FreshTrends

eCommerce Case Study

PAUL COWAN
CEO

Industry: eCommerce, Manufacturing (Jewelry)

Locations: 2 (CA & FL)

ParaSQL Applications In Production: 10+

Development Team Size: < 1 (outsourced)

Integration Use Cases: eCommerce platform integration (Shopify, Mi9)

Mobile Use Cases: Tablet app for warehouse pick-n-ship

Background

FreshTrends is the main brand behind parent companies ProPiercing and Kiwi Diamond. They specialize in manufacturing body jewelry and selling direct to consumer via a multi-channel e-commerce strategy (e.g. website, Amazon, eBay, etc). It is a family run company first incorporated in 2001. They have two primary locations, one in California that does the manufacturing, and one in Florida that handles design and e-commerce marketing.

As their product line has evolved over time they have managed between 10,000 and 80,000 SKUs with a complex mix of stocked, special order, drop ship, and manufacture-to-order items. Complicating factors include a fast-moving, fashion-based business and volatile input costs (e.g. gold, platinum, precious stones) to their manufacturing operation which necessitates frequent price changes. Further, the industry is highly competitive with margins that require efficient operations.

Paul Cowan has spent his entire career in e-commerce. He has extensive experience with multiple e-commerce systems and all the major e-commerce channels. Over the years FreshTrends has invested extensively in custom software development using a variety of companies, platforms and tools. This gives Paul an especially good understanding of the costs and trade-offs of using various approaches.

The Evaluation Process

Paul had always struggled with achieving efficiency and consistency of internal operations. Being a pioneer in multi-channel e-commerce meant that the tools were not always there. Often this meant that his team would spend long hours with spreadsheets trying to bridge the many gaps left by the various e-commerce packages, marketplaces and tools. A high volume of special order and manufactured-to-order items – often included on the same customer order – meant that simple things like knowing when an order could be shipped were very time consuming and error prone. As spreadsheets multiplied to try and solve specific problems, so did the amount of human error introduced into the process. Paul had tried to solve many of his specific problems by purchasing packaged software, but found that they often made the problem worse and integrations never worked as promised.

Paul’s evaluation of ParaSQL started off in an unusual way; a trusted advisor that was already familiar with ParaSQL suggested to Paul that he try ParaSQL on a single project that had already failed.

The evaluation project was to automate the restocking of inventory. Prior to ParaSQL, two people worked full time generating purchase orders by downloading inventory levels and special orders from the e-commerce platform and generating purchase orders and drop ship requests with spreadsheets. Previously Paul had tried several different packaged software solutions, none of which worked as advertised. He had also commissioned a custom project to solve the problem that took over 6 months and could never be debugged enough to put into production.

ParaSQL’s internal development team scoped the problem and suggested we iterate toward a solution rather than waste time scoping a big project that was still poorly defined. ParaSQL offered to have the first cut of the solution done the following week, and suggested that we could try to implement that and learn what else was needed. Paul was of course highly skeptical of a one week claim for custom anything, and rightly so. But with nothing to lose, he gave the green light.

That meeting was on Thursday and on Monday of the following week we asked for a meeting to review via screen share the first cut of the application. Numerous problems were identified, some software related and some data related. We suggested a follow on meeting for Wed to give us a day to work on a new version. On Wednesday’s software review it appeared that we were mostly there on the software side, but had a long way to go on the data side. We suggested another meeting for that Friday to review yet another iteration of the solution. By Friday, the system was deemed ready to test in production and ParaSQL also provided a couple of custom apps to help streamline the fixes needed to solve the data problems (primarily inaccurate inventory counts and inventory location assignments).

Paul said he could see the impact on his operations immediately, even with the bad data. Although resolving the data issues would take months, the amount of labor to restock and handle special orders was cut by 40 hours a week immediately (a 50% reduction in labor). Those man hours were re-purposed into getting the data issues resolved and reorganizing inventory locations (a significant undertaking with 15,000 stocked SKUs).

When I asked Paul what he thought about ParaSQL, he said “It’s a game changer. It finally gives me a way to solve problems that seemed impossible. The cost of the solutions and how fast they are developed is crazy. I never would have believed it.”

Paul didn’t know what model-driven, data-centric development was – and he didn’t care. He got the fact that since 95% of the ParaSQL work is done via drag-n-drop, that meant 95% less code which meant 95% fewer development hours and 95% fewer bugs. It meant solutions happened almost immediately, didn’t cost much, and started to have a pay-back in terms of benefit almost immediately.

Applications Developed

FreshTrends started by automating its restocking process. This lead to automating the drop ship requests. Then another app to automate the special order process. With a high level of manufactured-to-order transactions, this was the next process to automate. It was clear with this level of automation in place, that most of the remaining challenges were in order fulfillment. An application was built to reorganize the warehouse for more efficient storage of goods; in this case the application took a few days to develop but the warehouse reorganization took months. Once that was complete, there was an opportunity to implement an application that would allow the picking of orders in “batches” – so that a person could take a single walk through the warehouse (directed by a ParaSQL app on a tablet mounted to a push cart) and pick the goods for 50 orders at a time (previously each order was picked and processed individually). This “pick an ship” application could “stand on the shoulders” of all the previous applications and data improvements with the result being that order processing time was reduced by 80% (8 warehouse pickers replaced by only 2) and shipment errors (returns due to shipping the incorrect item) were reduced from a very high 1 in 20 orders to less than 1 error in 500 orders. Far beyond the savings in order processing labor, the reduced mistakes caused a 95% reduction in returns, which saved both on return shipping costs and the number of customer service reps required.

Paul runs his entire business on just ParaSQL and the e-commerce package. The applications implemented so far include:

- Restocking – automatic purchase order generation based on restock thresholds, tracking ETAs, and assess demand on over 50,000 SKUs across 80+ vendors, etc.
- Special Order Handling – order on demand for tens of thousands of items from approximately 100 suppliers in real time.
- Manufacture On Demand – to date only half of this application has been deployed, where manufacture-to-order items are provided to the manufacturing team. Paul anticipates full manufacture process management in the future.
- Pick and Ship – handles over 100,000 orders per year, batch picking 50 orders at a time rather that individually.
- Warehouse Receiving (ROG) – logging receipt of goods against the responsible PO, updating the e-commerce inventory in real time, tracking ETA dates and inbound quantities, etc.
- Bill of Materials – for our manufacturing division, detailed and constantly updated costing of labor and materials for all manufactured goods, detailed component inventory usage based on builds, feeds into purchase order generation for restocking components needed for manufacturing.
- Drop Ship Automation – splitting orders and automatically sending drop ship requests to suppliers while shipping the stocked goods themselves.
- Production Orders – yet to be built; this will manage the manufacturing process and allow a customer to see live information on where their order is in the production process.
- Merchandising – front end catalog merchandising app for our sites to market the right products to the right customers.
- General Warehousing – inventory recounts, location assignments, etc.
- Reports – all management reporting.

Although Paul wouldn’t describe his system as “Data-Centric”, this is exactly what it is. The various applications are merely different ways of working with the same set of data. Even the “external” data that is managed by the e-commerce platform is read and modified directly in real time as though it were all one big database; there are no “data extracts” or “syncs” that need to occur. Importantly, if this same set of functionality were attempted via the more traditional “application-centric” approach (buy an app to solve each problem) each application would therefore have its own database and require integration with multiple other applications, both read and write. It is precisely that “web of complexity” that dooms the application-centric approach to failure regardless of the time and money invested (suggested reading: *Software Wasteland: How the Application-Centric Mindset is Hobbling our Enterprises.* Available on Amazon.com).

Maintainability

The system has all the attributes that are far too often missing in deployments of this type, including ease of understanding and ease of change. I asked about the effort to make a routine change and a difficult change. By routine change we agreed that something like adding a field to the database, getting that on a form and providing formatting and validation would be a routine change. An example of a larger change would be something like adding a new module, which might interact with several modules, as it would have to be set up, maintained and carried all the way through the system and reporting.

He said the routine changes typically take an hour or two, and he can slip stream them into production. He has enough experience with the system that he knows which changes will not be disruptive. The platform is built such that these changes do not require data conversions, or even shutting the system down, a user of the system will get the change in the next refresh. The larger changes he said typically take 2-3 days.

I asked him to compare that with his experience with packaged implementations. Even the simplest of changes starts a work flow of definition, quoting, scheduling, development, QA, etc., etc., each of which typically takes weeks or more to get from requirements to in production.

A key consideration is this: each line of code that isn't written is a line of code that doesn't have to be maintained. This is especially true with tricky integration code that is trying to make incompatible systems play nicely together. Real-world model-driven, data-centric deployments (e.g. a ParaSQL-based system) achieve a 95% to 99% reduction in total code volume (application and integration code counted together) which is "why" they are so much more reliable and easier to maintain than an application-centric approach.

Team Size

All this functionality was built by a single ParaSQL consultant working a few hours a month on behalf of the client. Most would agree that traditional development would have required a medium sized team on each of the more than a dozen modules he has implemented over a lengthy time frame.

This study shows that even medium sized companies can benefit and benefit significantly from a data-centric approach to systems. It shows that the tools are real and the approach proven. And it shows that the data-centric approach delivers the promised value, even if you weren't aware that is what you set out to do.

Customer's Summary Evaluation

Pros: ParaSQL is amazing. We have built multiple cloud-based applications that integrate with our e-commerce platform API. These include our WMS / Pick and ship app which handles over 100,000 orders per year, purchase order app that maintains stock levels and assess demand on over 20,000 SKUs across 50+ vendors, bill of materials app for our manufacturing division, front end catalog merchandising app for our sites to market the right products to the right customers, and many others.

The thing makes ParaSQL special is that you can build solutions that are tailored to your specific business needs, for a fraction of the cost of paying licensing and implementation on out of the box solutions. ParaSQL applications have automated processes that were handled in spreadsheets, saved me a ton on payroll, and eliminated human error in business-critical processes. It is an affordable, powerful application platform. It is a game changer for anyone that applies it to their business.

Cons: I have had no issues with the platform, or the applications that we have built. The level of support has been outstanding.

Overall: 11/10. ParaSQL is a remarkable solution for a multitude of business problems and challenges. In 15 years of business I have never found a solution as complete and as adaptable as this. I cannot speak highly enough about Customer support and the team. I would highly recommend ParaSQL.