We present in this paper a proposal for developing efficient programs in the abstract data type (ADT) programming framework, keeping the modular structure of programs and without violating the information hiding principle. The proposal focuses on the concept of “shortcut” as an efficient way of accessing to data, alternative to the access by means of the primitive operations of the ADT. We develop our approach in a particular ADT, a store of items. We define shortcuts in a formal manner, using algebraic specifications interpreted with initial semantics, and so the result has a well-defined meaning and fits in the ADT framework. Efficiency is assured with an adequate representation of the type, which provides O(1) access to items in the store without penalising the primitive operations of the ADT.

INTRODUCTION

Modular programming with abstract data types (ADT) (Liskov & Guttag, 1986) is a widespread methodology for programming in the large. In this field, it is crucial the distinction between the specification and the implementation of ADTs, which results in the existence of different modules for them and which can be summarised with the information hiding principle: an ADT must be used just regarding the properties stated in the specification, without any knowledge of the characteristics of its implementation, which remains hidden. This principle simplifies the relationships between modules and supports the development of programs, because it is easier to code them, to test them, to reuse them and to maintain them.

However, the information hiding principle collides, often dramatically, with a very usual requirement on programs: their efficiency, mainly characterised by their execution time. The reason is that the access to a data structure implementing an
ADT must follow the properties that define it, which were stated in an abstract manner without taking into account the problems related to its subsequent implementation (as it must be). In case of a context using ADTs with strong efficiency requirements (for instance, program analysis tools construction, system programming, geometric computing and combinatorial computing), their full reusability can become impossible and it may be necessary to carry out many modifications to fit it to this context; even more, such modifications can be so important to decide throwing away the implementation and developing a new one.

This conflict between efficiency and modularity is a well known problem in the ADT framework, recognised as such in the most important textbooks on data types and data structures (Aho, Hopcroft & Ullman, 1983; Horowitz & Sahni, 1994; Cormen, Leiserson & Rivest, 1990), and solved in many cases sacrificing modularity to achieve efficiency. Fortunately, there are many widespread ADT-libraries that have coped the problem by incorporating the notion of location (i.e., a cursor — an integer referring to an array position — or a pointer) in ADT interfaces. This is the case for instance of STL (Musser & Saini, 1996) and LEDA (Mehlhorn & Naher, 1999), both of them providing a similar solution to the problem: when a new element is stored in the data structure, its location is returned as part of the result, being later usable as parameter in other operations (removal, lookup and modification). Unfortunately, these libraries present some drawbacks due to the fact that they are designed with the concept of location incorporated in the component from the very beginning. Therefore, the implementations that can be used for the ADT are often restricted to a fixed set (which makes these libraries not flexible enough), the behaviour is less clear (locations and elements appear at the same level) and some classical low-level problems appear (for instance, meaningless uses of cursors and pointers).

Our goal in this paper is to define a general framework to reconcile both criteria, efficiency and modularity, obtaining thus efficient programs reusing existing implementations of ADTs without any modification, and following the information hiding principle. The proposal is based on the definition of an alternative way to access data, that we call shortcuts. Shortcuts are added to existing ADTs in a systematic manner, obtaining new ADTs (compatible with the previous ones) that incorporate these alternative access paths. Then, the users of the new ADT will be able to access the data therein not only by means of the operations introduced in the original specification (that are the ones defining the underlying mathematical model), but also using other new ones which follow these alternative paths, when the use of the former operations is considered unacceptably expensive. We are going to develop the proposal on a particular ADT, a \textit{STORE} of items, although the conclusions of our work can be applied to any other container-like ADT, i.e. those ones arranging collections of items with an arbitrary (but completely defined) policy. More details can be found at (Marco & Franch, 1997).
Related Content

The Anatomy of the ArchiMate Language
[www.igi-global.com/article/anatomy-archimate-language/40951?camid=4v1a](www.igi-global.com/article/anatomy-archimate-language/40951?camid=4v1a)

Enforcing ASTD Access-Control Policies with WS-BPEL Processes in SOA Environments
[www.igi-global.com/chapter/enforcing-astd-access-control-policies/72001?camid=4v1a](www.igi-global.com/chapter/enforcing-astd-access-control-policies/72001?camid=4v1a)

ANFIS Modeling of Dynamic Load Balancing in LTE
[www.igi-global.com/chapter/anfis-modeling-of-dynamic-load-balancing-in-lte/79672?camid=4v1a](www.igi-global.com/chapter/anfis-modeling-of-dynamic-load-balancing-in-lte/79672?camid=4v1a)
A Model-Based Approach to Aligning Business Goals with Enterprise Architecture
Tony Clark and Balbir Barn (2013). Progressions and Innovations in Model-Driven Software Engineering (pp. 252-274).
www.igi-global.com/chapter/model-based-approach-aligning-business/78216?camid=4v1a