Objectivity Case History (Alan Ezust)

Customer Information

Customer:	Prior
Industry:	Aviation/Aerospace
Application Domain:	Air Traffic Control
Status:	Under Development
Platform:	HPUX 10.20
Compiler:	ACC 1.12 and planning to upgrade to 1.15
Other Tools:	Rational Rose, CSRCS Explorer, Rogue Wave (ACC) libraries STL + tools
Date:	December 4, 1998
Project Started:	October 97
Expected Completion:	December 99

Application Overview

The product under development is called FPS, or Flight Planning System. In many ways, it is similar to an Air Traffic Control system. The main difference between the two is the legal distinction.

An Air Traffic Control system actually controls the plane, instructing the pilot PRECISELY where to fly and where to approach and land. The FPS is used by Service Specialists, who can only give advice, rather than actual orders. Pilots are allowed to disregard the advice. You can think of an analogy of traffic lights versus other signs on the road which give you directions. You **must** obey the traffic lights but you can ignore a lot of the signs on the road because they just give you pointers on where to go. A Service Specialist is like highway road signs.

Some airports in Canada which use service specialists now are Kingston, Windsor and London Ontario, Edmonton and Calgary Alberta, Some airports have neither, such as Smith Falls.

The FPS simplifies the duties of a Service Specialist by collecting data from many different sources and providing analysis tools into one system. The kinds of data which are stored include flight info, weather, NOTAM, and other aeronautical information. FPS provides weather analysis and local observation entry tools, generates integrated pre-flight briefings, and validates flight data. Using electronic flight strips, forms, and automatic alerting capabilities, FPS supports en-route advisories, search and rescue operations, and local airport and tower movement advisories.

One of the features of FPS is its dual-redundant functionality. Using Objectivity FTO/DRO, they hope to implement a hot-failover. However, they want to use only two servers, each with one APs/Lockserver/DB group, and replicate everything. But to get true hot-failovers to work, they need to get around the quorum-restriction which is imposed by Objectivity. Therefore, we promised functionality which would give the customer API access to the quorum check. Basically, a client which fails to get a quorum on a DB can call a method which disables the quorum check so the client can still modify the replica. As long as only one replica is being updated by clients, the other can re-synch to it when the connection is restored. If the client "lies" about it being the only replica which is being updated, then the resynchronization will fail somehow. The exact scenarios are not known to me at this time, but this is the "2-machine FTO/DRO solution" which is referred to in the MRDs.

They also plan to use a dual-redundant LAN, so that one network failure will not interrupt service and packets can be routed to the alternate route. During the evaluation process, they wanted us to provide them with a system which would do this, but after a lot of wrangling, they agreed that this is really something they should get from network experts, rather than ODBMS experts such as ourselves. Afterall,

as long as TCP/IP packets can get from the client to a host by a particular hostname, objectivity clients should work too, right?

Development Environment

Currently developing on HP UX 10.20 HP B132L+ hardware. They have reservations about upgrading to HP UX 11 until it is more stable. For compilers they're using C++ ACC 1.12 and are planning to upgrade to version 1.15.

Their schema is captured in a Rational Rose model. The Rose code generator spits out most of what they need to get it working in DDL. However, it does not generate the proper stuff when parameter types are ooHandle(type) macro definitions. They filed a bug-report with Rose. I suspect they'll be very interested in our rose integration when it's ready.

They use a GUI Builder called ZAF - Zinc application framework.
They use a home-grown revision control system on HP
They also use Perl for data processing.

Data Storage Requirements

Schema

Their schema is still under development, but they estimate that approximately 50 classes will be in the schema. The inheritance tree will be only 0, 1, or 2 levels deep. The situation where inheritance will be used the most will be for representing different "kinds" of aviation messages.

Data Quantities

The amount of data in the database really depends on a lot of factors, but the following table can explain what kind of data amounts we are talking about

Kind of Event	Number of Objects per event	Accepted Frequency
Flight	4	400 flights/day
Weather Bulletin	3	5 bulletins/minute

The database only stores 1-2 days worth of data at any given time.

Concurrency

There could be up to 50 database clients running against the database on a large installation. Or as few as 1. The clients can be "weather trackers" or "flight trackers". The weather clients will only be reading data which is being stuffed in from external sources. The flight trackers will be performing reads and writes every time a plane takes off or lands.

Distribution / Containerization

They want to use different databases for different types of data, and within each database, they will have many containers responsible for different objects.

Data such as user accounts, userids, passwords, roles, and privileges, will all be stored in one container.

Perhaps each flight and all its related info will be in its own container, and these will all be stored in one "flights" DB.

Another DB will contain weather info.

And finally, another DB will be used to hold geographical data. Every point in Canada that is important from an aviation point of view will be represented as an object, such as weather beacons, airports, weather stations. There could be up to 10,000 objects in this database. Each such point has a "responsibility sector" (represented by one of the airports or sites where this software is running). Associations will be used to group these geographical objects together into such sectors.

Buying Criteria

ObjectStore was eliminated early because it does not support fault tolerance.

Versant can only replicate data twice. Objectivity can replicate many times. Multiple replicas just seems to be the right thing to do and this was seen as an asset because of future considerations of having several replicas of data across the country. Basically, we won this deal because of our FTO/DRO. And when these guys get up and running, they will be an excellent reference!

Why Objectivity

Vince's case histories have some excellent discussion of this. I recommend you check them out as well.

The features of Objectivity are rated on a 1-5 scale, 1 being lowest, and 5 being highest in importance.

Feature	Importance
Distribution/Storage Hierarchy	4
Clustering	2
One logical view	5
language interoperability	n/a
page service architecture	don't know
container level locking	useful because in other systems they are thinking of replicating the data across the federation and we just want to read from it. The lack of network traffic will be useful. The concept of the container becomes more important as a result
composite object support	4
backup/restore	4
indexes	3
schema migration	1-2
MROW	4
object versioning	I haven't used it. Might need it later
PQL	5
STL	5 but currently there are namespace conflicts so they are not using it now
multithreading	currently disabled for HP so they are unable to use it. Not happy about that. Looking forward to using it at some later time. Fortunately, it's not crucial right now
heterogeneous architecture	haven't used it yet
2-server FTO/DRO solution	5 (still waiting for it)

Contact Information

Objectivity Rep: Vince Ramoutar
Customer Contact: Mark Lovell
Phone:
Email: