Programowanie obiektowe

Symulacje

Paweł Daniluk

Wydział Fizyki

Jesień 2016

Czas

Paradygmat obiektowy doskonale nadaje się do opisywania struktury i stanu dużych systemów.

Niestety takie systemy rzadko kiedy pozostają niezmienne w czasie.

Czas

Paradygmat obiektowy doskonale nadaje się do opisywania struktury i stanu dużych systemów.

Niestety takie systemy rzadko kiedy pozostają niezmienne w czasie.

Atrybuty

Przechowują aktualny stan.

Metody

Odpowiadają za aktualizację stanu.

Czas

Paradygmat obiektowy doskonale nadaje się do opisywania struktury i stanu dużych systemów.

Niestety takie systemy rzadko kiedy pozostają niezmienne w czasie.

Atrybuty

Przechowują aktualny stan.

Metody

Odpowiadają za aktualizację stanu.

Jak reprezentować czas?

Czas

Paradygmat obiektowy doskonale nadaje się do opisywania struktury i stanu dużych systemów.

Niestety takie systemy rzadko kiedy pozostają niezmienne w czasie.

Atrybuty

Przechowują aktualny stan.

Metody

Odpowiadają za aktualizację stanu.

Jak reprezentować czas?

Skąd wiadomo kiedy jakie metody mają być wywoływane?

Czas – ciągły, czy dyskretny?

Zazwyczaj jest możliwe określenie stanu układu w dowolnej chwili, ale nie zawsze jest to niezbędne.

Niektóre stany można bez straty dokładności ze sobą utożsamić.

Przykłady

- Pociąg zajmuje odcinek toru, nie trzeba wiedzieć, gdzie dokładnie jest.
- Podwozie samolotu jest otwarte, zamknięte lub w trakcie otwierania/zamykania.

Jeżeli pewne stany są jednakowe, istotne są jedynie momenty przechodzenia pomiędzy nimi.

Czas ciągły (stały krok czasowy)

Symulacja oblicza stan układu w kolejnych krokach czasowych $t_1, t_2, t_3, \ldots$ ($\Delta t = t_{i+1} - t_i$). Kolejny stan obliczany jest na podstawie poprzedniego (lub kilku poprzednich).

Jeżeli układ zawiera wiele niezależnych obiektów, konieczne jest zaimplementowanie oddziaływań pomiędzy nimi. W przypadku modeli opartych o równania różniczkowe zwyczajne (ODE) algorytm całkowania można zawrzeć w osobnej klasie.

Konieczne może być utrzymywanie nienaruszonego stanu poprzedniego S_{t_i}, dopóki $S_{t_{i+1}}$ nie zostanie obliczony w całości.

Przykłady

Proces Markowa

Ciąg zmiennych losowych $X_1, X_2, X_3, \ldots$

$$P(X_{n+1} = j | X_n = i) = p_{i,j}$$

Równania Hamiltona

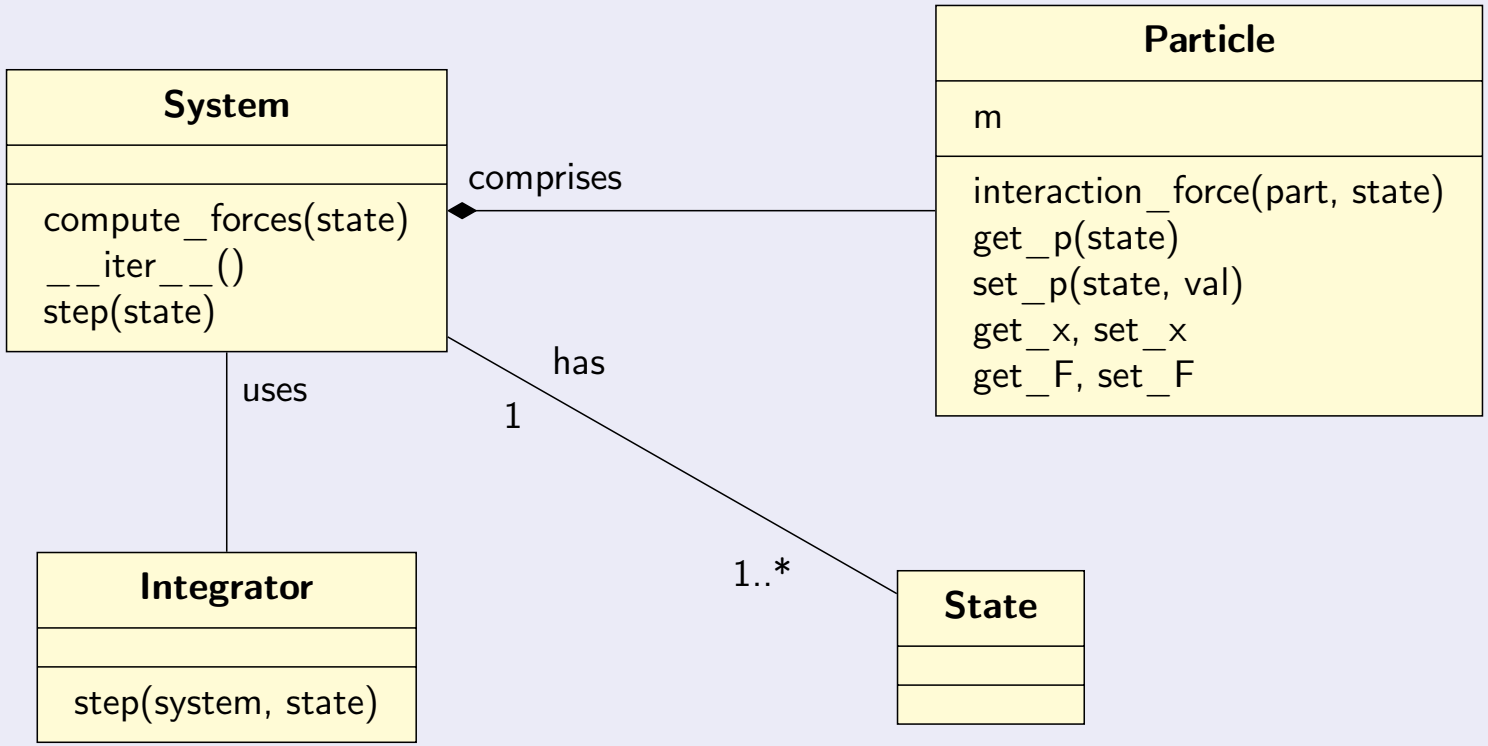
$$\frac{p^i}{dt} = -\frac{\partial H}{\partial x_i}, \quad \frac{x^i}{dt} = \frac{\partial H}{\partial p_i}$$

Całkowanie leapfrog

$$x_{i+1} = x_i + \frac{p_i}{m_i}\Delta t + \frac{1}{2}\frac{F_i}{m_i}\Delta t^2$$

$$v_{i+1} = v_i + \frac{1}{2}\frac{F_i + F_{i+1}}{m_i}\Delta t$$

Przykładowy projekt



Pętla główna

```
state=starting_state

for i in range(n_steps):
    state=system.step(state)
    statistics.add(state)
```

Zastosowania

- symulacje układów fizycznych
- w tym dynamiki molekularne
- gry komputerowe
- symulacje finansowe

Czas dyskretny

Zdarzenia

Zmiana stanu systemu następuje na skutek zajścia zdarzenia.
Zdarzenie może powodować wystąpienie kolejnych zdarzeń w przyszłości.

Czas

- Każde zdarzenie zachodzi w konkretnym czasie.
- Czas wystąpienia zdarzenia można obliczyć na podstawie zdarzeń je poprzedzających.
- Zdarzenia zachodzą nieuchronnie.

Zdarzenia

Przykłady

- Pociąg wyjeżdża ze stacji. Wiadomo, że za czas potrzebny do przejechania toru dojedzie do następnej.
- Kulka leci w kierunku ściany. Kiedy doleci, odbije się w przeciwnym kierunku.
- Wykład rozpoczął się. Skończy się za 1.5h.

Obsługę zdarzeń implementuje się jako metody w klasach obiektów, których dotyczą.

Zdarzenie jest obiektem, który zna:

- czas zajścia
- metodę (funkcję), która ma zostać wykonana
- argumenty do przekazania

Przypomnienie

Kolejka

Struktura danych pozwalająca na wstawianie elementów (operacja *Enqueue*) i pobieranie najwcześniej wstawionego elementu (operacja *Dequeue*) (FIFO – *First In First Out*)

Kolejka priorytetowa

Kolejka, w której *Dequeue* powoduje pobranie elementu o najwyższym priorytecie niezależnie od tego, kiedy był wstawiony.

Kolejka zdarzeń

Zdarzenia, które mają nastąpić, są wstawiane do kolejki.

Każdorazowo pobierane jest zdarzenie najwcześniejsze.

W szczególności możliwe jest wstawienie nowego zdarzenia, które ma nastąpić przed zdarzeniami już zarejestrowanymi.

Jest to naturalne zastosowanie kolejki priorytetowej.

Odpowiedzialność kaskadowa

1. Zdarzenie danego rodzaju może być obsłużone przez więcej niż jeden obiekt.
2. Obiekty niezależnie podejmują decyzję, czy obsłużą zdarzenie.
3. Nieobsłużone zdarzenie przechodzi do następnego obiektu.

Przykłady zastosowań

1. graficzne interfejsy użytkownika
2. filtrowanie
3. symulacje procedur zarządzania

Implementacja w Pythonie

Zdarzenia

Krotka, albo obiekt mający następujące atrybuty:

- czas
- metoda (funkcja)
- lista argumentów

Przykład

```
e1=(23, train1.leave, [track, speed_limit])

e1[1](*e1[2])    # train1.leave(track, speed_limit)
```

Implementacja w Pythonie

Kolejka priorytetowa

Listy w Pythonie mają metodę sort() , którą można wywoływać po każdym wstawieniu do listy. Sortowanie może być wykonywane według dowolnego klucza. Porządek sortowania ustala się przez opcjonalne argumenty key i reverse .

Przykłady

```
>>> sorted("This is a test string from Andrew".split(), key=str.lower)
['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']

>>> student_tuples = [
        ('john', 'A', 15),
        ('jane', 'B', 12),
        ('dave', 'B', 10),
]
>>> sorted(student_tuples, key=lambda student: student[2])   # sort by age
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

Przykłady c.d.

```
>>> class Student:
        def __init__(self, name, grade, age):
                self.name = name
                self.grade = grade
                self.age = age
        def __repr__(self):
                return repr((self.name, self.grade, self.age))
>>> student_objects = [
        Student('john', 'A', 15),
        Student('jane', 'B', 12),
        Student('dave', 'B', 10),
]
>>> sorted(student_objects, key=lambda student: student.age)   # sort by age
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

Przykłady c.d.

```
>>> from operator import itemgetter, attrgetter

>>> sorted(student_tuples, key=itemgetter(2))
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

>>> sorted(student_objects, key=attrgetter('age'))
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

>>> sorted(student_tuples, key=itemgetter(1,2))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]

>>> sorted(student_objects, key=attrgetter('grade', 'age'))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]
```

Implementacja w Pythonie

Kolejka priorytetowa

Alternatywnie można zastosować moduł heapq .

Przykłady

```
>>> h = []
>>> heapq.heappush(h, (5, 'write code'))
>>> heapq.heappush(h, (7, 'release product'))
>>> heapq.heappush(h, (1, 'write spec'))
>>> heapq.heappush(h, (3, 'create tests'))
>>> heapq.heappop(h)
(1, 'write spec')
```

Czas rzeczywisty

Symulacje czasu rzeczywistego

Upływ czasu symulowanego układu odpowiada rzeczywistemu (potencjalnie przeskalowanemu) czasowi modelowanych procesów. Jest to istotne np. przy wizualizacji.

Zdarzenia pochodzące z zewnątrz

- Interakcja z użytkownikiem
- Sygnały z urządzeń pomiarowych

Systemy czasu rzeczywistego

Przetwarzanie sygnałów zewnętrznych z zachowaniem ścisłych limitów czasowych. Np. autopilot, ABS, sterowanie procesem produkcyjnym.

Zdarzenia jeszcze raz

Formalizmu związanego ze zdarzeniami można również używać do rejestrowania przeszłości poprzez zapamiętywanie zmian przyrostowych. Dzięki temu można odtworzyć stan z przeszłości bez konieczności przechowywania wszystkich minionych stanów w całości.

Przykłady

- operacje na dokumentach
- transakcje

Systemy wieloagentowe

Agent

Autonomiczna jednostka (program, obiekt) zdolna do:

- postrzegania środowiska
- podejmowania decyzji
- wchodzenia w interakcję ze środowiskiem

Dodatkowo agenty mogą:

- komunikować się
- uczyć

Symulacje

Istnieją środowiska dedykowane do implementacji systemów wieloagentowych zapewniające niezależną równoległą pracę agentów. Takie środowisko można również symulować.