ISSN No:-2456-2165

A Generic Middleware for Integrating ERP and Web Applications

Abdulaziz Almuhaisin,
Saudi Aramco, PO Box 10760 Dhahran 31311 Saudi Arabia

Abstract:- A middleware is an application used to map and pipe outputs and inputs of differentheterogeneous applications to stick them together to enable them to share data back andforth. A client-side web application often reads and stores data in a server of backend Enterprise Resource Planning (ERP) system using a middleware.However, the current practice for building middleware is specific to each clientapplication such that the middleware needs to be updated each time when an interface in aclient side or a server side is changed. In addition, the middleware tends to be complex,and needs frequent maintenance.Accordingly, there exists a need for a generic middleware that can work with anyweb application to any data server without the need of updating the middleware in case ofan interface change, and that can be easily maintained without a frequent maintenance.

I. INTRODUCTION

A middleware is an application used to map and pipe outputs and input of different heterogeneous applications to stick them together to enable them to share data back and forth. Example of middleware include database middleware, application server middleware, message-oriented middleware, web middleware. (What is a middleware? [1]. In this more specific sense middleware can be described as the dash ("-") in client-server, or the -to- in peer-to-peer. Services that can be regarded as middleware include enterprise application integration, data integration, message oriented middleware (MOM), object request brokers (ORBs), and the enterprise service bus (ESB).[2] .

Each program typically provides messaging services so that different applications can communicate using messaging frameworks like simple object access protocol (SOAP), web services, representational state transfer (REST), and JavaScript object notation (JSON). While all middleware performs communication functions, the type a company chooses to use will depend on what service is being used and what type of information needs to be communicated. This can include security authentication, transaction management, message queues, applications servers, web servers, and directories. Middleware can also be used for distributed processing with actions occurring in real time rather than sending data back and forth [3].

II. MIDDLEWARE CHALLENGES

A. *Scalbility and application dynamic topology*

Scalability is defined as follows if an application grows or changed the middleware need to be flexible to adapt to such change without affecting the application performance or expected results. Middlewares that are efficient and generic must be capable of maintaining data integrity and performance levels as the application grows or enhanced. Application topology is dynamic with different requirement changes or feature additions or bug fixing. Middleware need to be robust and generic enough despite these dynamic changing of applications. [4]

B. *Quality of Service*

Increasing concerns about service quality have led to several proposals that advocate integrating QoS management into distribution infrastructures. QoS management at the middleware and application levels aims to control attributes such as response time, availability, data accuracy, consistency, and security level. Therefore, middleware should provide new mechanisms to maintain QoS over an extended period and even adjust itself when the required QoS and the state of the application changes.

C. *Performance*

Middleware systems rely on interception and indirection mechanisms, which induce performance penalties. Adaptable middleware introduces additional indirections, which make the situation even worse. This problem may be alleviated by various optimization methods, which aim at eliminating the unnecessary overheads by such techniques as in lining, i.e. injecting the middleware code directly into the application. Flexibility must be preserved, by allowing the effect of the optimizations to be reversed if needed [5]

III. IMPLEMENTING GENERIC MIDDLEWARE

To able to build a generic middleware, a protocol or a communicationstandard need to be established. The protocol will be divided into three main parts. Thefirst is agreeing on what the web application will send to the middleware. The second partis how the middleware will handle the request coming from a web application, how toauthenticate the application user, and how to fetch or post data to the ERP backend system.The final part is how to handle requests coming from the middleware, the design of thehttp handler in the backend ERP system, and an importing parameter that accepts theusername in each function that will be used to handle a client request.

ISSN No:-2456-2165

A. Creating the middleware API

This section presents the design principles considered in the architecture design. First fig 1.1 shows the architectural design to able to build a generic middle ware a protocol or a communication standard need to be established. The protocol will be divided into three main parts. The first is agreeing on what the web application will send to the middle ware. The second part is how the middle ware will handle the request coming from the web application and how to get the logged in user from active directory and how to fetch or post data to the SAP backend system. The final part is how to handle requests coming from themiddle ware and the design of the http handler in the backend ERP system.

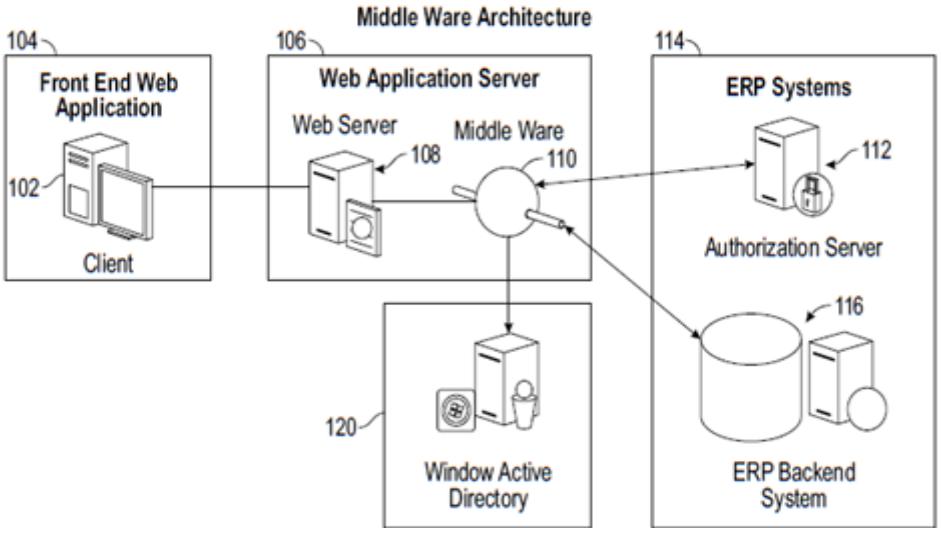

Fig. 1: Illustrates a middleware architecture in a system. Middlewarearchitecture includes a client, a web application server, ERP systems,and a Windows active directory. Client includes a front end web application.Web application server includes a web server and a middleware. ERP systemsincludes an authorization server and an ERP Backend system

B. Web application side

The web application need to send a POST request to the API with a JSON in the request body. The JSON request need to have the name of the data the same name of the SAP function module importing parameters. Also the POST request header need to consist of the following header fields:

- User name identifier (X-USER-IDNT)
- Password identifier (X-PASSWORD-IDNT)
- Target SAP system https handler URL (X-URL-IDNT)
- Operation code or function module name in the target SAP system (X-OPERATION_CODE)

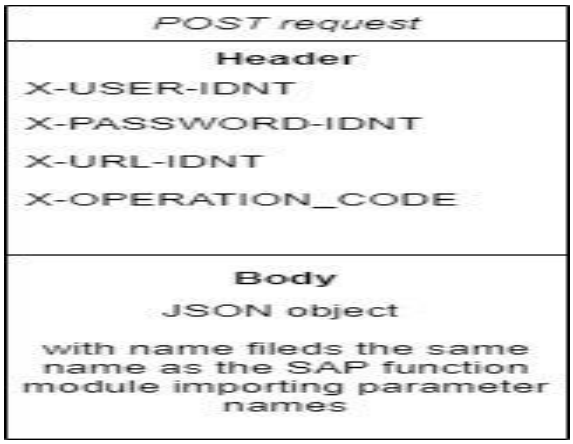

Fig. 3: Illustrates an exemplary POST request by a web application.

POST request comprises a header comprising a plurality of fieldsincluding a username identifier (X-USER-IDNT), a password identifier (X-PASSWORDIDNT),a target system HTTP Uniform Resource Locators (URL) identifier (X-URLIDNT),and an operation code (X-OPERATION_CODE). The operation code may be afunction module name in the target system. Values of these fields are identifiers and notthe actual information. For example, an identifier of a password is sent via a value in thepassword identifier, but not the actual password. As such, the username identifier identifiesa username. The password identifier identifies a password. The target system HTTP URL identifier identifies an address of a target system. The operation code identifies a code ora function module name to run on a target system.

ISSN No:-2456-2165

These are identifiers and not the actual information. For example, the identifier of the password is sent but not the actual password.

The web application will call the API asynchronously and wait for a reply from the API server.

C. Middleware

The middle ware is developed on a web application server using C# or any other programming language. In this article the REST API will be built in C#. First entry point for this API is a POST handler accepting a JSON object from the body. The client includes a web application that creates a first Hypertext Transfer Protocol (HTTP) POST request including a header and a body. The web application server includes a web server and a middleware.

The middleware receives the first HTTP POST request from the web application. Then, the middleware creates a HTTP GET request using the header. The middleware establishes a secure connection with an authorization server, and submits the HTTP GET request to the authorization server asynchronously. The middleware receives a response to the HTTP GET request regarding a validation of identifiers in a query string of the HTTP GET request, and decrypts a username and a password using the identifiers in response to the identifiers located in the authorization server. The middleware serializes an output of the decrypted username and password in a Java Script Object Notation (JSON) format, and places the output in a body of a second HTTP POST request. The middleware responds the second HTTP POST request to a data server. The ERP system includes the authorization server and the data server.

➢ *The design to handle the authorization request in the ERP authorization server*

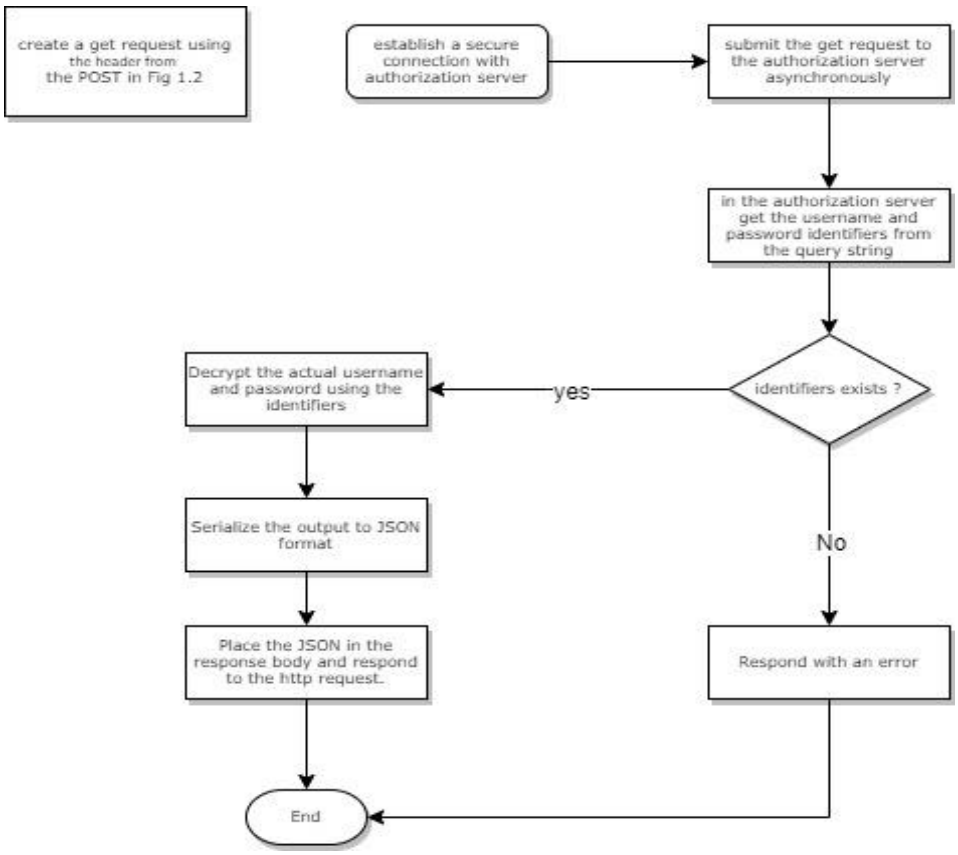

Fig. 4: is a flowchart of a method of providing interconnection of heterogeneousapplications by a middleware.

Once the authorization data is available from the authorization server, the middle will need to establish a POST request connection to the target data server using the user name and password from authorization server. This new post request needs to have content type as JSON and will two headers in the request. The headers are:

➢ *X-INITIATOR*

- This header will have the logged-on user from the network using the "http context" from the web server

➢ *X-OPERATION_CODE*

- Which the header from the previous request originating from the web application

The middleware then will establish an asynchronous call to the target data server and wait for a response. The target data server needs to have an http interface mapped to a handler class this class will map the request to a REST API handler class. The REST API handler class need to have a POST handler method. The method will read the headers X-INITIATOR and X-OPERATION_CODE and then will call the generic handler method that will accept the json from the original request, the X-INITIATOR and the X-OPERATION_CODE

ISSN No:-2456-2165

IV. GENERIC HANDLER

This generic handler method needs to be built using dynamic programming concept and need to be structured in a specific way explained below.

- *the generic handler will need to have six dynamic pointer variables*
- Dynamic pointer to a table variable
- Dynamic pointer variable to importing parameters
- Dynamic pointer variable to exporting parameters
- Dynamic pointer to a work area
- Dynamic pointer to temporary variable to importing parameters
- Dynamic pointer to temporary variable to exporting parameters.
- *get the interface of the method or function that will provide the required data or do the required operation based on the X-OPERATION_CODE*
- *Loop at the interface structure of the function to separate the importing parameters into one table or array and the exporting parameters to another table or an array.*
- *Point the dynamic pointer importing variable to a reference of data of same type as the importing parameters structure.*
- *Point the dynamic pointer exporting variable to a reference of data of same type as the exporting parameters structure.*
- *De-serialize the JSON object that came with original request and update importing dynamic pointer target memory location to hold the JSON object de-serialization output.*
- *Dynamically insert the initiator name into the importing parameters*
- *Call the function*
- *Serialize the output from the dynamic exporting variable to JSON format and return the output to the client via the middleware*.

V. CONCLUSION

This method will provide a generic middleware design that can work with any web application to any data server without the need to update the middleware in case of interface change and it is easy to maintain although it should not require any frequent maintenance.

REFERENCES

[1.] https://azure.microsoft.com/en-us/overview/what-is-middleware/
[2.] (Luckham, D. C. (2011). Event Processing for Business: Organizing the Real-Time Enterprise. John Wiley & Sons. pp. 27–28. ISBN 9781118171851).
[3.] https://azure.microsoft.com/en-us/overview/what-is-middleware/
[4.] K. Smith, D. Taniar and M. Ashrafi, "ODAM: An Optimized Distributed Association Rule Mining Algorithm" in IEEE Distributed Systems Online, vol. 7, no. 03, pp. 1, 2004.
[5.] doi: 10.1109/MDSO.2004.1285877
[6.] Sacha Krakowaik "Middleware Architecture with Patterns and Frameworks", Creative Commons license Feb 2009

First A. Author Abdulaziz Almuhaisin is an IT professional at Saudi Aramco. He received Bachelor degree in computer science from King Fahad University of Petroleum and Minerals. Currently he is a blockchain expert in Saudi Aramco