Adaptable Components

Grady H. Campbell, Jr.
Prosperity Heights Software
8457 Van Court
Annandale, VA 22003 USA
+1 703 573 3139
GradyCampbell@acm.org
www.domain-specific.com

ABSTRACT
The key to viable software reuse is the ability to rapidly retrieve and tailor software components for new uses. An adaptable component is a representation of a family of similar software components which supports tailoring as an intrinsic aspect of retrieval. Differences among the instances of a family are conceived as a set of feature decisions and represented as parameters of adaptability. Organized into a domain-specific taxonomy of architectural categories, adaptable components provide for rapid retrieval of components tailored to suit each particular use.

Keywords
Reuse, component, domain-specific, adaptable, family, generator, abstraction, metaprogram

1 INTRODUCTION
An adaptable component represents a family of similar software components. It provides the capability to derive an instance component that is tailored to a particular use, for reuse or in response to changed requirements. The diversity of instance components represented by an adaptable component is expressed as a set of parameters that define reusers' selection/tailoring decision criteria. An adaptable component represents a set of similar instance components and can express the form and content of any associated information, including code, documentation, and test cases. A reusable component can be derived to suit the specific needs of a particular application by instantiation of a suitable adaptable component. A collection of architecturally integrated adaptable components provides the basis for systematic derivation of complete software products.

2 BENEFITS OF ADAPTABLE COMPONENTS
The motivation for adaptable components is to represent reusable software in a form that makes comprehensive need-specific tailoring implicit to retrieval of each needed component. In this way, storage of reusable components is highly efficient, with each adaptable component implicitly representing an arbitrarily large number of retrievable instance components.

Adaptable components provide benefits both to component developers and to reusers:

- An adaptable component is a concise representation of an arbitrarily large collection of need-specific parts, avoiding the effort of handcrafting or tailoring of each one individually and resulting in fewer errors.
- A hierarchical file system is sufficient for effective, low cost storage of adaptable components; no special search or retrieval mechanisms are needed.
- Components of greater diversity are retrievable from an adaptable components repository, through mechanical tailoring to specific needs as part of their retrieval, providing more flexibility for new uses.
- Adaptable components enable reusers to be effective with less deep or detailed expertise in the various aspects of a problem and its solution, resulting in better solutions with less effort.

Obtaining these benefits requires an explicit investment in the development of adaptable components. Developing a high quality adaptable component costs more than a single instance component but pays off when a future need for three or more similar instance components is anticipated. This investment will provide substantially greater payoff sooner when managed as a coordinated independent service to product development efforts, involving an organization's best people and considering business and technical needs, current and future.

3 TWO MODELS OF REUSE
The usual procedure for reuse presumes a library of reusable instance components. This procedure consists of 4 steps:

- Retrieve a candidate set of components that approximate some needed capability.
- Select the component from the candidate set that most closely matches the specific need.
- Tailor the selected component to all aspects of the specific need.
- Return the tailored component to the repository as a candidate for future reuse.

The model of reuse underlying this procedure is weak, providing too little benefit to justify the associated costs. It presents several problems:

- Deciding whether an appropriate component exists requires indeterminate effort, linear to the number of components available or dependent on unreliable techniques of concept or content matching or complicated indexing schemes.
- Choosing among similar candidate components requires detailed, time-consuming comparative analyses of the components.
- Tailoring a component to meet slightly different needs can violate unstated developer assumptions resulting in errors. Failure to tailor or remove unneeded capabilities results in inefficient or bloated code.
- Every adaptation of a reusable component creates a new candidate for reuse resulting in uncontrolled growth in size and complexity of the repository.

In contrast, the procedure for reuse based on adaptable components is:

- From a taxonomy of provided adaptable components, select the adaptable component, if any, in which the needed reusable component should be found.
- Consistent with particular needs, resolve the decisions associated with the selected component, mechanically deriving the corresponding reusable component.
- If an appropriate category does not exist or the adaptable component does not support needed tailoring decisions, describe the unsatisfied need to adaptable component developers.

This procedure does not guarantee that a developer will find a needed reusable component but it guarantees that the effort to determine success or failure is very low. An adaptable components repository avoids the problems associated with the conventional model of reuse. In contrast:

- The thought that goes into abstracting similar reusable components into an adaptable component leads naturally into the identification of a hierarchy of well-defined categories that substantially reduces the effort of identifying the existence, or non-existence, of components appropriate to a particular need.
- An adaptable component establishes a certain level of standardization, retaining essential diversity but avoiding unnecessary redundancy that would exist in separately represented components and eliminating non-essential differences which arise when different people develop components that provide similar capabilities.
- Part of creating an adaptable component is envisioning future alternative uses that would be a natural extension of any existing set of similar components. This increases the likelihood that an adaptable component will suit each developer's particular needs without additional tailoring of a derived component.
- As reuser needs change, the need for new components arise in the context of the existing structure of adaptable components. This adds a degree of discipline in controlling the evolution of the repository, reducing the likelihood of redundant components or categories.
- The differences among the set of components represented by an adaptable component are formalized as a set of decisions that indicate the diversity represented. These decisions characterize alternative uses for a component and are a sufficient basis for identifying a single instance component for retrieval.

4 SUPPORTING THEORY

An adaptable component represents a family of similar components. Dijkstra [1] proposed and Parnas [2] elaborated the idea that families ought to be the basis for the systematic construction and evolution of programs. A family is characterized by an abstraction that expresses what is common about its instances. An adaptable component expresses a family in concrete form. The differences among instances are represented by a set of decision parameters that together are sufficient to distinguish each producible instance component from all others in the family.

Given a precise notation for representing a family as an adaptable component, a corresponding instantiation mechanism enables the derivation of instances. The body of an adaptable component is a definition of how common and varying component fragments are tailored and combined to derive any particular instance component. Goguen [3] and Dershowitz [4] discuss concepts of abstraction, parameterization, and instantiation that underlie adaptable components. A particular notation and mechanism that is easily adopted by programmers is described in [5].

Adaptable components are an effective medium for flexible reuse and program evolution with text-based instances. Future work needs to address application to graphical forms and uses of formal verification techniques for establishing correctness of a family of derivable instances.

REFERENCES

1. Dijkstra, E.W. Notes on Structured Programming. *Structured Programming*, Academic Press (London, 1972), 39-41.

2. Parnas, D.L. On the Design and Development of Program Families. *IEEE Trans. Software Eng.* SE-2 (March 1976), 1-9.

3. Goguen, J.A. Parameterized Programming. *IEEE Trans. Software Eng.* SE-10, 5 (Sep 1984), 528-543.

4. Dershowitz, N. Abstraction and Instantiation. *ACM Trans. Program. Lang. Syst.* 7, 3 (July 1985), 446-477.

5. PHS Web Site. On-line at http://www.domain-specific.com/MTP/index.html.