ABSTRACT
A key benefit of generic programming is its support for producing modules with clean separation. In particular, generic algorithms are written to work with a wide variety of unmodified types. The Runtime concept idiom extends this support by allowing unmodified concrete types to behave in a runtime polymorphic manner. In this paper, we describe one implementation of the runtime concept idiom, in the domain of the C++ standard template library (STL). We describe and measure the performance of runtime-polymorphic analogs of several STL algorithms. We augment the runtime concept idiom by employing a dispatch mechanism that considers both type and concept information to maximize performance when selecting algorithm implementations. We use our implementation to demonstrate the effects of different compile-time vs. run-time algorithm selection choices, and we indicate where improved language and compiler support would be useful.

Categories and Subject Descriptors
D.1.0 [Programming Techniques]: General

General Terms
Design, Languages

Keywords
Generic Programming, Runtime Polymorphism, C++, Standard Template Library

1. INTRODUCTION
ISO C++ [12] supports various programming paradigms, notably object-oriented programming and generic programming. Object-oriented techniques are used when runtime polymorphic behavior is desired. When runtime polymorphism is not required, generic programming is used, as it offers non-intrusive, high performance compile-time polymorphism; examples include the C++ Standard Template Library (STL) [5], the Boost Libraries [1], Blitz++ [17], STAPL [4].

Recent research has explored the possibility of a programming model that retains the advantages of generic programming, while borrowing elements from object-oriented programming, in order to support types to be used in a runtime-polymorphic manner. In [15], Parent introduces the notion of non-intrusive value-based runtime-polymorphism, which we will refer to as the runtime concept idiom. Marcus et al. [14], [3], and Parent [16] extend this idea, presenting a library that encapsulates the common tasks involved in the creation of efficient runtime concepts. Järvi et al. discuss generic polymorphism in the context of library adaptation [13].

A key idea in generic programming is the notion of a concept. A concept [10] is a set of semantic and syntactic requirements on types. Syntactic requirements stipulate the presence of operations and associated types. In the runtime concept idiom, a class $R$ is used to model these syntactic requirements as operations. The binding from $R$ to a particular concrete type $T$ is delayed until runtime. Any type $T$ that syntactically satisfies a concept’s requirements can be used with code that is written in terms of the runtime concept.

In this paper, we apply these principles to develop a runtime-polymorphic version of some STL sequence containers and their associated iterators. Runtime concepts allow the definition of functions that operate on a variety of container types.

Consider a traditional generic function expressed using C++ templates:

```cpp
// conventional template code
template <class Iterator>
Iterator random_elem(Iterator first, Iterator last)
{
   typename Iterator::difference_type dist = distance(first, last);
   return advance(first, rand() % dist);
}
```

Objects of any type that meet the iterator requirement...