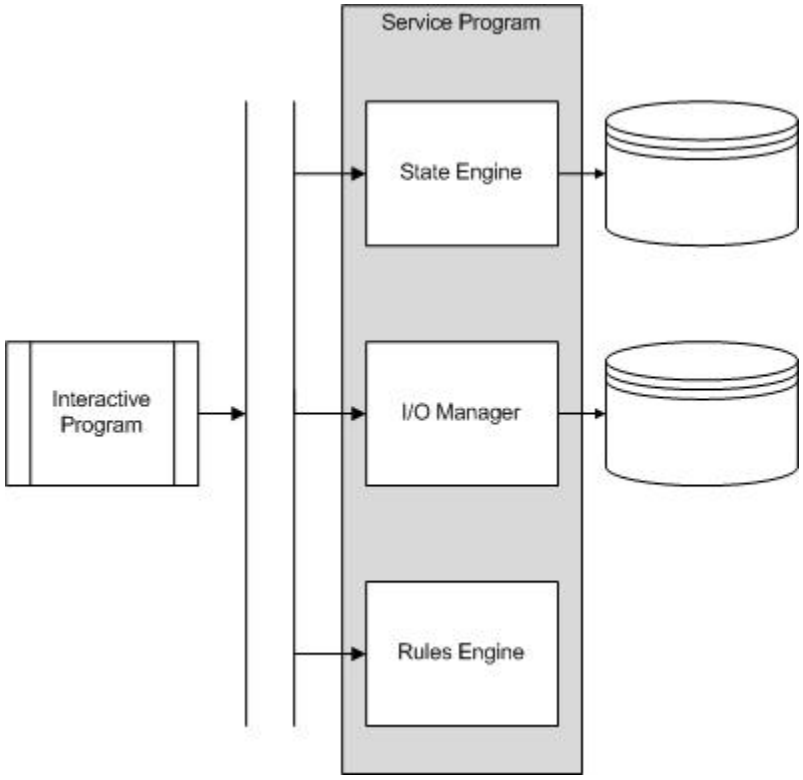
Table of Contents

iSoftwerks Design Concepts

The program design concepts used by iSoftwerks, Inc. are not necessarily unique in theory, but rare in implementation. Though written primarily in RPG, the development model varies greatly from the early models of top-down procedural codes familiar to most long-time RPG developers. It is far more similar to the modular, event driven concepts that are found in the stateless environment of CGI development, or web processes.

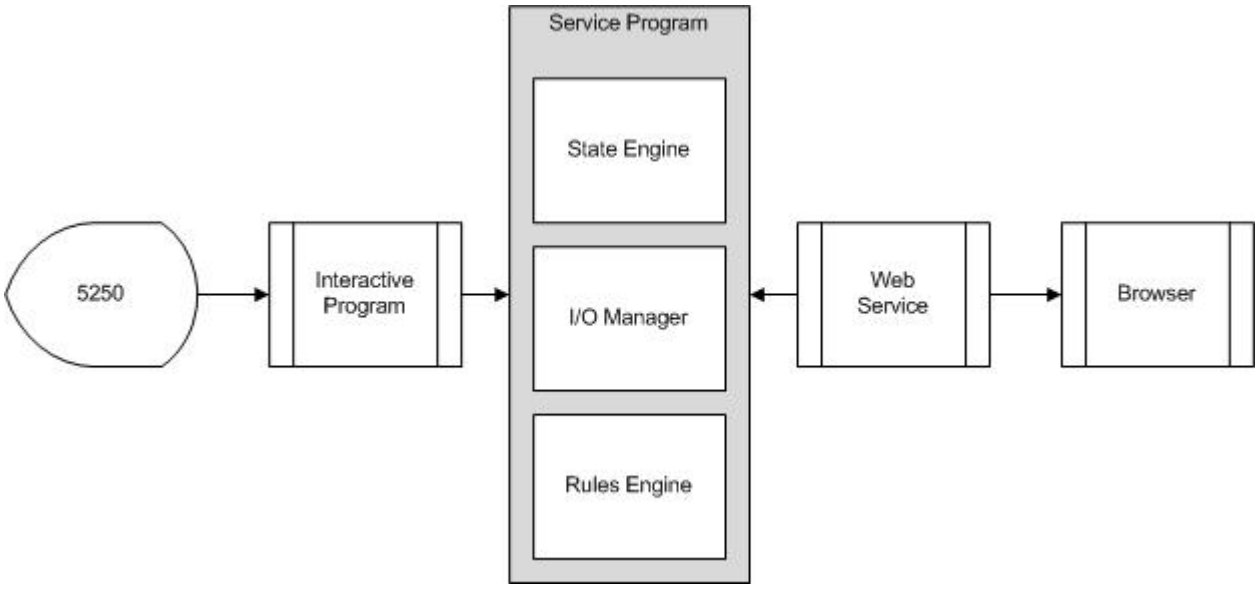


In the environment, programs are generally constructed without containing a database file. Instead the program is bound to a service program or file manager module which contains to data to be acted on. Typically an interactive program only manages the presentation of the data and the data management itself is left to the I/O manager.

This separation of presentation from data (and business rules) allows the RPG application to be relatively small. Keeping the code compact makes maintenance easier and faster. And since time is money, it also is cheaper to maintain throughout the Software Development Life Cycle (SDLC).

There is also a valuable by-product of this very modular style development; re-usable functions. Since the data management is separate from the data presentation, web applications and traditional 5250-style applications can share the same data and data services. (WDSC, (Websphere Development Studio Client, will allow service programs to be exposed as web services.)

What about SOA



There is no real need to replicate rules or I/O functions in order to serve up information. It has the added benefit of applying the same rules to data from the web side, or the 5250 side of the equation. Since the information is presented from a common source, the information displayed on a 5250 screen will match exactly the information presented through the browser—since the data source is the same. This type of application structure produces reusable code and fits into the idea of creating SOA type applications.

At the highest conceptual level, applications should simply provide services. The popular SOA (Service Oriented Architecture) concept describes the goal fairly well. Because ultimately, isn't that what a business, any business, is about—providing services to a customer? Of course, without further definition, SOA is just another TLA (Three Letter Acronym).

The W3C defines SOA as:

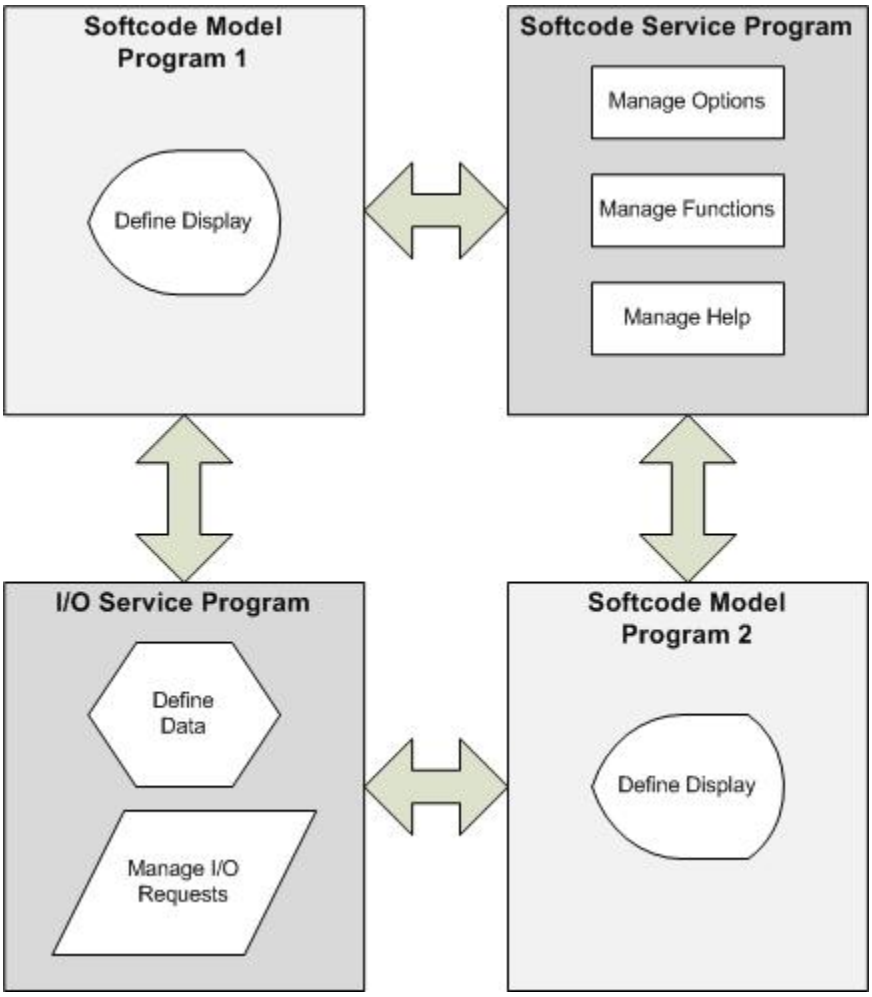
"*A set of components which can be invoked, and whose interface descriptions can be published and discovered*."

Quite frankly, with apologies to the W3C, that definition falls short of the mark. In the first place, components don't always appear as part of a set. Secondly, by W3C definition, SOA is only comprised of implemented and deployed components, rather than the model on which they were built, (the ARCHITECTURE, if you please). Architecture implies a style, deployed or not. A more practical definition of SOA might be:

The standards, practices, and models, that enable application functionality to be provided and consumed as services published to satisfy the requirements of the service consumer. Services may be invoked, published and discovered, regardless of implementation through a single interface, based on a common (standard) form.

Application Models

With the introduction of the Integrated Language Environment (ILE) IBM offered a model of how to build models aimed at providing services from a common interface. The current IBM Power Systems platform increases the opportunity to move forward toward SOA with current tools and application development languages—embracing the soft-coded application model, where functions are external to the program, rather than traditional top-down monolithic structures can help create re-useable functions that are easily maintained and easy to learn.



The iSoftwerks approach to development model takes advantage of the ILE structure to create service programs very much like Java classes. A Java class may be composed of a number of methods, bound together. If a class is imported to an application, the methods of the class are directly available to the application. The major difference is service programs may be constructed of modules of many different languages, RPG, C++, COBOL, CL, etc. The components of the service program are referred to as procedures instead of methods, but like a class, once a program is bound to the service program, the application will have access to the procedures contained in the service program.