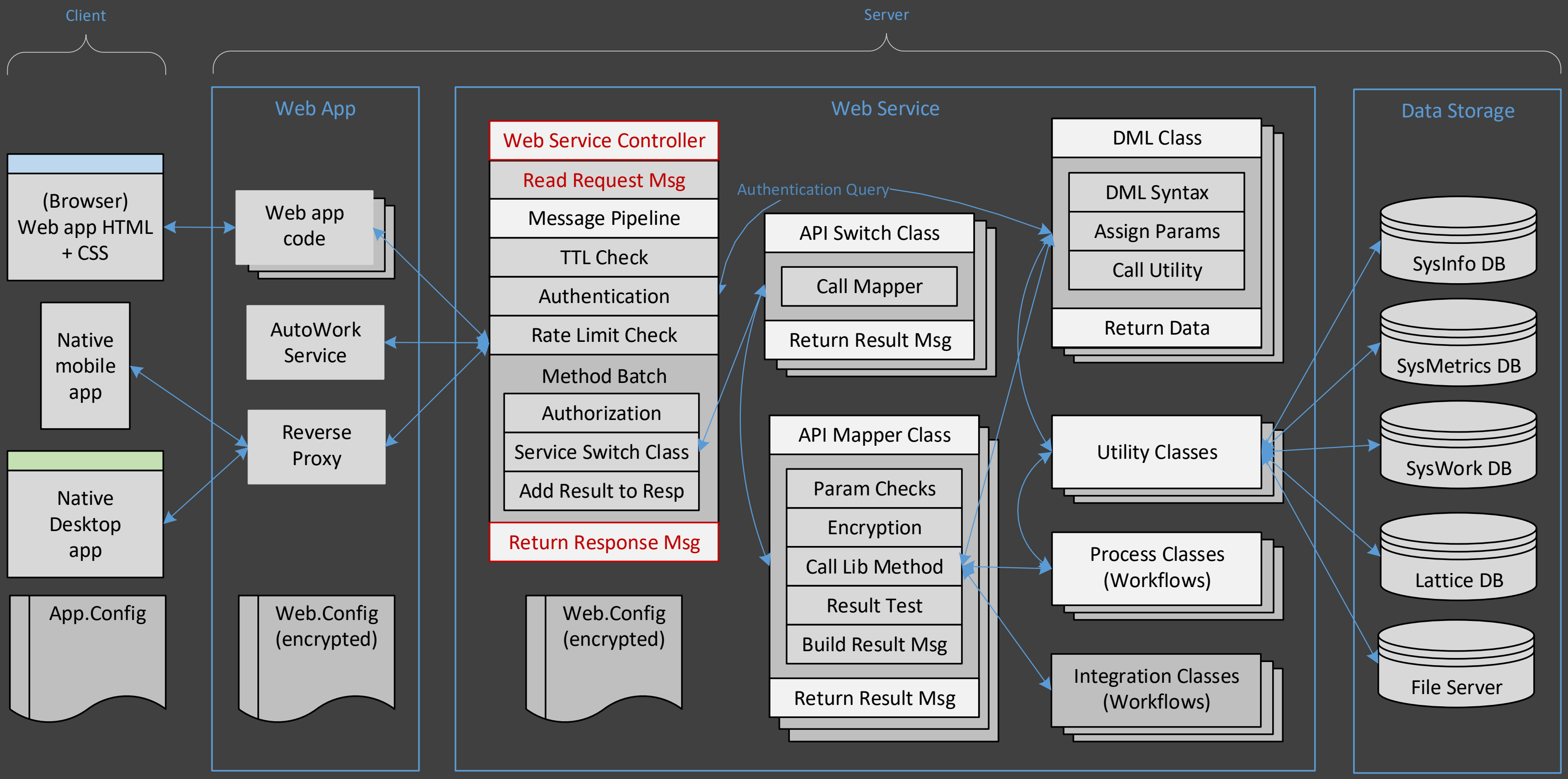
Distributed Grid Platform (DGP): Web Service Controller



The web service controller has very little logic, and only contains roughly 100 lines of code. Its purpose is mainly to accept HTTP requests posted to the controller, read them, pass them into the message pipeline for processing, get the finished API response message as the result, and return the result as the HTTP response to the calling application or system. All security and process logic is contained in the message pipeline, which calls internal library methods that do the actual work. From that perspective, the web service API's are just a mechanism used to securely call the internal library methods remotely.

The message-based API's allow for the two-way communication of a lot more additional data than a traditional method call. This communication along with the event-driven tolerant reader, enables the end-to-end unit testing of the API test harness as well as the remote monitoring functionality. It also provides a mechanism for pushing notifications to each client from the server, but that is not currently needed or used in DGP Lattice.

Service Switch Class

The Service Switch class is customized for each web service and contains case statements for all of the API Classes that are able to be called by the web service controller. Each case statement of the Service Switch class instantiates the correct API Switch class for the API being called in the API request message, and passes the request into the API switch object. This hierarchical structure of switch classes allows a degree of modularity for the web services.

Web service controller methods can access as many different API Switch classes as desired, and a system can also have as many web service controllers as desired. In general, a web service controller will be created for each application or integrated system, and a Service Switch class will be customized for that web service in order to access all of the API's needed by the calling program. In this way, the calling program only has to keep track of a single web service URL endpoint, which also makes failing over between locations by changing a single URL endpoint much easier as well. However, this convention is not mandatory.

Service evolution requires that most parts of the DGP functionality to follow the immutable append-only pattern (to avoid breaking backward compatibility), and also to provide mechanisms for versioning of the API methods to differentiate between them. These conventions are followed in all parts of a DGP system, but are most important in the controllers, switch classes and mapper classes that enable secure external/remote access to the functionality of the internal library classes.

Security

The web service controller uses the single-argument front controller pattern. The single argument front controller has the minimum possible public/external attack surface, which consists of a single method that accepts a single text input parameter (an XML API request message).

The controller limits the size of the API request messages and restricts the MIME type to text/xml. Therefore, the controller will not accept the multi-part form MIME type required for file uploads. The XML reader used by the controller is also locked down by its configuration. The most important configuration setting restricts the inputs to XML fragments. Most exploits of XML files depend on the XML header in order to work. Since XML fragments have no header, this effectively disables all of the potential XML exploits. This also means that the XML fragments have no metadata or strong data types, and for this reason all inputs are text values by default. Internal logic will cast the text values to whatever datatype is necessary “just in time”.

Finally, the controller uses the aspnet_regiis.exe utility to encrypt the AppSettings section of the web.config file in some environments (QA and Prod usually). This allows the “secret” data of a system such as ADO.NET connection strings to be stored in the web.config file securely, in a distributed fashion with no single points of failure or performance bottlenecks. The web.config files also make it easy to share some values that are identical for all locations while allowing other values to be customized for each location.

Performance

The API request messages are read as a forward-scrolling cursor in a single pass through the HTTP Request memory stream. This event-driven technique is both very flexible and very fast. It easily handles the variable structure of the request messages, and reading the memory stream is approximately 140 times faster than the fastest serialization/deserialization alternatives.

Reading the API Request messages as memory streams also improves security by eliminating serialization/deserialization exploits, which generally depend on strongly typed data inputs as well.