SOA based Data Architecture for HTML5 Web Applications

Cătălin STRÎMBEI
Al. I. Cuza University of Iasi, Faculty of Economics and Business Administration
Department of Business Information Systems
linus@uaic.ro, catalin.strimbei@gmail.com

Web Services based architectures have already been established as the preferred way to integrate SOA specific components, from the front-end to the back-end business services. One of the key elements of such architecture are data-based or entity services. In this context, SDO standard and SDO related technologies have been confirmed as a possible approach to aggregate such enterprise-wide federation of data services, mainly backed by database servers, but not limited to them. In the followings, we will discuss an architectural purpose based on SDO approach to seamlessly integrate presentation and data services within an enterprise SOA context. This way we will outline the benefits of a common end-to-end data integration strategy. Also, we will try to argue that using HTML5 based clients as front end services in conjunction with SDO data services could be an effective strategy to adopt the mobile computing in the enterprise context.
Keywords: *SDO-Service Data Objects, Data Access Services, HTML5, Web Services, Cloud Computing, SQL, Data Integration, Mobile Computing*

1 Introduction: Data Architectures and Enterprise Mobile Computing

Although Service Oriented Architecture (SOA) have became a well established and widely acknowledged computing paradigm, its proliferation into the enterprise environment still struggles to resolve some crucial issues as heterogeneous data source integration. In fact, an entire class of SOA services is responsible with the data level and most often these are known as entity services. Their most obvious function is to abstract and then to integrate different data source providers (mostly database servers) across the enterprise, often very distinctive in their internal nature [1:293-323].

As such, the Enterprise Data Source Integration [2] process ultimately assumes some kind of global and centralized data model, but in a decoupled, transparent and technologically independent manner. This process could be regarded as a Federative Data Integration strategy that assumes two aspects:

- schema integration/denomination: the data services will use a common meta-data layer to describe composite data structures, integrity rules and quality rules, that will hide the schema complexities and manage the changes of the data structures in evolution;
- actual data integration: the data services will use a common data type system to operate on the data coming from a wide range of heterogeneous data sources. This data type system will provide a common data format layer that will hide the specific data formats and manage their changes in evolution.

The *Service Data Objects* (SDO) based approach to integrate data services could be easily extend-able to standardize service-data-based communication across the entire SOA stack, as in Figure 1. The "traditional" web presentation services, heavily based on server-side computing, seem to be a natural fit, but the adoption of the HTML5-based autonomous applications, easy to deploy in mobile computing context, might prove to be a very promising approach.

DOI: 10.12948/issn14531305/17.2.2013.07

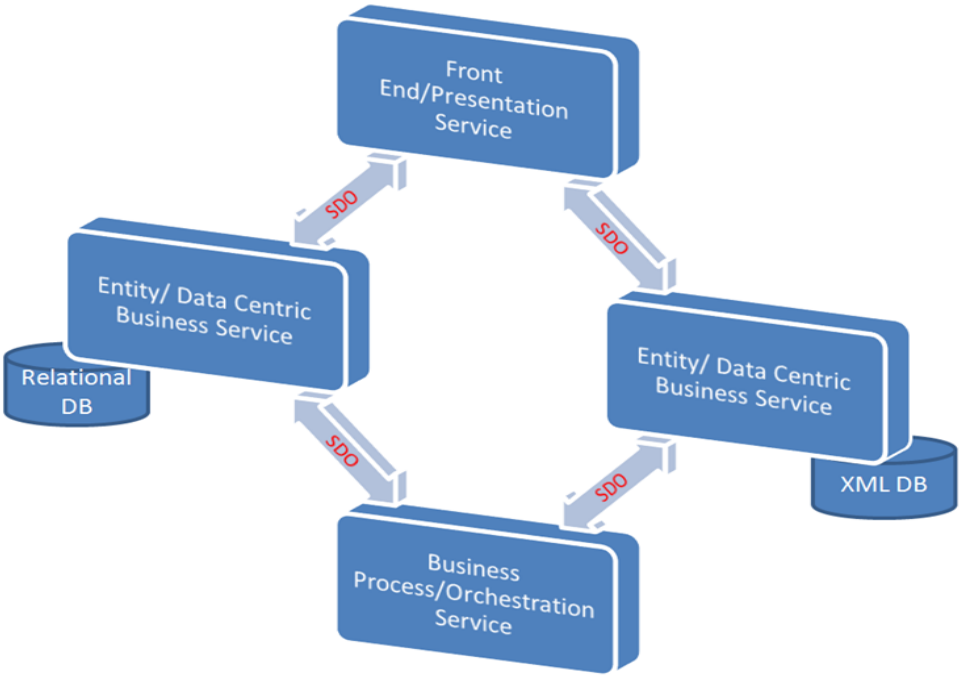

Fig. 1. SDO-based Data Integration Architecture

In short, SDO is just another standardized specification aimed to *handle data across multiple heterogeneous data source types*. Among the most interesting characteristics of the SDO programming model [3], [4] we could mention:

- offers a common platform to standardize access across heterogeneous data source types;
- offers support for both static and *dynamic data types*; also SDO data model is grounded on *data-graphs* which do not consist only into a single data set, but into a cluster of data-sets containing inter-referenced *data-objects*;
- offers support to query, update, and to *introspect data*: SDO data structures are strong typed by separated metadata objects that come along with data-graphs; moreover, the SDO data model defines a special kind of transactional structures, named *data object sequences*, to describe CRUD changes on interchangeable data objects;
- offers clean separation model of application code from data access code;
- uses a feature-rich, very adaptive XML based data type format.

The mobile computing paradigm proliferated in the consumer area, but still struggles in the enterprise area. In our opinion, the following issues could represent some key aspects to overcome this blocking stage in the mobile enterprise evolution:

- adopting a common access and processing paradigm to enterprise data (as in SOA);
- building feasible frameworks and architectures to develop presentation layer services across multi-device environments (as could be HTML5).

Anyone could note that the SOA approach already provides some reasonable answers to the first issue, but regarding the second one, many authors consider that the new HTML5 framework for web applications will represent one of the most prolific environments for the next generation of mobile applications taking into consideration the followings [5],[6],[7],[8]:

- the new HTML standard was headed to support multiple environments: desktop and mobile;
- there is already a strong mobile browser support to HTML5, taking into consideration HTML5 Mobile Frameworks like jQueryMobile (http://jquerymobile.com/), Jo (http://www.joapp.com/), Sproutcore

DOI: 10.12948/issn14531305/17.2.2013.07

(http://sproutcore.com/) or SenchaTouch (http://www.sencha.com/products/touch);
- HTML5 based applications could inherit the enterprise maturity of the existing web based computing frameworks;
- HTML5 proposes an improved Communication API and a new WebSocket protocol to integrate with business backend;
- in the HTML5 specs there are many other new and interesting capabilities like local storage, offline web applications, improved AJAX communication API, etc.

Consequently, the new HTML5 standard could represent a solid enterprise mobile computing foundation and a feasible path towards the proliferation of this kind of applications.

In the context of SOA and SDO marriage to effective integration of entity services, we will try to prove that the same SDO based integration approach could be used to achieve the integration of the HTML5-based presentation services with the enterprise data services. In figure 2 we have tried to envision a conceptual and synthetic perspective of such architecture.

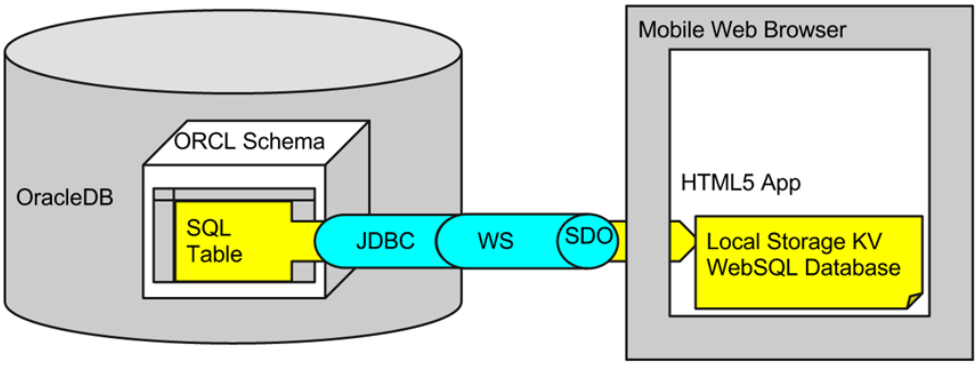

Fig. 2. SDO-based Conceptual Architecture for HTML5 based application

2 Server Side Architecture with SDO Data Services

At conceptual level, our vision concerning SDO potential to leverage data integration efforts is structured as a construction with three, partial concentric, levels (see figure 3):

- *Core-DIA (Data Interchange Architecture)* covering the reference data models formalized on SDO specifications;
- *DAS-DIA (Data Information Architecture)* covering the data interoperability protocols intra, inter and outer enterprise context, based on SDO data interchange specifications;
- *E-DIA (Enterprise Data Integration Architecture)* covering the enterprise view on data access services integrated at DAS-DIA level using SDO standard.

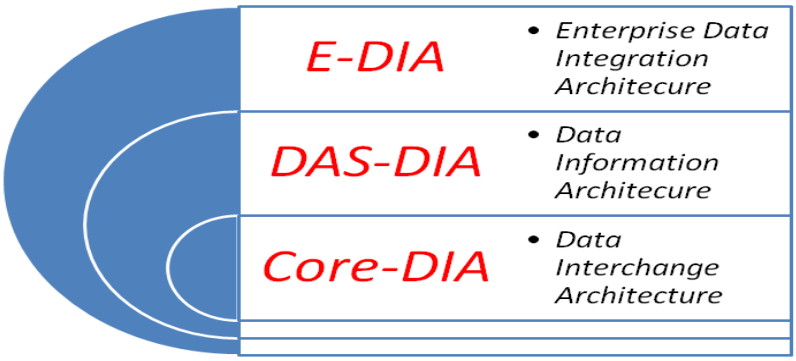

Fig. 3. SDO-based Architectural Layers

DOI: 10.12948/issn14531305/17.2.2013.07

Consequently, Core-DIA represents the key layer as the foundation to the entire integration infrastructure to be used among the enterprise-internal data services and further to internal or external presentation services, as mobile and autonomous HTML5 applications will be.

Our efforts to build a feasible Core-DIA layer yielded an architecture who's the most important components are rendered in Figure 4.

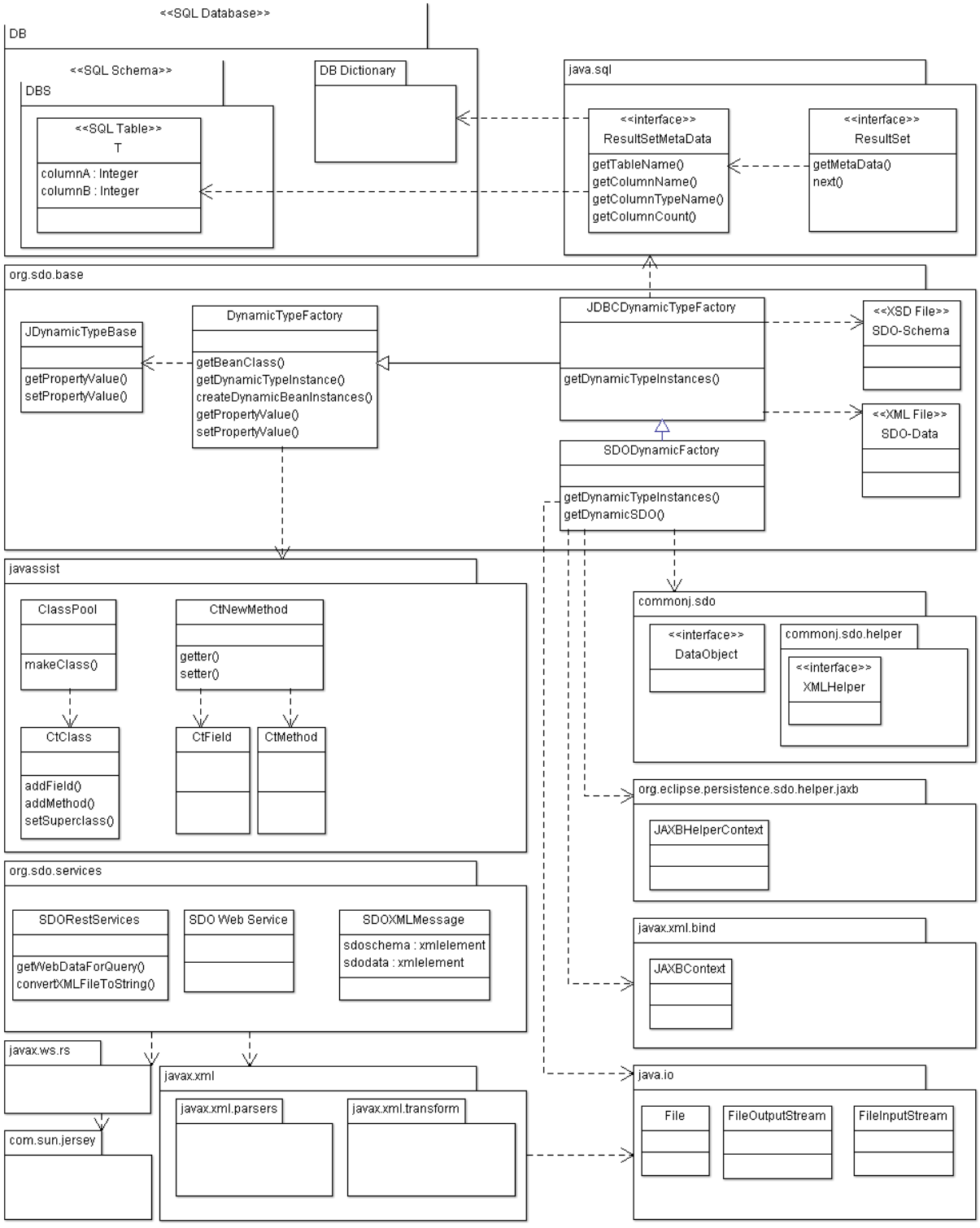

Fig. 4. SDO Data Integration Architectural Components

DOI: 10.12948/issn14531305/17.2.2013.07

To preserve the objective of achieving a SDO-based, dynamic, versatile and straightforward data integration model, we had to manage a complex of Java based technologies like Javaasist (to introduce dynamics into otherwise static and strong typed Java context), JDBC (to get a straightforward connection with SQL data source), JAXB (to convert Java beans to XML format), Eclipse-Link as an open source implementation of SDO standard.

Our first concern was to provide the possibility to work with dynamic typed data objects, a feature uncommon to most Java business applications. For this purpose we have used a special Java library to manipulate or to generate Java bytecode at runtime. Listing 1 tries to show how, starting from a collection containing field names, we've build a dynamic data type schema.

Listing 1. Generating Dynamic Data Object Types

```
public static Class getBeanClass(String beanName, Set<String> classFields)
throws Exception {
        // Create dynamic class
        ClassPool pool = ClassPool.getDefault();
        CtClass cc = null;
        try{
                cc = pool.get(beanName);
                return Class.forName(beanName);
        }
        catch(javassist.NotFoundException ex){
                cc = pool.makeClass(beanName);
                cc.setSuperclass(pool.get("org.open.core.GenericType"));
        }
        // Decorate with properties
        CtField field = null;
        String getterName = null;
        String setterName = null;
        CtMethod getter = null;
        CtMethod setter = null;
        for (String fieldName: classFields){
                field = CtField.make("private Object " + fieldName + ";", cc);
                cc.addField(field);
                getterName = "get" + fieldName.substring(0, 1).toUpperCase()
                        + fieldName.substring(1);
                getter = CtNewMethod.getter(getterName, field);
                cc.addMethod(getter);
                setterName = "set" + fieldName.substring(0, 1).toUpperCase()
                        + fieldName.substring(1);
                setter = CtNewMethod.setter(setterName, field);
                cc.addMethod(setter);
        }
        return cc.toClass();
}
```

Further we struggled to create or initialize the actual dynamic Java objects using the dynamic types from generated above. Listing 2 shows two sampling collections, one containing the property names and the other containing the actual values, and, finally, how to get, in first instance, a dynamic Java data bean, and further an entire Java data bean collection.

Listing 2. Generating Dynamic Java Data Beans

```
// Getting a dynamic data bean from a list of property values
public static Object getDynamicTypeInstance(Set<String> propList,
                List<Object> propBeanValues, Class dynaClass)
                throws Exception {
        Object bean = dynaClass.newInstance();
        Iterator<String> propListIterator = propList.iterator();
        Iterator<Object> propBeanValuesIterator = propBeanValues.iterator();
        String propBean;
        Object propValue;
        while (propListIterator.hasNext() && propBeanValuesIterator.hasNext()){
                propBean = propListIterator.next();
                propValue = propBeanValuesIterator.next();
                setPropertyValue(bean, propBean, propValue);
```

DOI: 10.12948/issn14531305/17.2.2013.07

```
        }
        return bean;
}
// Getting a dynamic data bean from an array of property values
public     static     Object     getDynamicTypeInstance(Set<String>     propList,     Object[]
propBeanValues,Class dynaClass)
                throws Exception {
        List<Object> propBeanValuesList = new ArrayList<Object>();
        for (Object propValue: propBeanValues){
                propBeanValuesList.add(propValue);
        }
        return getDynamicTypeInstance(propList, propBeanValuesList, dynaClass);
}
// Getting a dynamic data bean collection
public  static  List<Object>  getDynamicTypeInstances(Set<String>  propList,  List<Object[]>
propBeansValues)
                throws Exception {
        Class dynaClass = getBeanClass(DYNAMIC_CLASS_NAME, propList);
        List<Object> beans = new LinkedList<Object>();
        for (Object[] propBeanValues : propBeansValues){
                beans.add(getDynamicTypeInstance(propList, propBeanValues, dynaClass));
        }
        return beans;
}
```

But our main goal is to get a dynamic Java data bean collection from a data structure coming straight from the JDBC context. The data structure that is specific to the JDBC framework is the *java.sql.ResultSet*. Therefore we reused the logic from Listing 2 to build a method that could generate the data bean collection starting from a *ResultSet*, as in Listing 3.

Listing 3. Generating Dynamic Data Object for JDBC

```
public static List<Object> getDynamicTypeInstances(ResultSet rSetvalues)
throws Exception{
        Set<String> propList = new HashSet<String>();
        ResultSetMetaData meta = rSetvalues.getMetaData();
        for (int i=1; i <= meta.getColumnCount(); i++){
                propList.add(meta.getColumnName(i));
        }
        List<Object[]> propBeansValues = new LinkedList<Object[]>();
        Object[] propBeanValues;
        while (rSetvalues.next()){
                propBeanValues = new Object[meta.getColumnCount()];
                int i = -1;
                for(String prop: propList){
                        i++;
                        propBeanValues[i] = rSetvalues.getObject(prop);
                }
                propBeansValues.add(propBeanValues);
        }
        return getDynamicTypeInstances(propList, propBeansValues);
}
```

The dynamic nature of our process to generate Java data beans is very important because the *ResultSet* can be created in an offhand manner starting from a declarative SQL-SELECT phrase that could invoke any relation or table from database schema with any column projection on it.

The next step is essential regarding the conversion process of the Java data beans into Service Data Objects. Specifically, the final result consist in an XSD file that describe SDO schema and an XML file that will contain the serialization form of Service Data Objects. The Listing 4 suggests how to use SDO API implemented by the Eclipse-Link project to materialize SDO data objects from JDBC result set.

Listing 4. Generating Dynamic SDO schema and XML format

```
public void getDynamicSDO(ResultSet rSetvalues) throws Exception{
        // Generate Java Data beans to be converted in SDOs
```

DOI: 10.12948/issn14531305/17.2.2013.07

```
        List<Object> dataBeans = getDynamicTypeInstances(rSetvalues);
        if (dataBeans.isEmpty())
                throw new Exception("There is no JDBC data beans to convert!");
        Class dataBeanDynamicType = dataBeans.get(0).getClass();
        // Eclipse-Link SDO Infrastructure
        JAXBContext jaxbContext = JAXBContext.newInstance(dataBeanDynamicType);
        JAXBHelperContext jaxbHelperContext = new JAXBHelperContext(jaxbContext);
        // Generate SDO-XSD schema file
        MySchemaOutputResolver sor = new MySchemaOutputResolver(
                        "http://www.example.org",
                        "org/sdo/dynamicSDO.xsd");
        jaxbContext.generateSchema(sor);
        FileInputStream fileInputStream =
                new FileInputStream("org/sdo/dynamicSDO.xsd");
        jaxbHelperContext.getXSDHelper().define(fileInputStream, "dynamicSDO.xsd");
        File file = new File("org/sdo/dynamicSDO.xsd");
        fileInputStream = new FileInputStream(file);
        XMLDocument xsdDocument =
                jaxbHelperContext.getXMLHelper().load(fileInputStream);
        jaxbHelperContext.getXMLHelper().save(xsdDocument, System.out, null);

        // Generate SDO-XML data file
        DataObject sdo = null;
        File fileSDO = new File("org/sdo/dynamicSDO.xml");
        XMLDocument xmlDocument = jaxbHelperContext.getXMLHelper()
                .createDocument(sdo, "", "logs") ;
        for (Object dataBean: dataBeans){
                sdo = jaxbHelperContext.wrap(dataBean);
                jaxbHelperContext.getXMLHelper().save(sdo, null, null);
        }
        jaxbHelperContext.getXMLHelper().save(xmlDocument, System.out, null);
        jaxbHelperContext.getXMLHelper().save(xmlDocument,
                new FileOutputStream(fileSDO), null);
}
```

Some samples, of the two SDO related files (XSD and XML), that will be dynamically generated could look like in Listing 5.

Listing 5. SDO related files

```
File: dynamicSDO.xsd
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:complexType name="address">
                <xsd:sequence>
                        <xsd:element name="city" type="xsd:string" minOccurs="0" />
                        <xsd:element name="street" type="xsd:string" minOccurs="0" />
                        <xsd:element name="no" type="xsd:string" minOccurs="0" />
                        <xsd:element name="zipcode" type="xsd:string" minOccurs="0" />
                </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="customer">
                <xsd:sequence>
                        <xsd:element name="id" type="xsd:integer" minOccurs="0" />
                        <xsd:element name="name" type="xsd:string" minOccurs="0" />
                        <xsd:element name="address" type="address" minOccurs="0" />
                </xsd:sequence>
        </xsd:complexType>
</xsd:schema>

File: dynamicSDO.xml
<?xml version="1.0" encoding="UTF-8"?>
<customer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="customer">
   <id>1001</id>
   <name>First Customer</name>
   <address>
          <city>A City</city>
      <street>A Street</street>
      <no>123</no>
      <zipcode>700701</zipcode>
   </address>
</customer>
```

DOI: 10.12948/issn14531305/17.2.2013.07

The Java context, where SQL Data are translated to Service Data Objects, could be entirely operational within the Oracle database system (the Oracle instance). In this regard, we have managed to load all the support libraries into an Oracle schema in order to exploit the built-in support for Java-to-PL/SQL interoperability. This kind of interoperability was used to publish our OJDBC-EclipseLink-based SDO-engine as a PL/SQL Web service through DBMS_XDB and ULT_HTTP package functionality. Consequently, the starting point to deliver such Oracle database "native" web service is a plain simple PL/SQL procedure that will invoke the Java method from Listing 6. This method will be accessible in the PL/SQL context using an Oracle Object Type mapped on the Java class that will host this method. Loading and mapping Java classes as Oracle Object Types is a very interesting feature of Oracle 11g modern database. Another approach is to build a "natural" pure Java-based Rest service using JAX-RS API with an implementation like Jersey.

Listing 6. SDO related files

```
@GET
@Produces(MediaType.TEXT_XML)
@Path("/getsdocustomers")
public String getXMLCustomersData() throws IOException {
          String xmlTagSchema =
               "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>";
          String xmlTagData = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
          String beginEnvelope = "<envelope>\n";
          String endEnvelope = "\n</envelope>";

          String schema = convertXMLFileToString("sdo/data/customerSDO.xsd")
                          .replace(xmlTagSchema, "");
          String data = convertXMLFileToString("sdo/data/customerSDO.xml")
                          .replace(xmlTagSchema, "\n");

          String beginSDOData = "\n<sdodata>";
          String endSDOData = "\n</sdodata>";
          return beginEnvelope
                          + schema
                               + beginSDOData
                                      + data
                               + endSDOData
                          + endEnvelope;
}
private String convertXMLFileToString(String fileName) {
        try{
          DocumentBuilderFactory documentBuilderFactory =
                          DocumentBuilderFactory.newInstance();
          InputStream inputStream =
                          Thread.currentThread().getContextClassLoader()
                          .getResourceAsStream(fileName);
          org.w3c.dom.Document doc =
                          documentBuilderFactory.newDocumentBuilder().parse(inputStream);
          StringWriter stw = new StringWriter();
          Transformer serializer = TransformerFactory.newInstance().newTransformer();
          serializer.transform(new DOMSource(doc), new StreamResult(stw));
          return stw.toString();
        }
        catch (Exception e) {
          e.printStackTrace();
        }
        return null;
}
```

3 HMTL 5 Client Side Architecture to Consume SDO Based Data Services

In the previous section we've tried to outline how could look like the server-side backend of the proposed SDO operational architecture. In the followings we will try to outline the HTML5-based front end, or the client-side of our construction. The actual architectural components of the client side are represented in the Figure 5.

DOI: 10.12948/issn14531305/17.2.2013.07

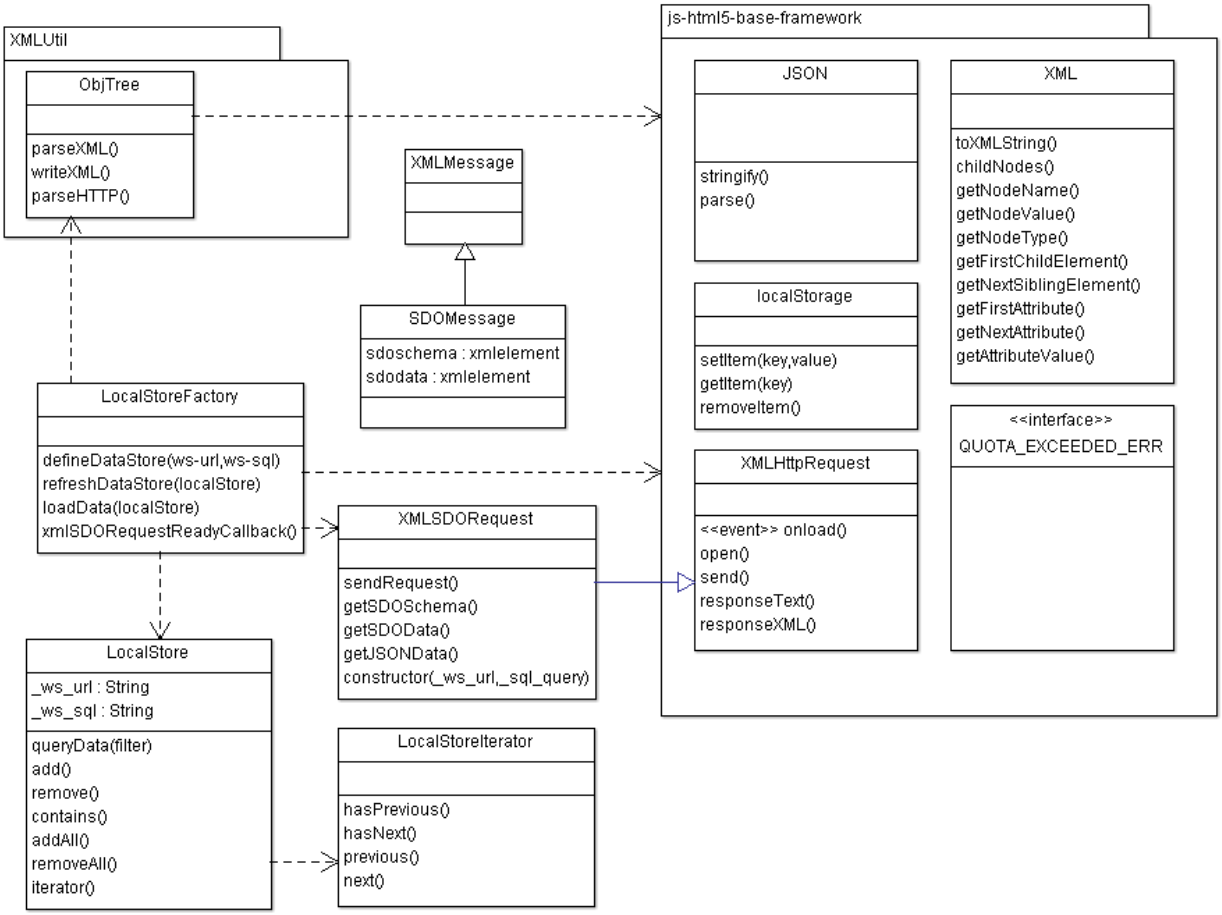

Fig. 5. Client side SDO architectural components

One could easily conclude that the architecture from the figure 5 is quite simple and straightforward, the only a bit more complex aspect is related to how to manage SDO data (*LocalStoreFactory*), consumed from the source Web Service (via *XMLSDORequest*), as a local updatable data cache (*LocalStore*) using HTML5 **localStorage** feature.

In this context, our request object used to invoke the backend data services could look like the one in the listing 7, where the *XMLSDORequest* extended the original *XMLHttpRequest* through the "traditional" prototyping mechanism specific to JavaScript language.

Listing 7. Adapted XML HTTP Request

```
var SDO = SDO || {};
SDO.XMLSDORequest = function(_wsurl, _wsquery){
        this.wsurl = _wsurl;
        this.wsquery = _wsquery;
        this.sdodata = null;
        this.storeFactoryCallback = null;

        this.execute = function(_storeFactoryCallback){
                this.storeFactoryCallback = _storeFactoryCallback;
                var url = wsurl + "?query=" + wsquery;
                open("GET", url);
                send(null);
        };

        this.onload = function(){
                if (this.status == 200){
                        var xotree = new XML.ObjTree();
```

DOI: 10.12948/issn14531305/17.2.2013.07

```
                        this.sdodata = xotree.parseXML(reqRest.responseText);
                        // callback
                        this.storeFactoryCallback(this);
                }else{
                        console.log("REST Request problem or ERROR !");
                }
        };

        this.getSDOSchema = function(){
                return this.sdodata["envelope"]["xsd:schema"];
        };

        this.getSDOData = function(){
                return this.sdodata["envelope"]["sdodata"];
        };

};
SDO.XMLSDORequest.prototype = new XMLHttpRequest();
```

We've implemented a rough local mini-database mechanism by leveraging the *localStorage* specific functionality of HTML5 through the *LocalStore* object summarized in Listing 8.

Listing 8. LocalStore as base data structures of HTML5 localStorage

```
SDO.LocalStore = function(_storeid, _wsurl,_wssql, _keyName){
        this.id = _storeid;
        this.wsurl = _wsurl;
        this.wsquery = _wsquery;
        this.keyName = _keyName;
        this.schema = null;
        // add new data
        this.add = function(data){
                if (!validateDataConformance(data))
                        throw new TypeError("Schema non conformance!");
                var storedHashKey = computeHashKey(this.id, data[this.keyName]);
                var storedData = getStoreData(storedHashKey);
                if (storedData)
                        throw new Error("Uniqueness non conformance!");
                // persist data
                localStorage.setItem(storedHashKey, data);
        };
        this.addAll = function(dataArray){
                for(var i in dataArray)
                        this.add(dataArray[i]);
        };
        // check data against this.schema
        var validateDataConformance = function(data){
                ... ... ...
                return true;
        };
        // restore persistent data from localStorage
        var getStoreData = function(hashKey){
                return localStorage.getItem(hashKey);
        };
        // return hash value from store id and key value
        var computeHashKey = function(){
                ... ... ...
                return hKey;
        };
        this.removeAll = function(){
                ... ... ...
        };
};
```

Additionally, taking into account the well known "Factory" design pattern, we've defined a separated JavaScript object to define and manage these data local stores. The *LocalStoreFactory* will create the local store definitions and will persist their metadata in the *localStorage*. Also, this object will invoke *XMLSDORequest* to asynchronously load data from remote/backend SDO-based data services

DOI: 10.12948/issn14531305/17.2.2013.07

into local data stores. A short definition of such object is presented in the Listing 9.

Listing 8. LocalStoreFactory to manage local stores

```
SDO.LocalStoreFactory = function(){
        // create new LocalStore and persist metadata
        this.defineDataStore = function(_wsurl,_wssql, _keyName){
                var storeHashId = computeStoreHashId(_wsurl,_wssql);
                var storeDef = {wsurl: _wsurl, wssql: _wssql};
                localStorage.setItem(storeHashId, storeDef);
        };
        // load data in store
        this.loadData = function(localStore){
                var dataRequest =
                        new SDO.XMLSDORequest(localStore.wsurl, localStore.wsquery);
                dataRequest.execute(this.xmlSDORequestReadyCallback);
        };
        // callback
        this.xmlSDORequestReadyCallback = function(sdoRequest){
                var storeHashId = computeStoreHashId(sdoRequest.wsurl,sdoRequest.wssql);
                var localStore = localStorage.getItem(storeHashId);
                if(localStore.schema === "undefined")
                        localStore.schema = sdoRequest.getSDOSchema();
                localStore.removeAll();
                localStore.addAll(sdoRequest.getSDOData());
        };
        // return hash value url and sql
        var computeStoreHashId = function(){
                ... ... ...
                return hKey;
        };
};
```

4 Conclusions

In this paper we have struggled to prove that the conceptual architecture proposed in Figure 1 could be feasible and materialized into a real computing environment as the one from Figure 6.

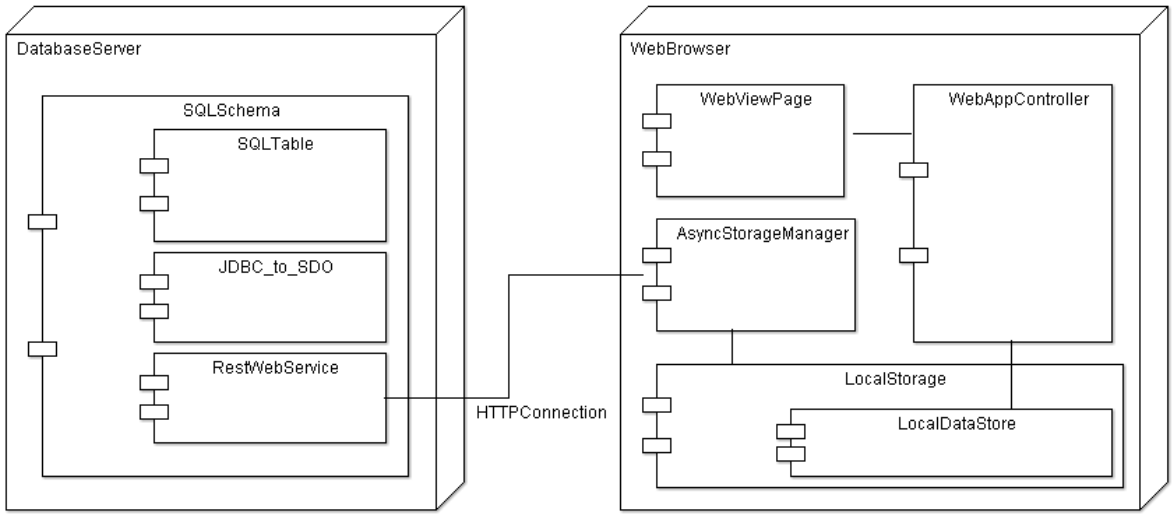

Fig. 6. The deployment architecture of a SDO-based Architecture ready for HTML5 based application

Beyond the SDO potential as a federative data integration strategy in the SOA context, we have tried to argue another two promising points in the actual and future enterprise environment:

- SDO versatility to be used in presentation layer;
- HTML based mobile computing path to enterprise data integration by leveraging SDO-based web services.

DOI: 10.12948/issn14531305/17.2.2013.07

We are aware that we only proved the possibility to build such wide-integration architecture. Other conceptual and experimental efforts are necessary to prove the real feasibility of this architecture taking into consideration at least the following questions:

- performance issues from simple to complex query tasks, or scalability (some benchmark tests need to be run);
- developing complexity and developers' acceptance;
- productivity and supporting tools (MDA with modeling tools based on UML meta-models seems to be a promising strategy);
- security issues concerning access control to enterprise data assets;
- SDO transaction management in the context of inter-architectural levels.

References

[1] T. Erl, SOA Principles of Service Design, Prentice Hall, Crawfordsville, Indiana, USA, 2008, ISBN-13: 9780132344821

[2] D. Fotache, L. Hurbean, O Dospinescu, V. Pavaloaia (2010). *Procese organizaţionale şi integrare informaţională*. Editura Universitatii "Al. I. Cuza" Iaşi.

[3] L. Resende, *Handling heterogeneous data sources in a SOA environment with service data objects (SDO)*, SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, ACM, New York, NY, USA, 2007, pp. 895–897.

[4] B. Portier, F. Budinsky, *Introduction to Service Data Objects. Next-generation data programming in the Java environment*, IBM developerWorks; 2004; http://www.ibm.com/developerworks/java/library/j-sdo/

[5] P. Lubbers, B. Albers, F. Salim, *Pro HTML5 Programming, Second Edition*, Publisher: Apress; 2 edition (November 30, 2011)

[6] D. Oehlman, S. Blanc, *Pro Android Web Apps: Develop for Android Using HTML5, CSS3 & JavaScript*, Publisher: Apress; 1 edition (February 22, 2011)

[7] S. Ritu, 10 OpenSource HTML5 Mobile App Development Frameworks, Available: http://www.toolsjournal.com/mobile-articles/item/1189-10-open source-html5-mobile-app-development-frameworks

[8] R. Ghatol, Y. Patel, *Beginning* PhoneGap*: Mobile Web Framework for JavaScript and HTML5*, Publisher: Apress; 1 edition (February 24, 2012)

Cătălin STRÎMBEI has graduated the Faculty of Economics and Business Administration of Al. I. Cuza University of Iaşi in 1997. He holds a PhD diploma in Cybernetics, Statistics and Business Informatics from 2006 and he has joined the staff of the Faculty of Economics and Business Administration as teaching assistant in 1998 and as senior lecturer in 2005. Currently he is teaching *Object Oriented Programming*, *Software Development Environments* and *Database Design and Administration* within the Department of Business Information Systems, Faculty of Economics and Business Administration, Al. I. Cuza University of Iaşi. He is the author and co-author of four books and over 25 journal articles in the field of object oriented development of business applications, databases and object oriented software engineering.

DOI: 10.12948/issn14531305/17.2.2013.07