Context-Oriented Programming: Beyond Layers

Martin v. Löwis

Marcus Denker

Oscar Nierstrasz

Agenda

- Context-dependent Behavior
- Method Layers (PyContext example)
- Implicit Layer Activation
- Case Studies
- Context Variables
- Implementation Notes

Context Dependencies

- Programs need to be aware of the context in which they operate
  - what is the state of the environment
  - what user is accessing the system
  - what mode is the program to be executed in
- Example: current user
  - different roles may cause completely different code to be executed (e.g. administrator may be offered different facilities)
    - can be modeled through method layers
  - different users acting in the same role access different data
    - modeling through method layers is not adequate
- Example: dependency of program output on output device
  - In OO system, rendering algorithm spreads over methods of different classes

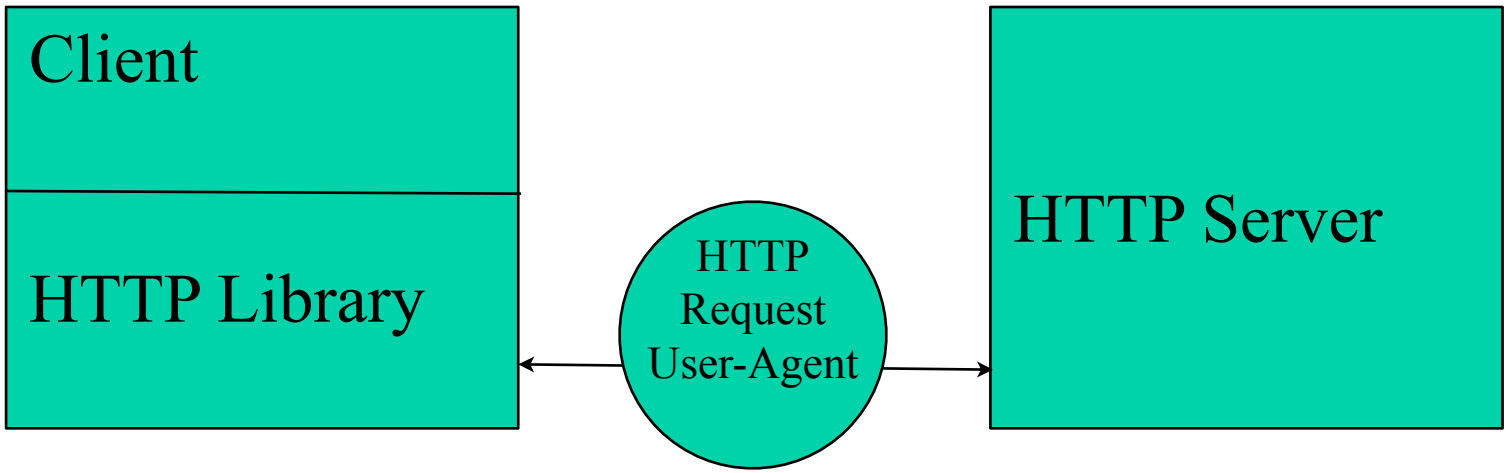
Layers

Method Layers

- addition of a few concepts to object-oriented programming
- layer: group of classes and methods to be used together in dynamic scope of execution
- layered class: collection of partial definitions of a class, for different layers
  - layered methods: definitions of methods for specific layers
  - layered slots: definition of instance attributes for specific layers
- (explicit) layer activation: specification of code block that runs in the context of a layer
  - inside the block, each sent message selects the method defined for that layer
  - nested activation: need to consider multiple layers in sequence

Example: User-Agent Header

- Web browsers sent User-Agent header to indicate client software (e.g. MSIE, Firefox, Safari, etc.)
  - "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)"
- Web servers sometimes have different behavior depending on User-Agent header
- Problem: automated web client might need to claim to operate as a specific user agent



Example: User-Agent Header (2)

- Assumption: client consists of multiple modules, each using different software layers to access underlying HTTP libraries
  - explicitly specifying User-Agent to the library is not possible
- Assumption: client is multi-threaded; different threads may need to operate in different contexts
  - setting User-Agent as a global variable is not possible
- HTTP libraries in Python:
  - httplib: direct access to protocol
  - urllib: unifying library for http, ftp, ...

PyContext: Using Method Layers

- with statement: automatic enter/leave semantics

```
from useragent import HTTPUserAgent
with HTTPUserAgent("WebCOP"):
    print "Using useragent layer"
    get1()
    get2()
```

- Importing useragent module automatically defines the layer and the layered methods
- Disabling layers

```
from layers import Disabled
with Disabled(Layer):
    code
```

Defining Layers

- Inherit from class Layer
  - Class can have arbitrary methods, instance variables, etc

```
class HTTPUserAgent(layers.Layer):
    def __init__(self, agent):
        self.agent = agent
```

Defining Layered Methods

- Inherit a class (with arbitrary name) from both the layer and the class to augment
- Define methods with the same name as the original methods
  - Each method has automatic second parameter "context" (after self, before explicit method parameters)
- Decorate each method with either before, after, or instead
- Context: Object indicating the layer activation
  - .layer: reference to the layer object
  - .result: result of the original method (for after-methods)
  - .proceed: callable object denoting the continuation to the original method (or the next layer)

```
class HTTPConnection(HTTPUserAgent, httplib.HTTPConnection):

    # Always add a User-Agent header
    @before
    def endheaders(self, context):
        with layers.Disabled(HTTPUserAgent):
            self.putheader("User-Agent", context.layer.agent)

    # suppress other User-Agent headers added
    @instead
    def putheader(self, context, header, value):
        if header.lower() == 'user-agent':
            return
        return context.proceed(header, value)
```

Implicit Activation

Implicit Activation

- Problem: explicit activation still needs to identify point in code where context might change or where context will be relevant
- Objective: allow addition of layers which get activated "automatically"
  - specifically, when a condition on the environment changes
- Design issues:
  - how can the system tell whether a condition becomes true?
    - each layer implements an **active** method
  - when should the active method be evaluated?
    - each time a layered method is executed whose meaning depends on whether the layer is active or not

Case Studies

Objective

- We tried to evaluate what aspects of context are common in application programs today
- Issue: how can we find code that depends on context?
  - Starting point: assume caller and callee are designed to run within the same context
  - Starting point: look for traditional examples of context
- Selected case studies: large Python applications/libraries
  - Django: web application framework
  - Roundup: bug tracker
  - SCons: automated build tool

Results

- Web applications (Django, Roundup) need to support concept of "current" request, including authenticated user, session data, target URL, etc.
- SCons keeps track of context in "environment": information about the current build goal
- These things were often referred to as "context", or showed up as pass-through parameters in methods
  - Searching for "context" revealed further context-dependent code fragments
  - Searching for pass-through parameters not easily possible with pure text searching; subject for further study
- Context information often not used to select different pieces of code, but merely as lookup keys in associative arrays

Dynamic Variables

Motivation

- case study results lead to identification of additional concept for context-oriented programming: Dynamic Variables
- in order to avoid pass-through parameters, a variable holding context should be set in a caller, and then read in a nested callee
  - similar to dynamic variables in functional languages
  - requires careful usage, to avoid old problems with dynamic variables (unintentional access due to naming collisions)
    - require explicit read and write operations

Dynamic Variables in PyContext

- Example: current HTTP session

1. Declare dynamic variable
   ```
   _session = Variable()
   ```
2. Obtain current variable (e.g. through helper function)
   ```
   def current_session():
       return _session.get()
   ```
3. Setup variable from dynamically-read context
   ```
   def process_request(request):
       session = lookup_session(request)
       with _session.set(session):
           dispatch_request(request)
   ```

Implementation Notes

- Method layers:
  - Dynamically replace methods with wrappers
- Dynamic variables:
  1. perform stack walk: O(stack-depth)
  2. use thread-local storage: O(1)

Summary

- current applications (in particular webapps) show high degree of context-awareness
- context-dependency is not made explicit in the code
- layers are a first step to making context explicit
- rehabilitation of dynamic variables necessary to support common cases of context