STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LV**, Number 2, 2010

A PROPOSED APPROACH FOR PLATFORM INTEROPERABILITY

PAUL HORAŢIU STAN AND CAMELIA ŞERBAN

Abstract. This paper presents a new approach regarding interoperability. The novelty of the proposed solution resides in using a proxy object for each parameter object in order to avoid serialization. The main guidelines and the architecture related to developing a framework in order to automatically generate the source code for communication between platforms are proposed. Finally, there are summarized the contributions of this work and future improvements.

1. Introduction

An actual software engineering problem is the improvement of the software development process and the final product quality. In order to realize this, the source code should be reusable [8]. Obviously, it is a good news for a developer if an old library developed in Java can be used in .NET without being necessary to rewrite the code. In this case, it saves time and improves the quality of the final software system, because the Java library was tested in the past and it works fine.

There are many frameworks written for both Java and .NET, for instance: Hibernate [9] respectively NHibernate, JUnit [10] respectively NUnit etc. These frameworks could be written for a platform and reused from another platform, for instance instead of rewriting Hibernate for .NET it can be reused directly.

The current paper presents the architecture and the main guidelines related to developing a framework in order to automatically generate the source code for communication between platforms.

The proposed approach is discussed as follows: Section 2 highlights the current interoperability approaches, and the proposed solution together with the motivation of the problem and the disadvantages of actual interoperability

Received by the editors: February 5, 2010.

2010 *Mathematics Subject Classification.* 68N15.

1998 *CR Categories and Descriptors.* D.2.12 [**Software**]: SOFTWARE ENGINEERING – *Interoperability*; D.2.13 [**Software**]: SOFTWARE ENGINEERING – *Reusable Software*;

Key words and phrases. Proxy, Automatic Source Code Generation, Interoperability.

approaches. Section 3 describes in details the technical details, the theoretical concepts used in order to implement the interoperability mechanism and an implementation solution for .NET-Java. Finally Section 4 summarizes the contributions of this work and presents future research directions.

2. The Problem

The general context of this article is focused on how to access remote objects developed for different programming languages in a transparent way, like they were written for client programming language. The problem is how the parameters can be passed to the remote methods without being necessary to serialize them on client platform and restored on remote platform in order to be used there. The main idea presented in this article is that each object should be executed in the address space of the process which has created itself.

This section is composed by two parts, first presents the current interoperability approaches and the second describes the proposed solution. The proposed solution is presented as follows: the motivation of the problem, the disadvantages of current approaches and a short description of the technical solution.

2.1. **Current interoperability approaches.** Nowadays there are many software applications that communicate and solve business problems. These applications are developed using different platforms, for instance Java, .NET, PHP, Perl, Python, Pascal etc. In order to realize the communication these applications should use the same protocol. There are many techniques used for implementing the communication between applications, for instance: COM (Microsoft Component Object Model), CORBA (Common Object Request Broker Architecture), Java RMI (Remote Method Invocation) [11], .NET Remoting [12], WEB Services based applications [2], SCA (Software Component Architecture) [3] etc.

The Component Object Model (COM) lets an object expose its functionality to other components and to host applications [16]. COM is Microsoft's initial component object model. A binary standard for the efficient interoperation across component boundaries. A COM component can implement several COM classes, each uniquely identified by a class ID (CLSID). Each COM class can implement several COM interfaces. A COM interface provides a set of operations and is uniquely identified by an interface ID (IID). A COM object is an instance of a COM class, but does not necessarily constitute a single object (split object). Clients use COM objects solely via the interfaces provided by that object. Each interface has a QueryInterface operation that can be used to ask for any of the other interfaces of the COM object based on

IIDs. COM object servers execute in processes that can be partitioned into COM apartments [1].

The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together, i.e. it supports multiple platforms [15].

First and foremost, CORBA covers the specification of the interfaces of object request brokers (ORBs). An ORB accepts requests for method invocations and relays them to the addressed object, bridging platform and language gaps. An ORB also provides interface and implementation repositories that make the system fully self-describing (reflection). CORBA also covers the binding of programming languages to ORB-understood interfaces. Such interfaces are described in a standardized interface definition language (IDL). CORBA 3 added the CORBA component model (CCM), which is designed to be a superset of EJB [1].

RMI-IIOP provides interoperability with other CORBA objects implemented in various languages - but only if all the remote interfaces are originally defined as Java RMI interfaces. It is of particular interest to programmers using Enterprise JavaBeans (EJB), since the remote object model for EJB components is based on the RMI API [17].

The .NET remoting support combines context and reflection infrastructure with flexible support for proxies, channels, and messages to provide building blocks for a wide variety of communication styles and patterns [1].

Web services are similar to deployed components from a client point of view as they offer features via standard interfaces. However, services do not reveal their implementations platform and infrastructure requirements. Web services are far more self-contained than typical components or even applications. By being almost completely self-contained, services cannot be reused in different contexts, but are merely good to be used as is. To enable reuse, services would have to have explicit context dependencies and offer reconfiguration by binding their required interfaces against selected provided interfaces of other services [1].

In the Simple Object Access Protocol (SOAP) model each object is serialized on the source platform, sent to the destination platform and deserialized there. If the serialized object is very large then the overall performance decreases.

Service Component Architecture (SCA) defines a way to create components and a mechanism for describing how those components work together. SCA was originally created by a group of vendors, including BEA, IBM, Oracle, SAP, and others [4].

Every SCA application is built from one or more components. The application might contain a few components implemented as Java classes, others written in C++, and still others defined using BPEL, all spread across a group of machines [4].

2.2. **Proposed Approach.** This paper intends to presents a technique for implementing the platform interoperability and the source code reusability between programming languages.

2.2.1. *Problem Motivation.* The platform interoperability approach allows the source code written for a given platform to be called from another platform, this will determine a more reusable source code. The software development process will be improved because the code rewriting tasks are skipped, the development time is minimized and the final deliverable quality is improved.

On the other hand, the libraries reusability instead of redesigning and rewriting them implies a better software systems functionality. In many cases when a functionality is rewritten, bugs appear in project, then in the software industry it's recommended to reuse modules that works fine instead of rewriting them [3].

2.2.2. *Drawbacks of current interoperability approaches.* Some current interoperability approaches are based on the COM technology, the Java and/or .NET classes are converted to COM objects and after that are imported using native methods for Java and using COM objects for .NET. There are business solutions that use this technique for interoperability. The COM technology disadvantage is that it is operating system dependent, so it does not allow platform interoperability.

In CORBA, the server objects are managed by an object broker that can be accessed by clients in order to send messages to server objects. The limitation of CORBA is that the server objects methods could not return and could not have parameters of types defined on client side.

Unlike COM and CORBA, proposed approach is platform independent and allows remote object's methods to return and to have parameters of client side defined types.

A web services based approach limitation is the objects passed from client to server and vice-versa should be serialized. The communication process is composed of the following steps:

- *Serialize the object in an XML format*
- *Send the serialized object*
- *Deserialize the object from the XML format*

Proposed solution solves this limitation, the objects should not be serializable because proxy objects are generated and it manage the communication with real object instead of serializing the real objects and deserializing them on the destination platform. This technique involves only few data exchanges between platforms used for creating and managing proxy objects. This solution is applicable for systems that does not involve data intensive processing but involves calling remote procedures and functions.

Unlike Web Services approaches, in the proposed solution for each remote object a proxy object is generated on the client side. This allows to avoid serialization/deserialization of huge objects. In some scenarios the overall performance is increased and in other ones the performance is decreased.

The Service Component Architecture model defines a top-down approach, in contrast with the proposed technique that is focused on reusing old software routines, that involves a bottom-up development process.

Presented solution can be used if parts of a distributed object should be used by many applications. For this use case our solution is better than SOAP model that implies serialization and deserialization of the remote object, in contrast, the proposed solution will exchange only short messages in order to manipulate the needed remote object parts.

2.2.3. *Conceptual view of the proposed solution.* The main solution idea is that each object is executed in the address space of the process which has created itself. This means these objects are not serialized, sent to remote platform and deserialized there in order to be used.

In the proposed solution for each remote object a proxy object is generated on the client side. There are three use cases for a proxy object: create itself, erase itself and invoke one of their methods, for all of the above situations the request is redirected to the remote object, when a proxy object is created a remote object is created too, when the proxy is erased from memory the remote object is erased too and when a proxy method is invoked the request is forwarded to the remote object. These three situations are presented in the figure 1.

3. Technical details

This section presents the high level architecture of the proposed approach and the implementation solution. This solution is applicable for .NET and Java, but it can be extended to support other development platforms.

We discuss the proposed approach as follows: Sections 3.1. presents the main architecture and introduces the terms and definitions used later in this article, Section 3.2 shows how the proposed solution works, finally, the main architecture of the developed interoperability framework for Java and .NET is presented in Section 3.3.

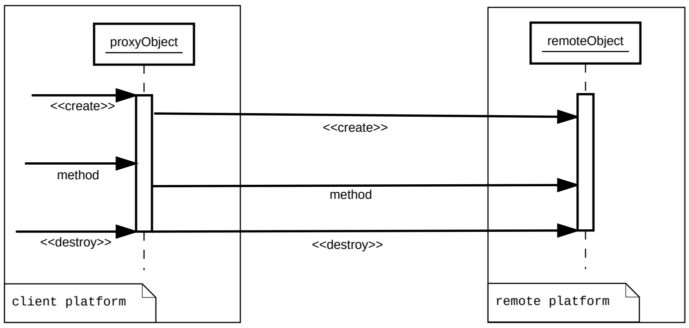

FIGURE 1. Proposed Architecture

3.1. **Architecture.** The proposed solution is based on the Proxy design pattern presented in [7].

In the proposed solution for each remote object a proxy object is generated on the client side. When the client platform creates the proxy then the server creates the real object and store them in a hash table based on an automatically generated identifier. When the client side platform erase the proxy object, for instance the garbage collector decides to remove from memory the unreferenced objects then the server side platform will remove from the remote objects hash table the real object. When a client object invokes a method on the proxy object the request is redirected to the remote object. These three situations are presented in the figure 2.

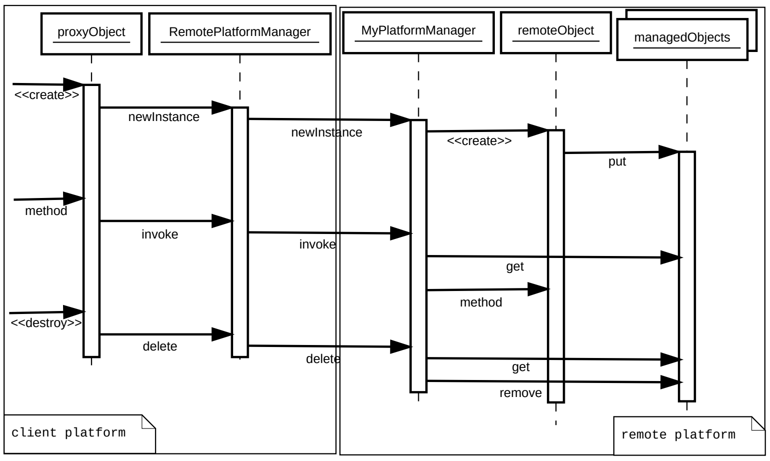

FIGURE 2. Communication Architecture

Definition 1. *RemotePlatform is the development environment and the programming language used in order to develop routines that should be imported.*

Definition 2. *ClientPlatform is the development environment and the programming language in which should be imported the remote routines.*

Definition 3. *RemoteType is a data type defined on the RemotePlatform that should be accessed from ClientPlatform.*

Definition 4. *ProxyType is a data type defined on the ClientPlatform. An instance of this type represents a proxy to an instance of RemoteType. This type is automatically generated.*

Definition 5. *RemotePlatformManager is a data type that manage the communication with RemotePlatform. This data type has been written for ClientPlatform and is used by ProxyType in order to redirect the methods requests. It is included in the proposed framework.*

Definition 6. *MyPlatformManager is a data type that manages the communication with the ClientPlatform. This data type has been written for RemotePlatform, it receives commands from RemotePlatformManager and redirects them to instances of RemoteType. It is included in the proposed framework.*

3.2. **How it works.** The general context is: there is a data type *RemoteType* on the developing platform *RemotePlatform* and it should be used in a class *C* on developing platform *ClientPlatform*. Based on the *RemoteType* an XML file is generated that specifies how the type can be used. Using this XML file the source code for the *ProxyType* is automatically generated. The *ProxyType* is written in the *ClientPlatform* programming language. This process is exposed in the figure 3.

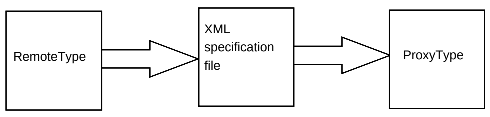

FIGURE 3. Automatically Generating the Proxy Type

After the *ProxyType* is generated it can be used on the *ClientPlatform*. The remote types in binary format should be available, that means .jar files if the remote objects are written in Java or .dll files if the remote objects are written in .NET.

On the *ClientPlatform* there is the class *RemotePlatformManager*, this class manages the communication with the *RemotePlatform*. On the *RemotePlatform* there is the class *MyPlatformManager* that manage the communication with the clients. *MyPlatformManager* stores all remote objects in a hash table using unique identifier for each object. When a method is invoked on the proxy object the *RemotePlatformManager* delegates the responsibility to the *MyPlatformManager* witch invokes the method on the remote object. When the proxy object is disposed from client side then the remote object is deleted from hash table.

A parameter type of a proxy object's method can be in one of the following situations:

- A standard type: *String*, *byte*, *short*, *int*, *long*, *char*, *float*, *double*, *boolean*. In this case the parameter is transmitted by value.
- A proxy type. In this situation the object identifier is sent to the *RemotePlatform*, in order to identify the real object.
- A client type. In this case the parameter is stored in a hash table on the *ClientPlatform* and an automatically generated identifier is passed to the *RemotePlatform*. The *RemotePlatform* will use the identifier in order to create a proxy object for the real client side object.

Figure 4 describes the passing parameter mechanism for the situations presented above.

The return type of a proxy object's method can be in one of the following situations:

- A standard type: *String*, *byte*, *short*, *int*, *long*, *char*, *float*, *double*, *boolean*. In this case the parameter is transmitted by value.
- A proxy type. In this situation the object identifier is sent from the *RemotePlatform* to the *ClientPlatform*, on the client side a proxy object is created that represents the real remote object.
- A client type. In this case on the *RemotePlatform* there is a proxy object to a real client object. The parameter object identifier is sent from *RemotePlatform* to the *ClientPlatform* in order to identify the client real object and this client object is returned by the proxy method.

RemotePlatformManager and *MyPlatformManager* are responsible for communication between platforms. The communication can be implemented using different protocols such as: TCP/IP, shared memory, HTTP etc. In this research the TCP/IP has been chosen for sending XML data in order to realize the communication.

3.3. **The developed framework for interoperability.** During this research a framework for interoperability between .NET and Java has been developed. Current version allows only Java classes to be used in .NET. On the following three listings the request XML files for the next three scenarios are presented:

- create a remote object
- invoke a method on a remote object
- delete a remote object

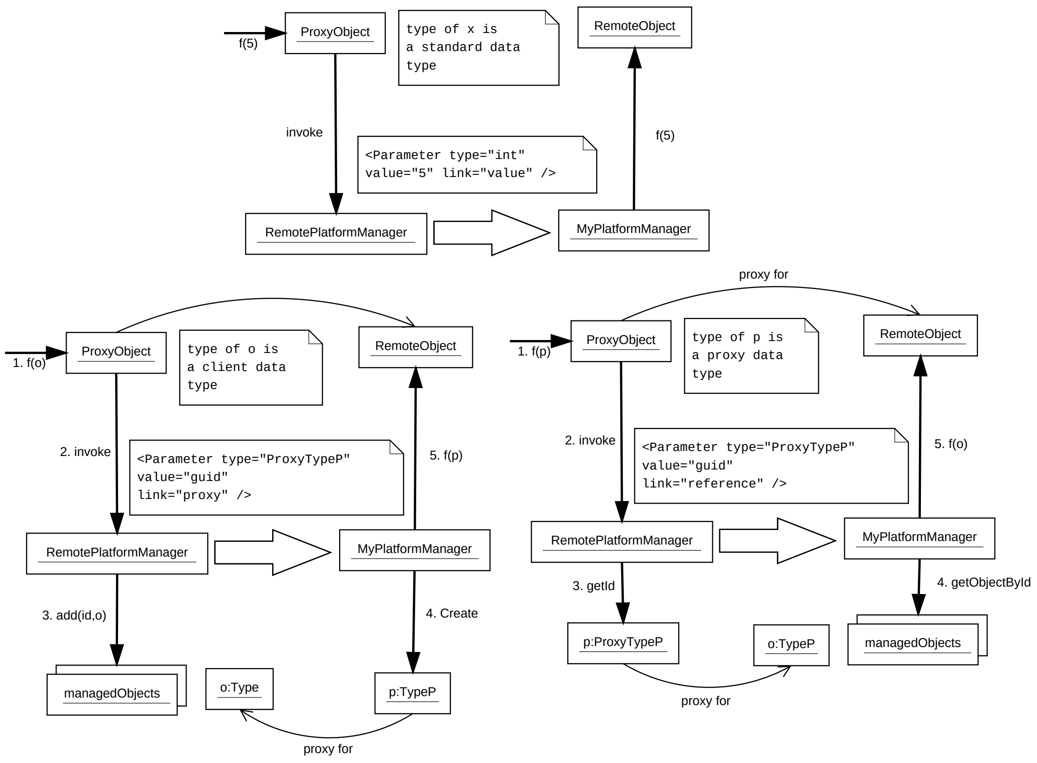

FIGURE 4. Passing parameter by value or by reference

Listing 1. Create Instance XML Request

```
<?xml version="1.0" encoding="utf-16"?>
<Request type="createInstance">
  <Class>cars.Car</Class>
  <Guid />
  <MethodName />
</Request>
```

Listing 2. Invoke XML Request

```
<?xml version="1.0" encoding="utf-16"?>
<Request type="invokeOnInstance">
  <Class />
  <Guid>63eba091-4ec1-4be1-bc9c-0f656efddc54</Guid>
  <MethodName>setPrice</MethodName>
  <Parameter type="int" value="12" link="value"/>
```

```
</Request>
```

Listing 3. Delete Instance XML Request

```
<?xml version="1.0" encoding="utf-16"?>
<Request type="delete">
  <Class />
  <Guid>63eba091-4ec1-4be1-bc9c-0f656efddc54</Guid>
  <MethodName />
</Request>
```

The type attribute of the Request element can have the following values:

- *createInstance* in order to create a remote object. In this situation the element Class is used in order to identify the remote object class. A constraint on the remote object is to have a parameter less constructor. Using reflection the remote object is instantiated and an object identifier is generated and returned.
- *invokeOnInstance* in order to invoke a method on a remote object. In this case the Guid element is used to identify the remote object. The method name and the list of parameters is used to identify the remote object's method. The method is invoked using reflection and the result is returned. The response XML will be presented later.
- *delete* in order to delete an instance of an object. In this case the Guid is used to identify the remote object and to remove it from the remote objects hash table.

In the following paragraph we discuss the *Parameter* XML element. The attribute *link* can have the following two values:

- *value* in this case the *value* attribute represents the parameter value
- *reference* in this case the *value* attribute represents the object identifier for a remote object.

The *type* attribute represents the parameter type, it can be one of the following standard type *String*, *byte*, *short*, *int*, *long*, *char*, *float*, *double*, *boolean* or a full remote type name.

The following three listings present the XML responses for the three request types:

Listing 4. Create Instance XML Response

```
<?xml version="1.0"?>
<Response type="Done">
  <ErrorMessage/>
  <Guid>85eec14b-f6ba-4783-af87-0d69fda884c2</Guid>
  <Value type="" value="" link="value"/>
</Response>
```

Listing 5. Invoke XML Response

```
<?xml version="1.0"?>
<Response type="Done">
  <ErrorMessage/>
  <Guid/>
  <Value type="void" value="null" link="value"/>
</Response>
```

Listing 6. Delete Instance XML Response

```
<?xml version="1.0"?>
<Response type="Done">
  <ErrorMessage/>
  <Guid/>
  <Value type="" value="" link="value"/>
</Response>
```

The *type* attribute can have the following two values:

- *Done* if the request was successfully executed. In this case the *Guid* is the object identifier if a create instance request was made and the *Value* element is the method return parameter if an invoke request was made. The *Value* has the same attributes like the *Parameter* attribute.
- *Error* if some error occurred on the *RemotePlatform*. In this case the *ErrorMessage* element contains the full message error.

4. Conclusions and Future Work

The main contributions of this paper are: a new approach of the interoperability issues between different platforms, regarding data types and objects, and a new development interoperability framework between Java and .NET, this can be easily extended to other platforms.

The novelty of the proposed solution resides in: using a proxy object in order to avoid serialization for each remote object and the parameter objects are executed in their own address space, that is the address space of the process which has created themselves.

A disadvantage of the proposed technique is the increasing of the execution time because there should be executed open connection operations and parameter values movements, but the main algorithms complexity is not affected. The proof of the proposed concept can be extended into a commercial application server that should allow the deploy/access process for the routines written on different platforms.

5. Acknowledgment

The authors wish to thank for the financial support provided from programs co-financed by the SECTORAL OPERATIONAL PROGRAMME HUMAN RESOURCES DEVELOPMENT, Contract **POSDRU 6/1.5/S/3** "Doctoral studies: through science towards society". (Paul Horaţiu Stan)

This research has been supported by the the Romanian CNCSIS through the PNII-IDEI research grant ID_550/2007. (Camelia Şerban)

The authors would like to thank professor Bazil Pârv and professor Ioan Lazăr for their precious help.

References

[1] Clemens Szyperski, *Beyond Object-Oriented Programming*, second edition, ACM Press New York 2002
[2] Ethan Cerami, *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI and WSDL*, O'Reilly Media, Inc. February 2002.
[3] Schach, Stephen R., *Object-Oriented Software Engineering* McGraw-Hill Science/Engineering September 2007.
[4] David Chappell. *Introducing SCA*, July 2007.
[5] *SCA Service Component Architecture, Assembly Model Specification* 2007.
[6] B Nolan, B Brown, L Balmelli, T Bohn, U Wahli *Model Driven Systems Development with Rational Products* IBM 2008.
[7] Erich Gamma, Richard Helm, and Ralph Johnson, and John Vlissides, *Design Patterns: 50 specific ways to improve your use of the standard template library.* Pearson Education, Inc 1995.
[8] Beck Kent. *Test Driven Development: By Example.* Addison-Wesley Professional, 2003.
[9] Christian Bauer, Gavin King, *Hibernate in Action: Practical Object/Relational Mapping*, Manning Publications August 2004.
[10] Vincent Massol, *JUnit in Action*, Manning Publications November 2003.
[11] William Grosso, *Java RMI (Java Series)*, O'Reilly Media, Inc. Octomber 2001.
[12] Ingo Szpuszta, Mario Rammer, *Advanced .NET Remoting 2nd Edition*, APRESS February 2005.
[13] James Snell *Programming Web Services with SOAP*
[14] http://www.w3.org/TR/soap, accessed on April 06, 2010
[15] http://en.wikipedia.org/wiki/CORBA, accessed on April 06, 2010
[16] http://msdn.microsoft.com/en-us/library/kew41ycz.aspx, accessed on April 06, 2010
[17] http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop, accessed on April 06, 2010

Babeş-Bolyai University, Department of Computer Science,, 1 Kogalniceanu St., 400084, Cluj-Napoca, Romania,

E-mail address: horatiu@cs.ubbcluj.ro, camelia@cs.ubbcluj.ro