Dynamisk programmering med effektiv kode

Traditionelt har dynamiske programmeringssprog tilbudt hurtig programudvikling, men langsom programafvikling. I det EU-støttede Open Source projekt PyPy arbejdes der på at ændre denne sammenhæng. På sigt er det målet at bane vejen for brug af dynamiske sprog til indlejret software-udvikling og DSP-udvikling.

Af Anders Lehmann,
DFKI (Deutsche Forschungzentrum für Künstliche Intelligens)

Python (www.python.org) er et af de dynamiske sprog fra Open Source bevægelsen, hvor grundstenene kaldes LAMP (Linux, Apache, MySQL og Python (eller PHP, Perl)). Python er et objektorienteret sprog, der tilbyder Garbage Collection, dynamisk typeerklæring og reflektion – og som samtidig udmærker sig ved en let læselig syntaks (eksekverbar pseudokode). Python er i øvrigt beskrevet mere detaljeret i en artikel i Elektronik & Data nr 10-2006.

Der findes på nuværende tidspunkt fire implementeringer af Python:

- CPython - der er den officielle version implementeret i C,
- IronPython - implementeret i C# til .NET (for en stor del financieret af Microsoft),
- Jython - implementeret i Java,
- samt PyPy - implementeret i Python, der er baseret på resultaterne fra et forsknings- og udviklingsprojekt, der er delvist finansieret af EU.

Formålet med EU-projektet, der fokuserer på PyPy, er at arbejde på at implementere en mere fleksibel Python-fortolker ved at skrive fortolkeren i Python. Projektet startede i december måned i 2004 og afsluttes i marts 2007. Der indgår syv partnere i projektet, og de kommer fra henholdsvis Sverige (Strakt AB og ChangeMaker), Tyskland (TismerSoft, Merlinux, Henrich Heine universitet og DFKI) samt Frankrig (Logilab). Det samlede budget er på ca. 2 millioner Euro.

Anders Lehmann er Cand. Polyt. fra DTU. Efter 15 år i udviklingsafdelinger hos Bang & Olufsen arbejder han nu for Deutsche Forschungzentrum für Künstliche Intelligens (DFKI) i Saarbrücken, Tyskland, hvor han beskæftiger sig med PyPy-projektet. Han deltog som privatperson i udarbejdelsen af PyPy-ansøgningen til EU i 2003. Fra den 1. december i år arbejder han igen i Danmark.

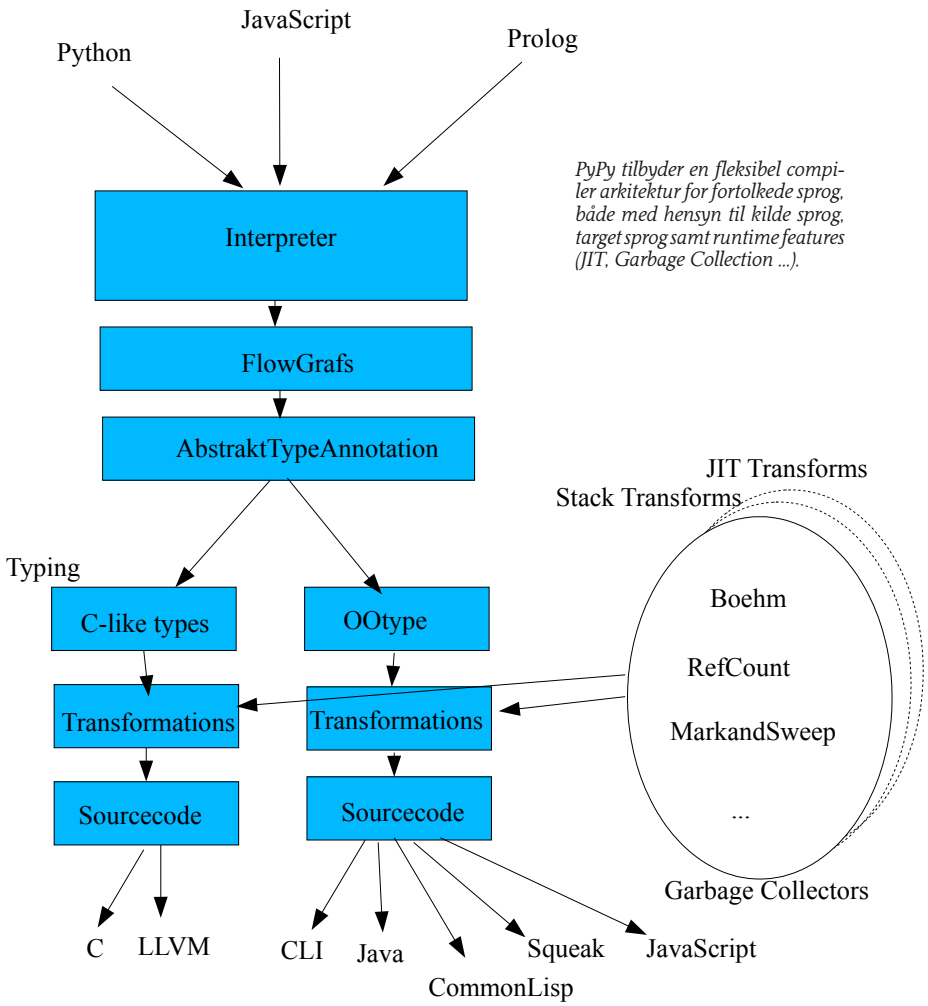
PyPy – Python skrevet i Python

I det følgende beskrives, hvorfor det kan være en god idé at implemetere Python i Python, og hvordan man kan få effektive programmer ud af en proces, som på overfladen ligner en sjov, men ikke særlig effektiv idé.

Den grundlæggende idé blev født i 2003, hvor tre Python-udviklere (Armin Rigo, Schweiz; Christian Tismer, Tyskland og Holger Krekel, Tyskland) satte hinanden stævne i Hildesheim med det mål at ændre på sammenhængen mellem runtime-effektivitet og programmøreffektivitet. De tre udviklere foreslog en løsning, der går ud på at implementere en Python-fortolker i Python på en sådan måde, at denne fortolker kan oversættes til et C-program. Denne idé blev grundlaget for etableringen af det EU-støttede PyPy-projekt.

Der har tidligere været gjort flere forsøg på at oversætte Python-programmer til C, men de tre udviklere valgte at gå helt nye veje for at nå målet. I stedet for at forsøge at oversætte Python kildekode til C kildekode (hvilket er umuligt på grund af den dynamiske type tildeling i Python), valgte de at lave oversættelsen til C ved hjælp af Pythons mulighed for reflektion.

Det vil sige, at istedet for at oversætte kildekode, oversættes 'levende' objekter til



PyPy tilbyder en fleksibel compiler arkitektur for fortolkede sprog, både med hensyn til kilde sprog, target sprog samt runtime features (JIT, Garbage Collection ...).

C-kode. Det Python program, der skal oversættes, bliver først eksekveret i en særlig Python-fortolker, der initialiserer objekter og kode (kode er også et objekt i Python). Disse initialiserede datastrukturer kan efterfølgende analyseres og oversættes af PyPy's oversætter framework.

Udover muligheden for en mere fleksibel fortolker, giver denne måde at analysere et program på også mulighed for forskellige former for optimeringer (som feks. at benytte "int"'s, hvis man kan bevise, at det er tilstrækkeligt). Disse optimeringer kan foregå automatisk og er uafhængige af fortolkerens opbygning og implementeringen af Pythons typer.

Den grundlæggende idé er at opbygge to dele: En Python-fortolker skrevet i Python samt et oversætterprogram.

Python-fortolkeren er delt i to dele: en parser + bytecode fortolker og et Objectspace. Bytecode fortolkeren kalder funktioner i et Objectspace for at udføre deres funktion. Alle typer er defineret i Objectspace, men da der er et entydigt interface, kan man bytte objectspaces ud efter behov.

Fortolkeren er derfor uden viden om den konkrete implementering af de objekter, som fortolkeren manipulerer. Hvilket giver mulighed for at definere nye features uden at ændre fortolkeren. Det har været brugt til at lave Objectspace med logiske variable, Objectspace med 'doven' semantik (variable bliver først tildelt, når der er brug for det), FlowObjectSpace (se senere) og standard Python selvfølgelig.

I Standard Object Space er alle Python's standard typer implementeret i et Python lignende sprog kaldet RPython.

Oversættelse fra Python objekter til C/CLI/Java

Et af de første Objectspaces, der blev udviklet, var FlowObjectSpace, der i stedet for at arbejde med konkrete typer, transformerer

...FORTSÆTTES NÆSTE SIDE

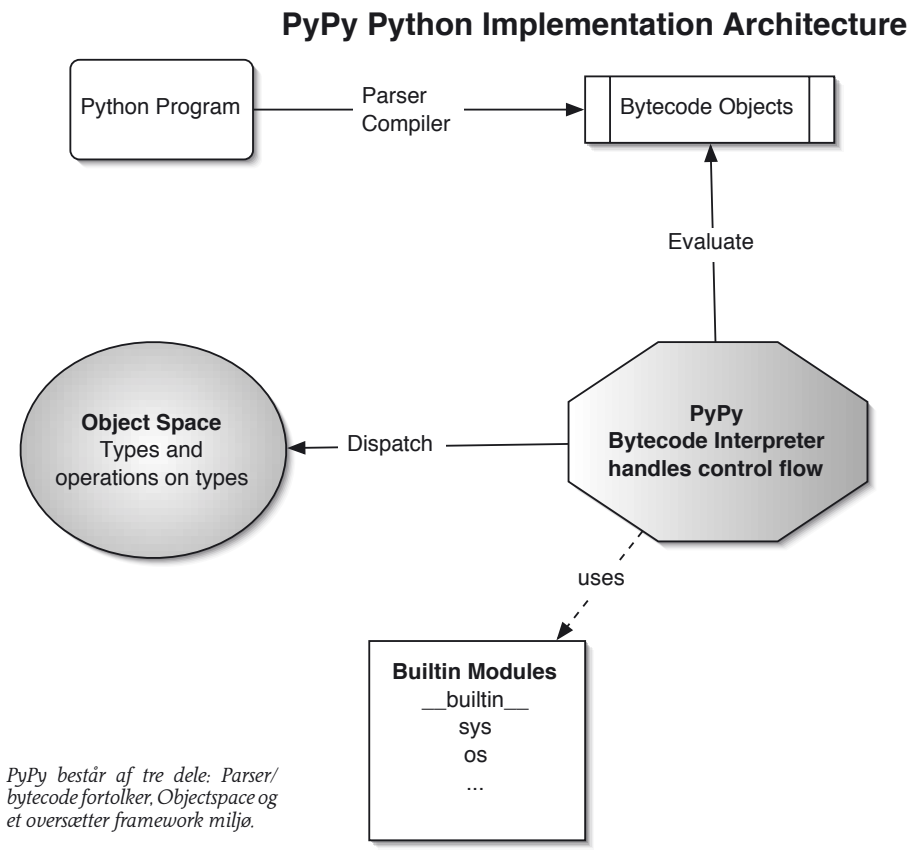
FORTSAT FRA SIDE 21:

Python bytecodes til en kontrol flow graf ved at 'optage' de funktioner bytecode, som fortolkeren kalder i Objectspace.

Disse grafer kan derefter dekoreres/annoteres med typeinformation, der findes som en global søgning for hele programmet. Det vil sige ud fra en given input type forsøger Annoterings-programmet at finde resulterende (abstrakte) typer for alle variable i programmet. For at denne proces kan lykkes kræves det, at programmet er rimelig statisk (alle variable skal have kompatible typer – heltal af forskellige længder (int og long) er f.eks. kompatible, mens heltal og tekststrenge ikke er kompatible). Dette semistatiske sprog kaldes RPython. RPython er en delmængde af Python. Der er ikke en formel definition af RPython, udover at 'det er, hvad annotationsprocessen kan forstå'.

Hvis Python-programmet, der undersøges, er tilpas statisk i typetildelingerne, vil alle variable i de annoterede grafer have en entydig (abstrakt) type. I en process kaldet 'rtyping' kan disse abstrakte datatyper konverteres til konkrete lavniveautyper. Der er to måder at gøre dette på: lltype, der er beregnet på C-lignende sprog, og ootype, der er beregnet på højniveausprog som f.eks. Java, .Net og Javascript.

Når de entydige abstrakte typer er blevet konverteret til konkrete typer, er resultatet en række flowgrafer annoteret med konkrete typer. Disse grafer kan nu manipuleres, og nye muligheder kan flettes ind. For eksempel kan man vente til dette punkt i oversættelsen med at tage stilling til, hvilken form for Garbage Collection, man ønsker sig, eller man kan indsætte stack manipulerende operationer strategiske steder i flowgraferne (for at muliggøre dybere rekursion eller continuations), eller manipulere graferne, så man kan generere optimeret kode (Just in Time compiler).



PyPy består af tre dele: Parser/ bytecode fortolker, Objectspace og et oversætter framework miljø.

Som det sidste led i oversættelseskæden er der kodegenerering, hvor de transformerede flowgrafer bliver omsat til kildekode. Dette foregår i en såkaldt backend. Der findes backends til C, LLVM (Low Level Virtual Machine), Javascript, Squeak, Common Lisp, CLI (Common Language Infrastructure – .NET/Mono), Python og en Java backend er undervejs.

Det største RPython-program, PyPy er lykkedes med at oversætte, er Python-fortolkeren skrevet i Python. Dette program fylder ca. 90.000 linier RPython-kode.

Effektive programmører og effektive programmer

I øjeblikket er PyPy's fortolker af Python omkring tre gange langsommere end den officielle håndkodede version. Dette skal ses i forhold til, at PyPy var 1000 gange langsommere for ét år siden. I løbet af efteråret vil der blive udviklet en Just In Time compiler til PyPy, hvilket vil bringe PyPy's ydelse tæt på eller over end CPython's.

PyPy kan dog allerede i dag benyttes til at skabe ganske effektive programmer, hvis

man ønsker at lave enkeltstående (uden fortolker) programmer. I disse tilfælde kan ydelsen måle sig med optimeret Java-kode. Der er allerede mindst ét firma, som udnytter dette til at forbedre deres programmer (et amerikansk investeringsfirma).

Det er også muligt at skrive Python-biblioteker (extension modules), som kan oversættes til C og derefter importeres i CPython (eller PyPy), hvorved man kan forøge performance af programmet. Før har man skullet skrive extension modules i C, men nu kan man gøre det i (R)Python.

Frem til EU-projektet slutter i marts 2007, vil fokus være på JIT compileren. Men der vil også blive gjort mere for at forbedre de enkelte backends (primært CLI) og arbejdet på at forbedre brugerinterfacet, så det bliver mere enkelt at vælge mellem alle de muligheder, som PyPy tilbyder.

NÅR EU- projektet er færdigt, fortsætter projektet som et Open Source projekt, men formodentlig med mindre fart.

Hvad med fremtiden?

Med PyPy er det muligt at få det bedste af to verdener: Man kan have fordelene med et dynamisk sprog (høj produktivitet, nemmere vedligeholdelse, osv) og samtidig generere effektive programmer.

PyPy vil øge det antal platforme, hvor Python kan bruges. Med PyPy vil det for eksempel være muligt at benytte et højniveau objektorienteret sprog til udvikling på embeddede platforme.

Hvis man ser lidt længere frem i tiden, vil der blive tilbudt nye frontends (Python og Prolog findes i dag, men Ruby er en oplagt kandidat). Der vil også blive introduceret nye backends (Java er under udarbejdelse, Verilog kunne være interessant), samt nye objectspaces: Security og persistence.

Fra at være et *legetøjssprog* har PyPy vist, at man kan bruge Python til seriøs forskning, samt at resultaterne kan bringes til anvendelse i industrien.

Yderligere oplysninger kan hentes på:
http://codespeak.net/pypy
http://pypy.org
eller ved at kontakte Anders Lehmann:
serendipity-soft@get2net.dk

Opdateret RoHS-katalog

RoHS er stadig et emne, der fylder rigtig meget i mange elektronik-virksomheder, og derfor vil det nye RoHS-katalog fra Farnell InOne sikkert blive hilst velkommen, når det lander hos firmaets kunder her i starten af november måned.

Sigtet med det nye katalog (som i tråd med ånden i RoHS-direktivet trykkes på miljøvenligt papir) er at give designingeniørerne detaljerede informationer om det voksende udbud af RoHS-kompatible produkter. Kataloget indeholder således spændende nyheder om RoHS-relaterede teknologier inden for områder som transistorer, loddeudstyr, kondensatorer, strømforsyninger samt diskrete halvledere.

Kataloget indeholder også vigtige opdateringer omkring direktivet, såsom information om undtagelser, herunder de nyeste kategorier 8 og 9, der dækker over medicinsk udstyr, og disse informationer kan være med til at gøre det lettere for designingeniørerne at arbejde med kravene i direktivet.

– Vores nye katalog er som altid blevet sammensat med henblik på designingeniørens behov. Resultatet er et brugervenligt værktøj, som giver et godt overblik over de nye RoHS-kompatible produkter, uden at der er spildt plads på unødig information, siger Gary Nevison, der er ansvarlig for 'product market strategy' hos Farnell InOne i Europa og Asien-Pacific.

Han påpeger, at det selvfølgelig også er muligt on-line – og i løbet at få sekunder – at identificere og bestille komponenter, der kan bruges i produkter, som er omfattet af RoHS-direktivet.