

James Foster

ESUG 2022



A VS Code Extension for GemStone/S



My Recent Environment

- Teaching Computer Science at Walla Walla University
 - From Intro to Programming to Artificial Intelligence
 - Include Smalltalk in Advanced Object-Oriented Programming
 - Give students industry experience

Python to Smalltalk



Grail A Python to Smalltalk Translator Will Hensel



ABSTRACT

This project seeks to make GemStone, an object database system sold by GemTalk Systems, more accessible by supporting Python through an external package. Grail takes Python source code as input and outputs GemStone Smalltalk source code.

Python was chosen because of its popularity among data science fields. GemStone makes storing large amounts of data very convenient and efficient without the need to write overhead database connection code.

INTRODUCTION

GemStone is a database system unlike traditional databases. It doesn't store rows in tables or documents in collections, but instead it stores Smalltalk objects, making it very convenient to use since there isn't a conversion layer for changing objects into a persistable format.

However, GemStone's reliance on Smalltalk is troublesome for most developers and companies. Smalltalk is a language born in the 80's that laid the groundwork for modern Object-Oriented languages, but the language itself never became very popular. Finding a modern Smalltalk programmer is becoming even more difficult with time.

Python boasts a large community of a wide range of programmers from data scientists to researchers in artificial intelligence. Because of these data oriented fields' reliance on Python, it was chosen to be the source language for Grail.

TRANSLATOR

Grail's translation procedure is a three step process

- An string-represented Abstract Syntax Tree (AST) is generated using the Python AST package.
- The string is taken and parsed into a new AST composed of Smalltalk objects.
- A second pass down the tree is made to translate each node to Smalltalk source code.

```
currentScope |
currentScope => Variables new.
(currentScope at: #print)
scope: currentScope
positional: ( (str __value: 'Hello world!'). )
named: Dictionary new
```

"Hello world" script generated by Grail

EXECUTION

The string output from the translation module is then executed, but the execution logic does not depend on the translator. The logic is instead defined in a Smalltalk dictionary called builtins which models the Python package of the same name.

The builtins package defines all the built-in behavior of Python. In order to accurately model Python, we have to model that exact same behavior in Smalltalk. We achieve this by defining every one of Python's built-in classes and methods in Smalltalk. This is tedious work and builtins is still in a partially complete state so the complexity of programs we can execute under Grail is limited.

CHALLENGES

The first challenge I faced was having to port the builtins dictionary from the Pharo Smalltalk dialect into the GemStone Smalltalk dialect. There are some differences between the two such as different suites of methods defined on various classes.

Another big challenge was due to some complex portions of Python's behavior, particularly when it came to scoping variables and function definitions. Python has hierarchical scoping with a root scope where builtins is defined, a module level scope called "global", and child scopes for the various structures defined in the module such as nested function definitions.

SUMMARY

While Grail as a whole is still limited in functionality, my part of the project was very successful. My original overarching goal was to translate some fairly sophisticated Python code into Smalltalk and I achieved that goal. What we have built is a great proof of the power that can be harnessed when combining a data-oriented language like Python with GemStone.

REFERENCES

- <https://gemtalksystems.com/>
- <https://pharo.org/>
- <https://www.python.org/>
- <https://docs.python.org/3/library/ast.html>



Dart to Smalltalk

- <https://www.youtube.com/watch?v=8rH9yMFKoPM>

Climate Research

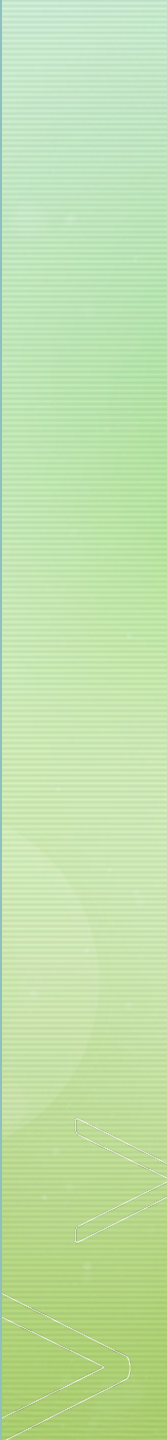


Tank Controller





Disclaimer

- This is a personal project
 - It is not a GemTalk Systems project or product
- 



About GemStone/S

- An object database
- Smalltalk

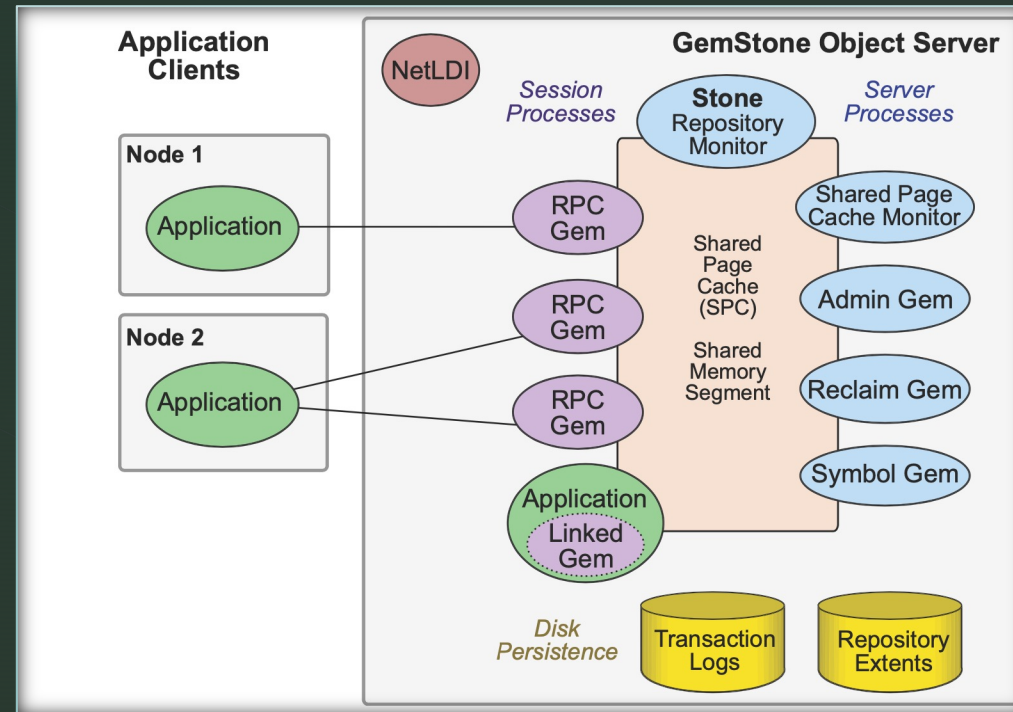


IDE Options for GemStone

- Topaz (command-line)
- GemBuilder for Smalltalk (e.g., VA Smalltalk)
- Jade[ite] (Microsoft Windows)
- Sparkle (Pharo)
- Visual Studio Code?

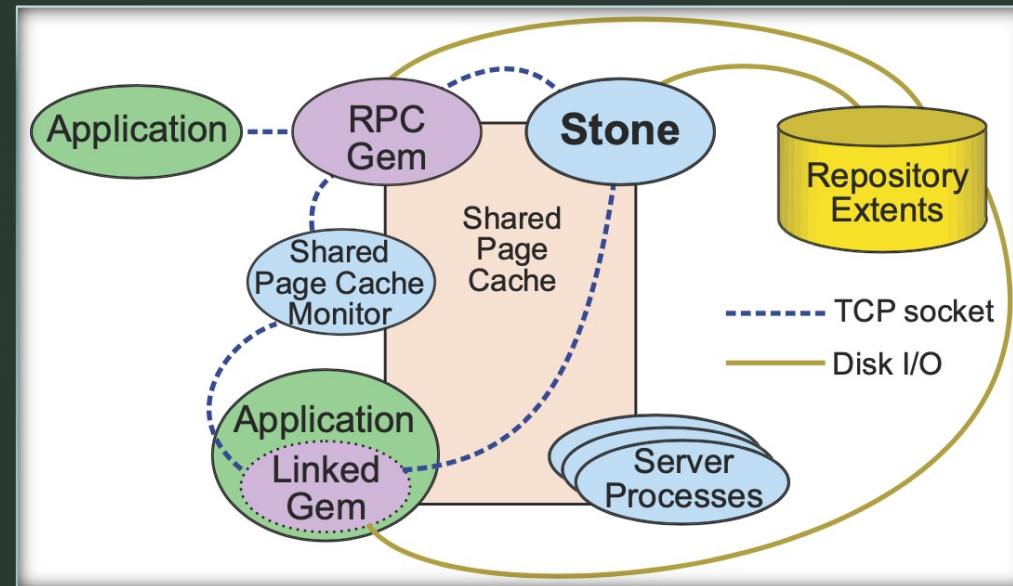
GemStone Architecture

- Objects are on disk in the Repository and in primary memory in the Shared Page Cache (SPC)
- A Gem provides a database session, manipulates objects, and executes Smalltalk code
- A client application interacts with a Gem



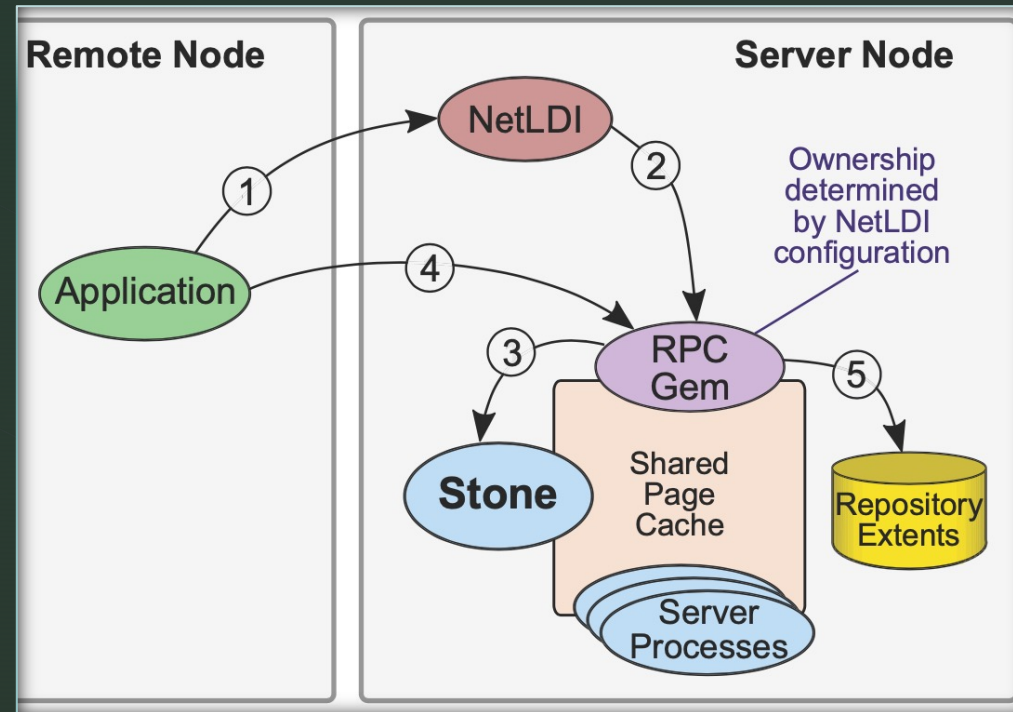
Application Architecture

- Smalltalk code runs in a Gem
- A Gem (server) requires a GCI (client) application
 - Topaz
 - Web server
 - Any FFI-enabled app
 - E.g., Pharo or VA Smalltalk
 - Another Gem!

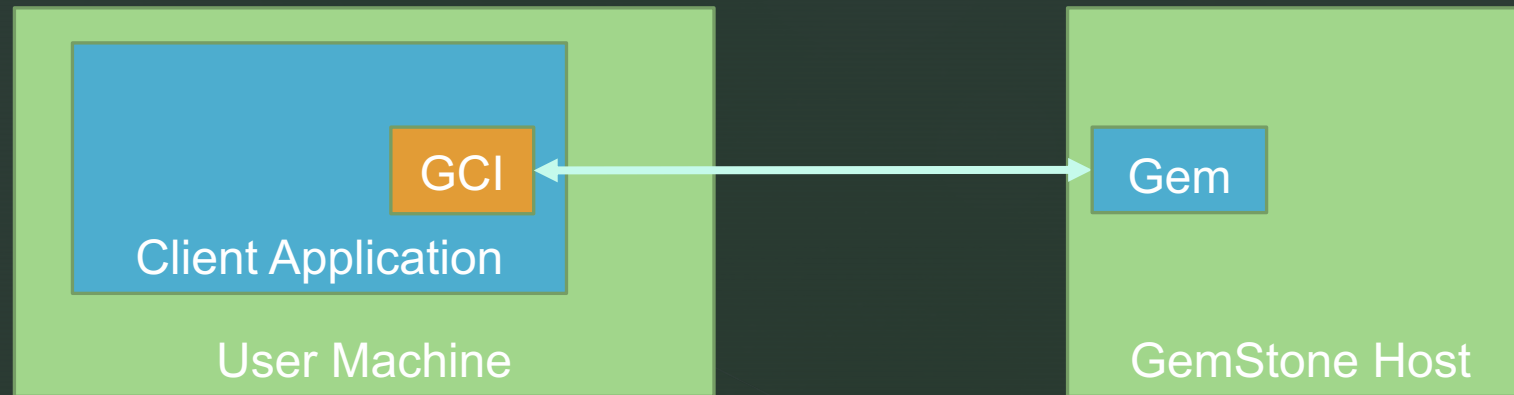


Traditional GCI Connection

1. Ask NetLDI for a Gem
2. NetLDI starts a Gem and passes it the socket
3. Gem connects to Stone
4. (see #2)
5. Gem opens the repository and connects to the SPC



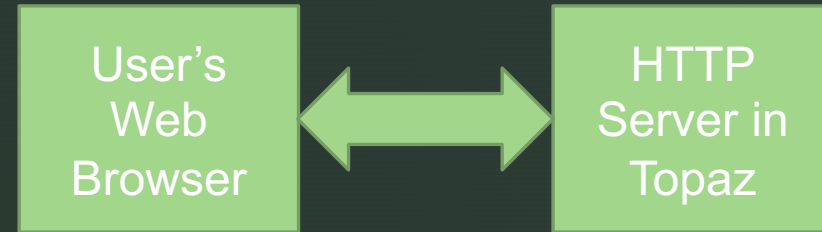
The GemStone C Interface Library



- Shared library: `libgcirpc-3.6.3-64.[dll | dylib | so]`

Web Server Model

- Login Topaz
- Listen on well-known port
- Receive HTTP request
- Send HTTP response
- Often have a traditional web server (Apache, Nginx) in between to handle HTTPS, serve static content, and reverse proxy dynamic requests to the Topaz Gem



About Visual Studio Code

- The most popular developer environment tool
 - 70% of 82,000 in [Stack Overflow](#) 2021 Developer Survey
 - Available from Microsoft on GitHub under the MIT license
- Syntax highlighting, bracket matching, code folding, Git, etc.
- Electron (web technology) runs on Chromium and Node.js
- Extended with (TypeScript) Extensions
 - Add support for languages, themes, debuggers, static code analyzers, and code linters



(Some) IDE Requirements

- Start a Gem (requires a user ID and password)
- Browse and edit code
- Evaluate expressions (identify an object and send a message)
- Object inspector
- Debugger

Connecting VS Code to GemStone

- Traditional approach: a GCI shared library
 - One library for each GemStone version and platform
 - Create a JavaScript wrapper (FFI) to the C library
- Simple Web approach: stateless HTTP request/response
 - Keep state on server
- WebSocket approach
 - WebGS supports WebSockets and a GCI-style API



Demo and Exploring Code

- `package.json`
- `extension.ts`
- ...

Code, Contact, and Questions

- Code
 - <https://github.com/jgfoster/vscode-gemstone>
 - <https://github.com/jgfoster/WebGS>
- Contact
 - James.Foster@GemTalkSystems.com
 - James.Foster@WallaWalla.edu
 - <https://programminggems.wordpress.com/>
- Credits
 - https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html
 - https://www.researchgate.net/figure/Web-socket-architecture_fig3_338553959