Interface Theory

Daniel Nikpayuk

March 2, 2018

The general formula for cleanly designing any interface:

1. Constructively define the structure being modeled.
2. Define every possible substructure.
3. Find all possible algebras (spaces) of substructures.
4. Compress each substructure space, this is an interface.
5. Determine how each interface relates to every other interface.
6. Use chains of interfaces to stratify and mitigate levels of interface privilege, this allows for the trade-off between optimization and safety.

This is the direct approach, although in practice if your initial structure is infinite in size, there will be infinite combinatorial possibilities so it becomes impractical. Even in finite cases, the combinatorial possibilities generally increase rapidly, so heuristic approaches will still need to be taken. Very rarely can you actually brute force the algorithm, but just knowing what it is actually helps quite a lot.

Following this, one of the first orders of business is to find a basis if possible. As our entire space is constructive, what are the constructive grammatical elements that allow us to build any given interface? Finding such bases reveals the bare minimum we'd need to equip a library with to build a given interface.

Another first order of business is to prove interface chains are *transitive*, meaning for every subinterface (indirect interface), there exists an isomorphic direct interface. From a user's perspective, this means you can interface indirectly, then subinterface, then subinterface, and so on, until you're working with your intended interface resolution, or you can work directly at that resolution level, or any level in-between. From a compiler's perspective, this means you can inline (translate) any indirect grammar into direct grammar for optimization purposes.