Flexible access control policy for SCOOP

Piotr Nienaltowski
Praxis High Integrity Systems Limited, 20 Manvers Street, Bath BA1 1PX, UK
E-mail: piotr.nienaltowski@praxis-his.com

Abstract. The SCOOP model extends Eiffel to support the construction of concurrent applications with little more effort than sequential ones. The model provides strong safety guarantees: mutual exclusion and atomicity at the routine level, and FIFO scheduling of clients' calls. Unfortunately, in the original proposal of the model (SCOOP_97) these guarantees come at a high price: they entail locking all the arguments of a feature call, even if the corresponding objects are never used by the feature. In most cases, the amount of locking is higher than necessary. Additionally, a client that holds a lock on a given processor cannot relinquish it temporarily when the lock is needed by one of its suppliers. This increases the likelihood of deadlocks; additionally, some interesting synchronisation scenarios, e.g. separate callbacks, cannot be implemented. We propose two refinements of the access control policy for SCOOP: a type-based mechanism to specify which arguments of a routine call should be locked, and a lock passing mechanism for safe handling of complex synchronisation scenarios with mutual locking of several separate objects. When combined, these refinements increase the expressive power of the model, give programmers more control over the computation, and enable more potential parallelism, thus reducing the risk of deadlock.

Keywords: Concurrency; Object-oriented programming; Design by Contract; SCOOP; Attached types; Locking; Callbacks

1. Introduction

The SCOOP model, initially proposed by Meyer [Mey97] and subsequently refined and implemented in [Nie07], offers a disciplined approach to building high-quality concurrent systems. The idea of SCOOP is to take object-oriented programming as given, in a simple and pure form based on the concepts of Design by Contract (DbC) which have proved highly successful in improving the quality of sequential programs [Mey92], and extend them in a minimal way to cover concurrency and distribution. Concurrency in SCOOP relies on the basic mechanism of object-oriented computation: the feature call. Each object is handled by a processor—a conceptual thread of control—referred to as the object's handler. All features of a given object are executed by its handler, i.e. only one processor is allowed to access the object. Several objects may have the same handler; the mapping between an object and its handler does not change over time. If the client and the supplier objects have the same handler, the feature call is synchronous; if they have different handlers, the call becomes asynchronous, i.e. the computation on the client's handler may move ahead without waiting. Objects handled by different processors are called separate; objects handled by the same processor are non-separate. A processor, together with the object structure it handles, forms a sequential system. Therefore, every concurrent system may be seen as a collection of interacting sequential systems; conversely, a sequential system may be seen as a particular case of a concurrent system (with only one processor).

Correspondence and offprint requests to: P. Nienaltowski, E-mail: piotr.nienaltowski@praxis-his.com
Since each object may be manipulated only by its handler, there is no object sharing between different threads of execution (no shared memory). Given the sequential nature of processors, this results in the absence of intra-object concurrency: there is never more than one action performed on a given object at a given time. Therefore, programs are data-race-free by construction. Locking is used to eliminate atomicity violations, i.e. illegal interleaving of calls from different clients. For a feature call to be valid, it must appear in a context where the client’s processor holds a lock on the supplier’s processor. Locking is achieved through the refined mechanism of feature application: the processor executing a routine blocks until the processors handling the objects represented by the actual arguments have been locked (atomically) for its exclusive use; the routine serves as a critical section. Since a processor may be locked by at most one other processor at any time, and all feature calls to objects handled by the same processor are executed in a FIFO order, no harmful interleaving occurs.

The lock-based access control policy provides strong safety guarantees but the price to pay is relatively high: all arguments of a feature call have to be locked, even if they are never used by the feature as targets of calls. Such unnecessary locking limits parallelism and increases the likelihood of deadlocks. Furthermore, a client holding a lock cannot relinquish it temporarily when the lock is needed by one of its suppliers. As a result, certain interesting concurrency scenarios require convoluted implementation patterns; other scenarios, e.g. separate callbacks, cannot be implemented at all. We propose two ways of relaxing the strict locking policy to solve the above problems: (1) a type-based mechanism to specify which arguments of a routine call should be locked, and (2) a lock passing mechanism, related to that introduced in [BPJ07], for safe handling of mutual locking between several separate objects.

The rest of the article is organised as follows. Section 2 analyses the access control policy of SCOOP and introduces the mechanism for selective locking. (The problems of precondition weakening and precursor calls discussed there are not concurrency-specific; their analysis and the proposed solutions may be regarded as contributions to DbC in general.) Sect. 3 introduces the lock-passing mechanism; Sect. 4 discusses its importance for proofs of software correctness. Section 5 presents related work. Finally, Sect. 6 concludes and points out future research directions.

In the rest of this article, we refer to the original model proposed in [Mey97] as SCOOP_97, and to the refined model presented in [Nie07] simply as SCOOP.

2. Eliminating unnecessary locks

The access control policy of SCOOP_97 requires all formal arguments of a feature to be reserved before the feature is applied, as expressed by the Feature Application Rule (Definition 1). The processor applying the feature to a target object blocks until all the processors handling the actual arguments have been reserved.

**Definition 1 (Feature application rule)** Before a feature is applied, its formal arguments must be reserved by the supplier’s handler, and its precondition must hold.

But this rule is too restrictive, as illustrated in Fig. 1. The handlers of x, y, and z must be locked on behalf of the executing processor before the body of r is executed (some precondition must also hold before executing...