The children’s game universe “Crimeville” from the game developers *Art of Crime* challenges the players to solve detective riddles cooperatively. In the on-line version of the game this means that the players in each session of the game can chat with each other.

To help the children write better – and to limit them being naughty – the chat is going through a language server\(^1\).

\(^1\)Written in Ada
Ada in the on-line multi-user game “Crimeville”

Jacob Sparre Andersen

Jacob Sparre Andersen Research & Innovation

4th February 2012
Jacob Sparre Andersen

Currently:
- Independent consultant.
- Co-founder of AdaHeads K/S.

Background:
- PhD in experimental physics
- BSc in mathematics
- Has taught software engineering.
- Worked with bioinformatics, biotechnology and modelling of investments in the financial market.

jacob@jacob-sparre.dk
www.jacob-sparre.dk
The context – “Crimeville”

Crimeville . . .

- is an on-line free to play MMO by the Danish game producer Art of Crime.
- mixes a cocktail of heartfelt wacky story and character driven crime fiction, interaction and gaming for tweens.
- is both an on-line game, trading cards, and a face-to-face game.

You can play the game at http://www.crimeville.com/.

Ada in the on-line multi-user game “Crimeville”
When Art of Crime contacted me, their problem was simply described as helping the players write correctly, and limit how much they insult each other. – Already at this stage the plan was to do this at the word level.

In short, every word written by a player should be categorised in one of four categories; correct, foul, misspelled or unknown.
I proposed a solution with network servers checking words using Ispell compatible Open Source spell checkers.

Some of the benefits:

- This makes the language server independent of the actual game server.
- This allows Art of Crime to reuse existing language data (dictionaries, etc.)
- Art of Crime can switch between different spell checkers with only a small modification of the system.
- I could choose whatever implementation language suited me for the task.
Communications protocol

We created a simple text based protocol for the interaction between the game server and the language servers.

```
42  Fuck I'm a gud speller!
42  -Fuck +I'm +a ~gud +speller!
```

To simplify the system, we decided that each language server instance should handle a specific language\(^2\).

\(^2\)I.e. language is selected by IP address and port number.
The server activates two spell checker (Aspell) instances. – One for foul words and one for correct words. – The server communicates with the spell checkers through pipes. Each spell checker is encapsulated in a protected object.

The server allocates a new task for each client.

The task receives words (sentences) from the client; classifies the words one at a time using the spell checker instances; and returns the classification data to the client.
Application logic

```ada
Foul_Words.Check (Word => Word,         
   Class => Class);

case Class is
  when Aspell.Found =>
    return Game_Communication.Foul_Word;
  when Aspell.Misspelled =>
    Dictionary.Check (Word => Word,        
       Class => Class);

      case Class is
        when Aspell.Found =>
          return Game_Communication.Correct_Word;
        when Aspell.Misspelled =>
          return Game_Communication.Misspelled_Word;
        when Aspell.Not_Found | Aspell.Timeout | Aspell.Error =>
          return Game_Communication.Foul_Word;
     end case;
  when Aspell.Not_Found | Aspell.Timeout | Aspell.Error =>
    Dictionary.Check (Word => Word,        
       Class => Class);

      case Class is
        when Aspell.Found =>
          return Game_Communication.Correct_Word;
        when Aspell.Misspelled =>
          return Game_Communication.Misspelled_Word;
        when Aspell.Not_Found | Aspell.Timeout | Aspell.Error =>
          return Game_Communication.Unknown_Word;
     end case;
end case;
end if;
```

Jacob Sparre Andersen

Ada in the on-line multi-user game “Crimeville”
Launching a spell checker

POSIX.IO.Create_Pipe (Write_End => To_Child, Read_End => From_Parent);
POSIX.IO.Create_Pipe (Write_End => To_Parent, Read_End => From_Child);
POSIX.IO.Create_Pipe (Write_End => Errors_To_Parent, Read_End => Errors_From_Child);

case Fork is
   when Parent =>
      POSIX.IO.Close (From_Parent);
      POSIX.IO.Close (To_Parent);
      POSIX.IO.Close (Errors_To_Parent);
   when Child =>
      POSIX.IO.Close (To_Child);
      POSIX.IO.Close (From_Child);
      POSIX.IO.Close (Errors_From_Child);

      Move (From => From_Parent,
            To  => POSIX.IO.Standard_Input);
      Move (From => To_Parent,
            To  => POSIX.IO.Standard_Output);
      Move (From => Errors_To_Parent,
            To  => POSIX.IO.Standard_Error);

      POSIX.Unsafe_Process_Primitives.Exec_Search (Program_Name, Arguments);
end case;
Supporting packages (pre-existing)

- **POSIX.IO**: For communication with clients and spell checkers.
- **POSIX.Unsafe_Process_Primitives**: To launch spell checkers.
- **GNAT.Sockets**: To set up network connections.
- **GNAT.Sockets.Compatibility**: To make GNAT sockets visible as POSIX file descriptors. (home-grown)
- **EUP.Sockets**: Short-cuts for some common patterns when using TCP/IP sockets. (home-grown)
Supporting packages (new, likely to be reused)

- **Buffered_IO**: Adds a minimal Ada.Text_IO-like interface on top of POSIX.IO.
- **Daemon**: Imports the C function daemon, which is used to disconnect a process from its terminal and parent process.
- **Logging**: Simple logging package. Encapsulates an Ada.Text_IO file in a protected object, which only allows writing whole lines.
- **Pipe_Fork_Exec_Search**: Launches an external program with POSIX pipes to its standard input, output and error files.
Supporting packages (application specific)

- **Aspell**: Encapsulates a spell checker instance in a protected object.
- **Game_Communication**: Encapsulates the communication with a client (game server).
- **Logs**: Declares the log files used by the server.
# Source code statistics

<table>
<thead>
<tr>
<th>Unit</th>
<th>Project-only</th>
<th>Reused</th>
<th>Reusable</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilation units</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>Subroutines</td>
<td>13</td>
<td>5</td>
<td>20</td>
<td>38</td>
</tr>
<tr>
<td>Lines</td>
<td>489</td>
<td>111</td>
<td>343</td>
<td>943</td>
</tr>
<tr>
<td>Characters</td>
<td>16592</td>
<td>3920</td>
<td>10934</td>
<td>31446</td>
</tr>
</tbody>
</table>

Complete source code at [http://www.jacob-sparre.dk/spelling/crimeville.zip](http://www.jacob-sparre.dk/spelling/crimeville.zip)
Jacob Sparre Andersen
Jacob Sparre Andersen Research & Innovation
jacob@jacob-sparre.dk
www.jacob-sparre.dk

Crimeville
www.crimeville.com