Service Oriented Architecture Based Electric Power Real-time Data Warehouse

T. Charan Singh¹, G. Soma Sekhar², M.V.Bramhananda Reddy³, C Venkateswarlu Sonagiri⁴
¹²³⁴Dept. of CSE, Holy Mary Institute of Technology & Science, Hyderabad, Andhra Pradesh, INDIA

ABSTRACT
Currently traditional data warehouse is no longer satisfying the electric power enterprises decision making support, the real time data warehouse can support the electric power enterprise in daily tactical operation support, and the real time data warehouse implementation method, can make feasible real time data warehouse architecture based on Service oriented architecture. This service oriented architecture based real time data warehouse architecture uses the web service to pack the various source electric power data base systems, and the various changed data was captured by the data capture web service. When it comes to the data transformation and flow, the update strategy based on message queue and XML are used for the real time updating. Also, the multi-level real-time data cache is used for real-time data storage. Using this real time data warehouse architecture, we can implement the electric power real-time data warehouse easily.

Keywords—Capture-Transformation-Flow, data transformation, electric power enterprise, Object Access Protocol Web Services Description Language,

I. INTRODUCTION
These tiers or layers must be seamlessly integrated and function as one to ensure the immediate success and long-term benefits of a real-time data warehouse. The most important technology is the real-time data update of the data warehouse. But the traditional ETL process cannot support the exact real time data updating. After the Service Oriented Architecture technology appeared, we can do the real-time acquisition based on Service Oriented Architecture and web service. Most researchers proposed the CTF (Capture-Transformation-Flow) process to update date in real-time data warehouse. CTF is a simple and effective data transfer technology in the heterogeneous systems. This is the real-time date updating process. Data can be transferred to data warehouse phase table from the OLTP system in a very low-delay when changed. This Service Oriented Architecture based real-time data warehouse architecture uses the web service to pack the various source electric power database systems, and the various changed data was captured by the data capture web service. When it comes to the data transformation and flow, the update strategy based on message queue and xml are used for the real-time updating. Also the multi-level real-time data cache is used for real-time data storage. Using this real-time data warehouse architecture, we can implement the electric power real-time data warehouse easily.

II. SERVICE ORIENTED ARCHITECTURE AND WEB SERVICES

The definition of "Service Oriented Architecture" is discussed in detail in "Introduction to Service-Oriented Architecture." Essentially, Service Oriented Architecture is a software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations and interface calls. Service Oriented Architecture would be better-named "interface-oriented architecture." Service Oriented Architecture is a relationship of services and service consumers, both software modules large enough to represent a complete business function. Services are software modules that are accessed by name via an interface, typically in a request-reply mode. Service consumers are software that embeds a service interface proxy (the client representation of the interface). Web services are defined elsewhere in Gartner research. Simply speaking, any software that uses the standards Web Services Description Language (WSDL), Simple Object Access Protocol (Service Oriented Architecture) or Universal Description, Discovery and Integration (UDDI) is a Web service. [2]

As evidenced by the definitions, Web services are about technology specifications, whereas Service Oriented Architecture is a software design principle. Notably, Web services' WSDL is an Service Oriented Architecture-suitable interface definition standard: this is where Web services and Service Oriented Architecture fundamentally connect. Those who see Web services as architecture regard WSDL as the definitive standard of Web services (others see Service Oriented Architecture as a definitive standard for Web services — this is
a view of Web services as a communication method). In practical use, the ubiquitous Web services standards enhance the mainstream appeal of Service Oriented Architecture design.

Bridging the gap between BI & Service Oriented Architecture

We already know that business intelligence (BI) can bring many benefits to an organization. Through consolidating, aggregating, and analyzing data, BI can provide many insights into what is currently happening, as well as what is going to happen within the organization. BI allows for identifying trends of where an organization is going or should be going. The road to BI usually starts with extract transform and load (ETL). ETL is, generally speaking, a process in data warehousing that involves:

- Extracting data from external and internal data sources.
- Transforming the data to fit business needs.
- Loading the transformed data into the data warehouse (or data mart).

Basically, for us to achieve BI Nirvana, all we need is "just" one input: data. BI needs the data that is hidden within the organization's systems. In the last few years, we have seen the advance of service-oriented architecture (Service Oriented Architecture) to the forefront of IT architectures. As the hype begins to clear and organizations make the transition to Service Oriented Architecture, the data that BI requires is suddenly scattered between multiple services and hidden behind contracts. Looking at the Service Oriented Architecture components in Figure 2.1, we can see that apart from the obvious component—the service—Service Oriented Architecture has several other components that are related to the interface of the service:

- Contract that the service implements
- Endpoints, where the service can be contacted
- Messages that are moved back and forth between the service and its consumers
- Policies to which the service adheres.
- Consumers that interact with the service

This, along with Service Oriented Architecture tenets like "share schema, not data" and "services should be autonomous," tells us Service Oriented Architecture really cares about its interfaces. This emphasis on communication through rigorously defined interfaces is exactly what brings technical and business advantages of loose coupling, flexibility, and agility to Service Oriented Architecture.

Figure 2.1: Service Oriented Architecture components and their relations

There is a real impedance mismatch here, with BI pooling in one direction of intimate understanding of the data and Service Oriented Architecture pooling in the other direction of isolating internal data behind interfaces. The service's internal data should never be exposed outside of the service, yet this is the very data that BI wants. Shows that only one-third of respondents reported that they believe their internal IT personnel to have the knowledge and skills to implement BI services.

It seems that there are two options: either to go directly at the data and invalidate some of our Service Oriented Architecture principles (like "share schema, not data") or to try to make do with the contracts that we have in place and hope that we will have enough data for BI.

The first option is to get the data that BI needs by using the same ETL processes that have proven themselves in the past. Service Oriented Architecture presents a little challenge to ETL, as you have to integrate data from many dispersed locations (services). However, converging data from multiple resources is not a new problem for either BI or ETL. Large enterprises already have a lot of data sources: ERP, CRM, all of those departmental data silos, and whatnot. ETL with Service Oriented Architecture might even be easier, considering that Service Oriented Architecture promises that the enterprise data would be woven into a cohesive fabric and not some point-to-point integration spaghetti. ("Promises" being the operative word here; actually achieving a cohesive fabric of services is not an easy feat. But that is a topic for another article—or book, for that matter.) [3]

As I mentioned earlier, ETL is mature and has a proven record as a basis for building successful BI solutions. However, using ETL basically negates most of the benefits that made us pursue Service Oriented Architecture in the first place. One of the main problems in the pre-Service Oriented Architecture era (which is still the reality in many organizations) is what is known as integration spaghetti. Consider the situation in Figure 2.2. Historically, each department builds its own systems. The result is isolated or stovepipe systems, as new business requirements emerged. Then, systems needed to share data, and new point-to-point interfaces were added to solve the integration needs. As people use the systems, they find that they need information from other systems, and point-to-point integration emerges. Figure 2.2 shows four types of point-to-point integrations: ETL (extract, transform Load), which is a DB-to-DB relationship; online and file-based, both of which are application-to-application relationships; and direct connection to a DB, which is an application-to-database relationship. Note that this is not an exhaustive list; there are additional relation types, such as replication, message-based, and others that are not expressed in Figure 2.2.

The end result is spaghetti of systems. Making changes in one system has ripple effects, with results that are unpredictable. The Service Oriented Architecture emphasis on general interfaces and autonomy aims to solve these problems.
Adding ETL as a direct pipeline into the services’ data just adds a new point-to-point interface—cracking the Service Oriented Architecture “interface armor” and introducing a dependency between the BI and the service. It also opens the door for other workarounds. (If we can do that for BI, why not do the same for other applications, services, or systems?)

A variation on doing ETL can be to replicate the Service Oriented Architecture data into an external database, and then do ETL on that data. However, it is exactly the same as using ETL on the service’s database, as we are still bypassing the contract and we are still coupled to the structure of the internal data. Okay, so using ETL is probably not the best option. So, let’s try to see if the second option of building on the Service Oriented Architecture principles by using contracts will fare better at integrating BI and Service Oriented Architecture. [3]

III. SERVICE-ORIENTED ARCHITECTURE

In computing, Service-oriented architecture (Service Oriented Architecture) provides methods for systems development and integration where systems group functionality around business processes and package these as interoperable services. Service Oriented Architecture also describes IT infrastructure which allows different applications to exchange data with one another as they participate in business processes. Service-orientation aims at a loose coupling of services with operating systems, programming languages and other technologies which underlie applications. Service Oriented Architecture separates functions into distinct units, or services, which developers make accessible over a network in order that users can combine and reuse them in the production of business applications. These services communicate with each other by passing data from one service to another, or by coordinating an activity between two or more services. Many commentators see Service Oriented Architecture concepts as built upon and evolving from older concepts of distributed computing and modular programming.

Companies have long sought to integrate existing systems in order to implement information technology (IT) support for business processes that cover all present and prospective systems requirements needed to run the business end-to-end. A variety of designs serve this end, ranging from rigid point-to-point electronic data interchange (EDI) interactions to web auctions. By updating older technologies, for example by Internet-enabling EDI-based systems, companies can make their IT systems available to internal or external customers; but the resulting systems have not proven flexible enough to meet business demands, which require a flexible, standardized architecture to better support the connection of various applications and the sharing of data. [5]
support, reliability, etc. This article defines Service Oriented Architecture and includes detailed discussion on several issues that arise when applying Service Oriented Architecture to industrial systems.

One can define a service-oriented architecture (Service Oriented Architecture) as a group of services that communicate with each other. The process of communication involves either simple data-passing or two or more services coordinating some activity. Intercommunication implies the need for some means of connecting services to each other. [7]

Architectures can operate independently of specific technologies. Designers can implement Service Oriented Architecture using a wide range of technologies, including Service Oriented ArchitectureP, REST, RPC, DCOM, CORBA, Web Services or WCF (Microsoft's implementation of Web service forms a part of WCF). Service Oriented Architecture can be implemented using one or more of these protocols and, for example, might use a file-system mechanism to communicate data conforming to a defined interface-specification between processes conforming to the Service Oriented Architecture concept. The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without a service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks. In order to efficiently use a Service Oriented Architecture, one must meet the following requirements: Interoperability between different systems and programming languages provides the basis for integration between applications on different platforms through a communication protocol. One example of such communication is based on the concept of messages. Using messages across defined message channels decreases the complexity of the end application thereby allowing the developer of the application to focus on true application functionality instead of the intricate needs of a communication protocol. Desire to create a federation of resources. Establish and maintain data flow to a federated data warehouse. This allows new functionality developed to reference a common business format for each data element. The following guiding principles define the ground rules for development, maintenance, and usage of the Service Oriented Architecture: Reuse, granularity, modularity, composability, componentization, portability, and interoperability Standards compliance (both common and industry-specific) Services identification and categorization, provisioning and delivery, and monitoring and tracking The following specific architectural principles for design and service definition focus on specific themes that influence the intrinsic behavior of a system and the style of its design:

**Service encapsulation** - Many web-services are consolidated to be used under the Service Oriented Architecture. Often such services were not planned to be under Service Oriented Architecture.

**Service loose coupling** - Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.

**Service contract** - Services adhere to a communications agreement, as defined collectively by one or more service description documents.

**Service abstraction** - Beyond what is described in the service contract, services hide logic from the outside world.

**Service reusability** - Logic is divided into services with the intention of promoting reuse. Service compensability - Collections of services can be coordinated and assembled to form composite services. Service autonomy – Services have control over the logic they encapsulate. Service optimization – All else equal, high-quality services are generally considered preferable to low-quality ones.

**Service discoverability** – Services are designed to be outwardly descriptive so that they can be found and accessed via available discovery mechanisms. The following references provide additional considerations for defining a Service Oriented Architecture implementation: Service Oriented Architecture Reference Architecture provides a working design of an enterprise-wide Service Oriented Architecture implementation with detailed architecture diagrams, component descriptions, detailed requirements, design patterns, opinions about standards, patterns on regulation compliance, standards templates etc. Life cycle management Service Oriented Architecture Practitioners Guide Part 3: Introduction to Services Lifecycle introduces the Services Lifecycle and provides a detailed process for services management though the service lifecycle, from inception through to retirement or repurposing of the services. It also contains an appendix that includes organization and governance best practices, templates, comments on key Service Oriented Architecture standards, and recommended links for more information. In addition, the following factors should be taken into account when defining a Service Oriented Architecture implementation: efficient use of system resources, service maturity and performance EAI (Enterprise Application Integration)

### IV. WEB SERVICES APPROACH

Web services can implement a service-oriented architecture. Web services make functional building blocks accessible over standard Internet protocols independent of platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network-enabled. Each Service Oriented Architecture building block can play one or both of two roles:

**Service provider**

The service provider creates a Web service and possibly publishes its interface and access information to the service registry. Each provider must decide which services to expose, how to make trade-offs between security and easy availability, how to price the services, or, if they are free, how to exploit them for other value. The provider also has to decide what category the service should be listed in for a given broker service and what sort of trading partner agreements are required to use the service. It registers what services are available within it, and lists all the services for which it can provide access information. In addition, the following factors should be taken into account when defining a Service Oriented Architecture implementation: efficient use of system resources, service maturity and performance EAI (Enterprise Application Integration)
Furthermore, the amount of the offered information has to be decided. Some brokers specialize in many listings. Others offer high levels of trust in the listed services. Some cover a broad landscape of services and others focus within an industry. There are also brokers that catalog other brokers. Depending on the business model, brokers can attempt to maximize look-up requests, number of listings or accuracy of the listings. The Universal Description Discovery and Integration (UDDI) specification defines a way to publish and discover information about Web services. Other service broker technologies include for example ebXML (Electronic Business using eXtensible Markup Language) and those based on the ISO/IEC 11179 Metadata Registry (MDR) standard. List all the services provided by various service providers.

**Service requester**

The service requester or Web service client locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its Web services. This Brokers will look for services. Which service they need they have to take into the Brokers then bind with respective service then use it. They can access multiple, if the service provide multiple services.

Architectures can operate independently of specific technologies. Designers can implement Service Oriented Architecture using a wide range of technologies, including Service Oriented Architecture, REST, RPC, DCOM, CORBA, Web Services or WCF (Microsoft's implementation of Webservice forms a part of WCF). Service Oriented Architecture can be implemented using one or more of these protocols and, for example, might use a file-system mechanism to communicate data conforming to a defined interface-specification between processes conforming to the Service Oriented Architecture concept. The key is independent services with defined interfaces that can be called to perform their tasks in a standard way, without a service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks. [7]

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Service-oriented architectures are not a new thing. The first service-oriented architecture for many people in the past was with the use DCOM or Object Request Brokers (ORBs) based on the CORBA specification. For more on DCOM and CORBA, see Prior service-oriented architectures.

**V. ALGORITHMS**

Data transmit based on XML and message Queue mechanism

Message queue service is a kind of loose coupling distribute application integration mode. In this mode, the sending and receiving are asynchronous. That is to say the sender and receiver can execute the other codes without wait for the succeed message from the other side. This method greatly increased the capability of affair process. The message send mechanism has the resume ability while breakdown, which make it possible for the integrating of the sender and receiver which were built on different physical platforms. In the data transmission process, the XML form is used as the intermediate data format, which provides a unified data accessing, transmission format. And it shields the difference of the data format stored in the different relational databases; facilitates the integration of the heterogeneous data source.

In this paper, the message queue manager (MQM) is used for the message management. It is the “heart” of the message middleware. It provides a message queue interface for the queue and message management of the procedure, in order to facilitate the procedure communication. The message queue manager uses the network equipment existed (e.g. TCP/IP, SNA or SPX) to transmit the message to other queue manager through the channel. The main function of the message manager is shown as following: Manage the message queue for the application procedure; Provide the program interface for application procedure (Message Queue Interface- MQI); Transmit the message to the other queue manager based on the existed network equipment. Update the database and queue simultaneously, so the PUT/GET can run simultaneously; Do the message partition and the reorganization if necessary, the manager can combine
some messages to one physical message and transmit it to the
destination, then the destination carries on automatic split and
reorganization (considering the capability). The data transmit
mode based on message queue and XML is shown in Figure 3.
Each cache maintains a message queue, and managed by the
message queue manager. The date format transmitted in the
queue is XML. The final data loads to the true data warehouse
similarly passes through a message queue maintains by the data
warehouse. After passed through all caches, the data was loaded
to the static data warehouse.
The system uses the multistage real-time data caches to
store the real-time data, simultaneously different caches store
data with different freshness, i.e. the data was generated in
different time. Along with the time increased, the preliminary
cache data will flow to high-level cache through message queue
Q, i.e. cache will be defers to the respective update cycle to carry
on the data the refresh, for example: In the system, Cache-0,
Cache-1, Cache-2, the Cache-3 update cycle respectively is 5
minute, 10 minute, 30 minute, 60 minutes, they deposit data in 5
minute, 10 minute, 30 minute, 60 minute separately. Figure 3.
Updating process

The algorithm for the cache-i and Qi updating:
```java
    The algorithm for the cache-i and Qi updating:
cache_updating {  While (Ti begins) { d = Qi.dequeue();
    foreach (m in M) integrate m into Ci; Select the dataset N
which older than Tt+1-Ti in Ci ; foreach (m in N) Qi+1.enqueue(n); return Ci, Qi+1; }
```
The algorithm for the data warehouse updating:
```java
Datawarehouse_Updation / While (Q is not empty) { d = Qi.dequeue (); Put d into dataset M; foreach (m in M), integrate
m into C; return C; } Cache-1 Data warehouse Cache-2 Cache-
Q1 Q2 Qi Q XML message …XML message
Use Case Specification for Cache Updating
Pre Condition: Data must be available from the transactions
being performed Main Flow: Create data queue Create data set
Until dataset empty insert into data queue
Until data queue empty write into data warehouse
Post Condition: Cache is update for data ware house access
Use Case Specification for Data warehouse Updating
Pre Condition: Data must be available in the XML format
Main Flow: Create Queue Retrieve XML messages Insert XML
message into dataset Until data queue empty write into data
warehouse Post Condition: The latest data is available to the user
for accessing information
Use Case Specification for Service Customer
Pre Condition: Consumer must request for the service.
Main Flow: Retrieve XML File Parse XML data Retrieve XML
Tags Write data into required format Post Condition: Data is
written in the required format which is specific to the service
being used.
Use Case Specification for Service Provider
Pre Condition: Web service is requested to use XML. Main Flow:
Request for web service Retrieve data from data warehouse Write
data into XML Send XML data Post Condition: Data is retrieved
from the warehouse and XML file being created is used as a
service.

VI. SYSTEM IMPLEMENTATION

The JAVA language was created by James Gosling in
June 1991 for use in a set top box project.[4] The language was
initially called Oak, after an oak tree that stood outside Gosling's
office - and also went by the name Green - and ended up later
being renamed to Java, from a list of random words.[5] Gosling's
goals were to implement a virtual machine and a language that
had a familiar C/C++ style of notation.[6] The first public
implementation was Java 1.0 in 1995. It promised "Write Once,
RunAnywhere" (WORA), providing no-cost runtimes on popular
platforms. It was fairly secure and its security was configurable,
allowing network and file access to be restricted. Major web
browsers soon incorporated the ability to run secure Java applets
within web pages. Java quickly became popular. With the advent
of Java 2, new versions had multiple configurations built for
different types of platforms. For example, J2EE was for
enterprise applications and the greatly stripped down version
J2ME was for mobile applications. J2SE was the designation for
the Standard Edition. In 2006, for marketing purposes, new J2
versions were renamed Java EE, Java ME, and Java SE,
respectively.
The Java programming language is a high-level language that
can be characterized by all of the following buzzwords:
In the Java programming language, all source code is first written
in plain text files ending with the .java extension. Those source
files are then compiled into .class files by the javac compiler. A
.class file does not contain code that is native to your processor; it
instead contains bytecodes — the machine language of the Java
Virtual Machine.[7] (Java VM). The java launcher tool then runs
your application with an instance of the Java Virtual Machine.

Figure 6.1 : An overview of the software development process.
Because the Java VM is available on many different operating
systems, the same .class files are capable of running on Microsoft
Windows, the Solaris™ Operating System (Solaris OS), Linux,
or Mac OS. Some virtual machines, such as the Java HotSpot
virtual machine, perform additional steps at runtime to give your
application a performance boost. This include various tasks such
as finding performance bottlenecks and recompiling (to native
code) frequently used sections of code.

TESTING AND SCREENS
Testing is a process, which reveals errors in the
program. It is the major quality measure employed during
software development. During testing, the program is executed
with a set of test cases and the output of the program for the test
cases is evaluated to determine if the program is performing as it
is expected to perform.

Testing Strategies:
The purpose of testing is to discover errors. Testing is
the process of trying to discover every conceivable fault or
weakness in a work product. It provides a way to check the
functionality of components, sub assemblies, assemblies and/or a
finished product. It is the process of exercising software with the
intent of ensuring that the Software system meets its
requirements and user expectations and does not fail in an
unacceptable manner. There are various types of test. Each test
type addresses a specific testing requirement.

Unit testing
Unit testing involves the design of test cases that
validate that the internal program logic is functioning properly,
and that program input produces valid outputs. All decision
branches and internal code flow should be validated. It is the
testing of individual software units of the application .it is done
after the completion of an individual unit before integration. This
is a structural testing, that relies on knowledge of its construction
and is invasive. Unit tests perform basic tests at component level
and test a specific business process, application, and/or system
configuration. Unit tests ensure that each unique path of a
business process performs accurately to the documented
specifications and contains clearly defined inputs and expected results.

**VII. RESULTS AND DISCUSSIONS**

**Figure 7.1:** The above screen shows the home page. It shows the services.

**Figure 7.2:** The above screen gives detailed information of electric database tables. It gives all the used tables.

**Figure 7.3:** The above screen shows here user is entering the Meter Number (Service Number) and its current reading.

**Figure 7.4:** The above screen shows after entering the data, pressing submit button shows that data is inserted successfully in the Data Ware-House.
Figure 7.5: The above screen shows checking the current reading of the meter.

Figure 7.6: The above screen shows Current Reading of the Meter is displayed.

Figure 7.7: The above screen shows this is the place where dynamic data is generated:

Figure 7.8: The above screen shows Dynamic data is generated and passed in the form of XML:

VIII. CONCLUSIONS

Using the Service Oriented Architecture technology, we can construct the loose coupling electrical power real-time data warehouse, not only uses the existing system, but also facilitates the system’s expansion. This system can provide reliable data transmission, satisfies the electric power enterprise’s request, and provides the formidable support for its decision-making. The real-time data warehouse application provide a good data environment for electric power enterprise’s information system, further more, the on-line analysis processing and the data mining technology provides enormous information support for its strategy and tactical decision.

FUTURE WORK

This system can be extended to work with mobile devices and efficient and dynamic update techniques can be incorporated in future so that the system efficiently works in the WI-FI mobile environment. This system can also replicate the data warehouse through several different geographical regions so as to maintain the consistency and integrity with efficient and faster access to the data.

REFERENCES

[1] LIU Jizhen, FANG Fang, NIU Yuguang. Real-time database in electric power enterprise
[3] Xu Li, Ma Ruixin. Research and Implementation of Service Oriented Architecture-Based Real-Time ETL.
[7] Zhou Long, Yao Yaowen. Discussion about ETL based on banking application