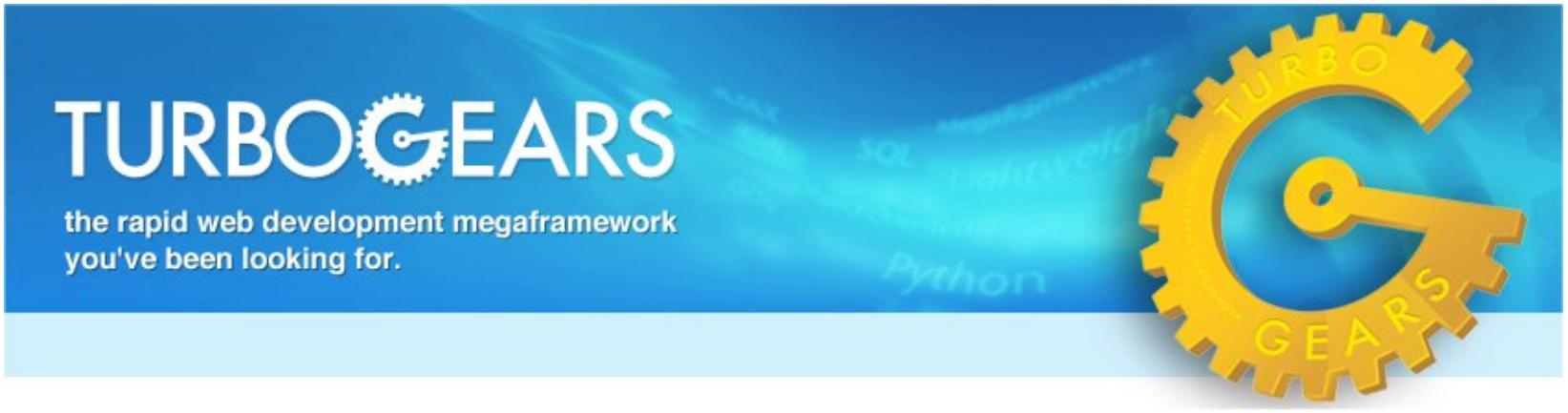
Generic functions in



Simon Belak
simon.belak@hruska.si

EuroPython 2006

What those not attending will miss

- Overview of generic functions
- Practical examples in TurboGears
- Generic functions from a framework designer’s point of view

A strange beast by the name of generic function

Generalisation of message passing

- Problem: a set of abstractly the same, practically different functions

- Solution: choose based on *context* (e.g. arguments)

All for one, **not** one for all

- Single dispatch

  Python, Java
- Multiple dispatch
  - Type based

    CLOS, PyPy, Python 3000?
  - Identity based

    CLOS
  - Predicate based

    Cecil, Mathematica, RuleDispatch

Pick me, pick me!

- Relation *more specific*
- Arbitrary dispatch strategies

Combine and Conquer

- Where is my super?
  - method chaining (dispatch strategy applies)
- How is this better than sliced bread?
  - before (all applicable)
  - after (dispatch strategy applies)
  - around (all applicable)

Something is rotten in the package of RuleDispatch

- Limited introspection
- Magic
- Scarce documentation
- Cryptic errors
- Once a generic function, always a generic function

Call now and get all these wonderful buzzwords free of charge

- DRY
- Ad-hoc event/plug-in system
- End-user framework extensions
- Data driven

Shifting to TurboGears

Use-case #1: JSON for everyone

- Data driven
- End-user framework extension

```
1 @jsonify.when('isinstance(obj, datetime) or isinstance(obj, date)')
2 def jsonify_datetime(obj):
3     return str(obj)
```

Use-case #2: errors bring joy to my heart

- Data driven
- Ad-hoc event system
- DRY (user and framework)

```
1 @error_handler(pagenum_eh, "'num' in tg_errors")
2 @error_handler(content_eh)
3 @exception_handler(fubar)
4 def page(self, num, content=None):
5     ...
```

Use-case #3: not all see me equally exposed

- Data driven
- DRY

```
1 @expose("mypackage.templates.xml", accept_format="application/xml")
2 @expose("json")
3 @expose("mypackage.templates.view")
4 def view(self):
5     ...
```

A word to the wise

RuleDispatch pitfalls

- Compile on definition (lexical scoping)
- Constant folding
  - Solution:

```
1 def getter(var):
2     frame = sys._getframe(1)
3     return lambda: var in frame.f_locals and frame.f_locals[var] or \
4                                              frame.f_globals[var]
```

- Limited inference

Hide but do not lock away

- Rule generation
  - Hides underlying complexity
  - Users need not know: solve the most common case implicitly
- *Registrator pattern*
  - Adjust the scope
- Always leave an out

The user takes precedence

- Custom dispatch strategies

Many words, few lines

```
 1 def register_handler(handler, rules=None):
 2     def register(func):
 3         when = "func_eq(tg_source, func)"
 4         if rules:
 5             when += " and (%s)" % rules
 6         dispatch_error.when(
 7             dispatch_error.parse(when, *combine_contexts(depth=[0, 1])),
 8             order=1
 9         )(func)
10         return func
11     return register
```

The future is bright, the future is generic

- PEAK-Rules
- Python 3000?

Questions?

References

- peak.telecommunity.com/DevCenter/CombiningResults
- www.gigamonkeys.com/book/object-reorientation-generic-functions.html
- www.python.org/pycon/2005/papers/53/img0.html
- www.artima.com/weblogs/viewpost.jsp?thread=155123
- codespeak.net/pypy/dist/pypy/doc/theory.html#multimethods
- http://citeseer.ist.psu.edu/ernst98predicate.html
- http://citeseer.ist.psu.edu/chambers99efficient.html