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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 January 2012 (26.01.2012)

PCT

(10) International Publication Number
WO 2012/012123 A2

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

(21) International Application Number:
PCT/US2011/042046

(22) International Filing Date:
27 June 2011 (27.06.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
201010236014.5 20 July 2010 (20.07.2010) CN
12/860,667 20 August 2010 (20.08.2010) US

(71) Applicant *(for all designated States except US)*: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, M/S 5op7, Redwood Shores, California 94065 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **LI, Haijun** [CN/CN]; Unit 1002, China Life Tower, No. 16, Chaowaidajie, Beijing, 100020 (CN). **XU, Xin** [CN/CN]; Unit 1002, China Life Tower, No. 16, Chaowaidajie, Beijing, 100020 (CN). **SUN, Peng** [CN/CN]; Unit 1002, China Life Tower, No. 16, Chaowaidajie, Beijing, 100020 (CN). **EGOROV, Vladimir** [US/US]; 1228 Floribunda Avenue, Apt. 12, Burlingame, California 94010 (US). **DU, Hongwei** [CN/CN]; Unit 1002, China Life Tower, No. 16, Chaowaidajie, Beijing, 100020 (CN).

(74) Agents: **MEYER, Sheldon, R.** et al.; Fliesler Meyer LLP, 650 California Street, Fourteenth Floor, San Francisco, California 94108 (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, 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).

[Continued on next page]

(54) Title: SYSTEM AND METHOD FOR SUPPORTING AN OBJECT ORIENTED SCRIPTING TOOL

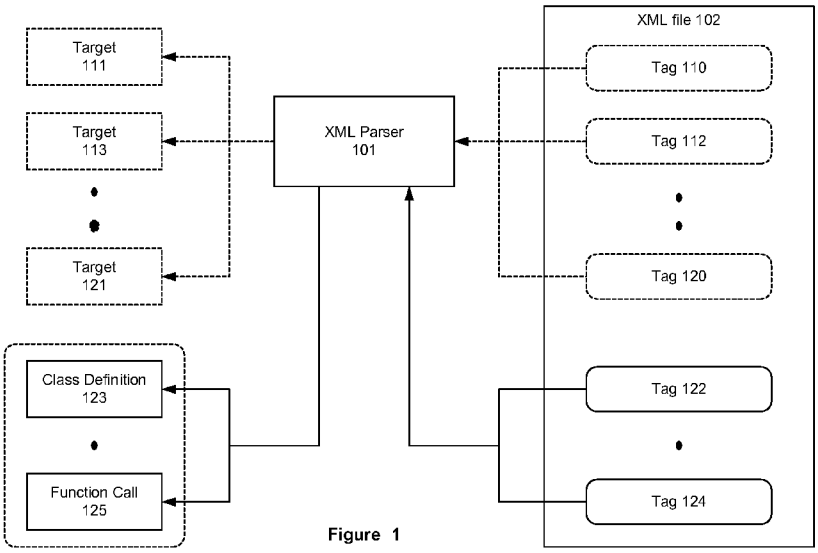

Figure 1

(57) Abstract: An object-oriented scripting tool uses a XML file for software development and domain management. The XML file includes at least a first tag that defines a scripting class in an object-oriented scripting language. The scripting class includes at least one method, which is defined in a second tag in the XML file. A generic software development and domain management script can be encapsulated into a general scripting class, which can be extended into individual scripting class for specific software development and domain management task.

WO 2012/012123 A2

Published:

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

SYSTEM AND METHOD FOR SUPPORTING AN OBJECT ORIENTED SCRIPTING TOOL

Field of Invention:

[0001] The present invention generally relates to a scripting tool for software development and domain management, and particularly to an XML-based scripting tool.

Background:

[0002] A scripting tool is a software tool that is implemented using a high-level scripting language. Typically, the scripts in the scripting tool can be interpreted by the execution environment at run time to perform certain programmed tasks.

[0003] One example of such a scripting tool is Ant (or "Another Neat tool"). Ant is a software tool, developed initially for automating software build processes. Ant is implemented using the JAVA programming language. Ant can be used for building JAVA projects in the JAVA platform. A typical Ant script file is in XML file format.

Summary

[0004] In accordance with an embodiment, an object-oriented scripting tool uses a XML file for software development and domain management. The XML file includes at least a first tag that defines a scripting class in an object-oriented scripting language. The scripting class includes at least one method, which is defined in a second tag in the XML file. A generic software development and domain management script can be encapsulated into a general scripting class, which can be extended into individual scripting class for specific software development and domain management task.

Brief Description of the Figures:

[0005] **Figure 1** illustrates an exemplary view of the object oriented Ant scripting tool environment.

[0006] **Figure 2** is an illustration that shows a sample mapping relationship between the syntax in objected oriented Ant scripting language and JAVA programming language, in accordance with an embodiment.

[0007] **Figure 3** is an illustration that shows an example of an objected oriented Ant class hierarchy for building software application, in accordance with an embodiment.

[0008] **Figure 4** is an illustration that shows a sample usage of object oriented Ant script classes for building application, in accordance with an embodiment.

[0009] **Figure 5** is an illustration that shows an example of a class hierarchy for managing application server domains, in accordance with an embodiment.

[0010] **Figure 6** is an illustration that shows a sample usage of object oriented Ant script classes for managing application server domains, in accordance with an embodiment.

[0011] **Figure 7** is an illustration that shows a sample usage of an object oriented Ant utility class, in accordance with an embodiment.

[0012] **Figure 8** is an illustration that shows the exemplary steps for testing software application in a distributed environment, in accordance with an embodiment.

Detailed Description:

[0013] The present invention is illustrated, by way of example and not by way of limitation, in the figures of the accompanying drawings in which like references indicate similar elements. It should be noted that references to "an" or "one" or "some" embodiment(s) in this disclosure are not necessarily to the same embodiment, and such references mean at least one.

[0014] The description of the embodiments of the invention as following uses the JAVA platform as an example for object oriented programming language platform. It will be apparent to those skilled in the art that other types of object oriented programming language platform can be used without limitation.

[0015] In accordance with an embodiment, an XML-based scripting tool, such as Ant, can be extended to include object oriented features. In one embodiment, the object oriented scripting tool can be implemented as an extension of the XML-based scripting tool. For example, the object oriented Ant scripting tool, or the object oriented Ant, can be implemented as an extension of Ant, using JAVA programming language. In one embodiment, the JAVA code that implements the object oriented Ant scripting tool can be compiled and deployed in a library directory that is designated by the Ant scripting environment to store extension tasks for the Ant scripting tool.

[0016] **Figure 1** illustrates an exemplary view of the object oriented Ant scripting tool environment.

[0017] As shown in Figure 1, the Ant scripting environment can use an XML parser 101 to parse the scripts that are contained in a XML file 102 with different tags 110, 112, and 120. Each tag 110,

112, or 120 in the XML file 102 is associated with a type of scripting task or target 111, 113, or 121.

[0018] In accordance with an embodiment, the object oriented scripting tool, such as the object oriented Ant, can make use of additional tags 122 and 124 that includes object oriented scripting syntax to support the object oriented features. As shown in Figure 1, a tag 122 can be used to support class definition task 123 in the object oriented Ant scripting environment; and another tag 124 can be used to support function call task 125 in the object oriented Ant scripting environment.

[0019] In accordance with an embodiment, the object oriented scripting syntax supports the use of different object oriented scripting types, which can be stored in a type definition table. The type definition table can be implemented using a harsh table, with the tag name as the key for the harsh table. Additionally, each task instance for the object oriented Ant scripting tool can be stored in an instance pool. The instance pool can also be implemented in a harsh table, using the id or reference for each instance as the key.

[0020] In another embodiment, such a type definition mechanism can be interpreted as a low priority, to prevent name collision with other usage of the same tag by the software developer for other purposes. In another example, a unique tag can be defined to initiate the object oriented Ant scripting environment, in order to prevent the name collision.

[0021] In accordance with an embodiment, the object oriented scripting tool can have similar features that are provided by an object oriented programming language. Furthermore, a software developer who is familiar with the object oriented programming language can easily understand the object oriented scripting language. For example, the objected oriented Ant which is based on Ant scripting language can have similar features as JAVA. This allows skilled JAVA developer to quickly learn to use the objected oriented Ant scripting tool.

[0022] **Figure 2** illustrates a sample mapping relationship between the syntax of the objected oriented Ant scripting language and the syntax of the JAVA programming language.

[0023] In accordance with one embodiment, the object oriented scripting tool can use a class tag to define a scripting class, just like the JAVA programming language can define a JAVA class. For example, as shown in Figure 2 at Line 1, a sample "Hello" class is defined in the objected oriented Ant scripting language, which can be mapped to a Hello class in the JAVA programming language.

[0024] In accordance with one embodiment, the class tag in object oriented scripting tool can contain arbitrary scripts that encapsulate the behavior of the scripting class. In one embodiment, there can be two attributes for the class tag. One attribute is a name attribute that defines a name for the underlying class. Another attribute is a base attribute that defines the class name of a base class. A default value for the base attribute is "Object," which indicates that the underlying class is a class extended directly from the root class.

[0025] In accordance with an embodiment, the object oriented scripting tool can use a method tag to define a method for a class, in a manner similar to defining a method in a JAVA class. In one

embodiment, the method tag can have a name attribute. For example, as shown in Figure 2 at Line 3, the name attribute for the method is given a value "sayHi." Additionally, the method tag also allows one or more attribute tags to be defined in the body of the method tag. Such attribute tags define one or more parameters that can be used by the method when it is invoked. In the example as shown in Figure 2 at Line 4, the method "sayHi" takes a parameter that specifies the content of a message.

[0026] In accordance with an embodiment, the object oriented scripting tool can assign a reference value to a class. For example, in Figure 2 at Line 7, a tag can be used for class initiation, which is similar to creating a new class in JAVA. The instance of the "Hello" class created using the tag in Figure 2 at Line 1 is assigned a reference value "h" in order to uniquely identify the instance.

[0027] In accordance with an embodiment, the object oriented scripting tool allows a user to use a class invocation tag to invoke a method in a scripting class, in the format of "[class reference].[method name]." For example, as shown in Figure 2 at Line 9, the "sayHi" method defined in Figure 2 at Lines 3-5 can be invoked using the syntax of "h.sayHi," which reassembles the class invocation syntax in JAVA. The "sayHi" method is provided with a parameter that that includes the content of the message, accordingly to the definition of the message attribute in Figure 2 at Figure 2 at 4.

[0028] In accordance with an embodiment, the object oriented scripting tool provides an echo tag that can write out the content of the message. For example, as shown in Figure 2 at Line 11, the echo tag can operate in a similar manner to System.out in JAVA.

[0029] In accordance with an embodiment, the object oriented scripting tool provides an <instance of> tag that can be used to check if some object oriented Ant object is some specific Ant class type. In one embodiment, the <instance of> tag provides an "id" attribute which describe an Ant object id; and a "classname" attribute that describes the name of an Ant class to be verified.

[0030] In accordance with an embodiment, object oriented scripting tool can provide other useful features, just like the object oriented programming languages. These useful features include allowing method overriding and special "this" and "super" instance, like JAVA provides. Also, similar to JAVA, the object oriented Ant can support class extension and inheritance; reflection flavor of instance method invocation syntax, and syntax of getting instance property.

[0031] In accordance with an embodiment, the object oriented scripting tool can be used to build a software application. A generic application build scripts can be encapsulated into a class. A software developer can then create his or her own version of the generic application build scripts, in order to incorporate in some special feature he or she likes.

[0032] **Figure 3** shows an example of a class hierarchy for building software application. As shown in Figure 3, App.class.xml 301 is the base class that encapsulates generic application build scripts. MyAppBase.class.xml 302 is a class that inherits from the base class ,App.class.xml 301. A software developer does not need to rewrite or copy and paste the implementation details already existing in App.class.xml 301. The software developer only needs to add new method or override an

existing method in MyAppBase.class.xml 302.

[0033] Additionally, based on MyAppBase.class.xml 302, the software developer can create different object oriented Ant script classes, MyApp1.class.xml 303 and MyApp2.class.xml 304, for different projects. The different object oriented Ant script classes, MyApp1.class.xml 303 and MyApp2.class.xml 304, includes all necessary information in order to successfully compile and build a project, such as project directory and links to useful libraries.

[0034] **Figure 4** shows a sample usage of object oriented Ant script classes for building application. As shown in Figure 4, two separate object oriented Ant script classes, MyApp1 and MyApp2 can be used to build two separate applications: app1 and app2.

[0035] In accordance with an embodiment, the object oriented scripting tool can also be used to manage different network or application server domains.

[0036] **Figure 5** shows an example of a class hierarchy for managing application server domains. As shown in Figure 5, generic domain related scripts can be encapsulated into a class, Domain.class.xml 501. A network administrator can specify his or her own version of the generic domain related scripts in another class MyDomainBase.class.xml 502. MyDomainBase.class.xml 502 includes customized features that fit to the network administrator. Here, MyDomainBase.class.xml 502 is a class that inherits the base class, Domain.class.xml 501. In such a scenario, the network administrator does not need to rewrite or copy and paste the implementation details already existing in the generic domain management scripts. The software developer only needs to add new method or override an existing method for his or her own need.

[0037] Additionally, based on MyDomainBase.class.xml 502, the network administrator can create a separate object oriented Ant script classes, MyDomain1.class.xml 503, for a specific domain. Here, the object oriented Ant script class includes all necessary information in order to successfully prepare and start up the specific domain.

[0038] **Figure 6** shows a sample usage of object oriented Ant script classes for managing application server domains. As shown in Figure 6, an application server domain, MyDomain1, can be prepared and start up using the object oriented Ant script class.

[0039] In accordance with an embodiment, the object oriented scripting tool can be used to make utility scripting class that can accept objects of other types of scripting classes as argument.

[0040] **Figure 7** a sample usage of an object oriented Ant utility class. As shown in Figure 7, the utility Ant class, "MyApp1Util" class, has a reference or id, "applutil." The "MyApp1Util" class has a process method, which can obtain all instance properties of the "MyApp1" class by its reference or id, "app1." Further operations can be performed based on the obtained instance properties of the "MyApp1" class. Such operations include, but are not limited to, managing deployment descriptor files and so on.

[0041] In accordance with an embodiment, the object oriented scripting tool can be used for

software application testing in a distributed environment, such as setting up a domain of application servers and linking to a database.

[0042] **Figure 8** shows the exemplary steps for testing software application in a distributed environment. As shown in Figure 8, the software application testing scripts can include the steps of first compiling the application 801, and developing and compiling different test cases of the application 802 for different purposes. Then, the target domain needs to be prepared 803 and started up 804, in order to be ready for the deployment of the application on the servers in the domain 805. After the test cases are done 806, the domain can be shut down 807 and ready for clean up 808.

[0043] Due to the complex nature of software testing, different test cases can be developed. There can be huge mount of script files for each testing scenario. The migration of the script files from one test case to another test case involves the usage of a lot of similar testing scripts, through a copy and paste approach, previously. The copy and paste approach is not efficient, since the testing scripts can be very large in size and involves a lot of details. Many times, the software developers are even willing to rewrite testing scripts without reusing the old testing scripts, in order to avoid mistakes. By using object oriented Ant, the software developer can easily extend from one test case to another test case without the need to copy and paste a huge amount of code.

[0044] In accordance with an embodiment, the introduction of object oriented syntax into Ant makes it much easier to reuse existing scripts. For example, an Ant macro "A" can internally invoke another macro "B." In one example, the software developer just wants to change the behavior of "B" and reuse "A". Using the copy and paste approach, the software developer needs to create two new macros: "A1" and "B1." the software developer needs to copy content of "A" into "A1" and change invoking of "B" into invoking of "B1." With the object oriented Ant, the software developer can simply extend the base class “B” with a new class “B1,” which overrides one or more methods in "B" with new implementation.

[0045] The present invention may be conveniently implemented using a conventional general purpose or a specialized digital computer or microprocessor programmed according to the teachings of the present disclosure. Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art.

[0046] In some embodiments, the present invention includes a computer program product which is a storage medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the processes of the present invention. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

[0047] The foregoing description of the present invention has been provided for the purposes of

illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the practitioner skilled in the art. The code examples given are presented for purposes of illustration. It will be evident that the techniques described herein may be applied using other code languages, and with different code.

[0048] The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalence.

Claims:

What is claimed is:

1. A computer-implemented method for supporting an object-oriented scripting tool using an XML file, comprising:

defining a software class in a scripting language using a first tag in the XML file, wherein the software class includes at least one method; and

invoking the at least one method in a second tag in the XML file.

2. The method according to Claim 1, wherein the scripting language is Ant.

3. The method according to Claim 1 or 2, further comprising:

using an XML parser to parse the XML file.

4. The method according to any preceding Claim, further comprising:

defining the at least one method using a tag in the XML file.

5. The method according to any preceding Claim, further comprising:

using a tag in the XML file to check whether a software object is an instance of the software class.

6. The method according to any preceding Claim, further comprising:

supporting at least one of:

class extension and inheritance,

method overriding,

instance polymorphism, and

special "this" and "super" instance.

7. The method according to any preceding Claim, further comprising:

supporting a syntax for instance method invocation in a tag of the XML file, wherein the syntax is in a format of "[class reference].[method name]".

8. The method according to Claim 7, further comprising:

associating a tag in the XML file with the syntax for instance method invocation, when the tag is determined to be an unknown element.

9. The method according to Claim 8, further comprising:

using a type definition table to define the syntax for instance method invocation.

10. The method according to any preceding Claim, further comprising:

encapsulating one or more generic application build scripts into the software class, and

executing one or more special application build scripts in another software class, wherein the another software class is extended or inherited from the software class.

11. The method according to any preceding Claim, further comprising:

encapsulating one or more generic domain management scripts into the software class, and

starting up a special domain using one or more domain management scripts in another software class, wherein the another software class is extended or inherited from the software class.

12. The method according to any preceding Claim, wherein the software class is a utility class that can accept an instance of another software class as an argument.

13. The method according to any preceding Claim, further comprising:

using the object-oriented scripting tool to link to a database.

14. The method according to any preceding Claim, further comprising:

using the object-oriented scripting tool to deploy an application to different servers.

15. A computer program comprising instructions that when executed by one or more computer systems implement the method of any preceding claim.

16. A computer-readable storage medium including the computer program of claim 15.

17. A computer system configured to implement the method of any of claims 1 to 14.

18. A computer system for supporting an object-oriented scripting tool using an XML file, said system comprising an XML file which includes:

a first tag in the XML file defining a software class in a scripting language, wherein the software class includes at least one method; and

a second tag in the XML file invoking the at least one method in the software class.

19. A machine readable medium having instructions stored thereon that when executed cause a system to:

define a software class in a scripting language using a first tag in an XML file, wherein the software class includes at least one method; and

invoke the at least one method in a second tag in the XML file.

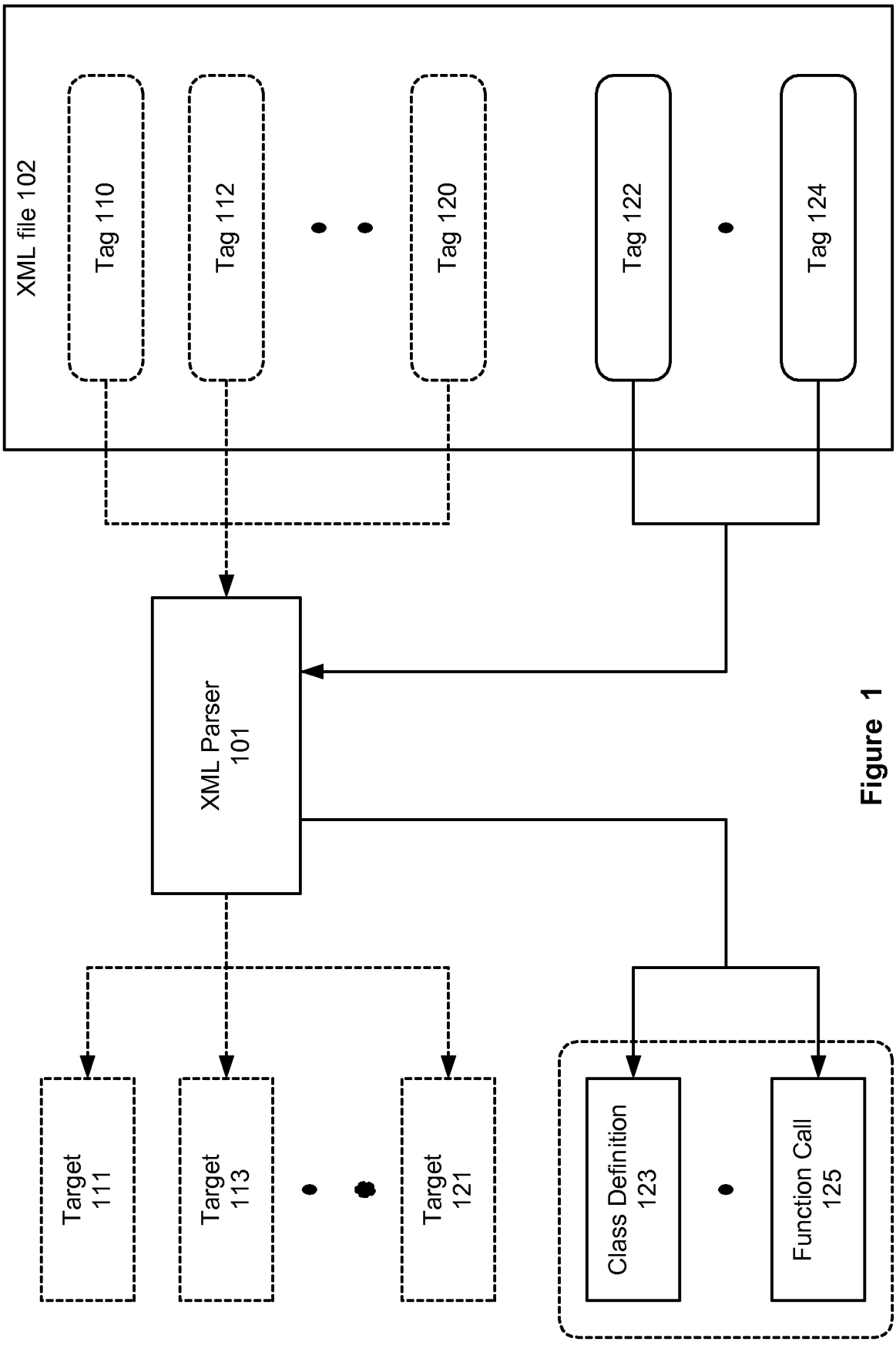
XML file 102
Tag 110
Tag 112
Tag 120
Tag 122
Tag 124
XML Parser 101
Target 111
Target 113
Target 121
Class Definition 123
Function Call 125

Figure 1

	Object Oriented Ant	JAVA
1	<class name="Hello"> ... </Class>	public class Hello {...}
2		
3	<Method name="sayHi">	public void sayHi(String message)
4	<attribute name="message"/>	
5	<Method/>	
6		
7	<Hello.init id="h"/>	Hellow h = new Hello();
8		
9	<h.sayHi message="Hi"/>	h.sayHi("Hi");
10		
11	<echo message="${h.message}"/>	System.out.prinitln(h.getMessage(());
12		
13	<super.sayHi message="Hi"/>	super.sayHi(Hi");
14		
15	<this.sayHi message="Hi"/>	this.sayHi(Hi");
16		

Figure 2

Figure 3

```
<MyApp1.init id="app1" dir=""/>
<MyApp2.init id="app2" dir=""/>

<app1.build/>
<app2.build/>
```

Figure 4

Figure 5

```
<MyDomain1.init id="domain1" domainNmae="xxxDomain"/>

<domain1.prepare/>
<domain1.startup/>
```

Figure 6

```
<MyAppl.init id="appl" dir=""/>
<MyApplUtil.init id="applutil"/>
<applutil.process app="appl"/>
```

Figure 7

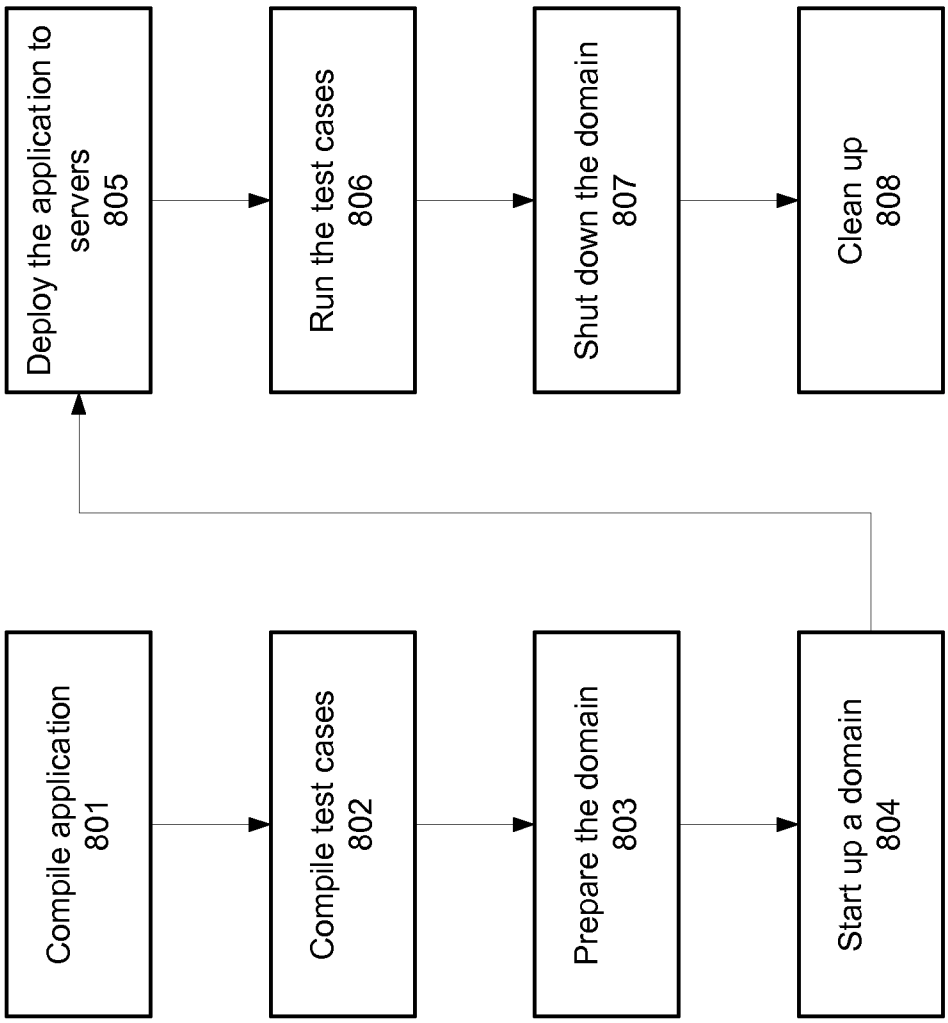
Compile application
801
Compile test cases
802
Prepare the domain
803
Start up a domain
804
Deploy the application to servers
805
Run the test cases
806
Shut down the domain
807
Clean up
808

Figure 8