Concept-Oriented Programming

Alexandr Savinov
University of Bonn, Germany

INTRODUCTION

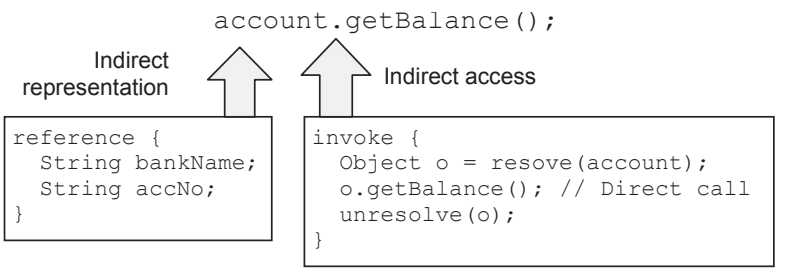
In the concept-oriented programming (CoP) (Savinov, 2005, 2007), the main idea is common to many other approaches and consists in raising the abstraction level of programming by introducing new language constructs and mechanisms. The distinguishing feature of CoP is that it aims at automating the way objects are represented and accessed (ORA). More specifically, one of the main concerns in CoP is modeling the format of object references and the procedures executed during object access.

For example, if we need to retrieve the current balance stored in a bank account object then we make the following simple method call: `account.getBalance()`. In object-oriented programming (OOP), it results in an *instantaneous* execution of the target method because this variable contains a *primitive* reference which is supposed to provide *direct* access to the represented object. In CoP, it is not so and everything depends on the format of the reference used to represent this account object. References in CoP have an arbitrary custom format defined by the programmer and hence objects are represented *indirectly* using abstract identifiers from a virtual address space. In this case, the real procedure executed during access depends on what is stored in the variable `account`. In particular, it may well happen that the account object is stored on a remote computer in another organization. Then, its reference can be rather complex and include such fields as `bankName` and `accNo` (Figure 1). Object access to such an indirectly represented account will involve many intermediate operations like security checks, transaction management, network packet transfer and operations with persistent storage. However, all these intermediate actions will be executed behind the scenes so that we have the illusion of instantaneous action. Then the programmer is still able to use the target objects as if they were local directly accessible objects, at the same time having a possibility to inject any intermediate code responsible for object representation and access (ORA).

References in CoP are as important as objects because both have arbitrary structure and behavior associated with them. If OOP deals with objects then CoP deals with both objects and references. The main role of references consists in representing objects, that is, they contain some data that makes it possible to access the object. Thus, references are *intermediate* elements which are activated each time the represented object is about to be accessed. For example, each time we read or write a field, or call a method, the object reference intercepts these requests and injects its own actions. Thus, any object access can trigger a rather complex sequence of intermediate actions which are executed behind the scenes. In large programs this hidden functionality associated with references can account for a great deal or even most of the overall complexity. The main task of CoP in this sense consists in providing adequate means for effectively describing this type of hidden intermediate functionality which has a cross-cutting nature. OOP does not provide any facilities for describing custom references and all objects are represented and accessed in one and the same way. CoP fills this gap and allows the programmer to effectively separate both concerns (Dijkstra, 1976): explicitly used business logic of objects and intermediate functions executed implicitly during object access.

The problem of indirect object representation and access can be solved by using such approaches as dynamic proxies (Blosser, 2000), mixins (Bracha & Cook, 1990; Smaragdakis & Batory, 1998), metaobject protocol (Kiczales et al.,

Figure 1. Indirect method call via custom references and intermediate operations



1991; Kiczales et al., 1993), remoting via some middleware (Monson-Haefel, 2006), smart pointers (Stroustrup, 1991), aspect-oriented programming (Kiczales et al., 1997) and others. However, CoP is the only technology that has been developed for precisely this problem and solves it in a principled manner. It is important that CoP generalizes OOP by providing a possibility of smooth transfer to the new technology.

BACKGROUND

Hierarchical Address Space

A concept-oriented program can be viewed as a set of nested spaces (Figure 2). Each space has one parent where it is identified by some local address. The parent space itself is identified by some address relative to its own parent and so on till the root. Thus, any element is identified by a sequence of local addresses where each next address identifies the next space. Such an identifier is referred to as a *complex address* while its constituents are referred to as *segments*. This structure is analogous to the conventional postal addresses where cities are identified by names within countries and streets have unique names within cities. For example, an element in the postal address space could be identified by the following complex address: `<"Germany," "Bonn," "University of Bonn">`.

An important consequence of such a design is that objects can interact only by intersecting intermediate space borders. An access request such as a method call or message cannot *directly* (instantaneously) reach its target. Instead, it follows some access path starting from the external space and leading to the internal target space (Figure 2). In order to access an element of the space it is necessary to resolve all segments of its complex reference starting from the high segment and ending with the low segment, which is the target object. The resolution procedure is responsible for locating the element identified by one segment in the context of the parent space. Thus, each intermediate border along the access path executes some special functions, which are triggered automatically as an access request intersects this border.

The same approach is used in CoP where objects are identified by complex references defined by the programmer rather than using primitive references. For example, if account reference consists of two segments—bank name and account number—then the balance could be obtained as usual by applying a method to this complex reference:

```
Account account = <"MyBank," "98765432">;
double balance = account.getBalance();
```

Since objects in CoP are represented by complex references each access requires several intermediate steps for locating the object. For example, in order to resolve the account object represented by its bank name and account number it is necessary to find the bank object and then to find the account object. Notice that the method applied to the reference is only the last step in this indirect access procedure. An important assumption of the concept-oriented approach to programming is that most of the functionality in large programs is concentrated on intermediate space borders. Target methods in this case account for a relatively small portion of the overall complexity. The goal of CoP in this sense can be formulated as providing support for describing such a hierarchical space at the level of the programming language rather than in middleware or hardware. The programmer then is able to describe an arbitrary *virtual address space* which serves as a container for objects. Such addresses are virtual because they are not directly connected with the real object position and hence they provide an additional level of abstraction.

References and Objects

In OOP, the programmer models objects by classes while all references have one and the same type. Thus we cannot influence how objects are represented and how they are

Figure 2. A program can be viewed as a hierarchical space