Revised October 2012

Creating a Single-Dealer Platform

The rational approach

Patrick Myles
CTO, Caplin Systems

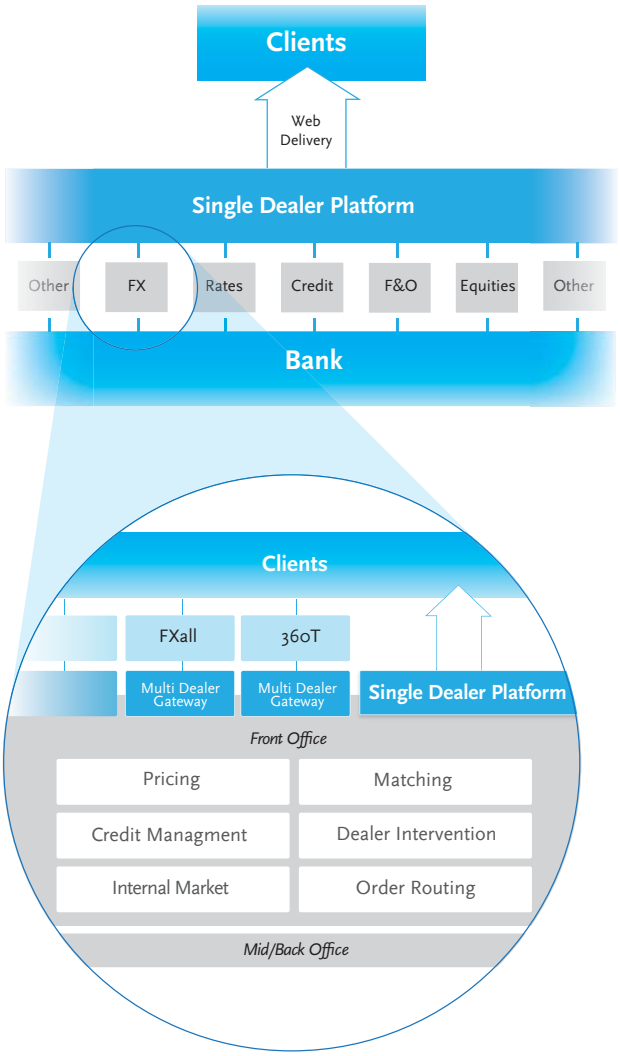
This document explains what an SDP is and outlines the complexities, costs and risks involved in building one. It discusses the importance of using a good SDP framework, and describes how Caplin Platform can be used to create a high-performance SDP quickly, at low cost and low risk, without in any way limiting your creativity or scope.

CAPLIN

A Single-Dealer Platform (SDP) is an eCommerce delivery layer that spans multiple electronic trading and information systems, and provides online trading and decision support direct to clients.

An SDP is not an electronic trading system. It does not provide such features as rate aggregation, liquidity management, order matching, credit checking, real-time risk, position management or STP.

An SDP sits between one or more electronic trading systems and a bank's clients. It is a single-bank gateway between the trading system and the clients, much as platforms such as Bloomberg, Tradeweb and FXall are multi-bank gateways.



Where does Caplin Platform fit?

Caplin Platform is an SDP framework. One key part of it is Caplin Liberator, the world's most advanced streaming server and financial Internet hub. Caplin Liberator includes many core capabilities that would otherwise need to be developed from scratch if you were using any other streaming server. But an SDP requires much more than just a streaming server to deliver a reliable, compelling and complete trading experience to Internet clients.

Caplin Platform provides everything you need to connect many electronic trading systems to many client applications. Each electronic trading system connected to a Caplin Platform SDP will have its pricing, trading and permissioning models presented, through a platform layer, in a standardised way to any number of client applications built in any mixture of user interface (UI) technologies. This platform enables the rapid development of a range of applications targeted at different client segments. These applications might range, for example, from a feature-heavy futures trading application built in .NET and aimed at high-end institutional clients, to a lightweight FX and Equity pricing website aimed at retail users, and anything in between. The platform acts as an anti-corruption layer: it decouples the UI applications from the various trading systems, allowing either to be extended or replaced quickly and with minimal cost.

Do I really need it? Isn't it easy to do this stuff?

What's the point of using an SDP framework? Why not just build all this from scratch, or pick "best of breed" infrastructure components and integrate them? Couldn't I create a more specific/functional/performant/bespoke/ tailored (delete as appropriate) solution that would meet my needs better this way? After all it's easy to build this stuff, isn't it?

In fact it's not easy at all. There are many "gotchas" and complexities involved in efficient Internet delivery of trading applications that are revealed only when you're at the coalface of implementing and operating an SDP. And most of these complexities require repetitive solutions and hard work, rather than being cool and interesting problems to solve.

Caplin has been at this coalface for 10 years and, while Caplin Platform is full of innovative solutions to these problems, it is predominantly the culmination of a huge amount of effort, trial and error, and feedback from real-world implementations.

Here are just some of the complex problems that Caplin Platform helps you solve in the delivery of an SDP:

Caplin Platform can deliver latency in the tens of microseconds

Highly scalable Internet distribution

Caplin Platform supports tens of thousands of simultaneous trading users per server with consistent low latency and horizontal and vertical scalability of all components.

Failover, load balancing and high availability

Caplin Platform will support your production deployment model, whatever it is. This includes live/live servers with failover between individual components and no single point of failure. All Caplin Platform components have been hardened and security tested to the highest level, meeting the needs of information security departments of tier-one investment banks.

Flexible and granular authentication, entitlements and per-user/group pricing

Caplin Platform integrates with your existing single sign-on (SSO) system(s) and represents entitlements from one or more permission systems in a standardised fashion, in real time. It can efficiently distribute (and if necessary, generate) per-user prices, multiple price tiers, spread groups and volume bands. In fact, Caplin Platform supports virtually any client price segmentation strategy.

Optimal transport for financial data structures and content-aware data management

Prices, trades, depth of market, order books, blotters and so on are all represented using optimized data structures in Caplin Platform (see "Why not just send my objects?" below for more on this). Only what the client actually needs at any moment is sent over the wire – for example, only the properties of an object

that (a) have changed and (b) are visible on-screen. Multiple price streams are managed efficiently with a throughput of over five million updates a second per Liberator server. Caplin Platform components are content-aware, so it is possible to share the load optimally between client and server for features such as list subscription management, filtering and sorting.

Reliable management of deal workflow

A key part of any SDP is execution lifecycle – both the technical implementation and user experience (UX). Caplin Platform makes it easy to integrate client applications with multiple trading systems by providing configurable domain objects for the modelling of any workflows and business processes, complete with automatic synchronisation of client and server state machines, guaranteed messaging, standardised error and failure handling, and managed failover and recovery.

Caplin Platform will support your production deployment model, whatever it is

System management and health checking

Caplin Platform provides monitoring and management capabilities that allow you to monitor everything from high-level information such as the health of each component, overall usage and trade volumes, to fine-grained details such as each user's subscriptions and price latencies. There is an audit trail for all price and trade messages, and business process latency tracking tools for troubleshooting.

Freedom of implementation approach

When implementing an SDP using Caplin Platform, you can use any UI technology and language of your choice – including native web apps, desktop, mobile or any mixture. Similarly, back-end integration can be performed using Java, .NET, C, C++, or scripting languages like Python and Lua. You have freedom to use an SOA architecture, frameworks such as Spring or Parsley, design patterns like MVC, IOC, and so on. Every care has been taken to allow Caplin Platform's application program interfaces (APIs), test frameworks, models and components to fit with your preferred implementation approach.

It's important to point out that Caplin Platform does not require that you represent your data in particular ways within your client application or trading systems

A huge amount of work has gone into making it simple to hook Caplin Platform up to any back-end system, regardless of the implementation language and data models of pricing, execution workflows and permissioning. It is easy to map any domain object data structures on to the Caplin Platform base datatypes in the server and client. This allows you to keep any standard data representation layers that may be shared between the client and the server while still taking advantage of Caplin Platform's features.

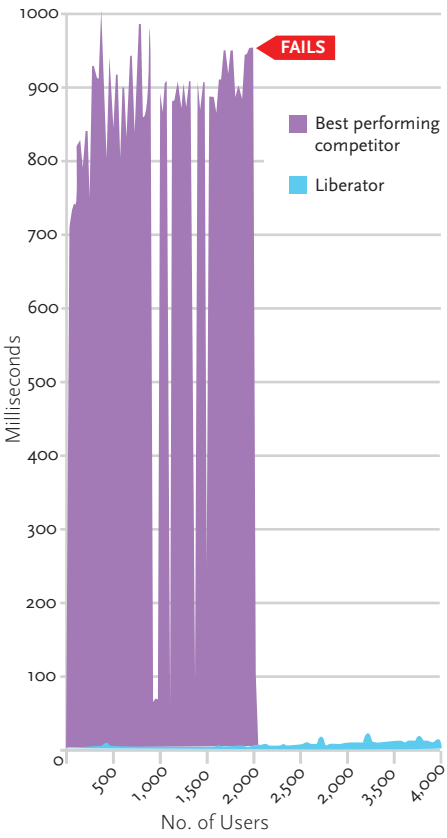
Does doing all this compromise latency?

Caplin Platform delivers all of these advanced and complex features while maintaining low and deterministic latency.

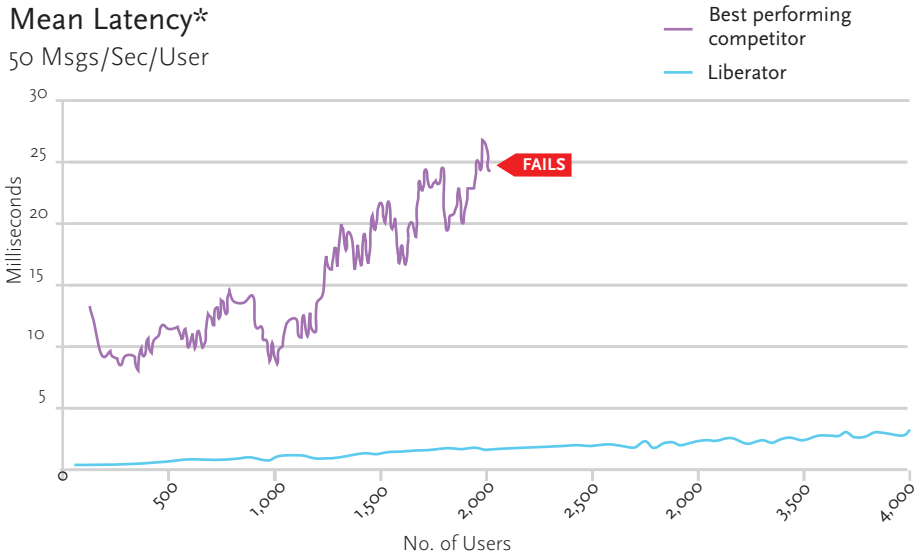
The initial challenge is making each component and feature of the platform operate with minimal latency overheads. But this is only the start of the story. In trading systems, consistent latency of every single price is vital.

It's no good delivering almost every price in a few milliseconds if some (potentially executable) prices take hundreds of milliseconds or even seconds to arrive. Any one of those prices might happen to be used for a high-value one-click trade. What if that is the one that was delayed, and the market has moved?

Latency Range*
50 Msgs/Sec/User



Mean Latency*
50 Msgs/Sec/User



Either the user or the bank will lose out (in terms of money or trust) and this quickly becomes unacceptable. If you aren't delivering consistent latency for every price then you can't reliably predict the latency of any given price, meaning risk models must have much higher tolerances and more complexity.

Building a platform that can deliver deterministic latency is hard. Things like garbage collection by virtual machines, interference from other user's actions and even excessive logging by components can all contribute to unpredictable system latency. Caplin Platform manages latency in two key ways. Firstly, because its key components are built around low-level real-time C application cores, it can consistently deliver every single price within a small latency tolerance window. No blips for garbage collection and the like. Secondly, it allows you to actively choose your trade-off between overall capacity (data volumes, number of users) and desired distribution latency. In an ideal world, every price would be sent with zero latency (and Caplin Platform can, if required, deliver latencies in the tens of microseconds). In reality, to provide more capacity and headroom, it's usually better to determine a latency ceiling for all users (perhaps based on risk tolerances) and then let Liberator optimise for these constraints, keeping all message latency within a predictable range up to a well-testable maximum capacity.

* Typically vendors will quote mean or 99th percentile latency - but it's important to know what the worst latency will be - because that message may contain the most valuable trade of the day. The Latency Range chart shows the best/worst range of latencies and compares Liberator with its best performing competitor.

The latency tests represented graphically in this document were conducted in our test labs using a Linux server and eight client machines. The server was equipped with dual, quad-core Opteron 2876 64-bit processors and 8Gb of RAM; it was running Red Hat Enterprise Linux Server release 5.3.

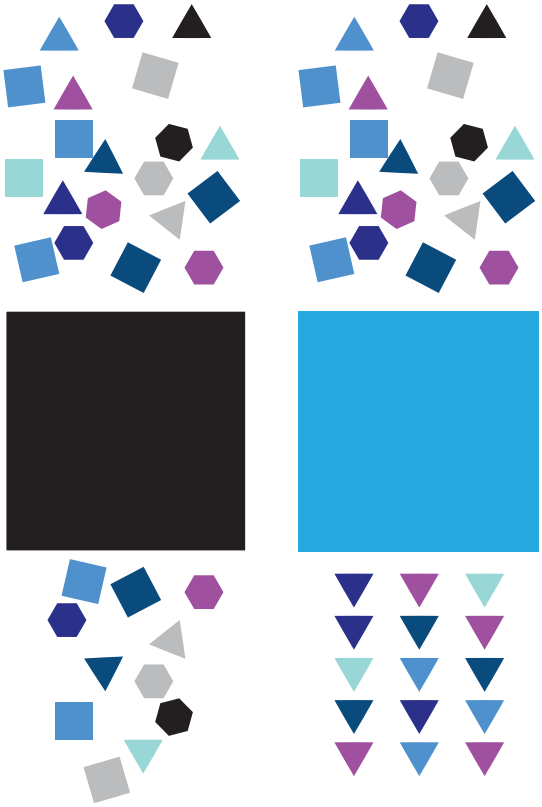
The client machines connected to the server via HTTP and simulated the real load on the server of updating live clients.

To represent a typical use case with a relatively high volume of updates, each client subscribed to a random 100 instruments from a set of 1,000 and was sent 50 updates per second.

The actual latency for each update was measured as the client load increased. Mean latencies were calculated from these measurements.

Why not just use a generic streaming server and send my existing data objects?

It might seem like a good idea to use a low-level streaming server to transport opaque, language-specific binary objects (serialised data structures, for example), XML, or other custom data. Why not treat the streaming server as a black box that just gets your data from bank to client? This way you can implement any data structures you like, you don't have to create an adaptor for the data types of the streaming server, and you have complete control over the format.



This approach turns out to be a bad idea for a number of reasons, not least performance and latency. Firstly, compared with using optimised protocols, sending financial price or trade data around like this is always inefficient in terms of bandwidth and message size – usually by at least an order of magnitude. This might not seem like a big deal, but it is expensive for both the server and the client in terms of data connection, pushing up latency and making hitting a bandwidth or performance ceiling much more likely. Additionally, in most cases the client-side parsing is far more processor-intensive for these structures than for an optimised protocol. Lastly, and most importantly, you cannot take advantage of the powerful features of the streaming infrastructure that need to be content-aware.

Caplin Platform provides various standard data structures that can be used to represent many different types of financial data including prices, trades, depth of market, order books, news, time series, inventories and reference information. It can also transport existing, opaque object formats should you choose to do so. But by taking advantage of the standard structures you not only ensure that the most efficient transport protocol is used; you gain access to the wealth of content-aware features throughout the platform.

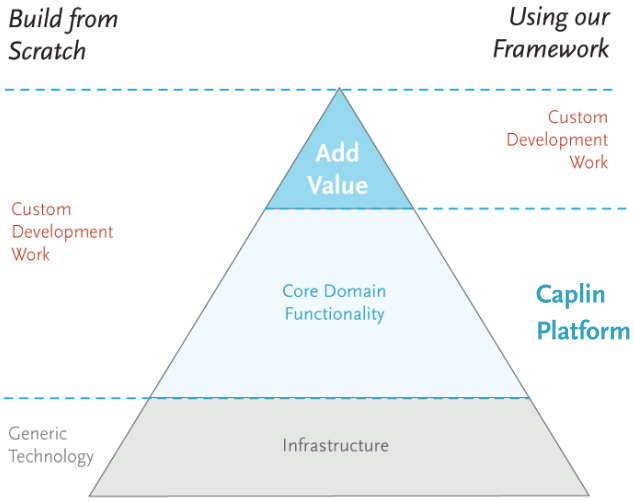
Improvements to the platform such as transport optimisations and new content-aware features can be added without affecting either the client or server implementations.

Using Caplin Platform as a normalisation and anti-corruption layer means that your client applications don't need to implement custom data structures for each back-end system and are therefore decoupled from these systems. The anti-corruption layer concept – from the Domain-Driven Design (DDD) community – is proven to reduce complexity and fragility in large systems. In contrast, transporting native binary objects results in the server and client being tightly coupled, so that even small changes to the back end can require a new client compilation and deployment. Debugging and auditing of opaque message formats through a complex Internet transport layer is also much harder and time consuming than using a standardised set of data structures to represent your models. Finally, the ability to implement UI code using standard object structures keeps it independent of the back end and removes the need to write a separate set of display components for each back-end system, which in turn makes adding an asset class to the client much simpler and quicker.

But I want to build cool stuff! Isn't building this stuff cool?

Trust us: it's not cool. Really – we know, we've done it. It can actually be pretty tedious and frustrating to implement and maintain reliable, scalable, bi- directional, asynchronous messaging over the internet, let alone trade workflow mapping and messaging complete with resilient client/server synchronisation, error states, failover, etc.

Concentrate on Adding Value



The cool stuff is creating unique and compelling product workflows with innovative user experience (UX) which provides real differentiation from competitors. The cool stuff is being able to quickly create and deploy new applications to meet emergent demand from the business and markets.

The cool stuff is implementing innovative pricing and liquidity management techniques, building real-time risk systems, managing smart order routing and so on. In other words, the cool stuff adds direct business value to your SDP.

Using Caplin Platform is cool, because:

- It provides sensible domain models that are easy to use; it understands and is optimised for the domain, but doesn't force particular workflows or data models on you.
- It allows you to use any integration technologies with any presentation technologies.
- It takes care of the routine stuff faster and at lower cost, freeing you to focus on the things that really matter in delivering a successful SDP.

Conclusion

If your idea of fun is writing hundreds of thousands of lines of low-level infrastructure code and then spending dozens of man-years (and millions of dollars) testing it, tuning it, refining it, debugging it, deploying it, scaling it and maintaining it, then you may prefer to build your SDP entirely from scratch.

And without an SDP framework, you'll have a chance to do it all over again each time you add a new asset class.

But if you would rather concentrate on adding business value, solving new problems, and creating an offering that is truly differentiated from your competitors, then Caplin Platform is the ideal starting point. It takes care of all the stuff that every SDP needs, leaving you free to devote all your available resource to creating a unique and advanced e-commerce solution tailored to your exact needs.

Patrick Myles

CTO of Caplin Systems

Patrick joined Caplin in 2000, shortly after the company was founded. He is now a board member and has held his current position since April 2009. He has pioneered the adoption of Agile processes and practices at Caplin, allowing the engineering department to rapidly deliver high quality software that provides significant business benefit to Caplin Systems' customers.

He holds a BSc in Computer Science from Warwick University.

CAPLIN

CAPLIN SYSTEMS LTD.
Cutlers Court, 115 Houndsditch, London EC3A 7BR, UK

+44 20 7826 9600

CAPLIN SYSTEMS, INC.

5 Penn Plaza, Suite 1982, New York, NY 10001, USA

+1 212-835 157

sales@caplin.com

www.caplin.com