The Benefits of Generators for Reuse

James M. Neighbors
Bayfront Technologies, Inc.
neighbrs@netcom.com

Generators are the mechanism I advocate for building systems from reusable components. In this panel we are not concerned with the details of generator features and implementation. The panel is concerned with what we expect to get by using a generator that combines reusable parts that are not necessarily code. Each section below describes a benefit of generator usage. Examples are drawn from our experience in generating communication protocols.

Aids System Description Using Domain Concepts
A generator input specification is not compilable code (such as C with if-defs). The specification might range from a feature checklist to a full specification language that enables an unbounded combination of domain concepts. The specification uses problem domain specific terms and identifies the domain-specific components that must be used. From a specification the interconnection of existing concepts in the reusable component library can be both constrained and determined. The allowed structure of this specification provides a framework for the system specifiers. It is a guide, education, and completeness check for the system specifiers. Code component users must determine the interactions and compatabilities between components themselves.

Avoids Code Corrosion
Domain analysis - the analysis of a particular problem domain to define objects (not necessarily the same as programming language objects) and operations to serve problems in that domain - is expensive [Neighbors84]. It is cost effective only if the analysis can be used in the construction and maintenance of many systems. The reusable information must be maintained for the multi-year lifetime of the system. Code-level descriptions are doomed to corrosion over time by the incessant change of. languages, compilers, and operating systems. If the domain analysis is captured in code-level fragments, it is subject to this corrosion. Defining a domain-specific problem specification language for use as generator input isolates the domain analysis from code corrosion. The low-level code components (such as I/O, process models, object models) used by the generator still need to be maintained but this maintenance becomes uniform, centralized, crosses system boundaries, and crosses domain boundaries.

Allows Implementation Variation
Inline code, threaded code, and threaded code interpreters are all valid schemes for implementing a subsystem. They are only a few of the general schemes for providing *Structural Architecture* - how a subsystem provides it function [Neighbors94]. If the components in a reusable library are abstracted to the code level, there is no good way to provide various implementations other than to multiply the components by the number of variations. A generator may make these general decisions during generation. These are important distinctions. In communication protocols a cellular phone protocol might be generated as a small, slow threaded code interpreter to be put into a cellular phone while the same protocol might be generated inline for use in a fast cell switch.

Allows Implementation Goal Variation
It goes without saying that the goal of system specification is to ultimately obtain a working system. However, during development many other needs must be met. In communication protocols some of these needs are analysis and simulation. Protocol analysis checks for protocol completeness and resource deadlocks. Protocol simulation produces data necessary to establish resource controls such as process priorities, buffer sizes, and timer time-out limits. With a generator the input specification of a protocol can be used to generate working protocol code, simulation code, and analysis tool input data all from the same description. It would be difficult to produce an assemblage of code-level components that could achieve this same effect.

Provides Optimization Above the Code Level
Optimization is an "after it works" task. No one cares how fast your system is if it doesn't meet it's requirements. This observation has reduced optimization to a process applied to code or by optimizing compilers of code. With a generator optimization can be performed on the input system specification. In communication protocols there are obvious examples. A series of actions triggered by a timer time-out can be removed from the description of a state if it can be shown that the timer can never be enabled in that state. These optimizations can be impossible to recognize in code because the code is a model of the domain concept [Neighbors84].

[Neighbors94] Neighbors, James M., An Assessment of Reuse Technology After Ten Years, *Third International Conference on Software Reuse*, IEEE Press, pp 6-13, November 1994

[Neighbors84] Neighbors, James M., The Draco Approach to Constructing Software from Reusable Components, *IEEE Trans. on Software Engineering*, vol SE-10(5), pp 564-574, September 1984.