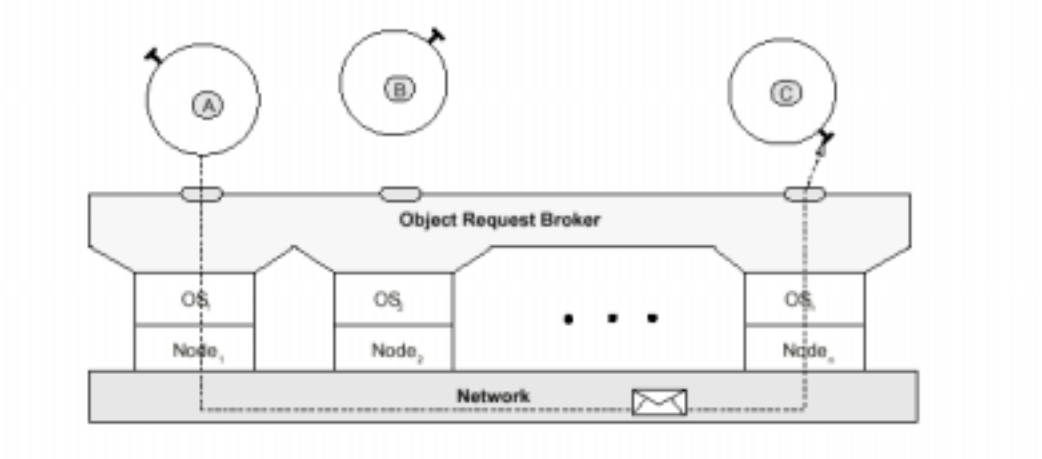
Arquitectura geral de funcionamento do MICO (CORBA)



Exemplo de uma aplicação em MICO

Classe Bank Account:

Interface e estado do objecto...

```
class Account
{
        long _current_balance;
public:
        Account ();
        void deposit (unsigned long amount);
        void withdraw (unsigned long amount);
        long balance ();
};
```

A implementação reflete o comportamento…

```
Account::Account ()
{
        _current_balance = 0;
}
void Account::deposit (unsigned long amount)
{
        _current_balance += amount;
}
void Account::withdraw (unsigned long amount)
{
        _current_balance -= amount;
}
long Account::balance ()
```

```
{
        return _current_balance;
}
```

Exemplo de um programa que usa a classe Account...

```
#include <iostream.h>

int main (int argc, char *argv[])
{
        Account acc;

        acc.deposit (700);
        acc.withdraw (250);
        cout << "balance is " << acc.balance() << endl;
        return 0;
}
```

Implementação da classe Account no MICO

Como os objectos CORBA podem ser implementados em diferentes linguagens de programação a especificação do interface do objecto e a sua implementação têm que ser separadas.

A implemetação é feita na linguagem de programação desejada e a interface na linguagem IDL (Interface Definition Language).

A especificação do interface Bank Account:

```
interface Account
{
        void deposit (in unsigned long amount);
        void withdraw (in unsigned long amount);
        long balance ();
};
```

O que se deve fazer de seguida é passar o IDL no compilador de IDL que irá gerar código na linguagem de programação seleccionada (neste caso C++).

O comando pode ser como o seguinte:

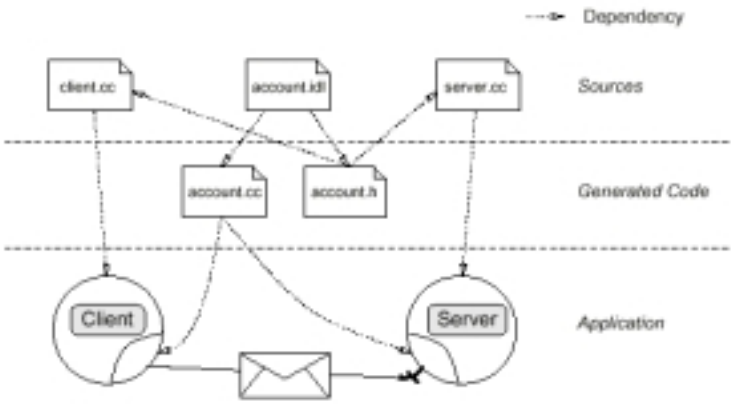
```
idl --boa --no-poa account.idl
```

O compilador de IDL gera dois ficheiros:

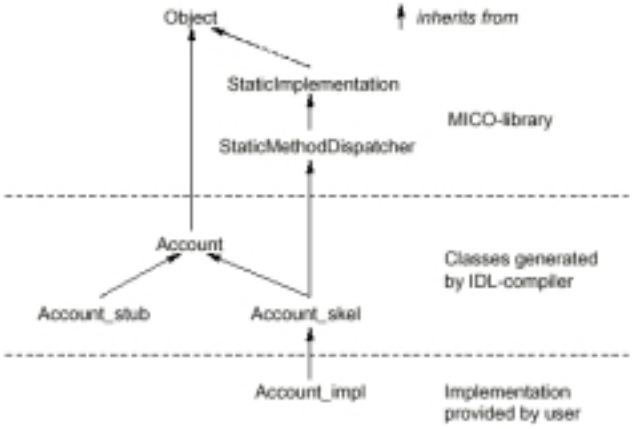
account.h – contem às declarações da classe base da implementação do objecto account e a classe stub que os clientes vão utilizar métodos nos objectos account remotos.

account.cc – contem a implemetação das classes anteriores e código de suporte. Para cada interface descrito no ficheiro IDL o compilador de IDL gera 3 classes C++.

A figura seguinte mostra o processo geral de criação de uma aplicação MICO



A figura seguinte mostra as classes geradas pelo compilador de IDL para o interface account.



Descrição das classes geradas pelo IDL

Classe **Account**

É a classe base a partir da qual as outras derivam.

Contem todas as definições que pertencem ao interface Account. Esta classe tambem define uma função virtual pura para cada operação pertencente ao interface.

```
// Code excerpt from account.h
class Account : virtual public CORBA::Object
{
        ...
public:
        ...
        virtual void deposit (CORBA::ULong amount) = 0;
        virtual void withdraw (CORBA::ULong amount) = 0;
        virtual CORBA::Long balance () = 0;
}
```

Classe **Account_skel**

É uma classe derivada de Account.

Esta classe acrescenta um *dispatcher* para as operações definidas na classe Account mas não define as funções virtuais puras da classe Account.

As classes Accout e Account_skel são classe abstratas na terminologia C++.

Para implementarmos o objecto account devemos criar uma subclasse de Account e impementar os métodos virtuais deposit(), withdraw() e balance().

Classe **Account_stub**

Esta classe tambem é derivada da classe Account.

Ao contrário da Account_skel esta classe implementa as funções virtuais. A implementação destas funções é gerada automaticamente pelo compilador de IDL e é responsável pelo *marshalling* dos parametros.

O aspecto desta classe é o seguinte:

```
// Code excerpt from account.h and account.cc
class Account;
typedef Account *Account_ptr;
class Account_stub : virtual public Account
{
...
public:
...
void deposit (CORBA::ULong amount)
{
        // Marshalling code for deposit
}
void withdraw (CORBA::ULong amount)
{
        // Marshalling code for withdraw
}
CORBA::Long balance ()
{
        // Marshalling code for balance
}
}
```

Desta forma a classe Account_stub é uma classe concreta (não abstrata) que pode ser instanciada. No entanto o programador nunca utiliza esta classe directamente. O acesso é feito através da classe Account, como veremos mais à frente.

Completar o exemplo

O que necessitamos de fazer agora é implementar a classe Account_skel (os seus métodos virtuais) e escrever um programa cliente.

```
1: #include "account.h"
2:
3: class Account_impl : virtual public Account_skel
4: {
5: private:
6: CORBA::Long _current_balance;
7:
8: public:
9: Account_impl()
10: {
11: _current_balance = 0;
12: };
13: void deposit( CORBA::ULong amount )
14: {
15: _current_balance += amount;
16: };
17: void withdraw( CORBA::ULong amount )
18: {
19: _current_balance -= amount;
20: };
21: CORBA::Long balance()
22: {
23: return _current_balance;
24: };
25: };
26:
27:
28: int main( int argc, char *argv[] )
29: {
30: // ORB initialization
31: CORBA::ORB_var orb = CORBA::ORB_init( argc, argv, "mico-local-orb" );
32: CORBA::BOA_var boa = orb->BOA_init( argc, argv, "mico-local-boa" );
33:
34: // server side
35: Account_impl* server = new Account_impl;
36: CORBA::String_var ref = orb->object_to_string( server );
37: cout << "Server reference: " << ref << endl;
38:
39: //--------------------------------------------------------------------
40:
41: // client side
42: CORBA::Object_var obj = orb->string_to_object( ref );
43: Account_var client = Account::_narrow( obj );
44:
45: client->deposit( 700 );
46: client->withdraw( 250 );
47: cout << "Balance is " << client->balance() << endl;
48:
49: // We don't need the server object any more. This code belongs
50: // to the server implementation
51: CORBA::release( server );
52: return 0;
53: }
```

Para compilar o programa, assumindo que o ficheiro se chama account_impl.cc temos que executar os seguintes comandos:

```
mico-c++ -I. -c account_impl.cc -o account_impl.o
mico-c++ -I. -c account.cc -o account.o
mico-ld -I. -o account account_impl.o account.o -lmico2.3.5
```