

# BV1 - Einführung in Orasis

Jan Carstens ([carstens@tzi.de](mailto:carstens@tzi.de))  
Sven Oesau ([soesau@tzi.de](mailto:soesau@tzi.de))

# Übersicht

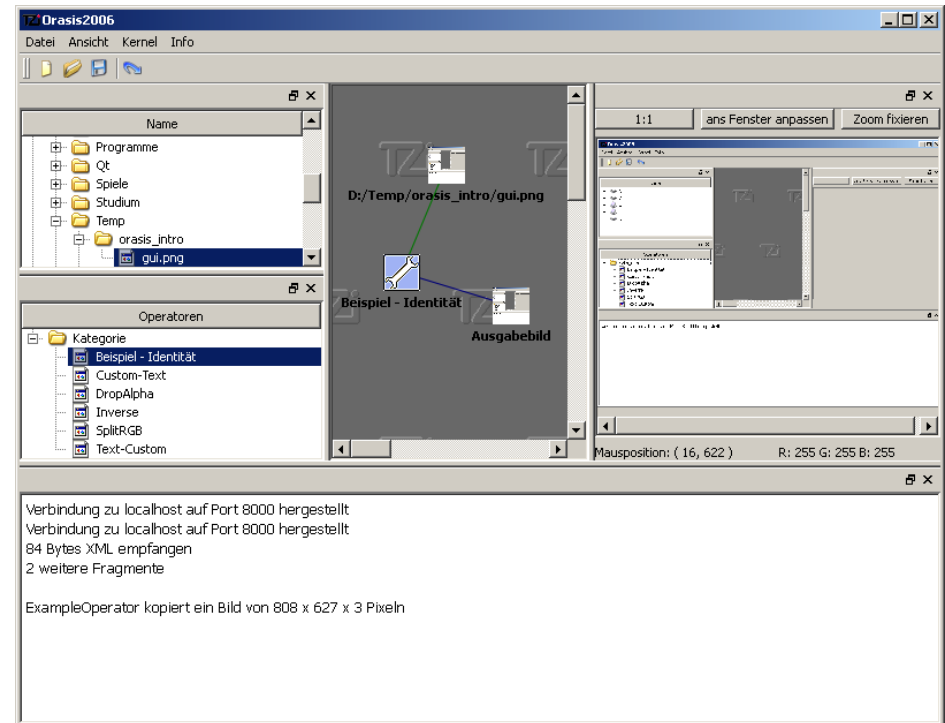
- Unterstützte Betriebssysteme
- Client/Server - Architektur
- Entwicklung eigener Operatoren
- Methoden für Objekte des Typs `Object::image`
- Fragen

# Unterstützte Betriebssysteme

- Windows / Microsoft Visual Studio
- Windows / MinGW
- Linux
- Mac OS X 10.3 (Panther)

# Client/Server – Architektur (1/2)

- Client stellt Qt-GUI zur Verfügung
- Graphen aus Bildern und Operatoren werden als XML kodiert an den Server übertragen



# Client/Server – Architektur (2/2)

- Server stellt Operatoren zur Verfügung und wendet diese auf Eingabebilder an
- kommuniziert mit dem Client über Port 8000



```
c:\D:\Studium\08 SoSe06\Orasis\kernel.exe
starte Kernel...
msvcDropAlpha.dll ist nicht kompatibel mit diesem Kernel
msvcExample.dll ist nicht kompatibel mit diesem Kernel
msvcInverse.dll ist nicht kompatibel mit diesem Kernel
msvcObjectText.dll ist nicht kompatibel mit diesem Kernel
msvcSplitRGB.dll ist nicht kompatibel mit diesem Kernel
msvcTextCustom.dll ist nicht kompatibel mit diesem Kernel
6 Operatoren geladen
warte auf Verbindung auf Port 8000
Kernel gestartet
GETOPERATORS
PROJECT
PNG: D:/Temp/orasis_intro/gui.png
1 Operator(en) und 2 Objekte
Beispiel - Identität ...
!!! setValue(-1,1000,0) in 800x627x3 Bild
!!! getValue(-1,1000,45) in 800x627x3 Bild
ok
-
```

# Entwicklung eigener Operatoren (1/5)

- geschrieben in C++, übersetzt als DLL
- abgeleitet von der Basisklasse Operator
  - einheitliches Interface für den Server
- für jeden Operator sind 5 virtuelle Methoden zu implementieren
  - getName()/getAuthor()/getDescription()
  - setParameter()
  - execute()

# Entwicklung eigener Operatoren (2/5)

- getName()/getAuthor()/getDescription()
  - dienen der Identifikation des Operators
  - stellen einen Teil des eindeutigen Namens des Operators
  - getName(): Name des Operators
  - getAuthor(): Name(n) der Gruppe(nmitglieder)
  - getDescription(): kurze Beschreibung des Operators

# Entwicklung eigener Operatoren (3/5)

- setParameter
  - Parameter sind eindeutig durch ihren Namen gekennzeichnet und haben einen definierten Typ
- Typen:
  - Object::image, Object::doubleimage
  - Object::integer, Object::real, Object::string
  - Object::custom
- Ein- und Ausgabeobjekte sind in setParameter festzulegen

# Entwicklung eigener Operatoren (4/5)

- `execute()`
  - leistet die eigentliche Arbeit des Operators, arbeitet auf Eingabeobjekten und erzeugt Ausgabeobjekte
  - liefert `true` im Erfolgsfall, `false` anderenfalls
  - mittels des `std::ostringstream` log können Ausgaben analog zu `std::cout` an den Client weitergeleitet werden

# Entwicklung eigener Operatoren (5/5)

- Testen der Operatoren
  - für Release/Debug passenden Server einsetzen
  - ist `FAST_GETSET` definiert, wird auf Bereichsüberprüfung beim Zugriff auf Pixel verzichtet!

# Methoden von Objekten des Typs Object::image (1/2)

- getHeight(), getWidth(), getNumChannels()
  - liefern Höhe, Breite, Anzahl der Kanäle
- operator bool()
  - true, falls Bild nicht dimensionslos ist
  - nichtinitialisierte Output-Objekte sind false!

# Methoden von Objekten des Typs Object::image (2/2)

- `init(x,y,z)`
  - initialisiert das Bild mit Breite `x`, Höhe `y` und `z` Kanälen und setzt alle Pixel auf 0
- `setValue(x,y,[z], value)`
  - Pixel `x`, `y` in Kanal `z` wird auf Wert `value` gesetzt
- `getValue(x,y,[z])`
  - der (`z`-)Wert des Pixels `x`, `y` wird zurückgegeben
- Kanäle: `ImageObject::R/G/B/Alpha`

Fragen?

# Fragen?

Vielen Dank für Eure Aufmerksamkeit!

E-Mail-Kontakt:

Jan Carstens ([carstens@tzi.de](mailto:carstens@tzi.de))

Sven Oesau([soeau@tzi.de](mailto:soeau@tzi.de))