demonstrate the recurring

The **error** procedure is an example of an escape procedure where everything that is waiting for the computation is

The

Rationale

We have now broken up our procedure into two parts, the control (loop) and the action. Using call/cc, we have a way of stopping the mpnite, our

Examples

If we need to compute the product of a non-empty list of numbers, we know that if 0 occurs in the list the product will be 0. So, we ca0 Td(ebAui Td(co25the)Tj 16.0799 0 T-799 0 Td53.0397j 12.47s:91 0

Loop via Continuations

This pattern presents a solution to creating a loop using

pass the continuation along as an argument. Inside these functions, if the condition to return to the beginning of the arthe ron1999 0 itseld(of)24.86.asR11

```
(define get-continuation-with-values
      (lambda (values)
            (let ((receiver (lambda (proc) (cons proc values))))
                  (call/cc receiver))))

(define loop-with-current-value
      (lambda (values)
            (let ((here-with-values (get-continuation-with-values values)))
                  (let ((continue
```

Escape from and Reentry into Recursion

This pattern presents a solution to the problem of escaping from a recursive process while allowing for the process to be reentered. We create a break procedure which will

Sample Code

The break procedure would be:

```
(define break
    (lambda (arg)
        (let ((exit-procedure
                (lambda (continuation)
                    (set! resume-computation (lambda () (continuation arg)))
                    (write-line ”Execution paused. Try (resume-computation)”)
                    ((escaper (lambda () arg))))))
            (call/cc exit-procedure))))
```

Howe

Coroutines

Coroutines allow for sequential control between procedures. Using call/cc, we obtain the current context of a procedure and store it so it can later be in

Sample Code

The **coroutine-maker** from [SF89] is:

```
(define coroutine-maker
      (lambda (proc)
            (let ((saved-continuation '()))
                  (let ((update-continuation!  (lambda (v)
                                                      (write-line x25lig97(9199 0sambda8j 26..9199 0(sav5j 17.64799 0 Td
```

Non-blind Backtracking

This pattern addresses the issue of allowing us to move to a previous or ”future” point of execution. Continuations are stored

were only partially complete. With these procedures, we have a simple isolated mechanism which can be reused in various programs. We can

given in [DH89] is used. With this timer, a tick is consumed by

3 Summary

We have presented in this pattern language the idea of first class continuations and several repetitive uses of continuations from non-local exits (with the *Escape from a Loop* and *Escape fromEsca*

4 Bibliography

References

[DH89] R. K. Dybvig and R. Hieb. Engines from Continuations. *Computer Languages*, pages 109–123, 1989.

[Dyb96] R. K. Dybvig. *The Scheme Programming Language*. Prentice Hall, 1996.

81 -289.319 7.24 0 Td(Pr)Tj 10.44 0 Td(o)Tj 5.27996 0 Td(gr)3j 9.47991 0 Td(a8.

[FHK84] D. P. Friedman, D. T. H3 3 Tm(,)Tj 12 0 Td(a9.)Tj 11.5E98 0 T3.43