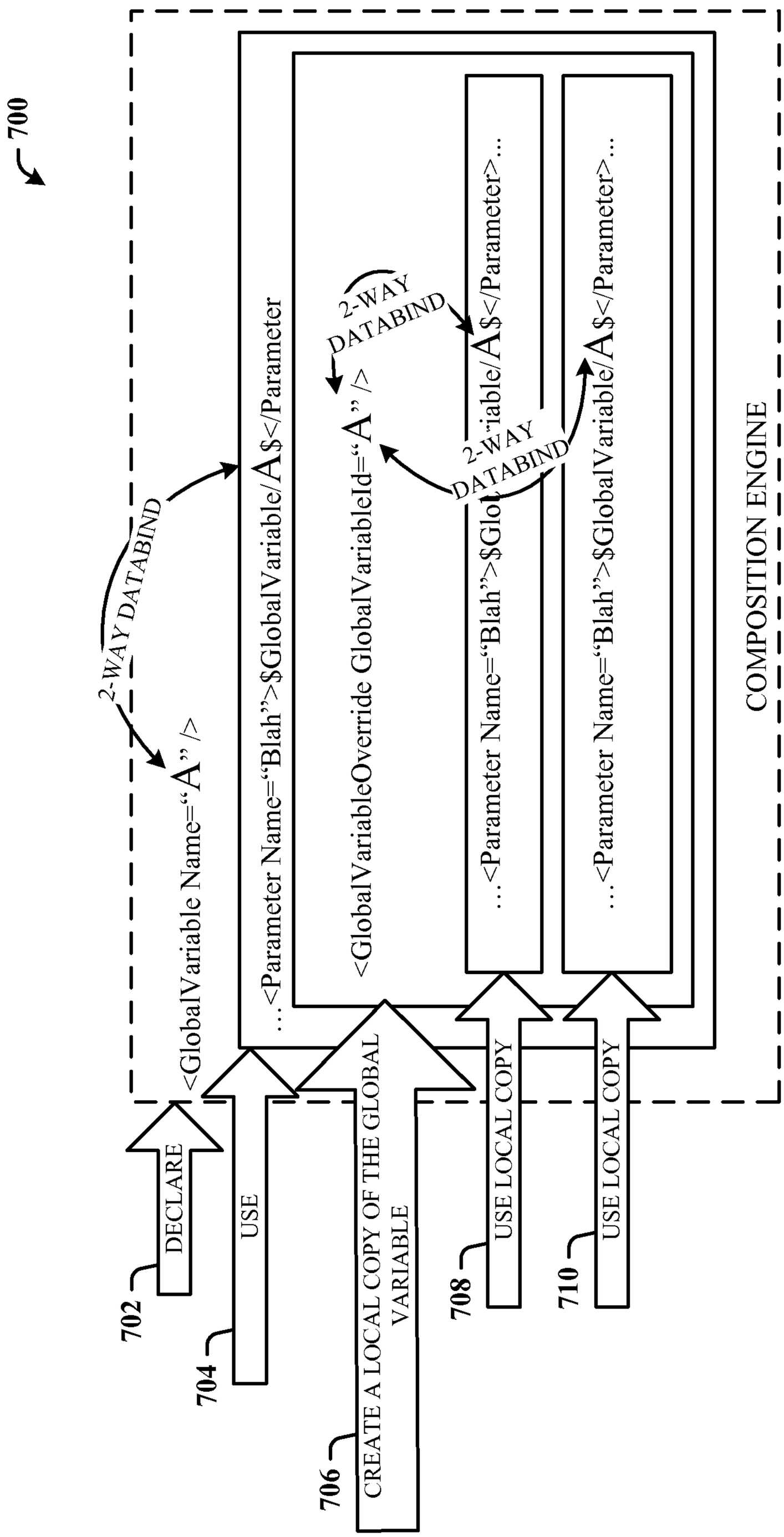
700
2-WAY DATABIND
<GlobalVariable Name="A" />
<GlobalVariable Name="Blah">$GlobalVariable/A$</Parameter
...<Parameter Name="Blah">$GlobalVariable/A$</Parameter
<GlobalVariableOverride GlobalVariableId="A" />
2-WAY DATABIND
2-WAY DATABIND
...<Parameter Name="Blah">$GlobalVariable/A$</Parameter
...<Parameter Name="Blah">$GlobalVariable/A$</Parameter
COMPOSITION ENGINE
DECLARE
USE
CREATE A LOCAL COPY OF THE GLOBAL VARIABLE
USE LOCAL COPY
USE LOCAL COPY
702
704
706
708
710

FIG. 7

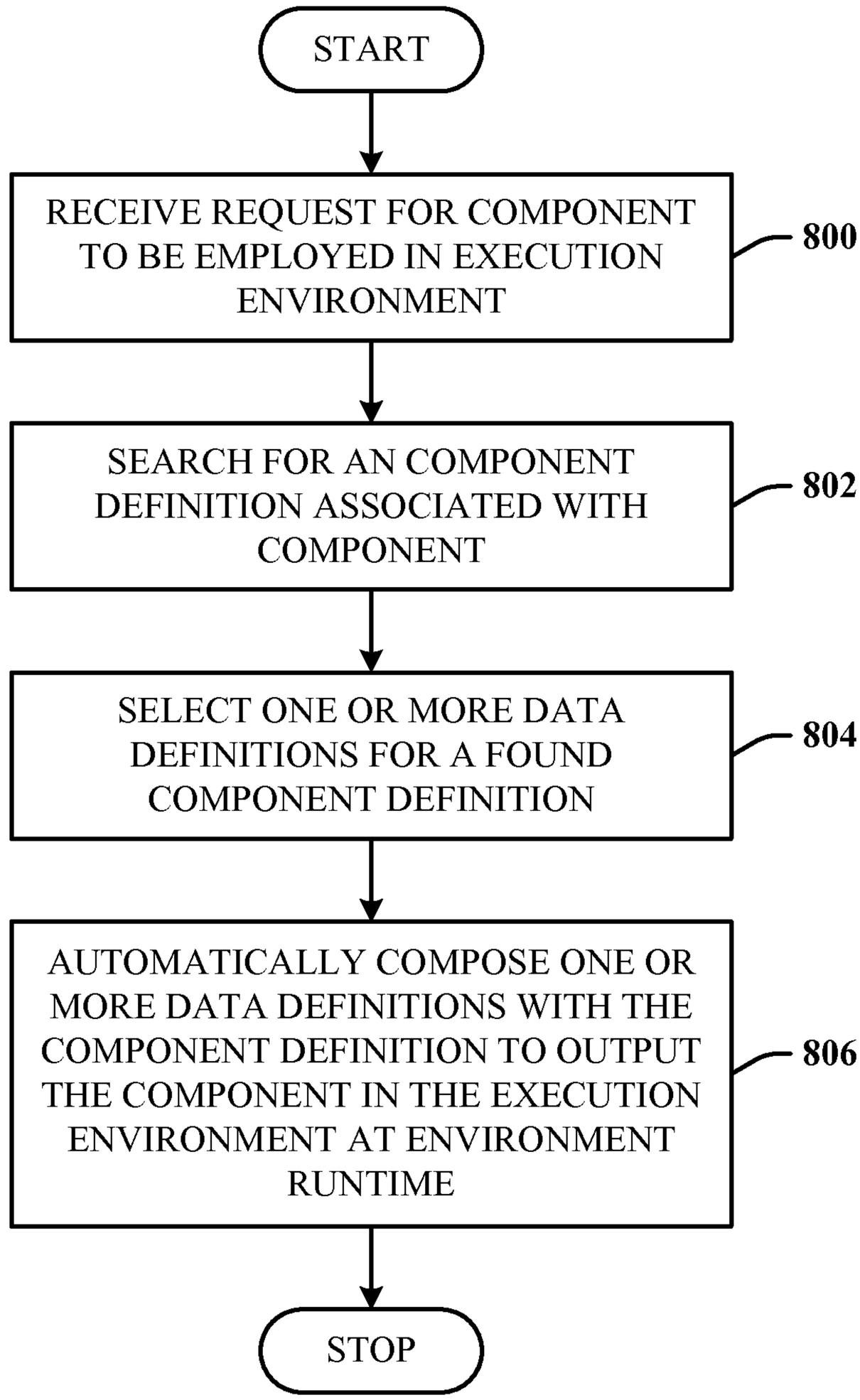
START
RECEIVE REQUEST FOR COMPONENT TO BE EMPLOYED IN EXECUTION ENVIRONMENT
800
SEARCH FOR AN COMPONENT DEFINITION ASSOCIATED WITH COMPONENT
802
SELECT ONE OR MORE DATA DEFINITIONS FOR A FOUND COMPONENT DEFINITION
804
AUTOMATICALLY COMPOSE ONE OR MORE DATA DEFINITIONS WITH THE COMPONENT DEFINITION TO OUTPUT THE COMPONENT IN THE EXECUTION ENVIRONMENT AT ENVIRONMENT RUNTIME
806
STOP

FIG. 8

FIG. 8

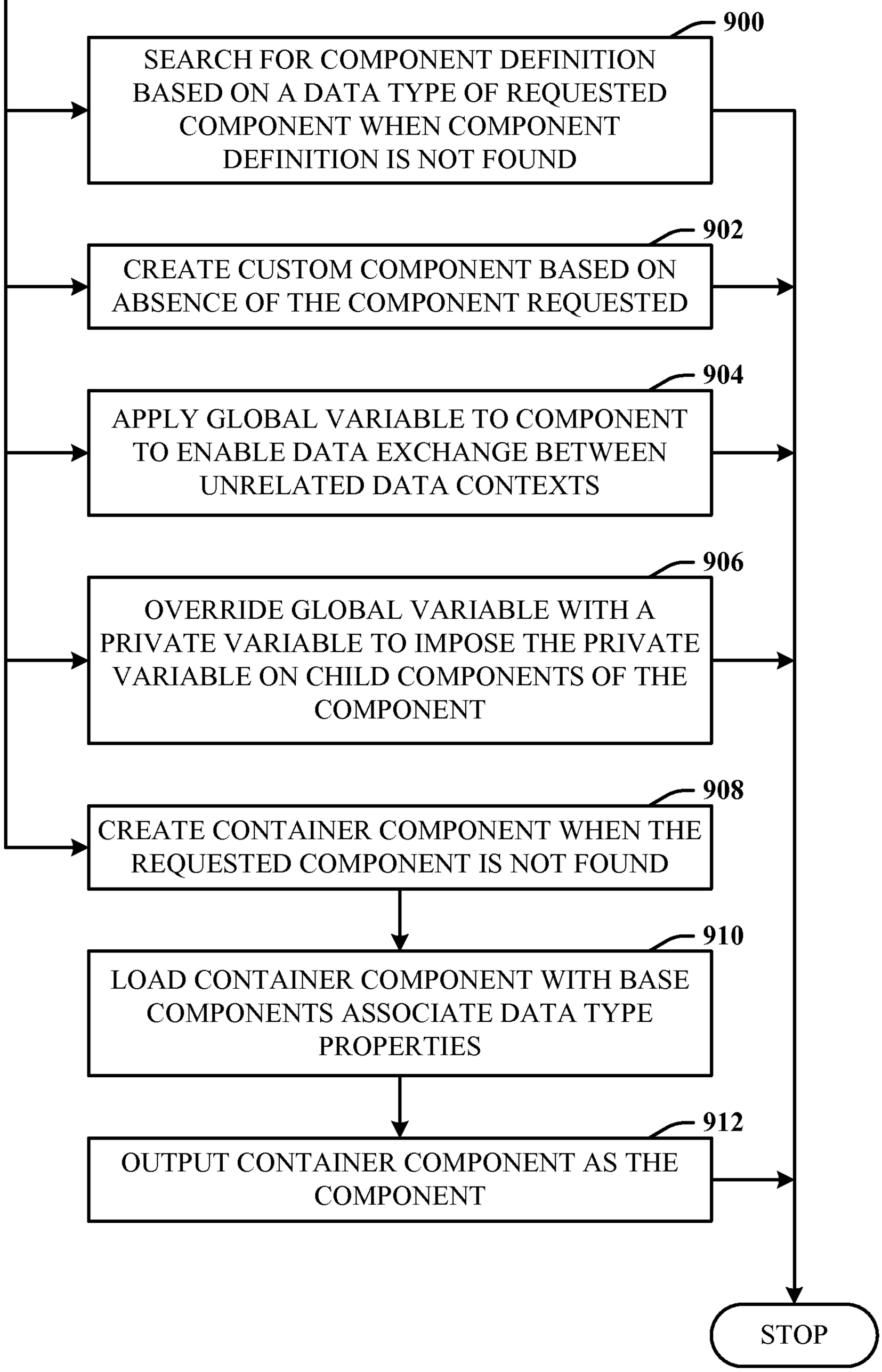
900
SEARCH FOR COMPONENT DEFINITION BASED ON A DATA TYPE OF REQUESTED COMPONENT WHEN COMPONENT DEFINITION IS NOT FOUND
902
CREATE CUSTOM COMPONENT BASED ON ABSENCE OF THE COMPONENT REQUESTED
904
APPLY GLOBAL VARIABLE TO COMPONENT TO ENABLE DATA EXCHANGE BETWEEN UNRELATED DATA CONTEXTS
906
OVERRIDE GLOBAL VARIABLE WITH A PRIVATE VARIABLE TO IMPOSE THE PRIVATE VARIABLE ON CHILD COMPONENTS OF THE COMPONENT
908
CREATE CONTAINER COMPONENT WHEN THE REQUESTED COMPONENT IS NOT FOUND
910
LOAD CONTAINER COMPONENT WITH BASE COMPONENTS ASSOCIATE DATA TYPE PROPERTIES
912
OUTPUT CONTAINER COMPONENT AS THE COMPONENT
STOP

FIG. 9

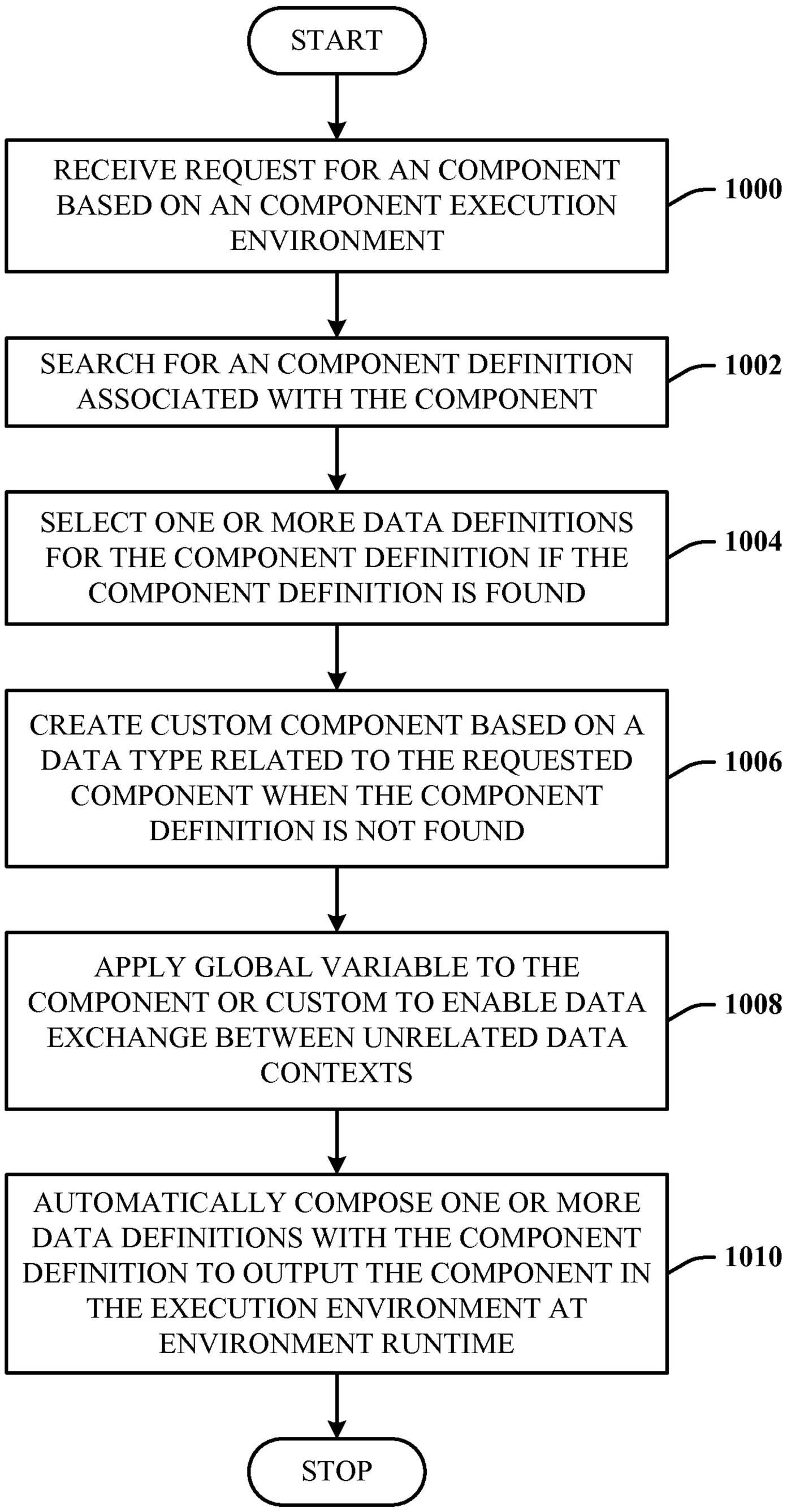
START
RECEIVE REQUEST FOR AN COMPONENT BASED ON AN COMPONENT EXECUTION ENVIRONMENT
1000
SEARCH FOR AN COMPONENT DEFINITION ASSOCIATED WITH THE COMPONENT
1002
SELECT ONE OR MORE DATA DEFINITIONS FOR THE COMPONENT DEFINITION IF THE COMPONENT DEFINITION IS FOUND
1004
CREATE CUSTOM COMPONENT BASED ON A DATA TYPE RELATED TO THE REQUESTED COMPONENT WHEN THE COMPONENT DEFINITION IS NOT FOUND
1006
APPLY GLOBAL VARIABLE TO THE COMPONENT OR CUSTOM TO ENABLE DATA EXCHANGE BETWEEN UNRELATED DATA CONTEXTS
1008
AUTOMATICALLY COMPOSE ONE OR MORE DATA DEFINITIONS WITH THE COMPONENT DEFINITION TO OUTPUT THE COMPONENT IN THE EXECUTION ENVIRONMENT AT ENVIRONMENT RUNTIME
1010
STOP

FIG. 10

FIG. 10

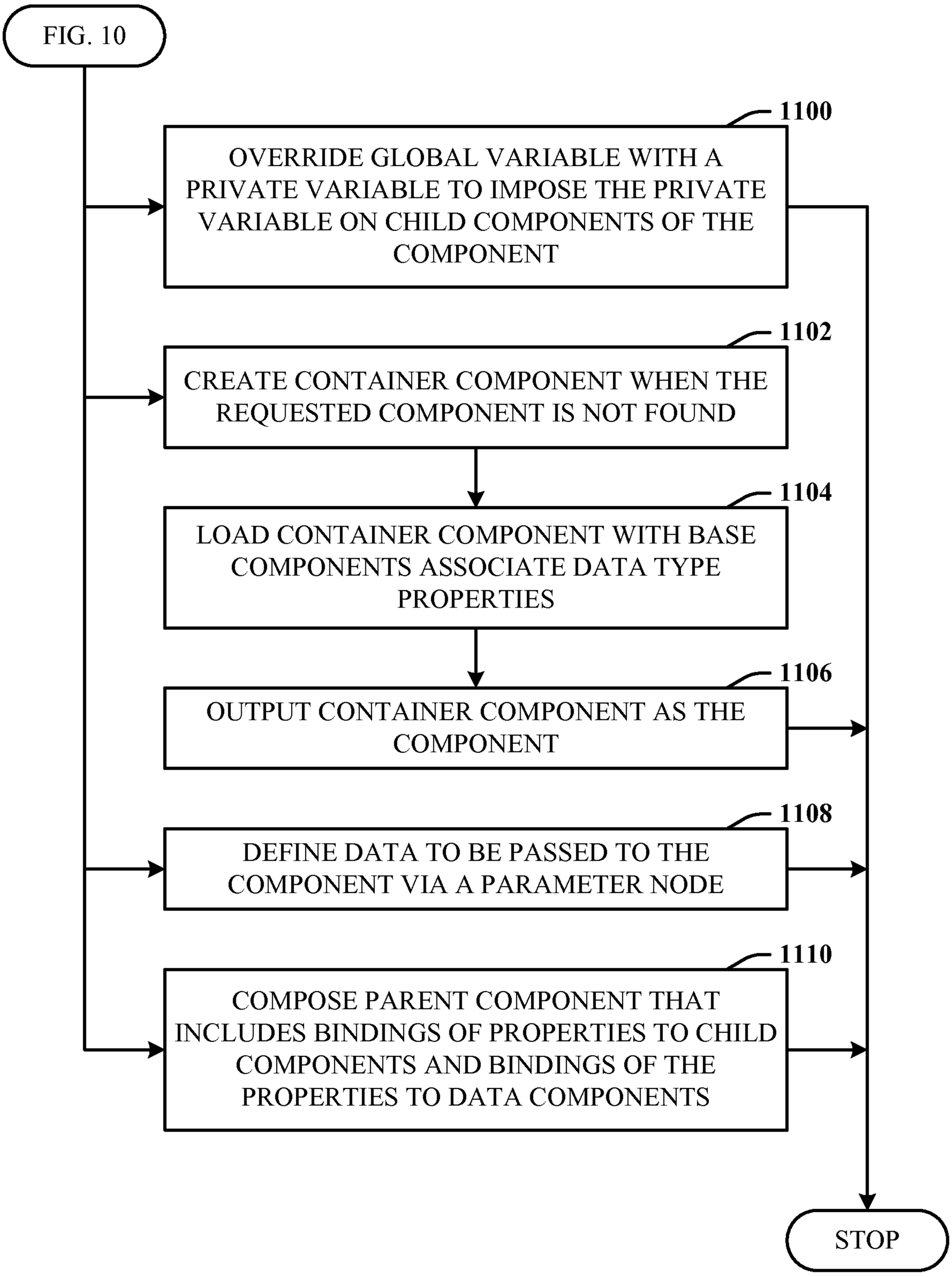
1100
OVERRIDE GLOBAL VARIABLE WITH A PRIVATE VARIABLE TO IMPOSE THE PRIVATE VARIABLE ON CHILD COMPONENTS OF THE COMPONENT
1102
CREATE CONTAINER COMPONENT WHEN THE REQUESTED COMPONENT IS NOT FOUND
1104
LOAD CONTAINER COMPONENT WITH BASE COMPONENTS ASSOCIATE DATA TYPE PROPERTIES
1106
OUTPUT CONTAINER COMPONENT AS THE COMPONENT
1108
DEFINE DATA TO BE PASSED TO THE COMPONENT VIA A PARAMETER NODE
1110
COMPOSE PARENT COMPONENT THAT INCLUDES BINDINGS OF PROPERTIES TO CHILD COMPONENTS AND BINDINGS OF THE PROPERTIES TO DATA COMPONENTS
STOP

FIG. 11

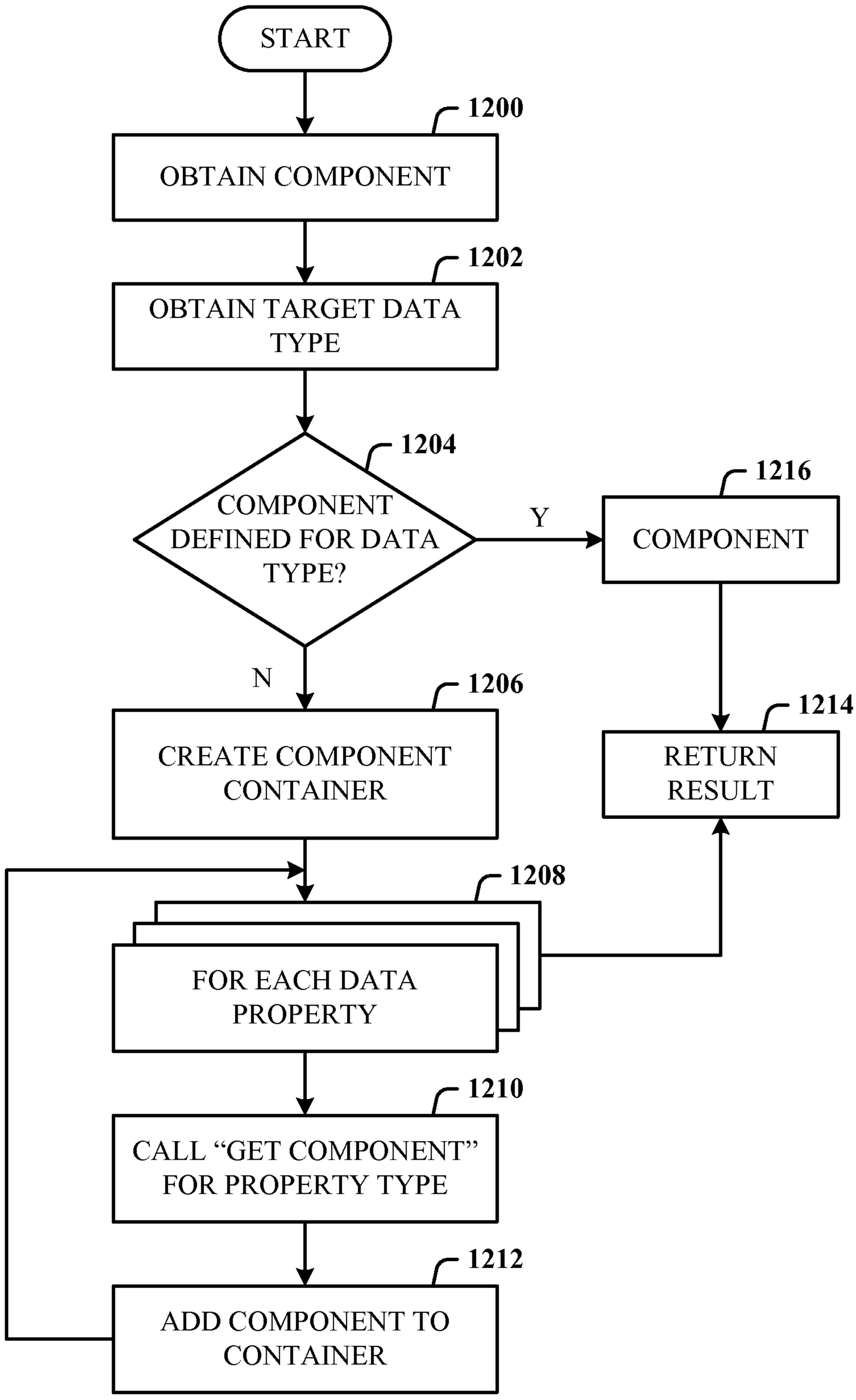
START
1200
OBTAIN COMPONENT
1202
OBTAIN TARGET DATA TYPE
1204
COMPONENT DEFINED FOR DATA TYPE?
Y
1216
COMPONENT
N
1206
CREATE COMPONENT CONTAINER
1214
RETURN RESULT
1208
FOR EACH DATA PROPERTY
1210
CALL "GET COMPONENT" FOR PROPERTY TYPE
1212
ADD COMPONENT TO CONTAINER

FIG. 12

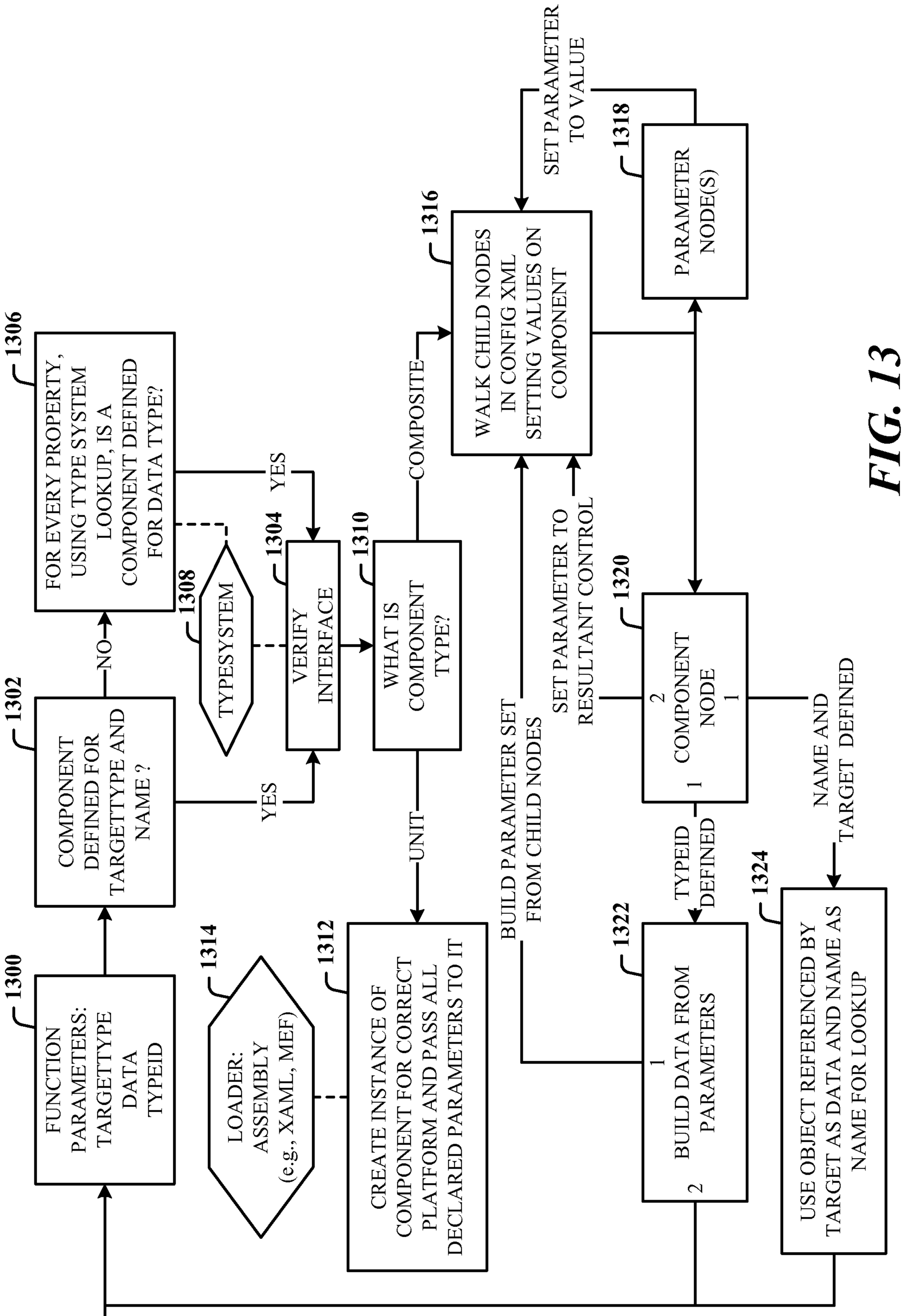
1300
FUNCTION PARAMETERS: TARGETTYPE DATA TYPEID
1302
COMPONENT DEFINED FOR TARGETTYPE AND NAME ?
NO
YES
1304
VERIFY INTERFACE
1306
FOR EVERY PROPERTY, USING TYPE SYSTEM LOOKUP, IS A COMPONENT DEFINED FOR DATA TYPE?
YES
1308
TYPESYSTEM
1310
WHAT IS COMPONENT TYPE?
COMPOSITE
UNIT
1312
CREATE INSTANCE OF COMPONENT FOR CORRECT PLATFORM AND PASS ALL DECLARED PARAMETERS TO IT
1314
LOADER: ASSEMBLY (e.g., XAML, MEF)
1316
WALK CHILD NODES IN CONFIG XML SETTING VALUES ON COMPONENT
BUILD PARAMETER SET FROM CHILD NODES
SET PARAMETER TO VALUE
1318
PARAMETER NODE(S)
SET PARAMETER TO RESULTANT CONTROL
1320
2 COMPONENT NODE 1
TYPEID DEFINED
1322
1 BUILD DATA FROM PARAMETERS
2
NAME AND TARGET DEFINED
1324
USE OBJECT REFERENCED BY TARGET AS DATA AND NAME AS NAME FOR LOOKUP

FIG. 13

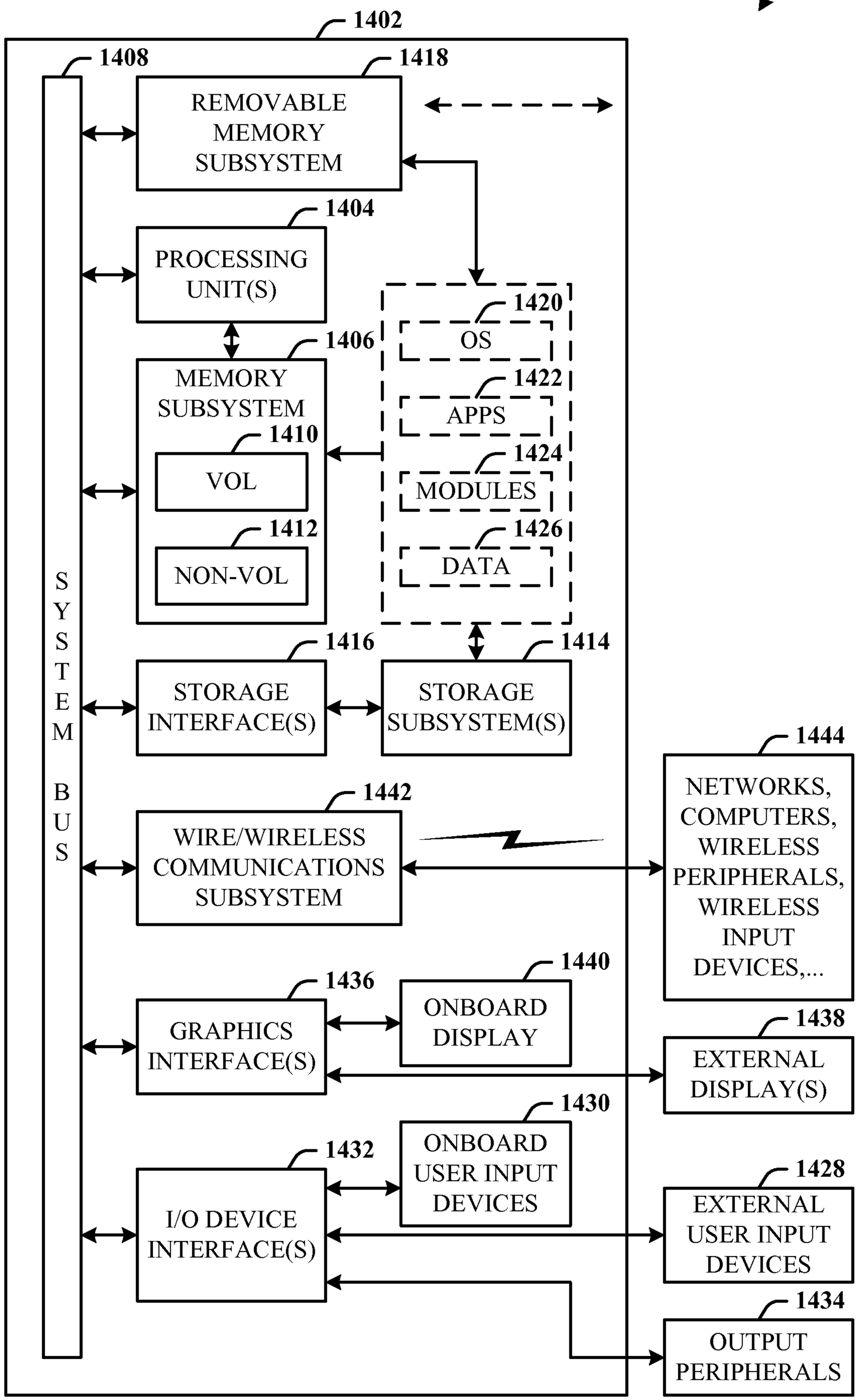
1400
1402
1408
1418
REMOVABLE MEMORY SUBSYSTEM
1404
PROCESSING UNIT(S)
1406
MEMORY SUBSYSTEM
1410
VOL
1412
NON-VOL
1420
OS
1422
APPS
1424
MODULES
1426
DATA
SYSTEM BUS
1416
STORAGE INTERFACE(S)
1414
STORAGE SUBSYSTEM(S)
1442
WIRE/WIRELESS COMMUNICATIONS SUBSYSTEM
1444
NETWORKS, COMPUTERS, WIRELESS PERIPHERALS, WIRELESS INPUT DEVICES,...
1436
GRAPHICS INTERFACE(S)
1440
ONBOARD DISPLAY
1438
EXTERNAL DISPLAY(S)
1430
ONBOARD USER INPUT DEVICES
1432
I/O DEVICE INTERFACE(S)
1428
EXTERNAL USER INPUT DEVICES
1434
OUTPUT PERIPHERALS

FIG. 14

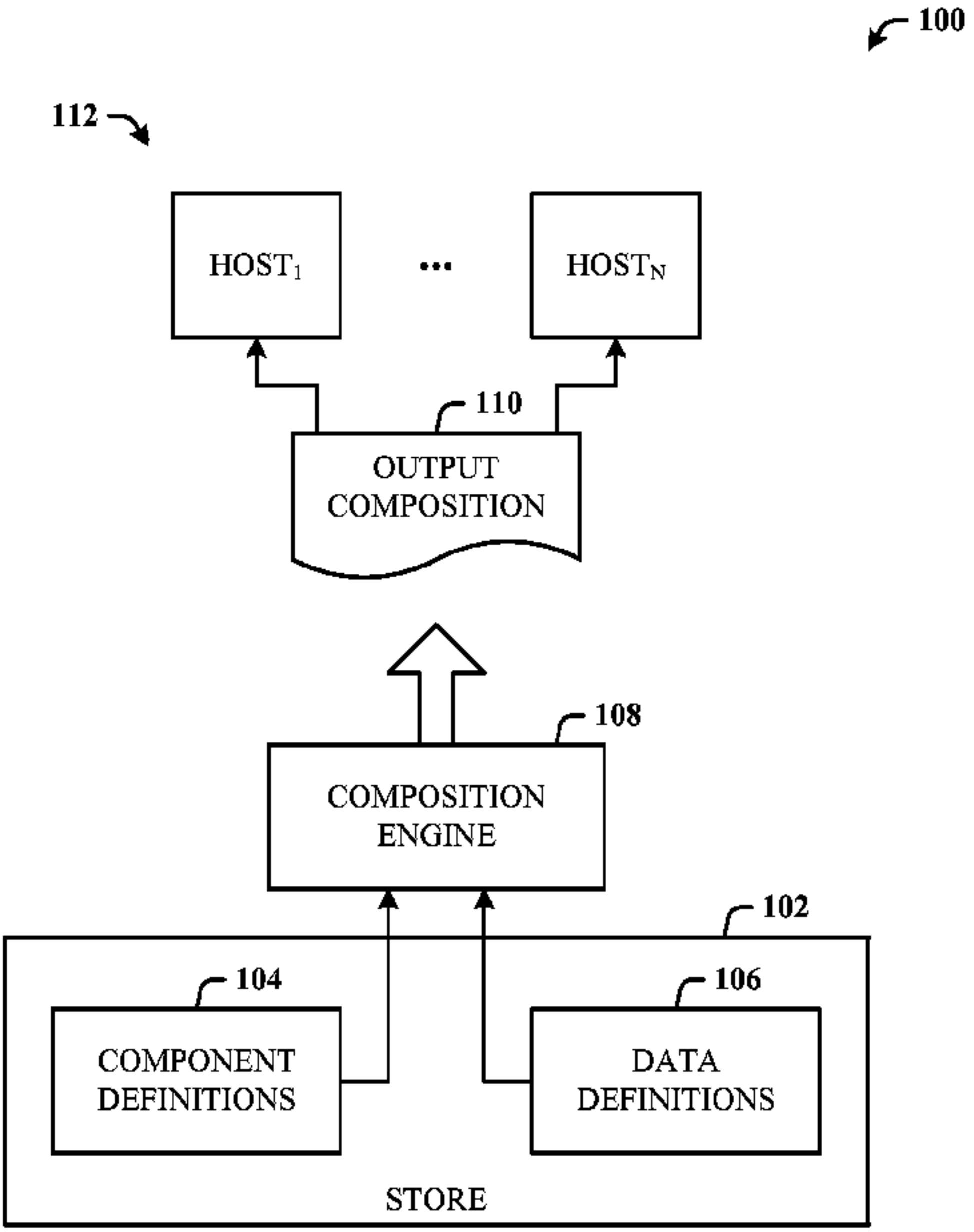
100
112
HOST$_1$
...
HOST$_N$
110
OUTPUT COMPOSITION
108
COMPOSITION ENGINE
102
104
COMPONENT DEFINITIONS
106
DATA DEFINITIONS
STORE

FIG. 1