Voyage-Tugrik: A new persistence option for Pharo using GemStone/S 64

Dale Henrichs
GemTalk Systems
ESUG 2016

Tugrik???

WAT???

1 Tugrik = 100 Mongo

Voyage

- Object-Document Mapper for NOSQL databases
- MongoDB and UnQLite databases
- Simplest API possible
    - 6 methods for save remove, and query
- No Voyage Query language

GemStone/S 64

- Multi-terabyte Smalltalk image
- 1000’s of concurrent Smalltalk vms (gems) making updates to shared image
- 1000’s of ACID transactions (concurrent image saves) per second
- Generous “Free for commercial use”
    - Cost for the following is an email
        - 50Gb data base
        - 20 gems - 20 concurrent transactions
    - https://gemtalksystems.com/licensing/#CWELicensing

GsDevKit/GLASS

- A compatibility layer for GemStone/S that provides a class library that more closely resembles a classical Smalltalk environment
  - Smalltalk global is present
  - Implements a large number of classes and methods that are present in Squeak and Pharo images

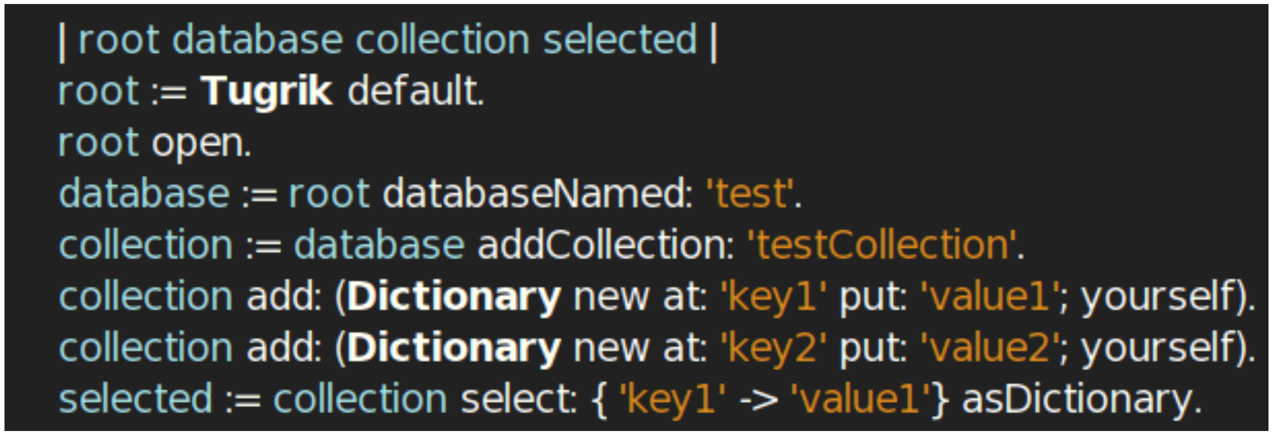
GsDevKit_home

- Bash scripts for creating/starting/stopping GemStone database instances (stones)
- Bash scripts for creating/starting/stopping Pharo clients
- Ubuntu 12.04/14.04 and MacOs
  - Ubuntu 16.04 soon

Tugrik
Proof of Concept

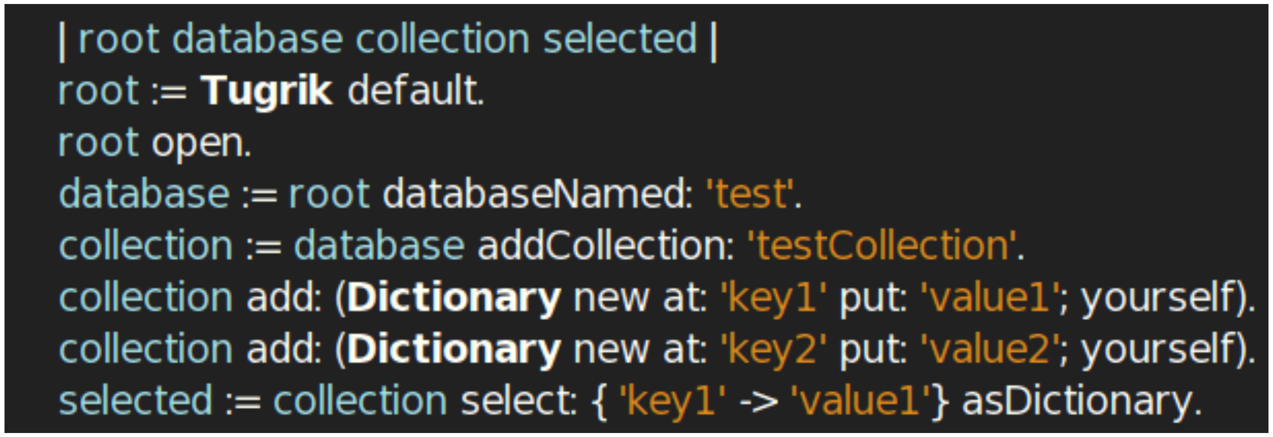
- Tugrik was created to make it possible to move a Voyage application written against MongoDB to GemStone/S
  - Tugrik implements the public MongoTalk Smalltalk API
  - Tugrik models the MongoDB collection organization
    - named database instance consisting of a number of named document store collections
  - Tugrik supports the MongoDB *query filters*
- Tugrik passes all of the MongoTalk unit tests

Tugrik

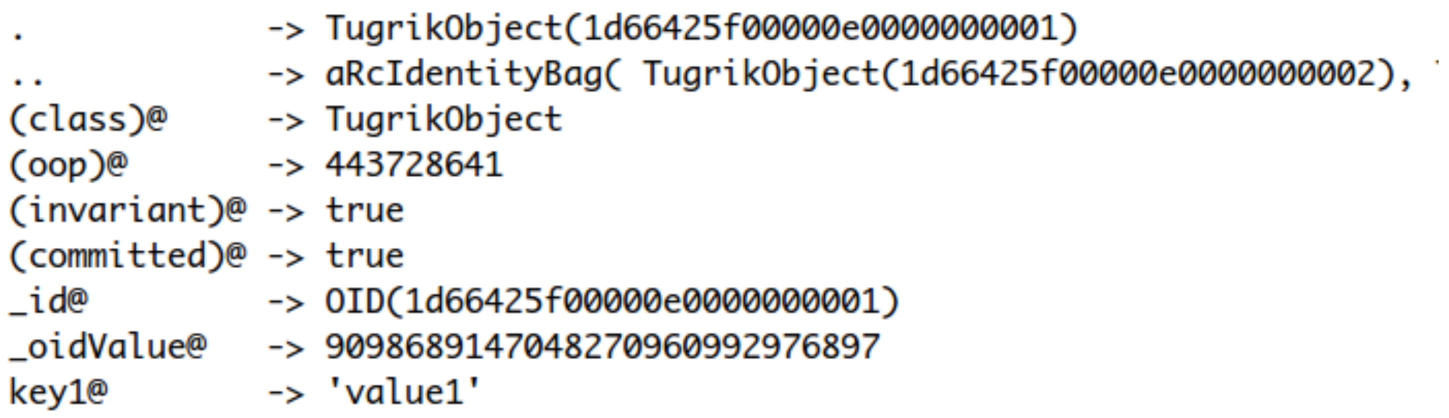


```
| root database collection selected |
root := Tugrik default.
root open.
database := root databaseNamed: 'test'.
collection := database addCollection: 'testCollection'.
collection add: (Dictionary new at: 'key1' put: 'value1'; yourself).
collection add: (Dictionary new at: 'key2' put: 'value2'; yourself).
selected := collection select: { 'key1' -> 'value1'} asDictionary.
```

Tugrik



```
| root database collection selected |
root := Tugrik default.
root open.
database := root databaseNamed: 'test'.
collection := database addCollection: 'testCollection'.
collection add: (Dictionary new at: 'key1' put: 'value1'; yourself).
collection add: (Dictionary new at: 'key2' put: 'value2'; yourself).
selected := collection select: { 'key1' -> 'value1'} asDictionary.
```



```
.              -> TugrikObject(1d66425f00000e0000000001)
..             -> aRcIdentityBag( TugrikObject(1d66425f00000e0000000002), '
(class)@       -> TugrikObject
(oop)@         -> 443728641
(invariant)@ -> true
(committed)@ -> true
_id@           -> OID(1d66425f00000e0000000001)
_oidValue@     -> 9098689147048270960992976897
key1@          -> 'value1'
```

Tugrik MongoDB query filters

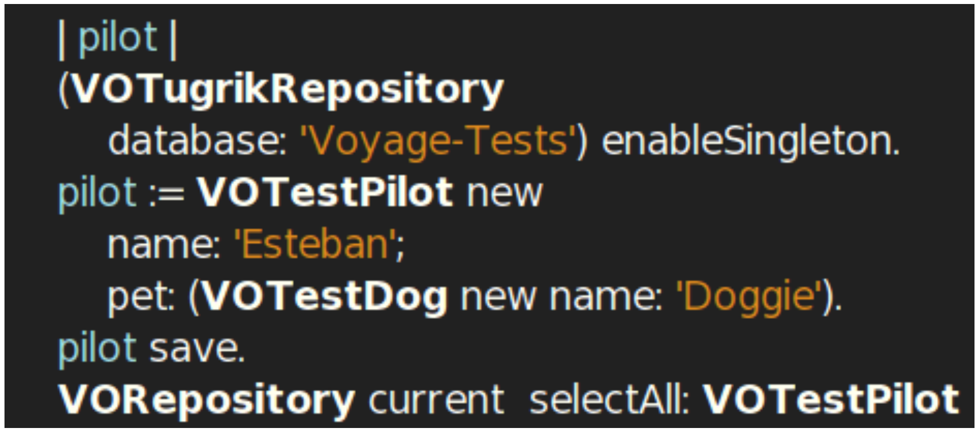
```
{ <field>: { $all: [ <value1> , <value2> ... ] } }
```

```
{ field: { $size: 2 } }
```

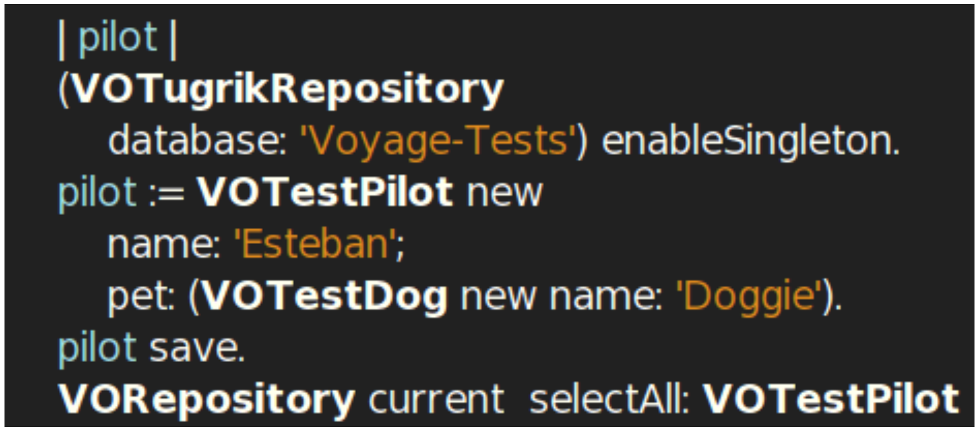
```
{ $nor: [ { price: 1.99 }, { sale: true } ]  }
```

Voyage-Tugrik

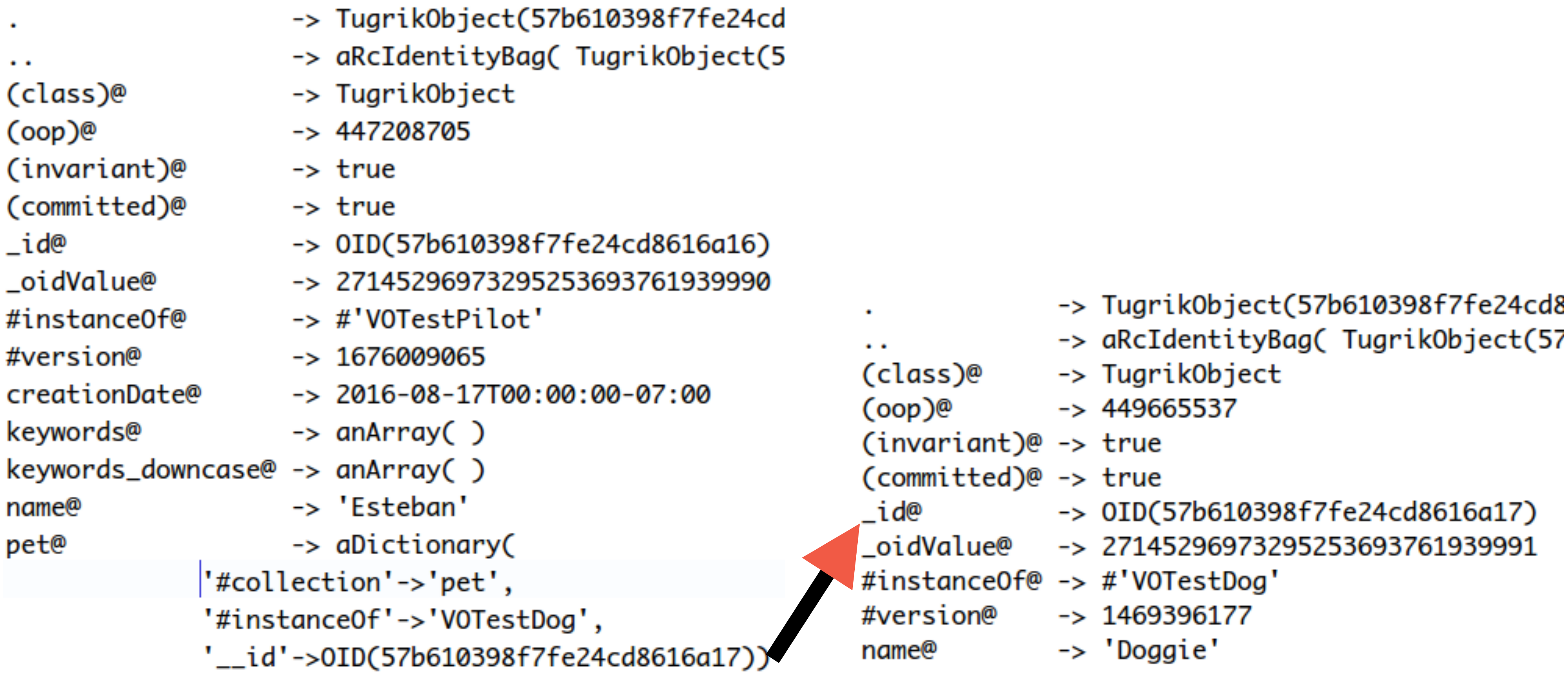
- Voyage-Tugrik specializes Voyage-Mongo.
- VOTugrikRepository is a subclass of VOMongoRepository



```
| pilot |
(VOTugrikRepository
    database: 'Voyage-Tests') enableSingleton.
pilot := VOTestPilot new
    name: 'Esteban';
    pet: (VOTestDog new name: 'Doggie').
pilot save.
VORepository current  selectAll: VOTestPilot
```



```
| pilot |
(VOTugrikRepository
    database: 'Voyage-Tests') enableSingleton.
pilot := VOTestPilot new
    name: 'Esteban';
    pet: (VOTestDog new name: 'Doggie').
pilot save.
VORepository current  selectAll: VOTestPilot
```



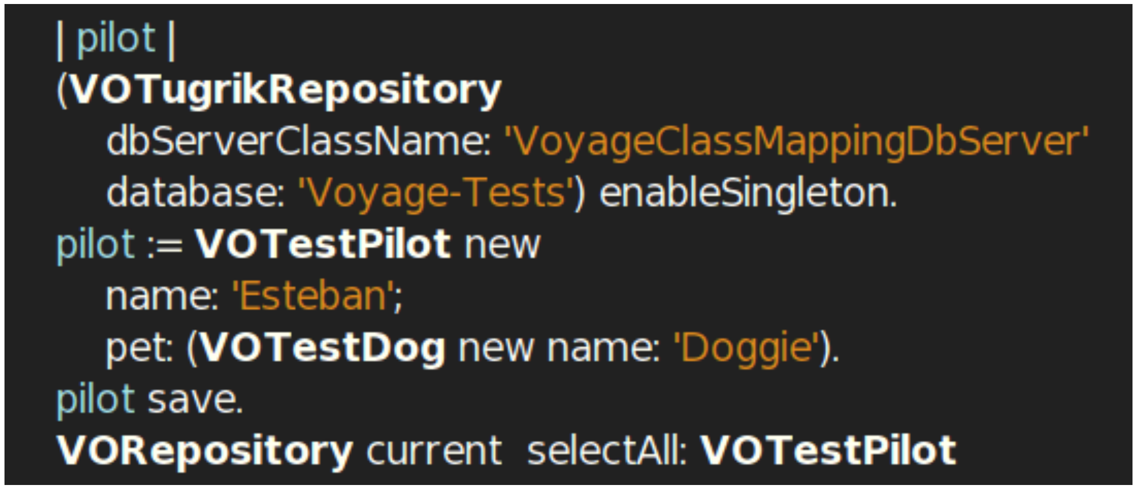
```
.                   -> TugrikObject(57b610398f7fe24cd
..                  -> aRcIdentityBag( TugrikObject(5
(class)@            -> TugrikObject
(oop)@              -> 447208705
(invariant)@        -> true
(committed)@        -> true
_id@                -> OID(57b610398f7fe24cd8616a16)
_oidValue@          -> 27145296973295253693761939990
#instanceOf@        -> #'VOTestPilot'
#version@           -> 1676009065
creationDate@       -> 2016-08-17T00:00:00-07:00
keywords@           -> anArray( )
keywords_downcase@  -> anArray( )
name@               -> 'Esteban'
pet@                -> aDictionary(
             '#collection'->'pet',
             '#instanceOf'->'VOTestDog',
             '__id'->OID(57b610398f7fe24cd8616a17))
```

```
.             -> TugrikObject(57b610398f7fe24cd8
..            -> aRcIdentityBag( TugrikObject(57
(class)@      -> TugrikObject
(oop)@        -> 449665537
(invariant)@  -> true
(committed)@  -> true
_id@          -> OID(57b610398f7fe24cd8616a17)
_oidValue@    -> 27145296973295253693761939991
#instanceOf@  -> #'VOTestDog'
#version@     -> 1469396177
name@         -> 'Doggie'
```

Voyage-Tugrik
Class Mapping

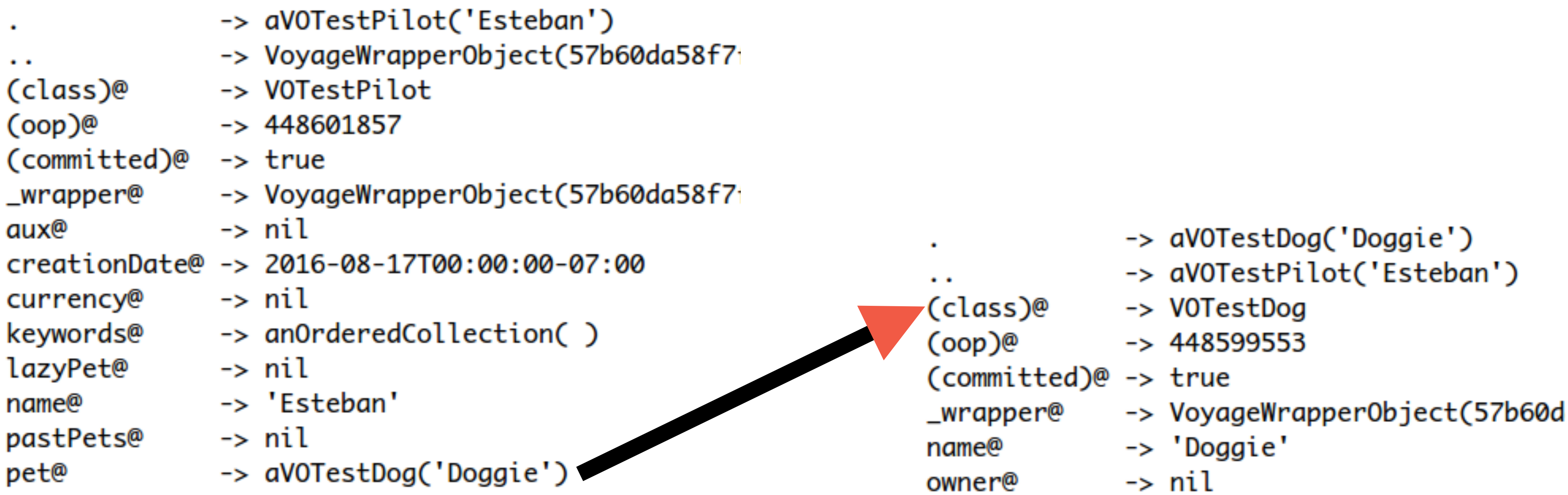
- With Voyage-Tugrik Class Mapping, instances of root object classes are created on server instead of anonymous TugrikObject instances

- Expect to share common packages between client and server



```
| pilot |
(VOTugrikRepository
    dbServerClassName: 'VoyageClassMappingDbServer'
    database: 'Voyage-Tests') enableSingleton.
pilot := VOTestPilot new
    name: 'Esteban';
    pet: (VOTestDog new name: 'Doggie').
pilot save.
VORepository current  selectAll: VOTestPilot
```

```
| pilot |
(VOTugrikRepository
    dbServerClassName: 'VoyageClassMappingDbServer'
    database: 'Voyage-Tests') enableSingleton.
pilot := VOTestPilot new
    name: 'Esteban';
    pet: (VOTestDog new name: 'Doggie').
pilot save.
VORepository current  selectAll: VOTestPilot
```

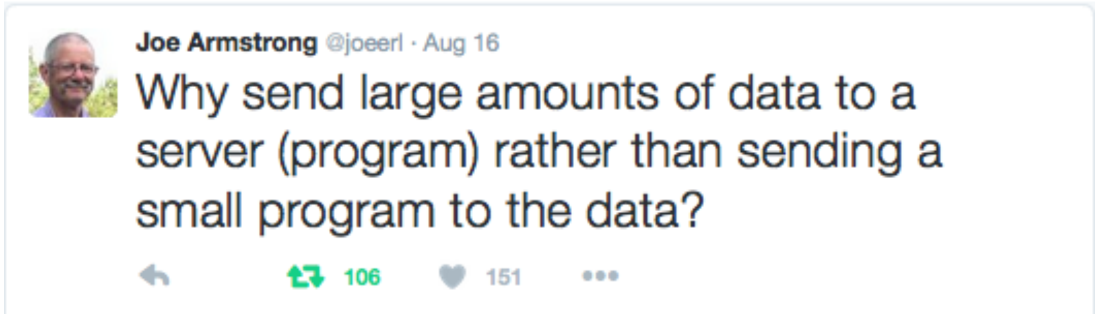


```
.               -> aVOTestPilot('Esteban')
..              -> VoyageWrapperObject(57b60da58f7
(class)@        -> VOTestPilot
(oop)@          -> 448601857
(committed)@    -> true
_wrapper@       -> VoyageWrapperObject(57b60da58f7
aux@            -> nil
creationDate@   -> 2016-08-17T00:00:00-07:00
currency@       -> nil
keywords@       -> anOrderedCollection( )
lazyPet@        -> nil
name@           -> 'Esteban'
pastPets@       -> nil
pet@            -> aVOTestDog('Doggie')
```

```
.               -> aVOTestDog('Doggie')
..              -> aVOTestPilot('Esteban')
(class)@        -> VOTestDog
(oop)@          -> 448599553
(committed)@    -> true
_wrapper@       -> VoyageWrapperObject(57b60d
name@           -> 'Doggie'
owner@          -> nil
```

Voyage Server Blocks: Unlocking Server-Side Execution

- Class instances WITHOUT behavior on the server is interesting and even useful in some cases
- More interesting would be to enable server-side execution


Joe Armstrong @joeerl · Aug 16

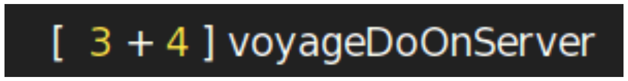
Why send large amounts of data to a server (program) rather than sending a small program to the data?

106 151

https://twitter.com/joeerl/status/765793779949395968

Voyage Server Blocks:

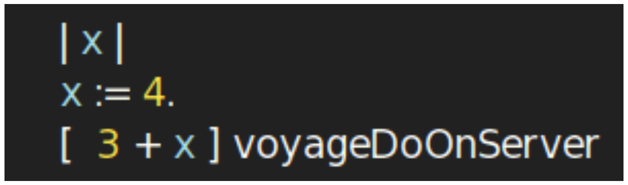
Send Small Programs to Data



```
[ 3 + 4 ] voyageDoOnServer
```

- The expression is embedded in your client code
- The block source is extracted, shipped to the server and evaluated.
- The return value is serialized and returned to client.

Voyage Server Blocks: non-local variables

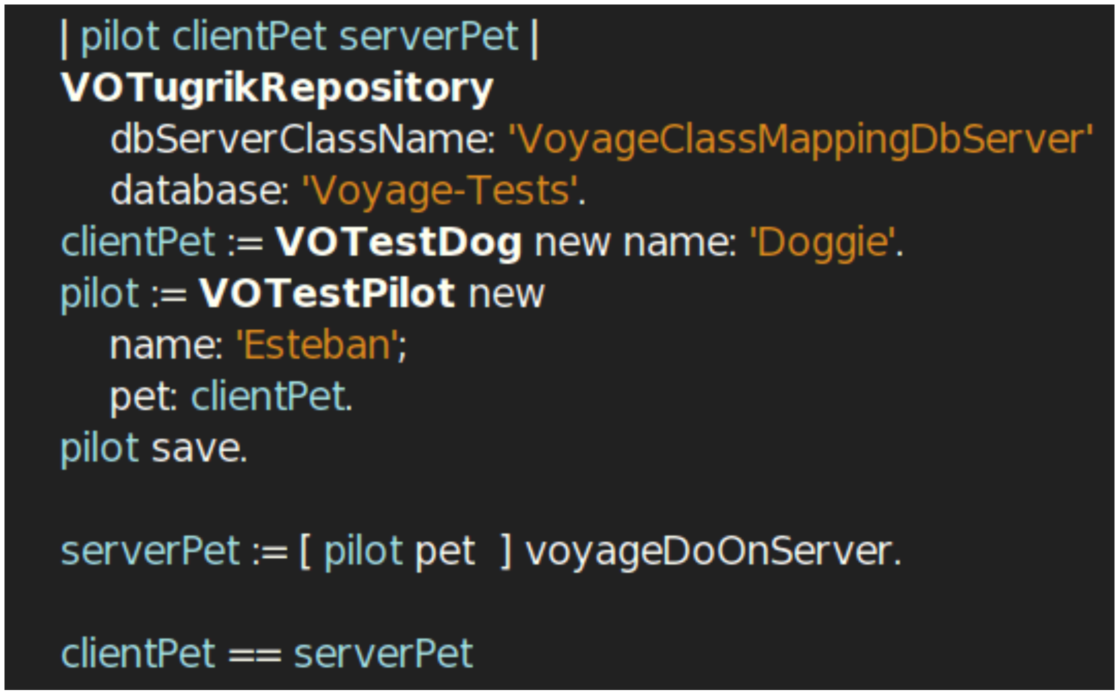


```
| x |
x := 4.
[ 3 + x ] voyageDoOnServer
```

- The values of non-local variable references in block are serialized and shipped to server where the values are bound to the variables before execution

Voyage Server Blocks: Voyage Root Objects

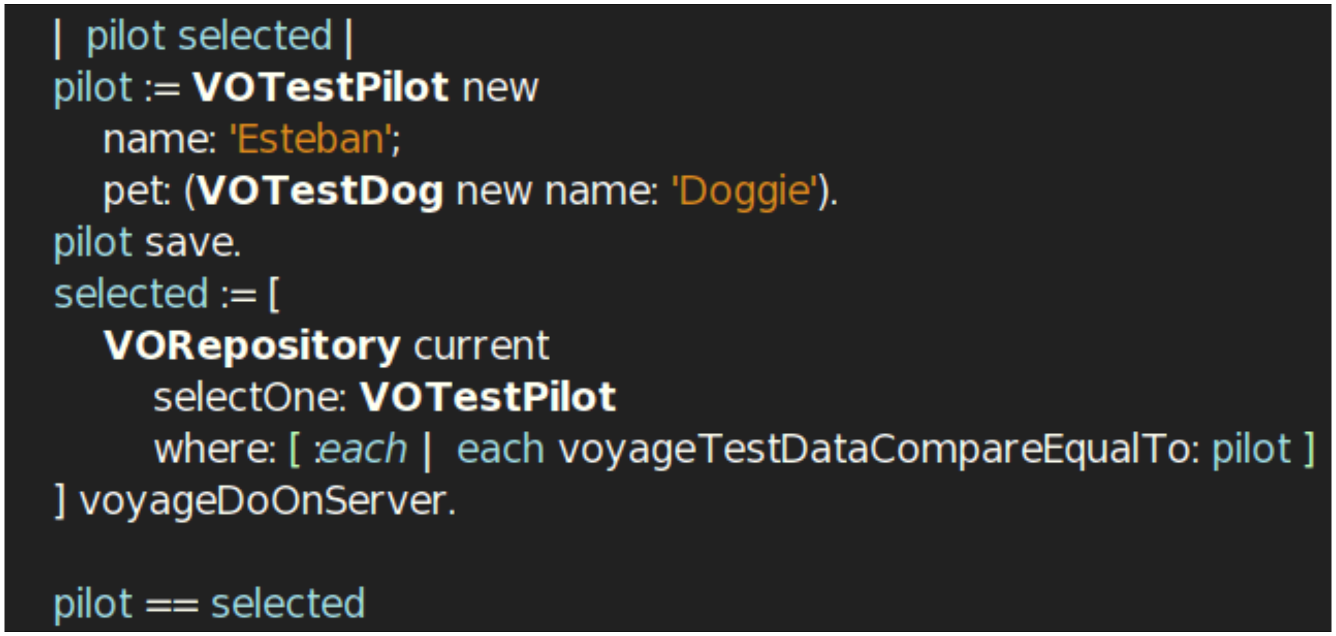
- A root object can be passed as the value of a non-local variable and a returned root object is mapped to client-side instance



```
| pilot clientPet serverPet |
VOTugrikRepository
    dbServerClassName: 'VoyageClassMappingDbServer'
    database: 'Voyage-Tests'.
clientPet := VOTestDog new name: 'Doggie'.
pilot := VOTestPilot new
    name: 'Esteban';
    pet: clientPet.
pilot save.

serverPet := [ pilot pet  ] voyageDoOnServer.

clientPet == serverPet
```

Voyage Server Blocks: Root Object behavior

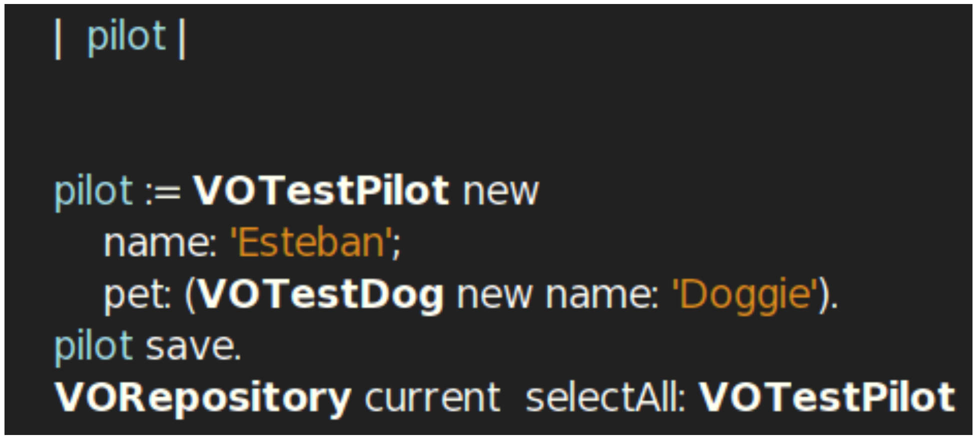
- Voyage query blocks can use full range of root object behavior on server (i.e., *real* select blocks)



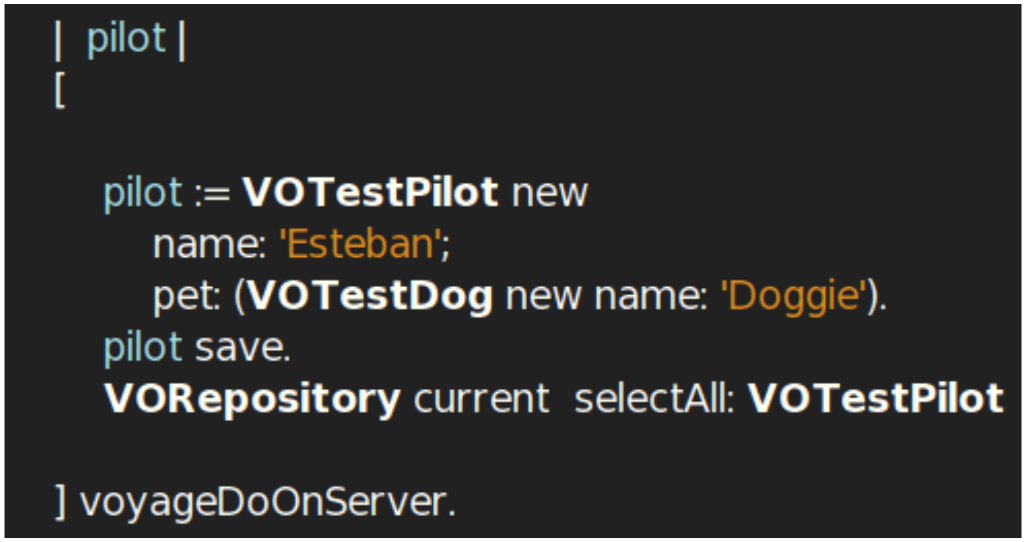
```
| pilot selected |
pilot := VOTestPilot new
    name: 'Esteban';
    pet: (VOTestDog new name: 'Doggie').
pilot save.
selected := [
    VORepository current
        selectOne: VOTestPilot
        where: [ :each | each voyageTestDataCompareEqualTo: pilot ]
] voyageDoOnServer.

pilot == selected
```

Voyage Server Blocks: Voyage API

- Full Voyage API (save, remove, select*) is implemented on server



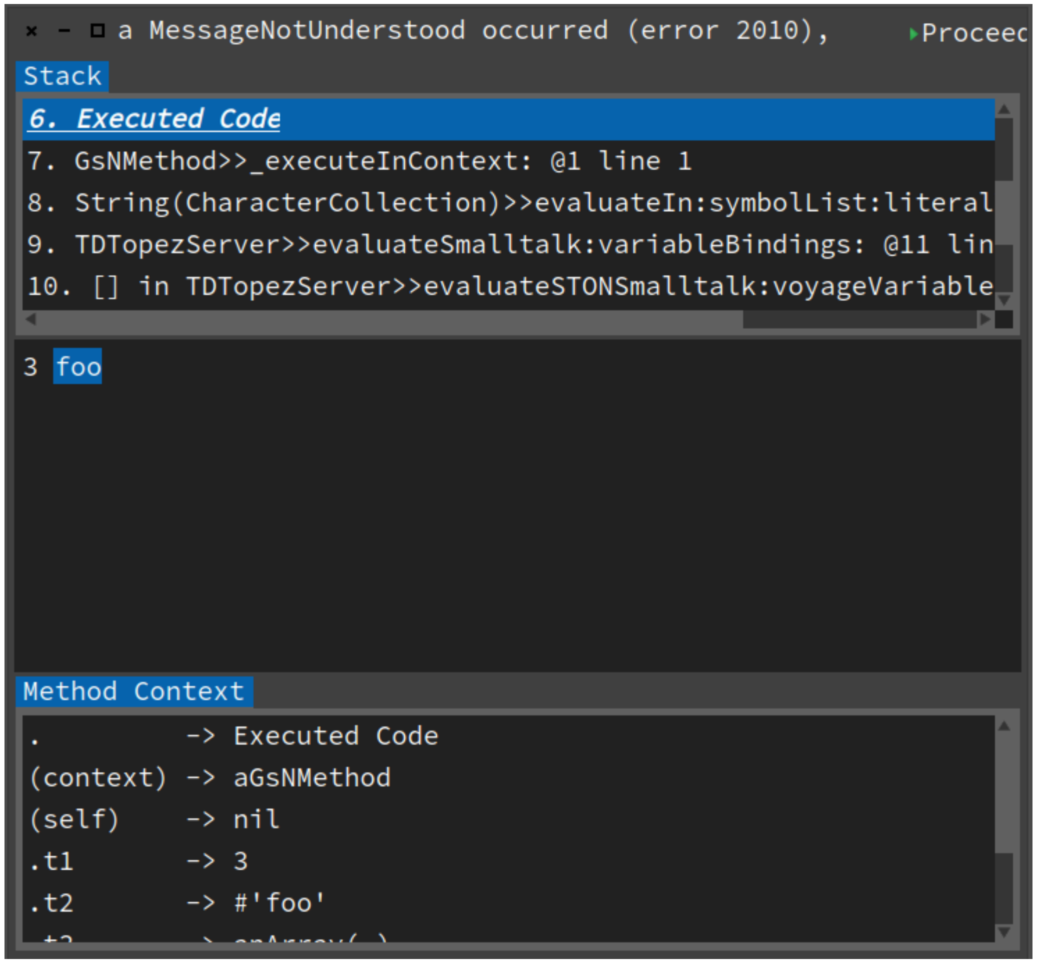
```
| pilot |


pilot := VOTestPilot new
    name: 'Esteban';
    pet: (VOTestDog new name: 'Doggie').
pilot save.
VORepository current  selectAll: VOTestPilot
```



```
| pilot |
[

    pilot := VOTestPilot new
        name: 'Esteban';
        pet: (VOTestDog new name: 'Doggie').
    pilot save.
    VORepository current  selectAll: VOTestPilot

] voyageDoOnServer.
```

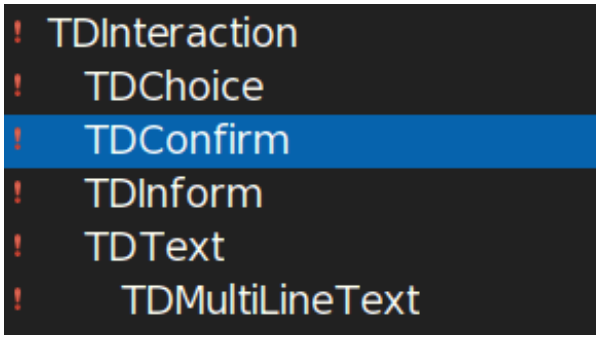
Tugrik Tools: Debugger

```
[ 3 foo ] voyageDoOnServer
```

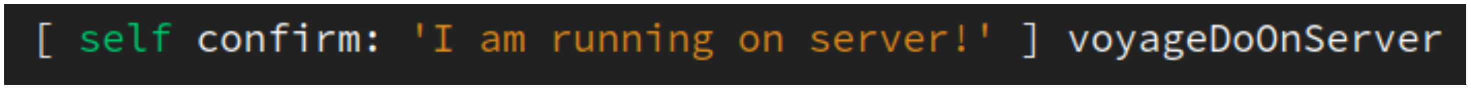


```
× – □ a MessageNotUnderstood occurred (error 2010),      ▸Proceec
Stack
6. Executed Code
7. GsNMethod>>_executeInContext: @1 line 1
8. String(CharacterCollection)>>evaluateIn:symbolList:literal
9. TDTopezServer>>evaluateSmalltalk:variableBindings: @11 lin
10. [] in TDTopezServer>>evaluateSTONSmalltalk:voyageVariable

3 foo

Method Context
.           -> Executed Code
(context)   -> aGsNMethod
(self)      -> nil
.t1         -> 3
.t2         -> #'foo'
```

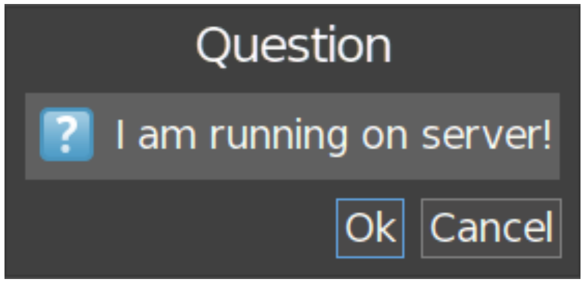
Tugrik Tools: Interactions



```
TDInteraction
  TDChoice
  TDConfirm
  TDInform
  TDText
    TDMultiLineText
```



```
[ self confirm: 'I am running on server!' ] voyageDoOnServer
```



Question

I am running on server!

Future Work

- Tugrik session pool and object cache refactoring
- Improve/add client-side tools
- Harden Tugrik client implementation
- GemStone indexing support(???)
- GemStone Transaction models(???)
- Replication and synchronization improvements(???)
- Automatically compile shared code on Server(???)
- Optimizations

Looking for Partners

- Set agenda and priorities for Future Work
- Contributions!!!!
- My Preferences:
  - incremental development with active participation from users
    - features added based on demand
    - simple over complicated
    - thin clients over fat clients

Path of Least Resistance

- Concurrent document update issues
- Large number of documents
- Interest in Thin Client/Fat Server application

Voyage-Tugrik Installation

```
# Install GsDevKit_home
git clone https://github.com/GsDevKit/GsDevKit_home.git

# Setup $GS_HOME env var and put $GS_HOME/bin in path
cd GsDevKit_home
. bin/defHOME_PATH.env

# Setup system for client/server operations (install OS prerequisites using `sudo`)
installServerClient

# Create tODE client
createClient tode

# Create Voyage stone
createStone -u http://gsdevkit.github.io/GsDevKit_home/Voyage.ston -i Voyage -l Voyage Voyage 3.3.1

# Create Pharo5.0 Voyage client
createClient -t pharo voyage -l -v Pharo5.0 -z $GS_HOME/shared/repos/voyage/.smalltalk-tugrik.ston

# Start Voyage client and register Voyage stone as default server
startClient voyage -s Voyage
```

Questions

Resources

- GemTalk Systems
  - http://gemtalksystems.com
  - https://gemtalksystems.com/products/gs64/versions33x/#33
- Voyage-Tugrik project
  - https://github.com/dalehenrich/voyage#voyage-tugrik
- GsDevKit_home Installation and Documentation
  - http://github.com/GsDevKit/GsDevKit_home
- GLASS (GemStone Open Source Community) mailing list
  - http://forum.world.st/GLASS-f1460844.html