

Andreas Hiltner & Georg Heeg

**ObjectStudio8 :=
VisualWorks -> ObjectStudio**

What does it mean to You?

Frankfurt, December 5th, 2006

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.

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)

User Conference in Frankfurt

- The First Window Opens
 - December 2004
- First customers see it and are impressed

Do it Completely

- Take the entire ObjectStudio C/C++ - Code
- Make it a VisualWorks callable DLL

June 2005

- Alpha1 Version made
- First time complete
 - But not tested or bug-free

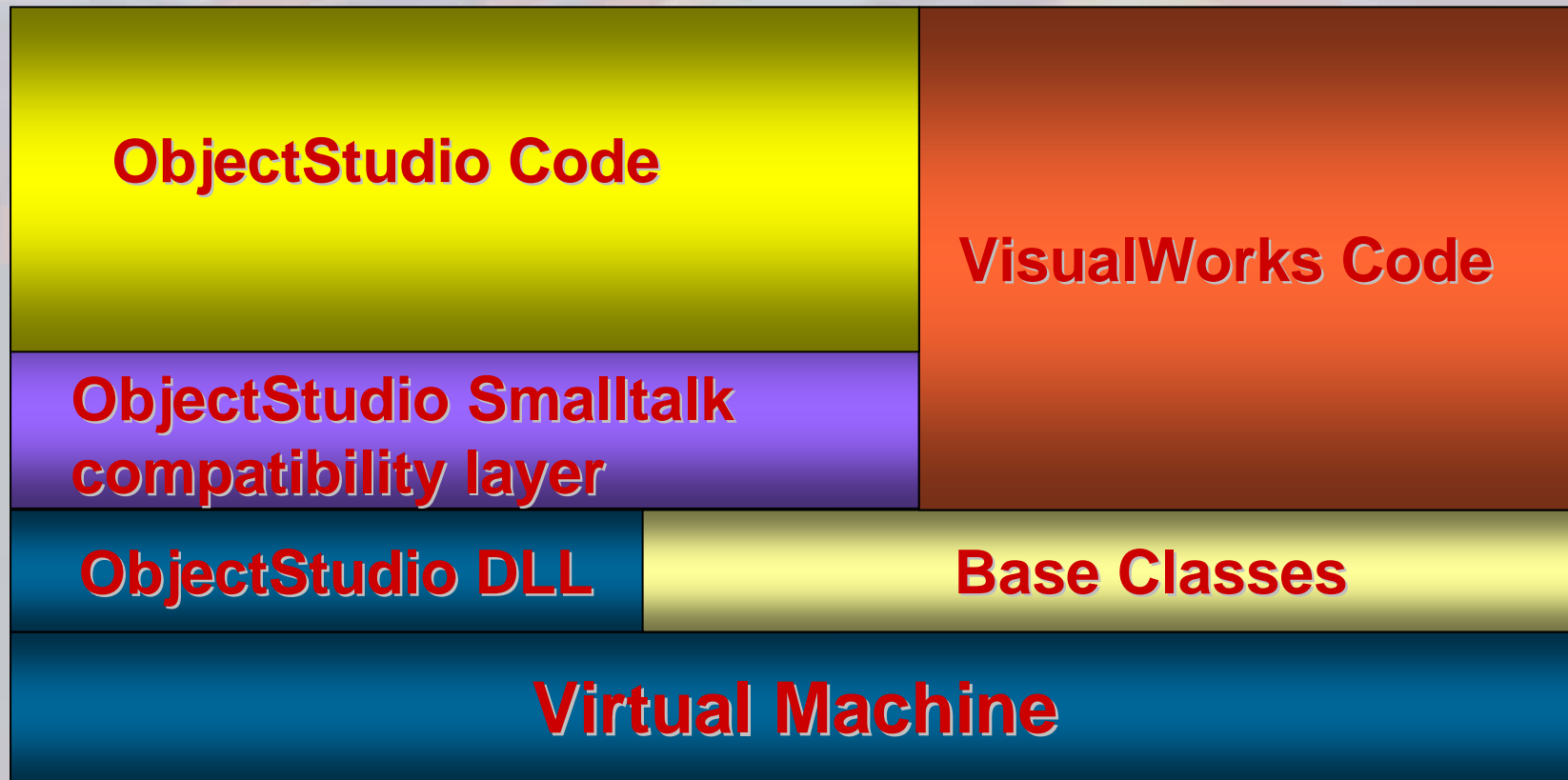
Forth step: Check for Compatibility

- 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

Availability

- December 2006
 - Ready for Beta1
- Summer 2007
 - First Product Release

Architecture overview



General Concepts

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

VisualWorks Namespaces

- Goal
 - (In-)visibility of global names
- Successor to
 - GlobalDictionary
 - PoolDictionary
 - ClassPools

Shared Classes

ActionSequence, Array, Association, Bag, Behavior, BlockClosure, Boolean, ByteArray, Character, Class, Collection, CompiledBlock, CompiledMethod, Compiler, Date, Dictionary, EncodedStream, False, Fraction, IdentityDictionary, IdentitySet, Integer, InternalStream, Magnitude, Metaclass, MethodContext, Number, OrderedCollection, PeekableStream, Point, PositionableStream, Random, ReadStream, ReadWriteStream, Rectangle, Semaphore, Set, SmallInteger, SortedCollection, Stream, String, Symbol, Time, Timestamp, True, UndefinedObject, UserPrimitiveMethod, WeakDictionary, WriteStream, Exception, Error, MessageNotUnderstood, Notification, StreamError, PositionOutOfBoundsError, IncompleteNextCountError, EndOfStreamNotification, SocketAccessor, SocketAddress

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']]
```

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

Source Code Integration

- ObjectStudio.ApplicationDefintionStream
 - 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

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 non 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.

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

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++

Decision of July 2006

- Move EventLoop to Smalltalk
 - Available in VisualWorks 7.5
 - since August 29, 2006

Eventloop in Smalltalk

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

processEvents

```
| eventRecord |  
handler canHandleMultipleEvents  
  ifFalse: [self blockEvents ifTrue: [^self]].  
  
[eventRecord := handler eventRecord.  
 [handler nextEvent: eventRecord]  
   whileTrue: [handler handleEvent: eventRecord].  
 handler postHandleEvents]  
  ensure: [self allowEvents]
```

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
```

Class Proxies

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

Application Loading

- Application loading works
 - Often project adaptations of ApplicationDefinitionStream methods must be adapted

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

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

What does it mean to You?

- Integration of Smalltalk systems is possible
 - Which other systems deserve to be integrated?
- Usage of all VisualWorks components by all of Cincom Smalltalk Customers
 - WebToolkit
 - WebServices
 - Contributions
 - GemStone, Objectivity, Seaside, June, VisualWaf, ...
- Usage of all ObjectStudio components by all of Cincom Smalltalk Customers
 - Native Windows GUI
 - EHLLAPI/APPC