

Glassbeam Product Analytics

Overview

Product Analytics is an untapped multi-billion-dollar market.

Gartner has recognized Product Analytics as a multi-billion-dollar market for the business value the analysis of product operational data (POD) can deliver to a wired product vendor. Product Analytics is the vendor-side, business-wide leverage of data generated by products once installed in the field.

Product Analytics shares value propositions with analytic CRM and with PLM, and is closely related to both. A vendor may attempt to draw value from an analytic CRM system when analyzing support cases, sales opportunities, and market trends, for example. For a wired product vendor, Product Analytics is a much more effective way to realize these kinds of value.

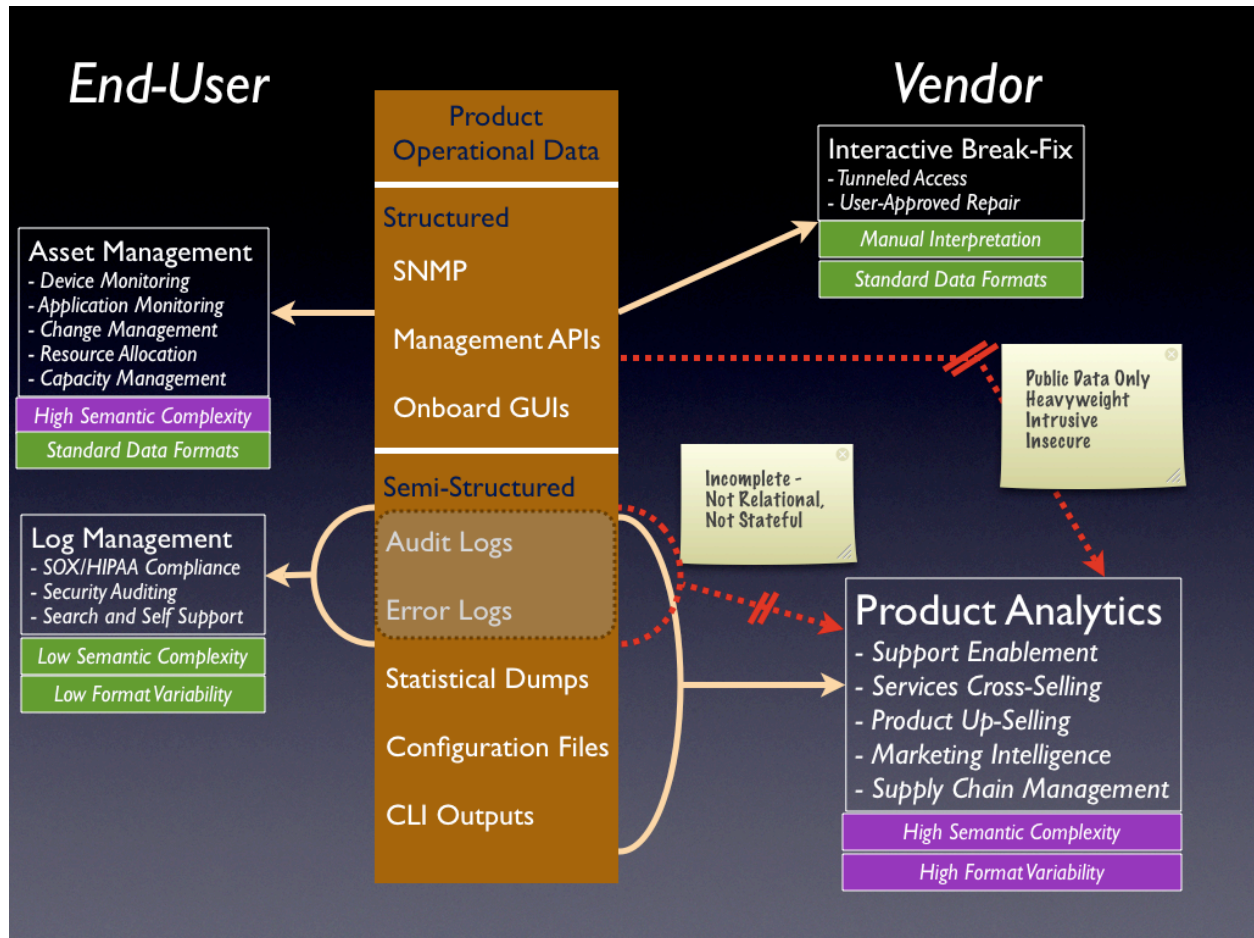
Product Analytics relies on semi-structured POD.

POD of the form necessary to serve vendors' interests is usually of a form we call semi-structured. It is text-based and has structure, but is intended for human consumption and so comes with no formal grammar or warranties for consistent formatting over time. Common POD examples include configuration files, command outputs, log files, and statistical dumps.

Structured product operational data underpins data center management; unstructured product operational data underpins call center data analytics. Similarly, semi-structured POD underpins Product Analytics. More detail on the importance and format of this sort of data is found further on. Its roots are in the universal practice of collecting data for manual debugging and support purposes.

Image 1 shows the relationship between POD types and market applications.

Image 1: POD Types in Relation to Markets.



Glassbeam is the first product to address the Product Analytics market.

Product Analytics is already the object of high spending in wired product vendors of all kinds. There are plenty of wired product vendors already pouring millions of dollars into internal projects to leverage POD for the business. Despite this trend, Glassbeam is the first Product Analytics market offering.

Our tools are designed specifically for Product Analytics. Where even most vendors haven't even thought through all the ways in which POD could serve their business, we have. From support and escalations tooling, to OLAP capabilities for account management and marketing, to a reporting platform for quality and operational tracking, we bring a comprehensive solution to the table.

Our SPL technology is uniquely suited for analytics on semi-structured POD.

We have opened up the Product Analytics market through our product's unique ability to cost-efficiently populate an analytical platform and workflow-oriented tools from this sort of data, for any wired product.

Our core technology is the Semiotic Parsing Language (SPL) and associated tools. It unifies source data modeling and parsing, ETL, target database modeling, and OLAP tooling into a single development step. It is a revolutionary innovation in the large-scale warehousing of semi-structured data, making it a great fit for a Product Analytics infrastructure.

The SPL description of a class of documents and its semantics is passed through an interpreter to generate database DDL for staging, database DDL for the final warehouse, DDL and DML for metadata to generate the UIs, and internal representations to perform a parse, a transactional or bulk load, and ETL transformations. Multiple databases are supported; the description of the data is independent of the particular database. However, the interpreter will emit specific optimization DDL (indexes, projections, or partitions, for example) as suits the particular target.

SPL is an intuitive, declarative language. Using SPL, an analyst can describe the structure and semantics of a class of documents without worrying about telling a program, database, or UI how to do things. For semi-structured data, this means taking advantage of the structure inherent in the data without any pre-defined grammars, and reflecting this structure in a high-performance data warehouse with a highly normalized and rich relational model. No specific understanding of databases or procedural programming is required. The SPL confines a customer's IP to a single document. The UIs and the interpreter are the same across all customers.

Image 2 shows a simple example of what SPL looks like. You can see that one line contains grammatical, relational, and semantic information naturally together. (Object / aspect definitions and ETL semantics are not shown here.) It says that a document section contains tuples from a relation which have been transformed according to a particular rule (the "icon"). Parsing then represents an inversion of this transformation to a set of rows in a staging table. So, the analyst describes what rows in the table "look like" in the document, and lets the interpreter take care of the rest. So far, we have found need for no more than five such icons to handle semi-structured data from multiple customers and industries.

among configuration / status changes, error events, and performance trends: can root-cause be ascertained? Without tools to bring this information out of operational data, the manual attempt will be time-consuming and cursory, at best.

An account manager, professional services engineer, or sales rep responsible for an account needs to be able to quickly spot reliability, performance, and other issues at an account: are the installed products functioning well? He needs to be able to quickly spot resource limits as they are being approached: is it time to make a sales call? Are any assets at the account approaching limits? He needs to be able to quickly spot misconfigurations or accidental changes at an account: is the account software / firmware up-to-date? Are on-site personnel using the equipment properly? Are on-site assets exposed to any bugs or issues as a result of current configuration or usage? Without tools to present this information out of operational data, the account can't be efficiently audited.

The above sorts of consumers require near-real-time analytics to work current cases.

A quality analyst or performance engineer need to be able to quickly spot patterns of events and statistics over time, as they relate to an arbitrarily specified population of components across the installed base. He needs to quickly spot error rates and performance problems correlated with system configuration: is one software version / OEM / firmware revision responsible for a quality or performance problem? He needs to classify product usage and compare performance and reliability: are enterprise customers using the data differently than others? Are large accounts experiencing reliability or performance issues? He needs to quickly spot correlations among configurations, utilization, and user experience: what subsystems need to be improved? What best practices ought to be enforced? Are customers updating software / firmware quickly enough to avoid issues? Without a database containing field-wide data at the level of detail necessary,

A marketing or program manager needs to be able to quickly spot feature adoption and trends in product demand: what features are being used? What products, options, and offerings are popular among enterprise accounts? He needs to be able to spot industry trends in utilization: what protocols / subsystems / features are becoming more or less popular? What additional demands will be placed on the product in the near future? Without a database containing field-wide data at the level of necessary, again, analysis will be extremely difficult and in many cases unfeasible.

Depending on who's driving an in-house Product Analytics project and how much budget is allocated, it's likely that only certain hot-button requirements out of the above will actually be addressed. Organizational issues are addressed later on.

Every product has a complex and unique representation for operational data.

For a set of data to be useful underneath Product Analytics, it needs to communicate detailed information about the configuration, events, and statistics of components of the

product's unique architecture. A wired product vendor will typically ask for access to a set of diagnostic files / data to work a case; aside from a core dump, this is the kind of data we're talking about when we discuss POD (product operational data). Applications, GPUs, appliances, etc. all log events, counters, and information about internal component state, down to the level of vendor-specific abstractions.

Much of the data will have little to do with the public information available to management APIs and other public interfaces. It will reveal a vendor's dirty laundry in the form of hidden errors and counters; it will expose the operation of various components of a vendor's particular implementation. It will be highly vendor-specific, unconcerned with conforming to a common information model for interoperability or end use; it will be largely undocumented and without formal grammar. We call this semi-structured data: configuration files, cli outputs, statistical captures, event logs. It is generally intended for human consumption.

A useful archive of such data will be vast.

To be of maximal benefit to all consumers, a call-home bundle needs to contain “everything”. Trending and field analysis require detail for all systems at all times under consideration, not just when problems are detected.

Support usage and engineering analysis of product operational data isn't like management. It's not about common cases of doing stuff when things are working; it's about understanding stuff when things aren't working, or understanding how things work in edge cases. Marketing analysis and service delivery are about understanding larger trends in the field or at an account. So, the data sets need to be large and comprehensive.

A successful product may have from thousands to millions of instances reporting home regularly, as well. The volume of data received by a vendor is likely to be in the range of hundreds of gigabytes to terabytes over the course of a year or even a month.

Formatting and semantics of the data will change quickly and without notice.

A product company will not hold up a product release for the documentation, review, approval, or downstream integration of non-customer-facing interfaces. It will not delay updating statistical counters, event formats, configuration file formats, or UI outputs because back-office tools may be unprepared. In the management space, requirements are around CIM creation and conformance, standards, and interoperability. In the Product Analytics space, requirements are around quick adaptation, edge-case coverage, and continuous business leverage. A Product Analytics solution cannot expect schematized or specially formatted data from engineering; it needs to be able to quickly adapt to new changes and new information in newly-formatted or newly-descriptive data while maintaining semantic continuity with existing tools.

Product Analytics implementation using a traditional approach is difficult.

It should now be clear that a successful Product Analytics solution will manifest as a complex, high-performance data warehouse and set of associated tools, both for loading and for OLAP tooling.

Given the above exposition, we have the following requirements for a typical data warehousing project:

- 1.) A complex, custom data model describing a wide array of data representations including text tables, name-value pairs, log files, CLI outputs, statistical dumps, and event listings.
- 2.) A parser or set of parsers conforming to (1) and generating some intermediate representations, such as staging tables.
- 3.) A highly detailed, custom target data model describing configuration, status, identification, relational, statistical, and event-related aspects of all relevant physical and virtual components of a product over time.
- 4.) A multi-terabyte, high-performance, near-real-time relational database instance to contain current and historical data loaded according to (3).
- 4.) A high-performance ETL integration of (1) through (4).
- 5.) A representation of the semantics of data in (1) and (3), and how that relates to necessary tooling for downstream users.
- 6.) A set of tools geared towards the needs of each distinct type of consumer of the POD. These tools will necessitate query tuning and extensive development for usability and comprehensiveness.
- 7.) All of the above need to be quickly and extensively modifiable based on changes in operational data representation or product architecture.

Such a project is inherently time-consuming and risky.

Taking an inventory of the above development challenges, we would expect to see members of each of the following roles in a typical project. Depending upon the specifics of the project, additional people or outside consulting services may be required.

- Procedural programmers - for (1) and (2), possibly (4)
- Data warehouse architects - for (3) and possibly (4)
- Project manager - for (5) and for coordination among the team
- GUI programmers - for (6)
- DBAs - for maintenance and tuning of (3) and (6)

IT operational resources for equipment procurement, maintenance, and integration

Because of (7) all of these roles must be ongoing.

The vendor may not recognize or prioritize Product Analytics goals properly.

Multi-terabyte data warehousing and high-performance distributed parsing and loading systems for complex data sets are not common skills among IT professionals. As a result, the high-performance and complexity requirements necessitate that each team member be proficient and masterful.

A-team resources at a healthy product company will be diverted to product development. Back-office projects with these sorts of requirements tend to succeed only as a result of particular personalities, or as a result of overstaffing in people, overspending in equipment and consulting, and overextending in time.

IT resources provide further complication and cost.

Typical data warehouses require manual source data modeling, manual database modeling, manual parser and loader writing, and extensive UI work for any major product revision or for any new product; these activities need to be coordinated and often take multiple man-years to complete.

A typical data warehousing project will require multiple instances of IT infrastructure for developing a new version while the old version continues to run; such requirements often bog the project down with multi-month to multi-year delays waiting on IT resources.

Reliance on IT also typically means that one of a very few expensive row-store databases will be used in the project. It is not uncommon for such databases to require, just for licenses and high-performance hardware, seven figure up-front capital expenditures. These costs are in addition to those discussed above.

Conclusion.

Compared with our combination of SPL technology and hosted delivery, we believe that a typical in-house Product Analytics infrastructure project is at least 100 times less efficient than it should be, both lasting and costing at least 10 times what it would using our platform... probably a lot more.

It should be clear that such a project has a significant chance of failure, or at least of very limited success. It requires a healthy seven figure per-annum commitment, a significant up-front investment of money and time while the solution is being developed from the ground-up, the right people, and an organization which can communicate the requirements to the project team quickly and accurately enough to guide ground-up development for the right feature sets for all audiences.

A Glassbeam vs. In-House Customer Comparison

The customer had been working on in-house Product Analytics for years.

At one current customer, an in-house team had been working to deliver Product Analytics value to the business for about three years before they purchased the Glassbeam product. During that time, the project had been staffed with 2-4 people at any given time, with multiple dedicated compute and storage servers for their use. They had clearly been paying significantly more for their in-house effort than they are currently paying for Glassbeam.

There was little in the way of a comprehensive Product Analytics roadmap in-place. The team was pushed and pulled according to which individuals were experiencing hot issues they figured Product Analytics could address. We came in with a complete solution.

To compare the results of the in-house effort with Glassbeam's capabilities, let's start by getting a glimpse at the sort of semi-structured POD Glassbeam operates on.

The data.

A POD bundle arrives at the vendor's site daily from each installed product in the field. It amounts to about 60GB per month. Here are just a couple of sanitized snippets:

```
X-bundle-location: unknown
X-bundle-time: 1184839212
X-autosupport-type: autosupport
Message-ID: <SJCEXFE039n1xxWn9R600000df9@mail.xxxxx.com>
X-OriginalArrivalTime: 19 Jul 2007 10:00:12.0954 (UTC) FILETIME=[9948EFA0:01C7C9EB]
```

```
===== GENERAL INFO =====
GENERATED_ON=Thu Jul 19 03:00:03 PDT 2007
VERSION=xxxxxxx 999.5.0.0-47161
SYSTEM_ID=5FP4144010
MODEL_NO=xxxxx
HOSTNAME=rig22.xxxxxxxx.com
LOCATION=
ADMIN_EMAIL=
UPTIME= 03:00:03 up 20 days, 13:54, 2 users, load average: 0.14, 0.06, 0.01
```

```
===== SERVER USAGE =====
Resource          Size GiB  Used GiB  Avail GiB  Use%
-----
/backup: pre-comp      -      10.0      -          -
```

/backup: post-comp 677.3 0.8 676.5 0%

.....

Current Alerts

```
-----  
Alert Time            Description  
-----  
Thu Jun 28 13:09    Encl 1 (5FP4144010) Disk 9 is absent and should be replaced  
Thu Jun 28 13:09    Encl 1 (5FP4144010) Disk 10 is absent and should be replaced
```

.....

===== MESSAGES =====

```
Jul 14 18:00:01 rig22 xxsh: NOTICE (tty=<>, session=15341, host=<>) root: command "filesys show space 2"  
Jul 14 18:00:01 rig22 xxsh: NOTICE (tty=<>, session=15347, host=<>) root: command "filesys show space 1"  
Jul 14 18:00:01 rig22 xxsh: NOTICE (tty=<>, session=15348, host=<>) root: command "system show hourly-  
status"  
Jul 14 18:00:01 rig22 logger: at 18:00:01 up 16 days, 4:54, 4395830 NFS ops, 0.1 GB data col. (0%)  
Jul 14 19:00:02 rig22 xxsh: NOTICE (tty=<>, session=17849, host=<>) root: command "filesys show space 2"  
Jul 14 19:00:02 rig22 xxsh: NOTICE (tty=<>, session=17855, host=<>) root: command "filesys show space 1"  
Jul 14 19:00:02 rig22 xxsh: NOTICE (tty=<>, session=17856, host=<>) root: command "system show hourly-  
status"  
Jul 14 19:00:02 rig22 logger: at 19:00:02 up 16 days, 5:54, 4395848 NFS ops, 0.1 GB data col. (0%)  
Jul 14 20:00:02 rig22 xxsh: NOTICE (tty=<>, session=20355, host=<>) root: command "filesys show space 2"  
Jul 14 20:00:02 rig22 xxsh: NOTICE (tty=<>, session=20361, host=<>) root: command "filesys show space 1"
```

.....

...and so on - in fact, there are 80+ such differently formatted sections, with formats for each differing over 100 releases. Source content includes configuration files, event logs, and statistical dumps for all objects and relationships deemed important for support debugging purposes.

The Glassbeam Model - Unification.

Our data model embodies:

- a unified source data model (document level, for parsing and associating the above data)
- a unified object data model (properties of disks, adapters, shelves, and so on)
- a unified relational data model (the way the DW is to be stored for optimal query performance and semantic representation)

Despite variations in content and semantics over the lifetime of the product, we were able to generate a single data model which naturally captured the attributes, component

relationships, and semantics of all of them. The in-house project needed if-then-else criteria pre-designed for each downstream query.

Complexity, Scalability, and Performance.

We know that our customers want to answer questions like "did this system, or any of its components, undergo a configuration change at any point in time?". This requires modeling the relationships among all components as either temporary or permanent, tracking those, allowing for different attribute classes (configuration vs. status, for example), storing them differently for optimization according to their cardinality, and so on.

Every event or statistic must be associated with the object(s) with which it is related in order to answer questions like "show me the media error rate by drive model over the whole install base". CRM data must be pre-joined in order to answer questions like "show me the aggregate throughput on all interfaces at this account".

Answering these sorts of questions with Glassbeam is, by design, a matter of a few seconds at most. By contrast, the in-house solution was incapable of answering these sorts of questions without custom-written scripts which would run for days. One infamous report, which we were able to duplicate and run in a few seconds, has been taking all day to run using the in-house solution, and impacting usability of other tools.

The in-house solution is a single table, storing about 250 columns in one row of data for each POD bundle. Many of those columns are dedicated to aggregated results over the bundle; count of this type or that type of component, concatenation of all instances of a particular attribute, etc. Queries which need different levels of aggregation or detail require that a script be run over the original POD bundles, raising query time to the level of hours or forcing a reload of the table.

Our approach was different. We realized that the complexity of the models involved, as well as the complexity of the questions which might be asked, demand a relational model capable of embedding these complexities. So far, we have not been asked a POD-based question we cannot answer from our database in a matter of seconds.

We track 600 data tables for the customer whose data is shown above. We have gone to quite a bit of trouble to make the model intelligible by an analyst, but there is a lot of data there and much of it is dedicated to tracking relationships and states of things. We track about 5K columns (about 2300 unique), over 36 objects. Our object model is a general DAG, not a tree. Our fact tables add up to about 2B rows; our UI joins average 15 tables per join (up to 40).

By comparison to the in-house solution's pre-aggregation approach, we're tracking hundreds of statistics down to a 10 minute interval, associated with all the objects related to those statistics (disk, shelf, raid group, adapter, volume, FC interface, ...), for every disk on every system in the entire installed base for a year. Roll-ups are to be

done for the UI at query time, depending on user choices; we have a self-imposed requirement of two seconds per page load for these views in the Support Portal. That's quite a challenge, and that's just one of the fact tables.

It would be a challenge just to maintain all the parsing, ETL code, DDL, and optimizations for our configuration by-hand. Our KB of SPL code generates the parsing, staging, and ETL of the 80+ POD sections over 100 different releases, 850KB of DDL for the data warehouse, 260KB of optimizations for the data warehouse, and all the metadata necessary to create the OLAP and workflow portals for this customer.

Conclusion.

Product vendor organizations do not have the business or technical structures in place to consistently innovate in the Product Analytics space. Fundamentally, there is a limit to the effective scope and complexity of back-office solutions. Product Analytics infrastructure needs require a lot of continued engineering innovation. Back-office projects are typically muddled in political infighting and beholden to disinterested parties such as IT.

When components of the solution can be bought on the market, though, then the back office can use what the A-team builds somewhere else, but at a much lower cost. There is a tipping point where in-house, back-office projects ought to be replaced by market solutions. With the availability of Glassbeam, that point has come for Product Analytics solutions.

W.r.t. other potential competitors, we believe that what we've achieved with SPL gives us a strong competitive advantage over anyone taking a more traditional approach. As our roadmap of continued Product Analytics innovation helps us to broaden that advantage, we should maintain our dominant position in this space for the foreseeable future.