Georg Heeg eK
Baroper Str. 337
44227 Dortmund
Germany
Tel: +49-231-97599-0
Fax: +49-231-97599-20

Email: georg@heeg.de
http://www.heeg.de

Georg Heeg eK
Mühlenstr. 19
06366 Köthen
Germany
Tel: +49-3496-214 328
Fax: +49-3496-214 712

Georg Heeg AG
Seestraße 135
8027 Zürich
Switzerland
Tel: +41(848) 43 34 24
Integration of Cincom Smalltalk Systems  
- e.g. ObjectStudio inside VisualWorks

Prague, September 7th, 2006
About Georg Heeg eK

- Founded 1987, headquarter in Dortmund, since 1996 in Zurich, since 1999 in Koethen/Anhalt
- Consulting- and training company in Smalltalk
- Hotline support, maintenance, bug-fixes for ObjectStudio, VisualWorks and Visual Smalltalk
- VM-laboratory for VisualWorks and ObjectStudio
- Porting service of old VisualWorks applications to 5i/7
- Technology-partner of CINCOM

Corporate Mission: Make Sophisticated Projects a Success for the Customer!
Some Background

- 1994/96 Cincom bought ObjectStudio
- 1999 Cincom bought VisualWorks
- ObjectStudio customers keep asking for VisualWorks features
- ObjectStudio 8 integrates VisualWorks and ObjectStudio
ObjectStudio and VisualWorks

- Both ObjectStudio and VisualWorks are Smalltalk systems.
- Both are owned by the same company Cincom.
- VisualWorks like Squeak goes back to the original Smalltalk-76/-78/-80 developments at Xerox PARC.
  - Originally called Smalltalk-80, ObjectWorks later.
- ObjectStudio
  - was developed as “Enterprise Object-Oriented Development Environment”
  - Originally called Enfin.
Key Features

- **Enfin/ObjectStudio**
  - Ease of Use
  - Enterprise Integration

- **Smalltalk-80/ObjectWorks/VisualWorks**
  - Execution speed
  - Sophisticated meta-modelling features

- **Customers want both**
  - Speed: always
  - Ease of use: to get started
  - Enterprise integration: mostly
  - meta-modeling: seldom
    - but if they need it, they outperform having it.
From Forbes.COM

- “A press conference is called.
- “The media is alerted.
- “The top executives gather for much backslapping and multisyllabic descriptions of the synergistic opportunities abounding in this, a fresh new partnership.
- “There's a champagne toast
- “and then the most extraordinary thing happens
  - “…nothing! Nothing at all.”
Synergy doesn’t Work

- VisualWorks VM for ObjectStudio
  - “HPOS”
  - Was started and
  - ... was not completed
- Ideas to integrate ObjectStudio into .NET
  - ... didn’t make it
Theory: Smalltalk is Different

- In Smalltalk programming is modeling.
- Thus every theory can be modeled well.
- Thus every computer technology can be modeled well.
- Thus every other Smalltalk system can be modeled well.
ObjectStudio 8

- Model ObjectStudio by VisualWorks in Smalltalk
- Use Meta-Modeling
- Use all reflection capabilities
  - Compiler
  - Debugger
  - Namespaces
The Goal

- Both ObjectStudio and VisualWorks live in the same image the same time.
- Both Environments share the same Smalltalk kernel and base classes to provide identical functionality.
- Mixing and matching is seamlessly possible.
- Both ObjectStudio and VisualWorks application Smalltalk code works unchanged.
  - There will always be exceptions to this rule.
The Team

- Suzanne Fortman (Program Director, Cincom Smalltalk)
- Andreas Hiltner (ObjectStudio 8 Lead Engineer)
- Mark Grinnell (ObjectStudio Lead Engineer)
- Eduard Maydanik (GUI Engineer)
- Kim Thomas (Testing, Packaging, Support)
- Eliot Miranda (VisualWorks Lead Engineer)
- Alan Knight (Store Consultant)
- Jörg Belger (VisualWorks Wrapper)
- James Robertson (Product Manager)
- Georg Heeg (Idea)
- Helge Nowak (Sales Engineer)
- Monika Laurent (Marketing)
The First Step

- Vocal chords surgery in June 2004
- No permission to speak for a week
ObjectStudio in VisualWorks

After the spring 2004 meetings in Santa Clara and Cincinnati Georg Heeg started to investigate how to embed ObjectStudio in VisualWorks.

He takes the following steps:

1. Define the environment in VisualWorks for ObjectStudio to live in > 2004-06-22
2. Provide a reader for ObjectStudio txt files > (list of txt and cls files) 2004-06-22
3. Get ObjectStudio class definitions defined in C not in Smalltalk > 2004-06-23
4. Add VisualWorks specifics to classes like Object and fix problems in the class definitions > 2004-06-23
5. Read the ObjectStudio boot file and fix language differences issues > 2004-06-23 to 2004-06-24
6. Check the result and publish in Store > 2004-06-24

This defines milestone 1.

2.1 Load back from Store and run all Initializers > 2004-06-24 to 2004-04-27

This defines milestone 2, reached 2004-04-27.
End of First Step

- Smalltalk works
- UI does not work
- Primitives don’t work
The Second Step: Primitives

- Jörg Belger starts working at Georg Heeg eK Köthen October 1, 2004
  - Coauthor of the original Ce VM as intern (100 days)
  - Coauthor of computer games
  - Background in C++
- ObjectStudio MFC DLL functions are implemented
  - One per one at a time
The First Window Opens

- User Conference in Frankfurt
  - December 2004
- First customers see it and are impressed
Second Step: Proof of Concept

- One primitive at a time
  - One day per primitive

- But ...
  - There are alone 450 MFC primitives

- Second step is stopped in January 2005
Third Step: Do it Completely

- Take the entire ObjectStudio C/C++ Code
- Make it a VisualWorks callable DLL
- Started in February 2005
June 2005

- Alpha1 Version made
- First time complete
  - But not tested or bug-free
Forth step: Check for Compatibility

- Run all components of the system
  - If they work differently
    - Add compatibility
    - Decide for incompatibility
      - seldom and easy to understand
      - E.g.: BOSS instead of BinaryObjectStreams

- Run real Customer Applications
  - Details: see later
September 2006

- Ready for Beta1 at the end of the month
Architecture overview

ObjectStudio Code

ObjectStudio Smalltalk compatibility layer

ObjectStudio DLL

Base Classes

Virtual Machine

VisualWorks Code
Implementation Strategy

- **KISS**
  - Keep modifications of ObjectStudio source code to a minimum
    - In Smalltalk and C/C++
  - Do not modify VisualWorks VM unnecessarily
    - Fix bugs
    - Additional features in VW 7.4.1/7.5
      - Use LoadLibraryEx() instead of LoadLibrary()
      - Event handling in the image
General Concepts

- Model ObjectStudio by VisualWorks Smalltalk
  - Preserve syntax and semantics
- ObjectStudioCompiler
- Embed in a NameSpace
- Share code
  - Collections
  - Magnitude
  - Semaphore
  - Boolean
  - Undefined Object
  - ...

www.heeg.de
VisualWorks Namespaces

- **Goal**
  - (In-)visibility of global names
- **Successor to**
  - GlobalDictionary
  - PoolDictionary
  - ClassPools
System, the GlobalDictionary

- In Classic ObjectStudio all global names live in the Globaldictionary called System
- In ObjectStudio 8
  - Classes live in Namespace `ObjectStudio`
  - Globals live in Namespace `ObjectStudio.Globals`
  - Non Identifier entries are Stored in an `IdentityDictionary`
PoolDictionaries

- In Classic ObjectStudio
  - PoolDictionaries are globals which are IdentityDictionaries

  OLEConstants := IdentityDictionary new.
  OLEConstants at: #ADVF_DATAONSTOP put: 64.

- In ObjectStudio 8
  - Extra declaration is needed (manually)

    ObjectStudio.Globals defineNameSpace: #OLEConstants

  - Entries in these Dictionaries become SharedVariables with initializers
    (automatically)

    ObjectStudio.Globals.OLEConstants
      defineSharedVariable: #ADVF_DATAONSTOP
      private: false
      constant: false
      category: 'As yet unclassified'
      initializer: '64'
Shared Classes

Special Renaming

ObjectStudio defineSharedVariable: # Float
private: false
constant: true
category: 'ObjectStudio Compatibility 2'
initializer: 'Core.Double'
Source Differences

- ObjectStudio cls files
  - One class per file
    - With class definition: primary files
    - Class extension: secondary file
  - Different chunk file format
- Method syntax differences
- Semantic differences
Parsing ObjectStudio Source Code

- **ObjectStudioCompiler**
  - Calls ObjectStudioParser
- **ObjectStudioParser**
  - Accepts ObjectStudio Syntax
  - Creates ObjectStudio Syntax compliant Syntax tree
- **Specifying to use ObjectStudio Syntax**
  - classCompilerClass
  - compilerClass
  - Automatic source transformation of methods of VisualWorks classes in cls files
Syntax differences: Arrays and cond

```
f: n
  ^ { { [n <= 1] [1]} [true] [(self f: n - 1) * n]} } cond

f: n
  ^ n < 1
    ifTrue: [1]
    ifFalse: [true
      ifTrue: [(self f: n - 1) * n]
        ifFalse: [ObjectStudio.Message
          newValue: 'No condition satisfied']]
```
Syntax differences: parameters

```plaintext
inc: n
   n isNil ifTrue: [n := 0].
   ^ n + 1

inc: ___n
    | n |
    n := ___n.
    n isNil ifTrue: [n := 0].
    ^ n + 1
```
### Some Semantic Differences

<table>
<thead>
<tr>
<th>Code</th>
<th>VisualWorks</th>
<th>ObjectStudio</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>1 = 0 ifTrue: [1]</code></td>
<td>nil</td>
<td>False</td>
</tr>
<tr>
<td>`[::n</td>
<td>] value: 1`</td>
<td>1</td>
</tr>
<tr>
<td>Array new add: 1</td>
<td>&lt;Error&gt;</td>
<td># (1)</td>
</tr>
<tr>
<td>(Array new: 1) at: 1 put: 7</td>
<td>7</td>
<td># (7)</td>
</tr>
</tbody>
</table>
Modifying ObjectStudio Parse Tree

- ProgramNodeEnumerator subclass OStudioTreeTransformer modifies parse tree
  - <osprim ...>
  - Blocks in literal arrays
  - Assignments to arguments
  - Enabling inner returns
  - Condition results
  - Substitutions

- All this allows for VisualWorks Code generation
Messages with different Semantics

Kernel.OStudioTreeTransformer defineSharedVariable:
  # Substitutions
  private: false
  constant: false
  category: 'constants'
  initializer: '(Dictionary new)
    at: #basicAt:put: put: #os_basicAt:put;
    ...
    yourself'
Substituted Selectors

,, os_comma:
// quo:
< os_LessThan:
<= os_LessEqualThan:
= os_Equal:
> os_GreaterThan:
=> os_GreaterEqualThan:
add: os_add:
add:after: os_add:after:
add:before: os_add:before:
add:beforeIndex: os_add:beforeIndex:
add:withOccurences: os_add:withOccurences:
addAll: os_addAll:
addAllFirst: os_addAllFirst:
addAllLast: os_addAllLast:
addDependent: os_addDependent:
addFirst: os_addFirst:
addLast: os_addLast:
add:Time: os_add:Time:
allInstVarNames os_allInstVarNames
allSubclasses os_allSubclasses
asByteArray os_asByteArray
asFilename os_asFilename
asFloat os_asFloat
asInteger os_asInteger
asNumber os_asNumber
asSet os_asSet
associationAt: os_associationAt:
associationsDo: os_associationsDo:
asString os_asString
asText os_asText
asTime os_asTime
asTimestamp os_asTimestamp
asValue os_asValue
at os_at:
at:put: os_at:put:
basicAt:put: os_basicAt:put:
between:and: os_between:and:
changeSizeTo: os_changeSizeTo:
coerce os_coerce:
collection os_collection:
copyFrom: os_copyFrom:
copyReplaceFrom:to:with: os_copyReplaceFrom:to:with:
copyWithout: os_copyWithout:
copyWithout:dependents os_copyWithout:dependents
detect os_detect:
fileName os_fileName
first os_first
firstDayOfMonth os_firstDayOfMonth
flush os_flush
fromDays: os_fromDays:
ifFalse: os_ifFalse:
ifTrue: os_ifTrue:
includesKey: os_includesKey:
indexOf: os_indexOf:
inheritsFrom: os_inheritsFrom:
inspect os_inspect
isAlphaNumeric os_isAlphaNumeric
last os_last
leapYear: os_leapYear:
methods os_methods
nameOfDay: os_nameOfDay:
new os_new
new: os_new:
nextString os_nextString
origin:corner: os_origin:corner:
origin:extent: os_origin:extent:
perform: os_perform:
perform:with: os_perform:with:
perform:with:with:os_perform:with:with:
perform:with:with:with:os_perform:with:with:with:
perform:withArguments: os_perform:withArguments:
printOn: os_printOn:
printOn:base: os_printOn:base:
printString os_printString
readFrom: os_readFrom:
remove: os_remove:
remove:ifAbsent: os_remove:ifAbsent:
removeAll: os_removeAll:
removeDependent: os_removeDependent:
removeKey: os_removeKey:
signal os_signal
size os_size
storeOn: os_storeOn:
storeString os_storeString
subclasses os_subclasses
subtractTime: os_subtractTime:
to:by:do: os_to:by:do:
to:do: os_to:do:
totalSeconds os_totalSeconds
trimBlanks os_trimBlanks
value os_value
wait os_wait
whileFalse os_whileFalse
whileFalse: os_whileFalse:
whileTrue os_whileTrue
whileTrue: os_whileTrue:
with:do: os_with:do:
withAllSubclasses os_withAllSubclasses
\= os_NotEqual:
\^ os^
Different file structure, first version

- Reader for .txt and .cls files to get ObjectStudio code into VisualWorks
  - ClassReader
    - Reads txt files and calls ObjectStudioChangeScanner
  - ObjectStudioChangeScanner
    - Reads cls files
Second Version: Source Code Integration

- **ObjectStudio.ApplicationDefinitionStream**
  - Original Classic ObjectStudio class
  - Reads .txt files
- **OStudioChunkSourceFileFormat**
  - Lives in VisualWorks SourceFileFormat class hierarchy
  - Reads and writes .cls files
  - Controls that .cls files
ObjectStudio Smalltalk and C

- Strong interconnection between C and Smalltalk
  - They call each other all the time
- There are five ways to call C
  - Direct primitives
    - PrimSpecRec
  - Numbered Primitives
    - `<primitive: 123>`
  - Named Primitives
    - `<osprim: MFC openWindow>`
    - `<osprim: #myDll myfunc>`
  - Module/ENFINModule
    - Programmatic creation of interface methods using osprim
  - ExternalProcedures
    - Programmatic creation of interface methods with C datatypes
ObjectStudioWrapperDLL

- Regular VisualWorks DLL
- Uses _oop as parameter and return type
- Calls ObjectStudio primitives using a simulated ObjectStudio stack
OPTR and _oop

- In ObjectStudio object pointers OPTR never move
- In VisualWorks object pointer _oop are moved by the garbage collector
- ObjectStudio C code relies on not moving OPTR

Solution:
- Define class OPTR with wraps VisualWorks Oops for ObjectStudio including C++ reference counting
- The real object pointers _oop are stored in a Smalltalk array
  - which is registered in the Visualworks registry
  - the index to this array is stored in class OPTR as variable
- Reference counts are stored in a normal C array.
class __declspec(dllexport) OPTR;
typedef OPTR* POPTR;

class OPTR
{
    public:
        static OPTR& cpp_S2O(LONG value);
    [...]
    private:
    [...]
    enum Types { TYPE_NIL = 0, TYPE_NORMALOBJECT,
                TYPE_NOTNORMALOBJECT, TYPE_UNKNOWN} ;
    INT m_IndexOrOop; // index in objecttable or not normal object
    INT m_Type;       // type of object
    static INT s_NumObjects; // number of registered objects
    static INT s_NextFreeObject; // index of next free object in objecttable
    static INT s_ObjectTable; // index of objecttable array in visualworks registry
    static INT* s_LinkTable;  // array with linked registry slots
Direct Primitives

- Look at definition
- Re-implement
  - mostly in Smalltalk using existing VisualWorks functionality
  - In rare cases replace by osprim
Numbered Primitives

- `<Primitive: 45>`
- Some are replaced by
  - VisualWorks Primitive
    - E.g. `Object>>==`  
    - `<vwprimitive: 110>`
  - VisualWorks implementation
- For most
  - Auto-generate calling code
disable
<primitive: 7>
^ self primitiveFailed.

disable
< osprimitive: 7>
| argumentArray |
argumentArray := Core.Array os_new: 0.
^ ObjectStudioCallInterface os_new
  callPrim: 7
  self: self
  arguments: argumentArray
  ifFail: [^self primitiveFailed]
Class ObjectStudioCallI nterface

callPrim: primitiveNumber self: oSelf arguments: args ifFail: failBlock

self init.
^[self callPrim: primitiveNumber self: oSelf arguments: args]
on: PrimitiveFailException
do: [:ex| ^failBlock value]

callPrim: primitiveNumber self: oSelf arguments: args
<C: _oop callPrim(int primitiveNumber, _oop oSelf, _oop args)>
_errorCode = 0 ifTrue: [PrimitiveFailException raise].
self externalAccessFailedWith: _errorCode
_oop callPrim(int primitiveNumber, _oop oSelf, _oop args)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    [...]
    primReturnCode = CallUserPrimitiveFunc(primMethodOPTR);
    oSelf = PPop().getObject();
    [...]
    if (primReturnCode == FALSE)
        UPfail(0);
    return oSelf;
}
activateExcludeChild: fExclude
  < osprim: MFC ActivateForm>
  ^ self primitiveFailed.

activateExcludeChild: fExclude
  < osprim: MFC ActivateForm>
  < methodClass: ExternalAnnotatedMethod>
  | argumentArray |
  (argumentArray := Array new: 1) os_at: 1 put: fExclude.
  ^ ObjectStudio.External.ObjectStudioCallInterface new
callOSPrim: thisContext method functionSpecification
  self: self
  arguments: argumentArray
  ifFail: [^self primitiveFailed]
translateFromForm: aFormOrFormItem to: anOther
"<osprim: TRACK POINTTRANSLATEFROMFORM>"

| args functionSpecification |
args := Array new: 2.
args at: 1 put: aFormOrFormItem.
args at: 2 put: anOther.
functionSpecification := OSPrimSpecifier new.
functionSpecification
    module: Root.ObjectStudio.ENFINModule versionTag , 'TRACK'.
functionSpecification function: 'POINTTRANSLATEFROMFORM'.
functionSpecification numArgs: 2.
^ Root.ObjectStudio.ObjectStudioCallInterface new
callOSPrim: functionSpecification
self: self
arguments: args
ifFail: [^self primitiveFailed]
OSPrimSpecifier

Smalltalk.Kernel defineClass: # OSPrimSpecifier
  superclass: # { Object }
  indexedType: # none
  private: false
  instanceVariableNames: 'numArgs numTemps handle function module '
  classInstanceVariableNames: ''
  imports: ''
  category: 'ObjectStudio Tools,'
Smalltalk.Kernel defineClass: # ExternalAnnotatedMethod
superclass: #{ Kernel.AnnotatedMethod}
indexedType: # objects
private: false
instanceVariableNames: 'functionSpecification '
classInstanceVariableNames: "
imports: "
category: 'ExternalAnnotatedMethod'

• Variable functionSpecification stores instances of OSPrimSpecifier
Module/ EnfinModule

- Autogenerate osprim

**privateQueryVolumeName**: aSmallInteger
< osprim: FILES PRIVATEQUERYDRIVENAME>

":Section Reference
privateQueryVolumeName: aSmallInteger

Description: Returns the name of a drive.
Assumptions: The drive number passed in is a small integer
starting with 1 for drive A:. This function returns
allways an empty strings on operating systems, which
do not support a drive concept.

[...]
External Procedures

- Just work unchanged
ObjectStudio Unicode

- In ObjectStudio 7
  - Two installations
  - Two Virtual Machines
  - Incompatible Images
  - Compatible Source Code

- In ObjectStudio 8
  - One installation
  - One Virtual Machine
  - Two sets of DLLs
  - One image
  - Decision at start-up
Native Widgets

- Use MFC
- Callbacks to VisualWorks
Calling Smalltalk from C

- **PSend** is implemented using
  - `OESendMessageMany()`
- **ASend** is implemented using
  - `OESendMessageMany()`

``` Smalltalk
whenIdleSend: aSelector to: aReceiver withArguments: anArray
"inserts the asend into the queue"
ASendSemaphore critical:
  [ASendQueue addLast: (Core.MessageSend
    receiver: aReceiver
    selector: aSelector
    arguments: (anArrayOrObject isArray
      ifTrue: [anArrayOrObject asArray]
      ifFalse: [anArrayOrObject]))]
```
The Process Model

- **VisualWorks process model**
  - Many Smalltalk processes are running in different priority.
  - Windows events are handled (almost) any time
  - Shared queues move the events to target processes
  - This is implemented in C

- **ObjectStudio process model**
  - The caller of all execution is the Windows Event Loop
  - All Message Sends in the ASendQueue are executed when the Windows Event is empty
  - This is implemented in C++
• Move EventLoop to Smalltalk
  - Available in vw-dev since August 29, 2006

• Initializing in class EventProcessingManager:
  \texttt{primCallbackEventInstall: \texttt{aMethodSelector}}
  "Inform the VM of the callback class (the receiver) and the callback
  invocation method selector for event processing."

\begin{verbatim}
| regNames |
regNames := ObjectMemory registrationNames.
((regNames includes: 'callbackEventClass')
 and: [regNames includes: 'callbackEventSelector'])
ifTrue:
  [ObjectMemory registerObject: self
   withEngineFor: 'callbackEventClass'.
   ObjectMemory registerObject: aMethodSelector
   withEngineFor: 'callbackEventSelector']
\end{verbatim}
Eventloop in Smalltalk

- Class EventProcessingManager
  - runs the loop
  - delegates all actions to the handler
    - Implemented in EventHandler class hierarchy

```smalltalk
processEvents
| eventRecord |
handler canHandleMultipleEvents
  ifFalse: [self blockEvents ifTrue: [^self]].

[eventRecord := handler eventRecord.
[handler nextEvent: eventRecord]
  whileTrue: [handler handleEvent: eventRecord].
handler postHandleEvents]
  ensure: [self allowEvents]
```
Subclasses of EventHandler

- Example methods in Win32EventHandler

```plaintext
eventRecord
  ^ self MSG gcMalloc

nextEvent: anEventRecord
  ^(self
    peekMessageA: anEventRecord
    window: 0
    filterMin: 0
    filterMax: 0
    flags: (self PM_REMOVE bitOr: self PM_NOYIELD)) ~= self FALSE

postHandleEvents
  ^ self
```
OStudioEventHandler

- Example methods in OStudioEventHandler

preTranslateMessage: msg
  <C: BOOL PreTranslateMessage(LPMSG msg)>
  ^ self externalAccessFailedWith: _errorCode

postHandleEvents
  System wantsBusyCursor
  ifTrue: [UI.Cursor wait
    showWhile: [System drainASendQueue]]
  ifFalse: [System drainASendQueue].
  self onIdleApp

onIdleApp
  <C: void OnIdleApp(void)>
  ^ self externalAccessFailedWith: _errorCode
Handling an event

- In Win32EventHandler

  `handleEvent: anEventRecord`
  `self translateMessage: anEventRecord`
  `self dispatchMessageA: anEventRecord`

- In OStudioEventHandler

  `handleEvent: anEventRecord`
  `(self preTranslateMessage: anEventRecord) = self FALSE`
  `ifTrue:`
  `  [self translateMessage: anEventRecord`
  `    self dispatchMessageA: anEventRecord]`
User Interrupts

- `<ctrl> Y` and `<ctrl> <shift> Y` work as in VisualWorks
- Implemented as Accelerator Keys
- `<ctrl> Y` interrupts the running process by sending `self halt:` in a callback from C
Class Proxies

- Class Proxies work
- They are shared variables in Root.ObjectStudio.Global
- Loading puts class in Root.ObjectStudio
Application Loading

- Application loading works
Performance Issues

- In Classic ObjectStudio
  - Smalltalk execution is slow
  - Calling C primitives cost nothing
- In VisualWorks
  - Smalltalk execution is fast
  - Calling C is expensive
- In Classic ObjectStudio
  - Performance critical methods are moved to C
- In ObjectStudio 8
  - Performance critical primitives are moved back to Smalltalk
VisualWorks .st file-in is slow

- Of 1000 seconds
  - 900 are RB refresh
  - Of 100 seconds
    - 90 are repeatedly relinking the system
    - Of 10 Seconds
      - 9 are disk flush (Changes file)
      - 1 second compiling

- At the end of the optimizations:
  - Loading as alternative to file-in
  - Loading a .st is similar speed as parcel loading
    - Small .st/.cls files load faster than parcels
    - Large parcel files load faster than .st/.cls files
New Concepts for ObjectStudio

- **Source Code Management**
  - **Store**
    - Versioning system
    - Database based
  - Smalltalk Archives
    - File based
    - One file per application

- **Code purity**
  - Undeclared
  - Code Critic
  - Overrides
Time Line

- June 2005
  - Internal Alpha
- October 2005
  - Internal Alpha2
- March 2006
  - Cincom internal testing
- April 2006
  - First Customer
- August 2006
  - First independent Service Partner
- May-September 2006
  - Solve incompatibilities, Refactoring, Performance issues
- October 2006
  - Next few customers
- December 2006
  - Preview
Porting Process Theory

- Theory
  - Load application
  - Start it
Porting Process Practice

- Look at base system modifications
  - Like any other porting this is a source of many problems
- Known incompatibilities
  - Immutability
  - Declare Pool Dictionaries
  - Export Modeling Tool definitions in ASCII
    - Binary format is incompatible
Undeclareds

- **In Classic ObjectStudio**
  - Each undeclared variable is always considered a global
- **In VisualWorks and ObjectStudio 8**
  - Each undeclared variable is considered undeclared
- **GHUndeclaredBrowser**
  - Shows all Undeclareds and their references
  - It uncovered some very old ObjectStudio problems due to typos.
Target Look
After loading Modeling Tool
What can ESUG Audience learn?

- Integration of Smalltalk systems is possible
- Which systems deserve to be integrated?
  - VW and Squeak
    - Very common systems
  - VSE
  - VA
  - Smalltalk-X
  - Dolphin
  - Ruby
  - Java
    - Frost
  - GemStone