Inuit: from printf to interactive user-interfaces

Frédéric Bour

OCL, University of Cambridge

Introduction

As OCaml doesn't favor introspection, long-running processes risk turning into black boxes.
Common solutions to observe internal state are printf-debugging/logging or relying on heavier GUI toolkits. The former is limited; the latter is harder to develop and has higher requirements.
Inuit extends the printf approach and narrow the gap with GUI toolkits.
It aims at exposing a model that is:

- easy to use for interactive monitoring, tracing, and configuration of services;
- good enough for simple text-based UI.

Programming model

Inuit is a library to build and update a semi-structured text document.
The central concept is the **region**. A region abstracts a range of characters and can contain sub-regions. The following primitives can be applied:

- *append* inserts text;
- *clear* erases everything in the region;
- *sub* creates and returns a new sub-region.

From a root region and these primitives one can build and update a tree of textual regions.

Rendering the document

Inuit keeps track of the document structure and turns the content into simple *patches* – a patch inserts, removes or replaces a range of text. Text itself is not stored by Inuit. A *backend* consumes the stream of patches.
This design makes it easy to switch between backends. The default one serializes patches to a channel, to be consumed by external processes. No dependency are introduced.

Region API

```
type 'flags t
(** Type of a region.
    'flags are associated to text content. They don't affect the document
    structure but allow to extend the text with backend specific information
    (styling, interactions, etc).  *)

val append : 'flags t → 'flags list → string → unit
(** Add text to the right end of the region. *)

val clear : 'flags t → unit
(** Erase the content, including sub-regions. *)

val sub : ?observer:'flags observer → 'flags t → 'flags t
(** Append a new sub-region.  Multiple calls to sub form a tree of regions.
    [observer] is a callback that allow to observe changes. *)

val make : unit → 'flags t * 'flags patch socket
(** Creating a root region. It also returns the patch stream for the backend. *)
```

Flexible deployment

The only backend available today displays content in Emacs. The UI is presented in a buffer. Communication is done with a custom sexp-based protocol.
Different setups are supported:

Commands communicating over stdin/stdout.

Services listening on a UNIX Domain Socket.

Remote display. The protocol handles long latencies well. Applications can run transparently over a network connection such as SSH.

Extended features

Extra information is communicated via flags. They don't affect the structure, just the interaction. If a flag is not supported by a backend, functionalities are degraded but the content remains available.
A reasonable set of flags is still being devised. Current implementation includes *clickable* and *editable* areas, *focus* management, and *custom face* selection.

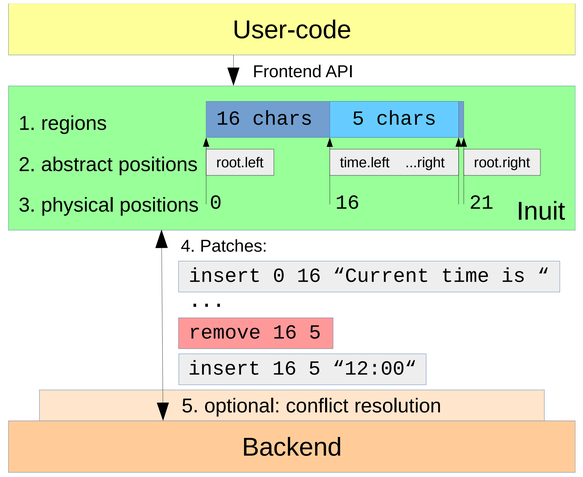
Simple trace

root region:
```
# append root "Current time is ";;
```
Current time is
```
# let time = sub root;;
```
Current time is
```
# append time "11:59";;
```
Current time is 11:59
```
# clear time;;
```
Current time is
```
# append time "12:00";;
```
Current time is 12:00

Practical applications

A few applications have been developed to validate the library.
Top-level integration, a replayable trace library, a frontend for interactive fictions. Introspection for Merlin (state monitoring, interactive logging).
Frontends for OCaml tools: *landmarks* and *spacetime* profilers and *ocp-index* library browser.

Rendering pipeline details



Inuit exposes regions (1), forming a tree of nested positions (2). Positions can be mapped to physical offsets (3). User actions update this tree and are turned into a stream of patches (4).
Since backend might executes asynchronously, an optional conflict resolution pass can be applied (5).

Future work

A TTY backend is planned for development soon.
A NeoVim backend is being considered too, but cleaning up the protocol specification and providing stronger typing is a prerequisite.
Among possible applications for Inuit we would like to experiment hyper-linked navigation in OCaml documentation.

Implementation prototype

The library is licensed under ISC license.
Development happens on github:

Inuit library: github.com/let-def/inuit

Emacs backend: github.com/let-def/sturgeon

OCaml