Dynamic Cuis Libraries

A platform-independent framework for sharing Smalltalk classes and methods in a “binary” way

Smalltalks 2023

Dynamic Cuis Libraries

Goal

- Share classes and methods among Cuis images without requiring the Compiler and without distributing source code

Dynamic Cuis Libraries

The problem, the challenge

- Provide a straightforward mechanism for sharing Smalltalk classes and methods
- Define a clear and simple format for libraries
- Keep what is necessary in source space to create a library, to a minimum
- Keep what is necessary in target space to receive the code, to a minimum
- Destination image not to require Compiler to install new code
- Be Platform independent
- Be fast

Dynamic Cuis Libraries

The model: ClassSpec (Cuis class description transcription)

- A ClassSpec describes a class. It includes enough detail for a real class to be created from it. But it is not a class:
- It can not create instances. You need a real class for that.
- It is a regular object. Doesn't include any detail specific to Cuis ObjectMemory or VM (Spur). Doesn't refer to globals.
- It is easy to serialize and deserialize. Nothing special to take care of.

Dynamic Cuis Libraries

The model: MethodSpec (Cuis class description transcription)

- A MethodSpec describes a CompiledMethod. It includes enough detail for a real CompiledMethod to be created from it. But it is not a CompiledMethod
- It can't be run. Can't be added to a class.
- It is a regular object. Doesn't include any detail specific to Cuis ObjectMemory or VM (Spur). Doesn't refer to globals.
- It is easy to serialize and deserialize. Nothing special to take care of.

Dynamic Cuis Libraries

The model: DynamicCuisLibrary (Cuis class description transcription)

- A DynamicCuisLibrary defines a set of classes and methods that can be loaded into a running Cuis image. Its parts, instances of ClassSpec and MethodSpec are not actual Classes and CompiledMethods, but regular objects. Actual Classes and CompiledMethods are created from them at load time.

Dynamic Cuis Libraries

Main operations

- Create DCL
  - #withClasses:classSelectors:instanceSelectors:
- Test DCL “installability”
  - #canBeInstalled
- Install DCL
  - installLibrary

Dynamic Cuis Libraries

What is supported

- Dynamic Cuis Library enable:
  - Adding new classes
  - Adding instance variables to existing classes
  - **Support automatic class re-shape**
  - Add new/overwrite existing methods
  - If class is present, ensure same hierarchy path to the root
  - Destination image does not require Compiler
  - **DCL does not include source code**

Dynamic Cuis Libraries

Let’s do some demo (with some “testing sugar”)

- Just a “Testing sugar” to aid writing tests
- Do testing in the same image
- This is only to achieve agility in testing and of course by no means is part of the framework
- Use testing “prototype” classes that serve as a base for creating and modifying classes and methods
- Testing prototype classes begin with ‘noclash’
- Including a class named ‘NoclashXYZ’ in a DCL will impact or materialize a class named ‘XYZ’, so NoclashXYZ will **not clash** with XYZ. And NoclashXYZ remains intact

Dynamic Cuis Libraries

Implementation notes :)

- “Binary way”: bytecodes
- CompileMethod
- ClassSpec
- MethodSpec
- **Bonus track**: **Class reshaping** (involves Juan’s elegant design solution decision about bytecode length that notably simplifies underlying reshaping algorithm) :)

Dynamic Cuis Libraries

Performance: an example, a benchmark

- 10 classes
  - With 3850 methods
  - 1290 Kb source code
  - 280 Kb DCL file
  - FileIn time: 10.000 ms

- **DCL total load time: 58 ms**
  - **ReferenceStream: 38 ms**
  - **#canBeInstalled: 1 ms**
  - **Pure Installation: 19 ms**

Dynamic Cuis Libraries

Some conclusions

- It’s simple
- Lightweight
- Platform-independent
- Foundation for tools
- No source code for sharing classes and methods
- It’s very fast

Dynamic Cuis Libraries

Cuis packages

- DynamicCuisLibraries.pck.st
- TestDynamicCuisLibraries.pck.st