

# Toward Assessing Approaches to Persistence for Java

John V. E. Ridgway    Craig Thrall    Jack C. Wileden

Convergent Computing Systems Laboratory  
Department of Computer Science  
University of Massachusetts  
Amherst, Massachusetts 01003 USA

`{ridgway,cthral1,wileden}@cs.umass.edu`

## Introduction

- Several approaches to persistence for Java now exist
  - Explicit file I/O
  - JDBC
  - Orthogonal persistence systems
- We have developed such a system: **Jspin**
- It is now time to assess these various approaches

# The Rest of this Talk

- Introduction to Jspin
- Assessment of persistence approaches
  - Quantitative
  - Qualitative
- Conclusions and future work

## The Jspin Approach

- Rationale - why did we build YAPJ (Yet Another Persistent Java)?
  - The Jspin API is very similar to that of OPJ
  - Others are working on similar approaches
- A seamless extension of Java
- Minimal barriers to adoption
- Maximal opportunities for Interoperability
- Suitable basis for future research

# Jspin APIs

- Per-class methods added:
  - `public void persist(String name)`
  - `public static class fetch(String name)`
- PersistentStore abstract class:
  - `public void beginTransaction()`
  - `public void commitTransaction()`
  - `public void abortTransaction()`
- All of these methods throw  
`PersistentStoreException`

# Jspin Platform

- JDK 1.0.2
- Two persistent store interfaces exist
  - TI/DARPA Open OODB
  - Single-user Mneme
- Unmodified 1.0.2 Java Virtual Machine
- “No” native method calls in kernel
  - Except for allocation of arrays of arbitrary class
  - Particular persistent object stores may use native methods

# Implementation Strategy

- **Jspin** does all of the in-memory maintenance
- Persistent stores are given the objects as an array of bytes and an array of object handles (identifiers)

## Persistence Proxies

- Each resident (potentially) persistent object has a persistence proxy
- Instances of classes compiled with the **Jspin** compiler act as their own persistence proxy
- Classes that cannot be compiled with the **Jspin** compiler must have explicit related proxy classes
- The proxy holds information about the status of the object and its memory and store locations
- Mappings from object to proxy are maintained by the `PersistentStore` object

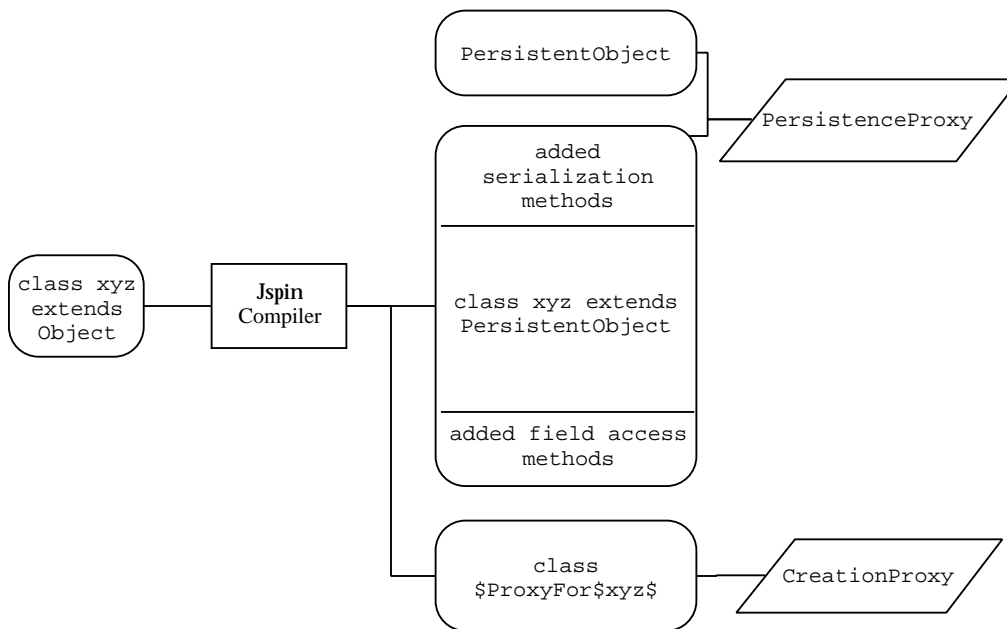
# The PersistentStore Class

- A partially-abstract class that is the kernel of Jspin
- Determines which objects persist
- Maintains mappings between objects and their proxies and between handles and objects
- Allocates object proxies and handles as necessary

## Persistent Store Interfaces

- The PersistentStore class is a partially abstract class that is extended to yield particular persistent store interfaces
  - This allows the particular implementation to override the behavior of the PersistentStore class
- The particular persistent store must provide methods to
  - create, read, and write objects of a specified size
  - begin, commit, and abort transactions
  - create an empty object handle and an array of empty object handles

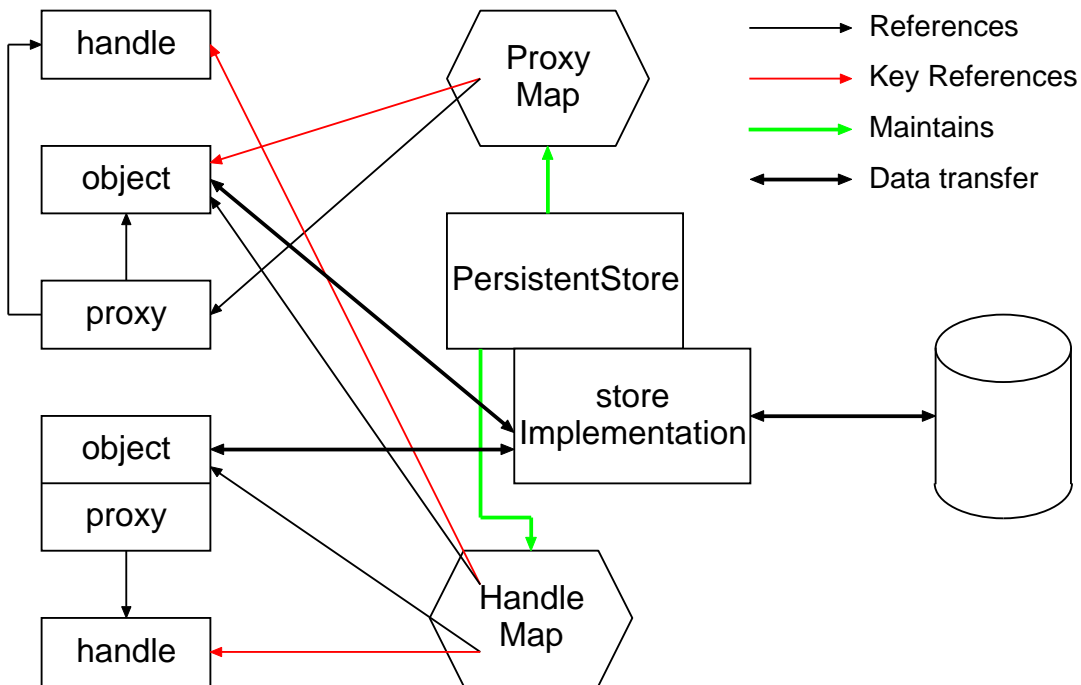
# The Jspin Compiler



## Limitations

- Unfortunately it does use one native method call
- Can only make persistent those objects that have been processed by the **Jspin** compiler, or those (special) objects that it knows about
- Must fetch arrays eagerly, as it does not modify array access code, nor can it modify the array class code
- Resident objects are never garbage-collected
- Multiple transactions in the same execution may fail

# The Jspin Architecture



## Assessment

- Criteria for assessment

- Performance
- Ease of use

- Types of assessment

- Quantitative
- Qualitative

- Systems assessed

- Jspin with both Mneme and Open OODB interfaces
- OPJ
- JDBC using MySQL and the GWE MySQL JDBC interface
- Raw Persistent Store - Mneme with C calls, single and multi-user

# Quantitative Assessment

- Ideally we would quantify all assessment criteria
- Performance Measure - the OO1 benchmark
  - Run on Tiny (2000 part) and Small (20,000 part) databases only
  - Run with larger buffer sizes than specified (except raw Mneme)
  - Mixture of “Local” and “Remote”
- Quantitative Usability Measures
  - I wish we had some, but we don't, yet...

## OO1 Results on Small Database

Measure	Cache	Jspin	Jspin	C/Mneme C/Mneme			
		Mneme	OpenOODB	OPJ	JDBC	Local	Remote
DB Size(Kbytes)		4296	28240	7160	4755	3208	
Load		667.82	3981.13	245.71	1086.28	21.541	
Lookup	cold	20.043	62.397	2.237	26.743	1.398	1.428
	warm	10.428	166.506	0.63	26.712	0.144	0.09
Traversal	cold	25.837	92.14	4.36	51.939	1.073	3.07
	warm	5.974	N/A	0.607	49.926	0.065	0.064
Insert	cold	11.159	26.704	1.44	7.939	0.668	7.939
	warm	8.548	N/A	1.073	5.282	0.516	6.141
Total	cold	57.039	181.241	8.037	86.621	3.139	12.437
	warm	24.95	N/A	2.31	81.92	0.725	6.295

# OO1 Results on Tiny Database

Measure	Cache	Jspin	Jspin	OPJ	JDBC	C/Mneme	C/Mneme
		Mneme	OpenOODB			Local	Remote
DB Size(Kbytes)		488	2400	972	N/A	392	
Load		44.33	145.63	18.98	100.06	2.28	
Lookup	cold	8.979	31.119	0.285	26.65	0.077	0.12
	warm	1.146	1.608	0.285	26.443	0.064	0.062
Traversal	cold	3.326	16.257	0.088	49.347	0.165	0.333
	warm	0.812	2.877	0.084	52.184	0.08	0.055
Insert	cold	3.894	13.221	0.967	5.255	0.284	1.565
	warm	4.217	N/A	0.9	4.98	0.316	2.441
Total	cold	16.199	34.155	1.34	81.252	0.526	2.018
	warm	6.175	N/A	1.269	93.607	0.46	2.558

## Qualitative Assessment

- **Seamlessness**
  - Code modifications
  - Support tools
- **Barriers to Adoption**
  - Language extensions
  - Compiler modifications
  - Virtual machine modifications
  - Operating system dependencies
- **Interoperability**
  - With preexisting Java classes
  - With code and data from other languages

# Qualitative Assessments

Aspect	Criterion	Jspin			C
		Mneme	OPJ	JDBC	Mneme
Seamlessness	Code Modifications	Little	Little	Much	N/A
	Support Tools	Moderate	None	None	N/A
Barriers	Language Extensions	None	None	Much	None
	Compiler Modifications	Yes	No	No	No
	VM Modifications	No	Yes	No	N/A
	OS Dependencies	Unix	Solaris	Any	Unix
Interoperability	Preexisting classes	Some	Yes	No	N/A
	Other languages	Future	No	No	N/A

## Future Work

- **Work on Jspin**
  - Migrate to JDK 1.1.x
  - Add capabilities to use Serialization and Reflection
  - Name management API
  - Migrate to Open OODB 1.1 and use kernel rather than C++ objects
- **Assessment**
  - Develop better qualitative assessment tools
  - Finish OO1 “large” database runs
  - Experiment with OO7
  - Other benchmarks
- **Polylingual Interoperability**