Efficient Database Implementation of EXPRESS Information Models

Efficient Database Implementation of EXPRESS Information Models
PhD Thesis, David Loffredo

I have made a PDF version of my PhD thesis and defense slides available for download. Although I am happy to have you download, read, and use this, please note that re-distribution of these documents are not allowed without permission. If someone else would like a copy, please send them back to my official page:

`http://www.steptools.com/~loffredo/`

If you plan to reference any of this material in another document, the proper citation is:

> Loffredo, David, *Efficient Database Implementation of EXPRESS Information Models*. PhD Thesis, Rensselaer Polytechnic Institute, Troy, New York, May 1998.

Questions and comments can be mailed to me at `loffredo@steptools.com`.

Thanks and happy reading!

Efficient Database Implementation of EXPRESS Information Models

David Loffredo
loffredo@steptools.com

Doctoral Thesis Defense
April 10th, 1998

Overview

Problem Statement

SDAI Database Implementation Framework
- Upload/Download, Cached, and Direct Access Architectures
- Implementation Case Studies

SDAI Operational Benchmarks
- PartStone, BOMStone, and NURBStone

Benchmark Results
- Benchmarks on Oracle and ObjectStore
- Effect of optimizations
- Comparison of Direct and Alternate Bindings

Conclusions

Problem Statement

Product databases are essential for integrated design and manufacturing.

- Product databases today are rare and not open.

EXPRESS information models can define open engineering product databases.

- Part of International Standard ISO-10303 (STEP)
- Standard Data Access Interface (SDAI) — the API for EXPRESS-defined information
- Engineering apps tightly tied to model, standard model and standard API make open product databases possible.

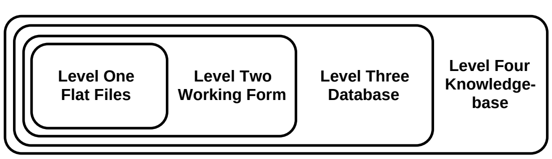
How can we provide SDAI access to product databases?

Previous Work

Access to EXPRESS-defined information using files and working-form is well understood.

- Many CAD and PDM systems have file exchange implementations.
- Several working-form SDAI implementations exist.

SDAI access to EXPRESS-defined information in database or knowledgebase not well explored.



Overview

Problem Statement

SDAI Database Implementation Framework

- Upload/Download, Cached, and Direct Access Architectures
- Implementation Case Studies

SDAI Operational Benchmarks

- PartStone, BOMStone, and NURBStone

Benchmark Results

- Benchmarks on Oracle and ObjectStore
- Effect of optimizations
- Comparison of Direct and Alternate Bindings

Conclusions

Framework for EXPRESS Database Implementation

Database Schema Definition

Choose Information Model

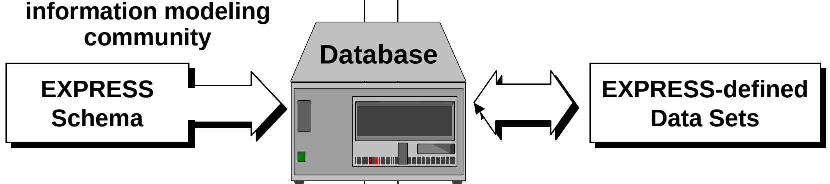
Map EXPRESS Structures into Native Database DDL

Well-explored by the information modeling community

Data Access Software Architecture

Construct SDAI Access Software

Move Model-defined data into and out of database with DML

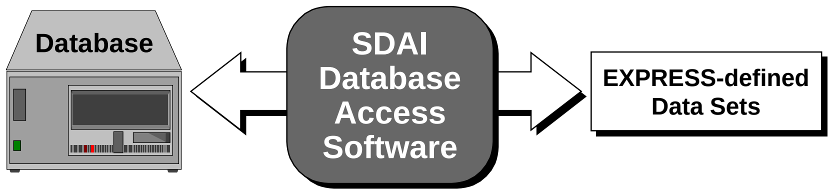


Construct SDAI Access Software

Software must be able to move EXPRESS-defined data into and out of the database system. Some of the design parameters:

Access Style
- Upload / Download SDAI
- Cached SDAI Binding
- Direct SDAI Binding

Binding to EXPRESS
- Code Generation
- Data Dictionary



SDAI Access Architectures

Based on the quantity of data and time of transfer, we can identify three architectures:

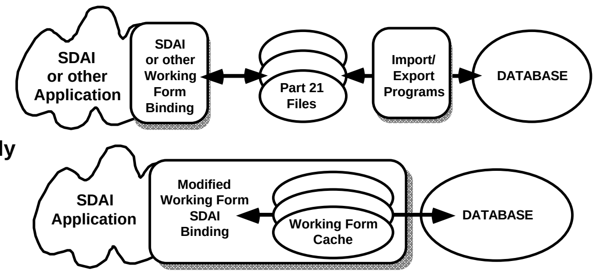
Upload/Download SDAI Binding
- Entire model, off-line batch transfer
- Move an entire model from database to physical file and vice versa.

Cached SDAI Binding
- Entire model, on-line batch transfer
- Move an entire model to and from main memory. Operate on it in main memory with SDAI operations.

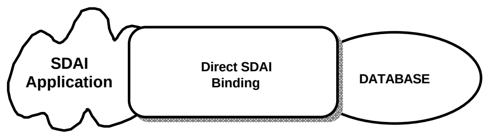
• Direct SDAI Binding
- Individual values, on-line incremental transfer
- Operate on a model incrementally within the database, using the SDAI operations.

SDAI Access Architectures

Upload/Download Cached
- Easier batch algorithms
- Can reuse working form binding
- DB features on model only
- High latency, but access at main-memory speeds
- Potential for multiple DB systems



Direct
- More complex interactive algorithms
- Minimal code reuse
- Can use special DB features (locks, concurrent update)
- Low latency, but access at DB operation speeds
- One DB system only

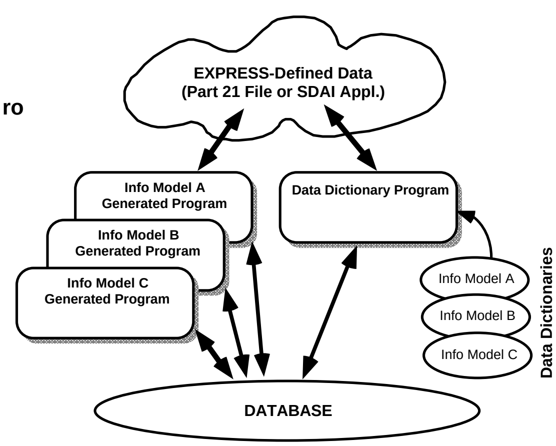


EXPRESS Definition Binding Styles

Determines how interface software is configured to use a particular EXPRESS schema.

Code Generation
- Configure the interface at development time.
- Use an EXPRESS compiler ro generate program code.

Data Dictionary
- Configure the interface at execution time.
- Match data dictionaries for EXPRESS and the database system



Implementation Case Studies

Cross-section of engineering database systems and implementation techniques.

- ORACLE —Relational
- HP OpenODB — Relational / OO Hybrid
- Versant — Object Oriented / Multiple Languages
- ObjectStore — Object Oriented / Persistent

Access Style

Binding Style			
	Oracle and OpenODB Early-bound Import/Export	Oracle Early-bound Cached SDAI	Oracle ObjectStore Early-bound Direct SDAI
	Late-bound Import/Export	Versant Late-bound Cached SDAI	Late-bound Direct SDAI

Implementation Efforts

	System	Effort	Reuse
Upload Download	Oracle	5000 lines	40,000 lines
	OpenODB	6000	40,000
Cached	Oracle	5000+	40,000
	Versant	3000	40,000
Direct	Oracle	11,500 (partial)	none
		91,000 (full est)	
	ObjectStore	200+	40,000

Overview

Problem Statement

SDAI Database Implementation Framework

- Upload/Download, Cached, and Direct Access Architectures
- Implementation Case Studies

SDAI Operational Benchmarks

- PartStone, BOMStone, and NURBStone

Benchmark Results

- Benchmarks on Oracle and ObjectStore
- Effect of optimizations
- Comparison of Direct and Alternate Bindings

Conclusions

Operational Benchmarks

First choose a STEP information model as the basis for the benchmarks.

AP-203 used as the basis for the benchmarks.

- Most widely used application protocol.
- First to be standardized.
- 14 Units of Functionality (UOFs) that cover a wide range of CAD and PDM information.
- Contains data common to many of the STEP APs.

Identify the Benchmarks

Looking at UOFs, we can identify three styles of engineering information:

- **Navigational — Hierarchical references (Geometry)**
- **Existence-dependant — Property-of references (Part Identification)**
- **Mixed — A combination of both (Bill of Material)**

Create benchmarks to exercise each style.

- **Consider data access operations only.**
- **Update operations out of scope.**

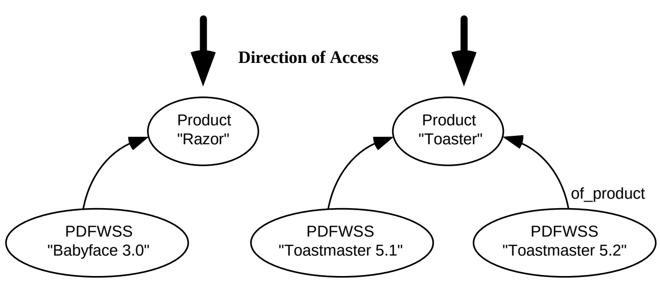
PartStone Benchmark

Traverse Part Identification Information

- **Existence-dependant modeling style, all definitions properties of a "product"**
- **Used by all STEP APs**

Print all versions of a single part.

- **Loop over all versions to find the one that points to a specific product**
- **Repeat operation on all products in a data set to scale up.**



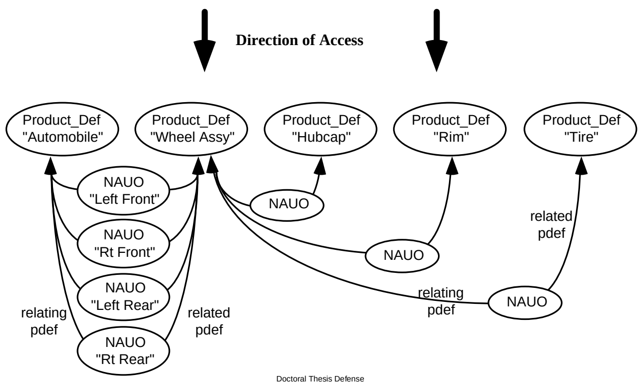
BOMStone Benchmark

Traverse Bill of Materials Information

- **Mixed modeling style, relationship from product to assy nodes existence-dependant , all others navigational.**

Print an assembly hierarchy.

- **For each node, print, then find all children. Repeat recursively.**



NURBStone Benchmark

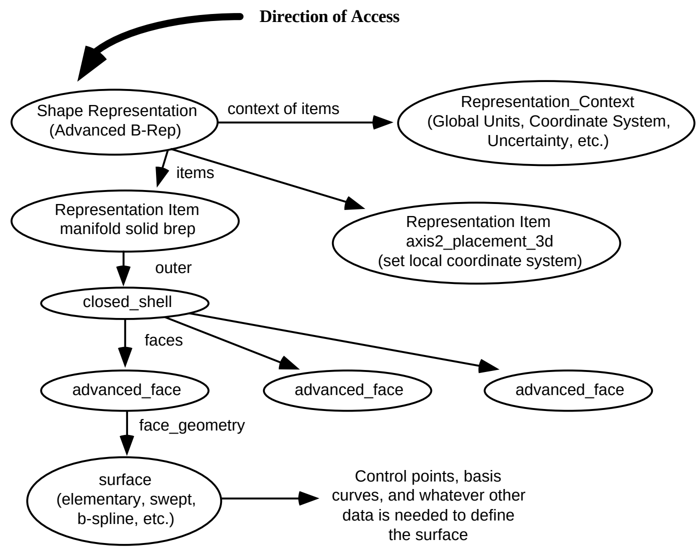
Traverse Geometry Information

- **Navigational modeling style, all definitions reachable from a “shape representation.”**
- **Used by all STEP APs**

Print the structure and attributes of a shape from the top-level down to the cartesian points.

- **Perform a depth-first search of the shape data. Like a recursive descent parse algorithm.**
- **Benchmark covers 50 different geometry definitions.**

Shape Representation Data



Overview

Problem Statement

SDAI Database Implementation Framework

- Upload/Download, Cached, and Direct Access Architectures
- Implementation Case Studies

SDAI Operational Benchmarks

- PartStone, BOMStone, and NURBStone

Benchmark Results

- Benchmarks on Oracle and ObjectStore
- Effect of optimizations
- Comparison of Direct and Alternate Bindings

Conclusions

Benchmark Experiments

Run the benchmark experiments on:

- Direct Binding on Oracle Database
- Direct Binding on ObjectStore Database
- Working-Form Binding using Files

Use data sets with 100 to 100,000 objects

Look at the effect of database optimizations on the benchmarks.

Also measure database load/extract performance to estimate performance of alternate binding architectures.

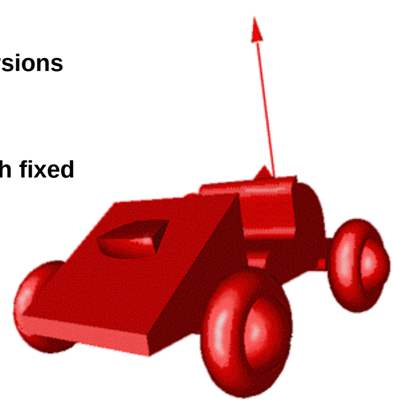
Benchmark Data

Programs were developed to build large data sets for the benchmark tests.

PartStone — Generate parts and versions with fixed num of versions per part.

BOMStone — Generate assy's with fixed num children and depth.

NURBStone — Duplicate the geometry from the STEPnet moonbuggy.



Optimizations

Several non-SDAI database optimizations were explored during the benchmark experiments.

Oracle

- **All Benchmarks — Collapsed many SDAI get_attribute calls into one SQL select.**
- **All Benchmarks — Added indices on important columns**
- **PartStone and BOMStone — Replaced SDAI loop with SQL join to improve USEDIN() operation.**

ObjectStore and Working Form

- **PartStone and BOMStone — Replaced SDAI loop with backpointers to improve USEDIN() operation.**

Benchmark Results Outline

NURBStone Results

- **Effect of Access Performance**

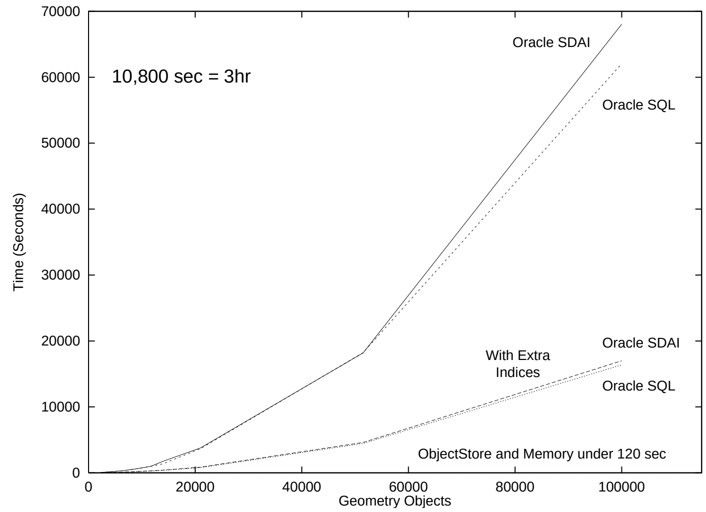
PartStone and BOMStone Results

- **Effect of Usedin() Optimizations**
- **Effect of Relational Indices**

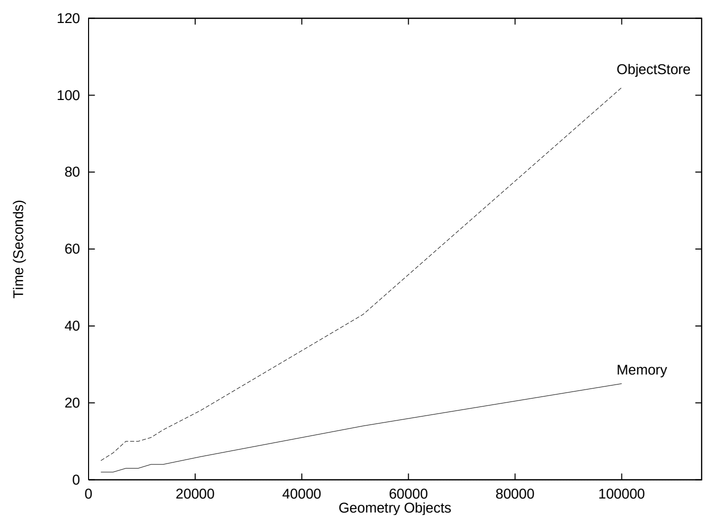
Load/Extract Results

- **Effect of SDAI Architecture**
- **ObjectStore Alternate Bindings**
- **Oracle Alternate Bindings**

NURBStone Runs



NURBStone Runs — Under 120 Sec.



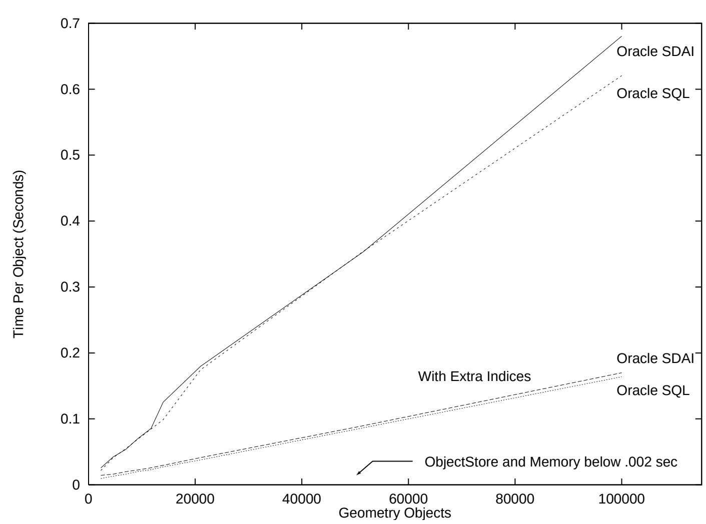
Effect of Access Performance

Using the NURBStone results, we calculated the relative speeds of the systems.

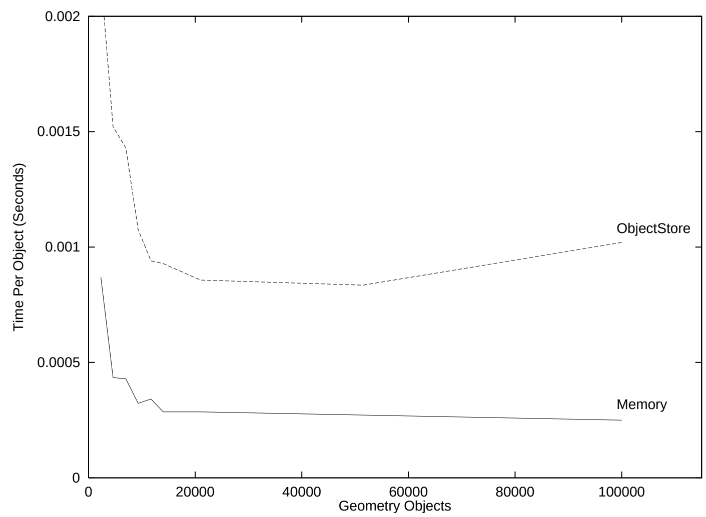
- **Oracle results were not constant. Cost increased with the size of the database. Appears to be O(n).**

System	Cost	Objs/second
Oracle	~.05-.7 sec/obj	1.4-20 obj/sec
ObjectStore	~.001 sec/obj	1000 obj/sec
Working-Form	~.00025 sec/obj	4000 obj/sec

Oracle Access Performance



ObjectStore and Working Form Access



Benchmark Results Outline

NURBStone Results

- **Effect of Access Performance**

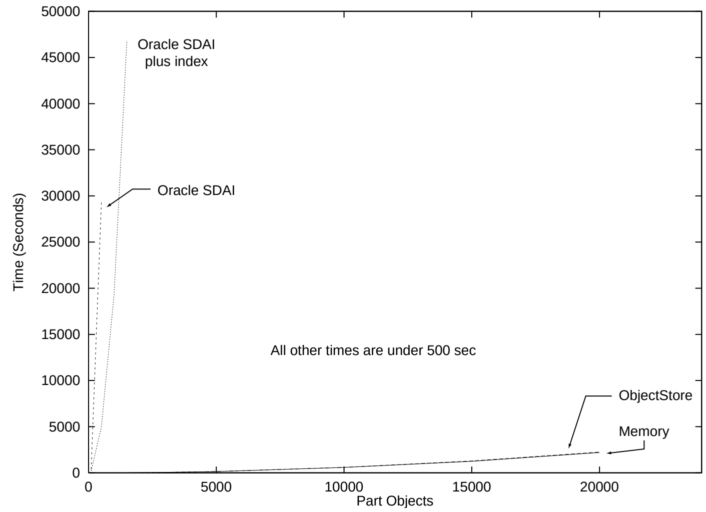
PartStone and BOMStone Results

- **Effect of Usedin() Optimizations**
- **Effect of Relational Indices**

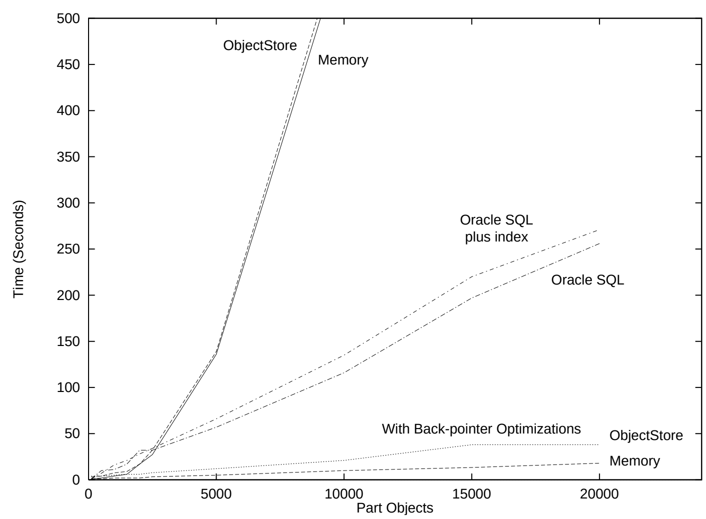
Load/Extract Results

- **Effect of SDAI Architecture**
- **ObjectStore Alternate Bindings**
- **Oracle Alternate Bindings**

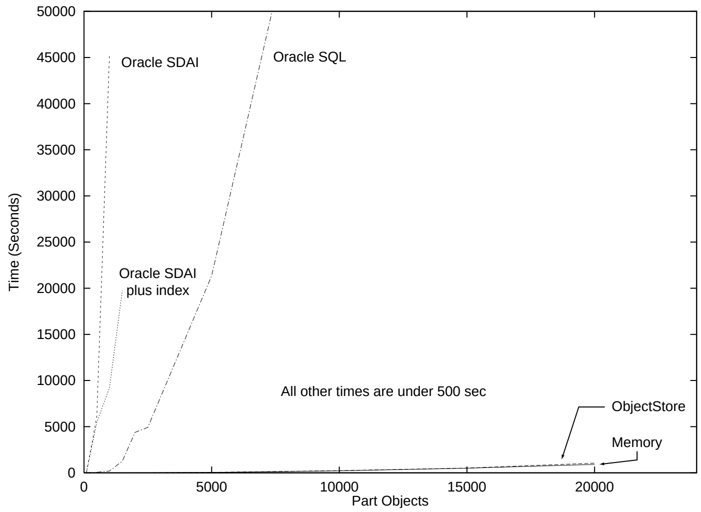
PartStone



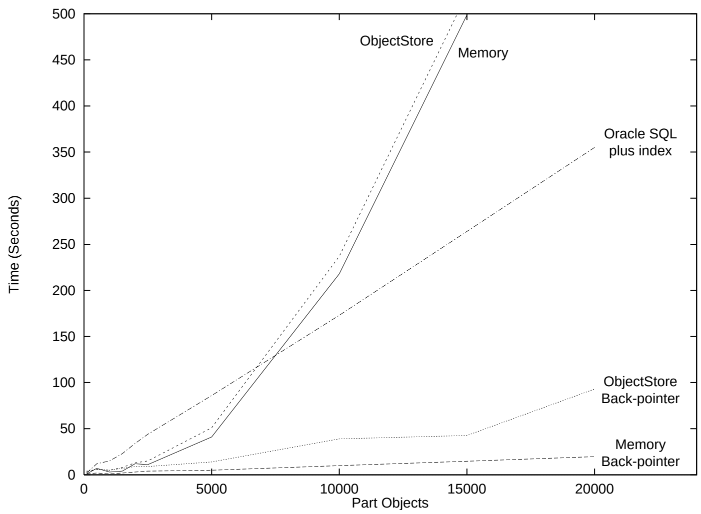
PartStone — Under 500 Sec.



BOMStone



BOMStone — Under 500 Sec.



Effects of Usedin Optimizations

Oracle

- Replacing SDAI loop with SQL query improved $O(N^3)$ behavior to roughly linear behavior.
- Some odd behavior WRT indices.

ObjectStore and Working-Form

- Adding backpointers reduced the algorithm complexity from $O(N^2)$ to linear.

Effect of Relational Index Optimizations

Oracle indices had different effects on the Oracle benchmarks.

NURBStone

- Most effective optimization. Improved both SDAI and SQL versions. SQL optimization of little value.

PartStone

- Of minimal importance. Improved SDAI-only case slightly, but actually slowed the SQL join slightly.

BOMStone

- Very effective. Improved SDAI-only case slightly, but improves the SQL joins dramatically.

Benchmark Results Outline

NURBStone Results

- Effect of Access Performance

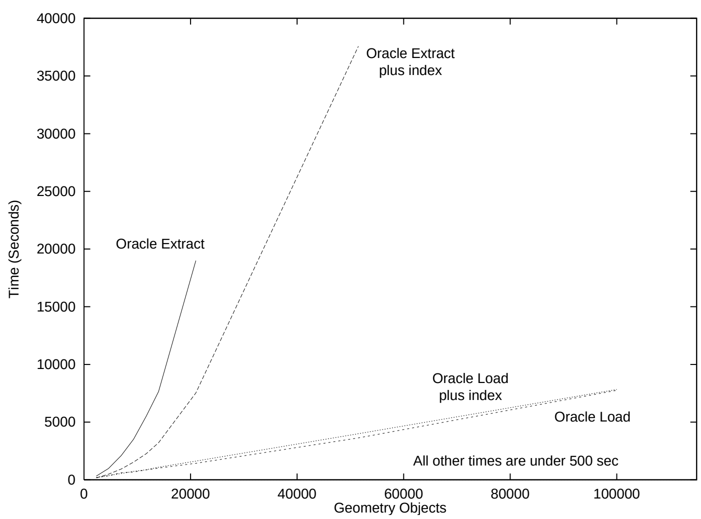
PartStone and BOMStone Results

- Effect of Usedin() Optimizations
- Effect of Relational Indices

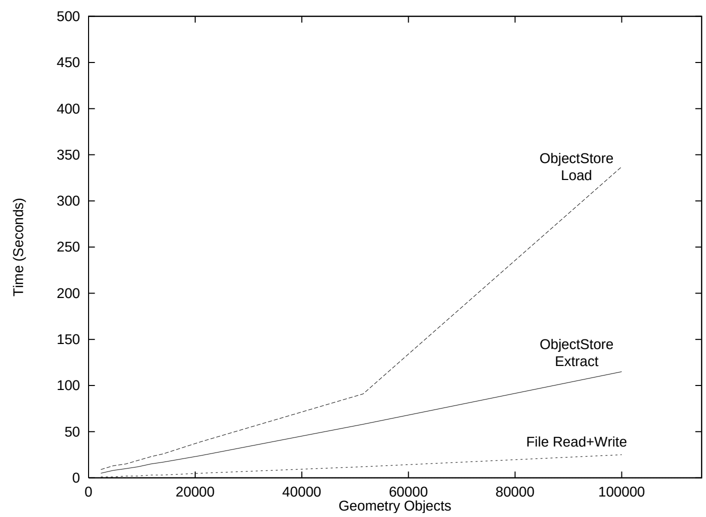
Load/Extract Results

- Effect of SDAI Architecture
- ObjectStore Alternate Bindings
- Oracle Alternate Bindings

Load/Extract Measurements



Load/Extract Measurements — Under 500 Sec.

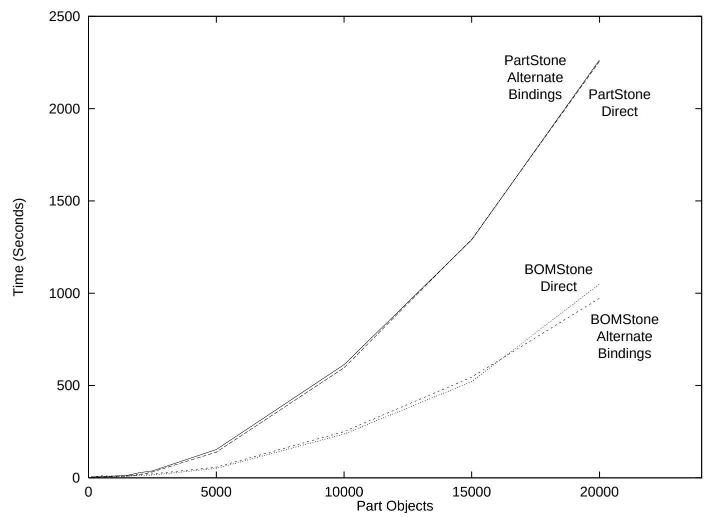


Effect of Access Architecture

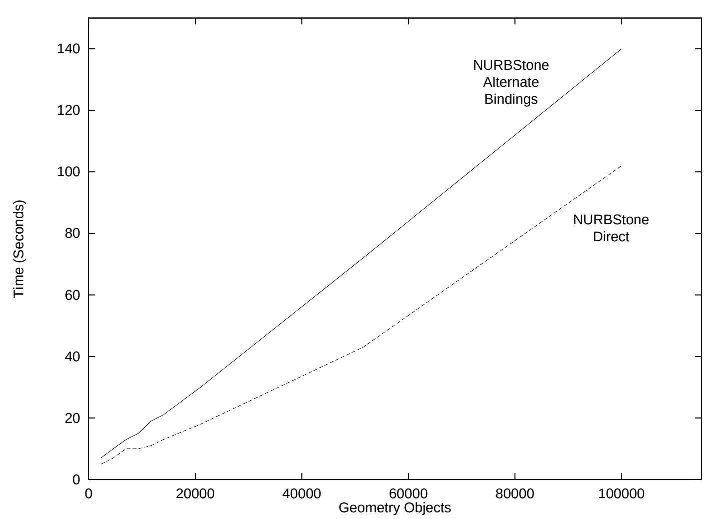
Estimate the performance of the alternate bindings by combining working form binding times with the load and extract times on databases.

- **Estimate for upload/download and cached bindings.**
- **Compare with results for direct bindings.**

ObjectStore — PartStone and BOMStone



ObjectStore — NURBStone

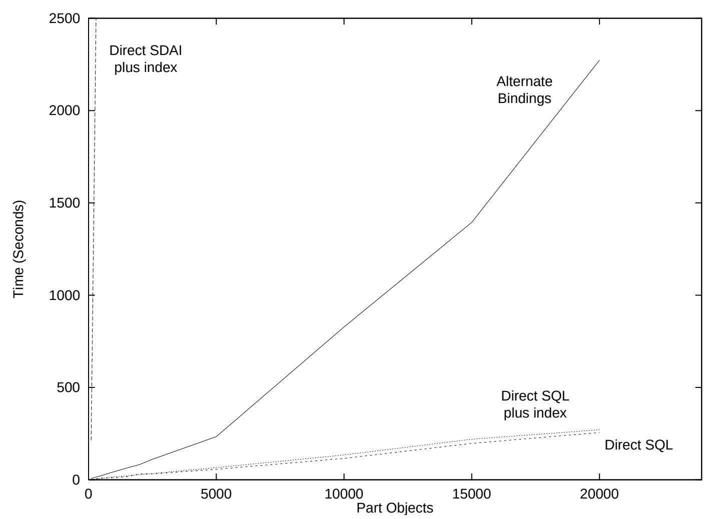


Effect of Architecture — ObjectStore

ObjectStore

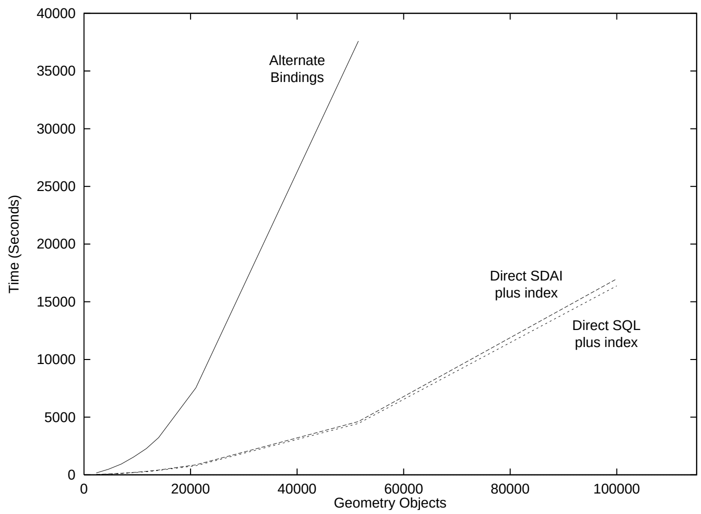
- **Very little difference between direct and alternate bindings.**
- **Alternate bindings a cost-effective choice.**
- **Validates choice of cached binding for Versant binding.**

Oracle — PartStone



- **BOMStone results are similar.**

Oracle — NURBStone



Effect of Architecture — Oracle

Oracle

- **Alternate bindings better than unoptimized SDAI, but not as good as optimized SQL access.**
- **For NURBStone-type access, both SDAI and SQL are better.**

Overview

Problem Statement

SDAI Database Implementation Framework

- Upload/Download, Cached, and Direct Access Architectures
- Implementation Case Studies

SDAI Operational Benchmarks

- PartStone, BOMStone, and NURBStone

Benchmark Results

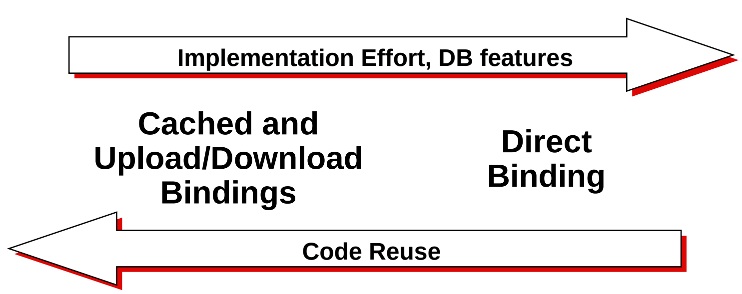
- Benchmarks on Oracle and ObjectStore
- Effect of optimizations
- Comparison of Direct and Alternate Bindings

Conclusions

Conclusions

Identified data access architecture for building SDAI database bindings:

- Upload/Download, Cached, and Direct
- Direct bindings can take advantage of most DB features, but are the most difficult to implement.
- Other styles require less effort and may satisfy all application requirements.



Conclusions

Defined benchmarks to measure SDAI access behavior

- Based on AP-203, but definitions shared by many of the STEP application protocols.
- Covers Navigational, Existence-dependant, and mixed modeling styles for product data.
- Usedin() optimizations extremely important for existence-dependant (Part) and mixed (BOM) data.

Cached and upload/download bindings are a useful alternative to direct bindings.

- Much lower implementation effort. Allows code reuse.
- Performance influenced by load/extract behavior.
- Equal performance for ObjectStore.
- Better performance than plain SDAI for Oracle, not as good as custon SQL. Depends on algorithm and optimizations.

Future Work

There are a number of areas that could benefit from more exploration

- Range of algorithms appropriate for implementing SDAI operations
- SDAI access to non-database systems, like CAD or Analysis systems.
- Cached SDAI bindings across the network (Java, Corba, etc)
- Extend benchmarks to evaluate database update behavior.
- Explore some irregularities seen in Oracle extract behavior with indices
- Look at non-SQL RDB batch load/extract methods