3
Binding entre Charm++ e Python

Este capítulo descreve um binding já existente entre Charm++ e Python. Este estudo foi importante para avaliarmos a flexibilidade do framework e a dificuldade de estendê-lo, e também para avaliarmos as deficiências presentes nessa integração.

3.1
Overview

A integração entre Charm++ e Python foi desenvolvida pela equipe de Charm++ com o objetivo de permitir a execução de trechos de scripts em uma aplicação durante a sua execução (05). O objetivo do binding não é que métodos do chare sejam implementados em Python, mas sim que um trecho de código possa ser carregado no chare e executado, e que esse código Python possa fazer chamadas a métodos do chare. Um exemplo de aplicação que utiliza esse binding é o charmdebug (35), um debugger para Charm++.

Para que o script seja executado no chare, primeiramente ele deve ser carregado a partir de uma aplicação externa ao programa Charm++. Essa carga do script é feita através do envio de uma mensagem contendo o código a ser executado e instruções de como lidar com o ambiente de execução Python.

Os dois componentes dessa integração são o cliente, que é uma aplicação externa ao programa Charm++, e o servidor, que é o processo Charm++ que executa os chares. A comunicação desse cliente com o servidor exige identificação por IP e porta. O servidor, nesta integração, é responsável por receber o código, executá-lo e manter o ambiente de execução de acordo com as instruções vindas do cliente.

Para que o objeto Charm++ (Chare, Array, Group, Node ou Nodegroup) possa executar código Python, é necessário que ele seja definido como compatível com essas requisições. Isso é feito através do atributo *python* colocado entre colchetes antes da declaração do objeto no arquivo de interface. Por exemplo:

```
mainchare [python] main {...}
array [1D] [python] myArray {...}
```

```
group [python] myGroup {...}
```

Além disso, quando um chare é criado, ele registra uma string identificadora que será usada por Charm++ para saber direcionar corretamente o script para o chare desejado, conforme exemplificado adiante.

Para registrar a string identificadora, o método *registerPython* deve ser chamado após a criação do chare. O método registerPython é disponibilizado pela classe do proxy, que é responsável pela instanciação do chare. Por exemplo, no trecho de código abaixo, um chare do tipo *MyChare* é criado e registrado para receber chamadas Python. O argumento de `registerPython` é a string identificadora do chare.

```
CProxy_MyChare localVar = CProxy_MyChare::ckNew();
localVar.registerPython("pyCode");
```

A API de integração oferece três tipos de requisições feitas pelo cliente para o servidor:

Execute chamada para executar um trecho de código. Ela contém o código a ser executado no servidor, juntamente com as instruções de como lidar com o ambiente;

Print requisição para que o servidor envie todas as strings que o script imprimiu até o momento;

Finished consulta ao servidor para saber se o script terminou ou se ainda está em execução.

Essas requisições são feitas síncrona ou assincronamente, dependendo dos parâmetros enviados na requisição. Charm++ disponibiliza uma API para comunicação e envio das requisições, que é explicada na sua documentação (36).

Quando uma mensagem do tipo *Execute* é recebida pelo objeto, antes da execução do código Python, o chare cria o ambiente de execução e exporta duas bibliotecas para Python. A primeira biblioteca é a *ck*, que contém funções básicas de Charm++. Os métodos presentes em ck são:

printstr Aceita uma string como parâmetro. Escreve na saída padrão do servidor.

printclient Aceita uma string como parâmetro. Irá enviar a string para o cliente quando ele fizer uma chamada do tipo Print.

mype Retorna um inteiro representando o processador corrente onde o código está executando.

numpes Retorna o número de processadores que a aplicação está usando.

myindex Retorna o índice do elemento dentro do array, se o objeto onde o código está sendo executado for do tipo *Array*, ou não retorna nada caso contrário.

A segunda biblioteca exportada para o ambiente Charm++ é a *charm*, que contém os métodos do chare que foram exportados para Python. Para um entry method ser acessível de Python, ele deve ser declarado com o modificador *python* na sua interface, da mesma forma que a declaração do objeto, e deve retornar `void` e receber como parâmetro um inteiro, que será uma referência ao ambiente Python onde ele está sendo executado. Essa referência será importante para o recebimento de parâmetros e para o retorno de valores pela função. Um exemplo de entry method é:

```
entry [python] void highMethod(int handle);
```

O método exportado acima será acessível de dentro de python através da seguinte chamada:

```
result = charm.highMethod(var1, var2, var3)
```

sendo que a função pode receber qualquer tipo de parâmetro, inclusive tipos complexos como tuplas ou dicionários. Ele também pode retornar um objeto tão complexo quanto desejado. O recebimento de parâmetros e o retorno de dados é feito através de chamadas da API de Charm++, usando a referência recebida para indicar o estado de execução.

3.2 Avaliação do binding com Python

O estudo deste binding foi importante por oferecer uma oportunidade de um estudo mais aprofundado do funcionamento interno de Charm++, principalmente no referente à API de envio de mensagens, ao parser do arquivo de interface e à integração com novas funcionalidades. Através deste estudo, pudemos decidir tomar um rumo diferente no desenvolvimento do binding com Lua, descrito no próximo capítulo.

Como explicado anteriormente, o binding com Python não visava permitir que chares fossem implementados na linguagem de script, mas apenas oferecer uma ferramenta para possibilitar uma maior interação do usuário com o chare em tempo de execução, através da carga de código dinâmico. Assim, o modelo é bastante restrito para possibilitar a integração da linguagem de script com a programação paralela. Algumas das restrições encontradas neste modelo de desenvolvimento são as seguintes:

– Execução dependente de um ator externo para inserir o código de script no objeto Charm++.

  Toda classe que é capaz de receber e executar scripts deve estender uma classe chamada PythonObject. Essa classe faz parte da implementação do binding e é responsável por toda a parte de tratamento de mensagens e de manutenção do interpretador Python.

  Como todo o controle e administração de ambientes são feitos pelo código da integração e o programador não tem a possibilidade de alterar o seu comportamento, não existe uma maneira do usuário, ao criar um chare, carregar e executar um script. Para isso é necessária a presença de um programa externo que se conecte com o chare e envie para ele uma mensagem com o script.

– Não é possível criar ou chamar um método de um objeto que não seja o responsável pela execução do script.

  Como foi descrito na seção anterior, a interação entre o script e o chare é feita através da exportação de algumas funções através da biblioteca charm. Ao criar a biblioteca charm, apenas os métodos do chare que têm o atributo [python] são exportados para o script. Assim, para que um método remoto seja chamado ou um chare criado a partir do script, é necessário que essa implementação exista em C++ e seja exportada.

– Não é possível para um chare chamar uma função do script carregado na instância do interpretador presente no próprio chare ou em um outro.

  Como toda a parte de controle dos estados e execução dos script é feita dentro do código do binding, e o chare não tem acesso ao interpretador Python, não é possível para um chare chamar uma função ou executar um script. Assim sendo, a única maneira de um chare chamar um método do script é ele enviar uma mensagem, sendo que o envio dessas mensagens é trabalhoso e verboso, fugindo aos padrões e à simplicidade das chamadas assíncronas transparentes de Charm++.

– Todas as chamadas feitas pelo script são síncronas.

  Apesar de Charm++ oferecer a possibilidade do usuário criar e executar threads, a sua implementação padrão é de threads não-preemptivas. Assim, enquanto um script está sendo executado, apesar da sua execução acontecer em uma thread separada, ela é bloqueante em relação ao resto do chare. Como todas as chamadas do script são locais, ao chamar um método do chare, o script interrompe sua execução e permanece em estado bloqueado até que o código C++ explicitamente retorne o controle da execução para o script.

Apesar da limitação das chamadas do script, durante a execução de um método do chare chamado pela biblioteca `charm`, existe a possibilidade de que o chare chame métodos remotos assíncronamente, e receba então chamadas pendentes antes de retornar o controle da execução para o script.

Essas restrições tornam o modelo utilizado nessa integração pouco abrangente. Nos estudos subseqüentes será explorada uma outra forma de tornar o uso de uma linguagem de script dentro de Charm++ mais integrada com o sistema e assim acabar com a necessidade de código C++ para a inicialização do chare e para chamada de métodos de outros objetos de Charm++. Nesta forma a implementação dos objetos pode ser toda feita na linguagem de script.