WIKIPEDIA

Language interoperability

Language interoperability is the capability of two different programming languages to natively interact as part of the same system and operate on the same kind of data structures.[1]

There are many ways programming languages are interoperable with one another. HTML, CSS, and JavaScript are interoperable as they are used in tandem in webpages. Some object oriented languages are interoperable thanks to their shared hosting virtual machine (e.g. .NET CLI compliant languages in the Common Language Runtime and JVM compliant languages in the Java Virtual Machine).[2]

Contents

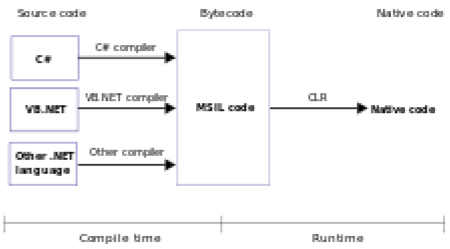
Methods for interoperability

Object models

Object models are standardised models which allow objects to be represented in a language-agnostic way, such that the same objects may be used across programs and across languages. CORBA and the COM are the most popular object models.

Virtual machines

A virtual machine (VM) is a specialised intermediate language that several different languages compile down to. Languages that use the same virtual machine can interoperate, as they will share a memory model and compiler and thus libraries from one language can be re-used for others on the same VM. VMs can incorporate type systems to ensure the correctness of participating languages and give languages a common ground for their type information. The use of an intermediate language during compilation or interpretation can provide more opportunities for optimisation.[1]



Different Languages compile into a shared runtime

Challenges

Object model differences

Object oriented languages attempt to pair containers of data with code, but how each language chooses how to do that may be slightly different. Those design decisions do not always map to other languages easily. For instance, classes using multiple inheritance from a language that permits it will not translate well to a language that does not permit multiple inheritance. A common approach to this issue is defining a subset of a language that is compatible with another language's features.[3] This approach does mean in order for the code using features outside the subset to interoperate it will need to wrap some of its interfaces into classes that can be understood by the subset.

Memory models

Differences in how programming languages handle de-allocation of memory is another issue when trying create interoperability. Languages with automatic de-allocation will not interoperate well with those with manual de-allocation, and those with deterministic destruction will be incompatible with those with nondeterministic destruction. Based on the constraints of the language there are many different strategies for bridging the different behaviors. For example: C++ programs, which normally use manual de-allocation, could interoperate with a Java style garbage collector by changing de-allocation behavior to delete the object, but not reclaim the memory. This requires that each object will have to manually be de-allocated, in order for the garbage collector to release the memory safely.

Mutability

Mutability becomes an issue when trying to create interoperability between pure functional and procedural languages. Languages like Haskell have no mutable types, whereas C++ does not provide such rigorous guarantees. Many functional types when bridged to object oriented languages can not guarantee that the underlying objects won't be modified.

See also

- Foreign function interface
- Language-independent specification
- Language binding
- Glue language
- API reuse
- JVM languages
- CLI Languages
- SWIG

References

1. Malone, Todd (2014). "Interoperability in Programming Languages". CiteSeerX 10.1.1.684.337 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.684.337).
2. "Cross-Language Interoperability" (https://msdn.microsoft.com/en-us/subscriptions/50b25433-59f2-4c6f-9774-7d777cde6b0a%28v=vs.90%29). Microsoft Developer Network

(msdn.microsoft.com).

3. Chisnall, David (2013-10-01). "The Challenge of Cross-language Interoperability" (http://dl.acm.org/citation.cfm?id=2542661.2543971). *Queue*. **11** (10): 20–28. doi:10.1145/2542661.2543971 (https://doi.org/10.1145%2F2542661.2543971). ISSN 1542-7730 (https://www.worldcat.org/issn/1542-7730).