Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Echtzeitprogrammierung mit Ada

Frank Feinbube

Seminar: Prozesssteuerung und Robotik

Agenda

- Einführung
  - Geschichte
  - Sprachfeatures
- **Echtzeitprogrammierung**
  - Zeiten und Uhren
  - **Multitasking & Rendezvous**
  - Prioritäten
  - Ravenscar-Profil
- Zusammenfassung

Geschichte

- **Verteidigungsministerium der USA** verwendete über **450** Programmiersprachen in seinen Projekten (viele eingebettet)
  - ➢ Schlechte Wartbarkeit und Wiederverwendung
- Lösung: Entwicklung einer *Hochsprache für eingebettete Systeme*
  - 1977: Ausschreibung → vier Kandidaten
  - ➢ ***10.12.***1980 wurde Ada von *Jean Ichbiah* angenommen
- Einführung **1983**: *ANSI*-Norm (MIL-STD ***1815***), später *ISO*-Norm
- Revision **1995**: erste objektorientierte Programmiersprache mit einer *ISO*-Norm

Ada Lovelace

- Augusta Ada Byron, Countess of Lovelace (*10.12.1815, †1852)
- Übersetzte Beschreibung der ***Analytical Engine***
- Plan, zur Berechnung von Bernoulli-Zahlen mit der Maschine
- **„erste ProgrammiererIn“**




Sprachfeatures

- Pakete / Namespaces
- Fehlerbehandlung
- Generische Programmeinheiten
- **Parallele / Nebenläufige Verarbeitung**

- **Ada95**
  - Objekt-Orientierung
- Ada2005
  - (Java-artige) Interfaces

„Hallo Ada“


```
with Ada.Text_IO;           -- include
use Ada.Text_IO;            -- import / using


                            -- Programmbaustein
procedure hello is          -- Name unabhängig von Datei
begin                       -- Blockstruktur
  Put_Line("Hello World");
end hello;                  -- Namenswiederholung optional
```

Zeiten und Uhren

- Normale Zeit: im Paket **Ada.Calendar**
- **Hochauflösender Timer**: im Paket **Ada.Real_Time**

- Funktionen
  - **Clock**: gibt die aktuelle Zeit zurück (vom Typ **TIME**)
  - **Split**: extrahiert Datum und Zeit (für einen Tag) aus TIME
  - **Time_Of**: Umkehrfunktion zu Split

- **delay <milliseconds>**: bis 86.400.0 in mind. 20er Schritten
- **delay until <time>**: unterbricht Programmausführung

Multitasking & Rendezvous

- **Tasks sind Objekte**
  - Spezifikation und Implementierung (**body**)
  - Instanziierung von **Task-Typen** mit **new**
  - Sind **limited private** → keine Zuweisungen oder Vergleiche
- Spezifikation definiert Einstiegspunkte (**entries**)
  - Können von außen mit Parametern aufgerufen werden
  - **body** definiert zugehöriges Verhalten
  - Immer nur ein aktiver Task → **FIFO** Warteschlange
- Ausführungsreihenfolge kann beliebig sein

Ein Beispiel-Task


```
task Gourmet is
   entry Make_A_Hot_Dog;
end Gourmet;
```

```
task body Gourmet is
begin
  Put_Line("I am ready.");
  for Index in 1..4 loop
    accept Make_A_Hot_Dog do
      delay 0.8;
      Put("Make Hot Dog.");
    end Make_A_Hot_Dog;
  end loop;
  Put_Line("I am finished.");
end Gourmet;
```

Aufruf des Einstiegspunktes:

```
for Index in 1..4 loop
  Gourmet.Make_A_Hot_Dog;
end loop;
```

Entries und Accepts

- **Entries**
  - Unbegrenzte Anzahl von Einstiegspunkten
  - Parameter Optional: **IN, INOUT, OUT**

- **Accepts**
  - Unbegrenzte Anzahl von accept-Ausdrücken
  - Erreichen → **Task schläft**, bis ein anderer Task ihn aufruft
  - Ausführung → **aufrufender Task schläft**
  - Leeres accept für Synchronisation
  - **FIFO-Warteschlange** für mehrere gleichzeitige Aufrufe

Rendezvous‘ kontrollieren mit Select


```
loop
   select
      accept Stock_With_A_Hot_Dog do
         Put_Line("Add a hot dog to the shelf");
      end Stock_With_A_Hot_Dog;
   or
      accept Deliver_A_Hot_Dog do
         Put_Line("Remove a hot dog from the shelf");
      end Deliver_A_Hot_Dog;
   end select;
end loop;
```

Guarded Select


```
loop
   select
      when Number_Of_Dogs < 8 =>
        accept Stock_With_A_Hot_Dog do
           Put_Line("Add a hot dog to the shelf");
        end Stock_With_A_Hot_Dog;
   or
      when Number_Of_Dogs > 0 =>
        accept Deliver_A_Hot_Dog do
           Put_Line("Remove a hot dog from the shelf");
        end Deliver_A_Hot_Dog;
   end select;
end loop;
```

… mehr zu Select


```
select
   Restaurant.Eat_A_Meal(My_Name);
else
   Burger_Boy.Eat_A_Meal(My_Name);
end select;

select
   Restaurant.Eat_A_Meal(My_Name);
or
   delay 600;
   Burger_Boy.Eat_A_Meal(My_Name);
end select;
```

… und noch mehr zu Select


```
select
   accept Answer_Query(FAST)(Counter : INTEGER) do
      Put("FAST Query made");
   end;
or
   when Answer_Query(FAST)'COUNT = 0 =>
      accept Answer_Query(MEDIUM)(Counter : INTEGER) do
         Put("MEDIUM Query made");
      end;
or
   terminate;
end select;
```

Select auf einen Blick

- Verwendung vom Gerufenen als auch vom Aufrufer

- Verwendungsarten von Select
  - Normal
  - mit **else**
  - mit **delay** / **delay until**
  - mit **guards**
  - mit **terminate**
  - mit **COUNT**

Protected Objects

- **Geschützter Bereich** → bewahrt Integrität
- Enthält normale Prozeduren, Funktionen, Entries
- Immer nur **ein aufrufender Task**, andere warten

- Prozeduren: dürfen private Daten lesen und schreiben
- Funktionen: dürfen nur lesen → mehrere parallele Zugriffe möglich

```
protected type <name> is
    <operations>
private
    <operations or declarations>
end <Name>;
```

Prioritäten: ein Bespiel


```
task type SHORT_LINE is
   pragma PRIORITY (5);
end SHORT_LINE;

task type LONG_LINE is
   pragma PRIORITY (1);
end LONG_LINE;

Cow, Dog: SHORT_LINE;
Whale, Bear: LONG_LINE;
```

```
task body SHORT_LINE is
begin
  for Index in 1..4 loop
    delay 0.0; Put_Line("s");
  end loop;
end SHORT_LINE;

task body LONG_LINE is
begin
  for Index in 1..3 loop
    delay 0.0; Put_Line("xxl");
  end loop;
end LONG_LINE;
```

Prioritäten

- **Größere Nummer = höhere Priorität**

- **Vererbung von Prioritäten**
  - Tasks erben die Priorität der Elterntask bei ihrer Aktivierung
  - Aufgerufener Task erbt die Priorität des Aufrufers
  - Bei Operationen auf protected objects, erben Tasks deren **ceiling priority** (höchste Priorität, aller zugreifenden Tasks)

- `pragma Queuing_Policy(Priority_Queuing);`
  - Tasks werden nach Prioritäten in Entry-Warteschlangen gefüllt
  - Bei gleicher Priorität → FIFO

Ravenscar

- Subset der Sprache: `pragma Profile(Ravenscar)`

- Garantiert
  - **Determinismus**
  - **Analysierbarkeit** mit Werkzeugen zur statischen Analyse
  - **Task-Planbarkeit**
  - Speicherbeschränktheit

- Zertifizierung bis zu den **höchsten Sicherheits-Levels**
  - Entwicklung hochzuverlässiger Software für Echtzeitsysteme
  - Verbietet Rendezvous und viele andere unsichere Features

Zusammenfassung / Fazit

- Entwicklung von **zuverlässiger Software**, großen Softwaresystemen **wiederverwendbarer** Komponenten im Team
- „**highly recommended**“ für sicherheitskritische Bereiche
- Viele Features für die Programmierung von **eingebetteten Echtzeitsystemen**:
  - Unterstützung von hochauflösenden Zeitgebern, delay until
  - **Multitasking, Rendesvouz, protected objects**
  - **Prioritäten**
  - Interrupt Handlern

Vertiefung

- **Überblick**
  - ADA 95 TUTORIAL, Gordon Dodrill, 1998, http://www.infres.enst.fr/~pautet/Ada95/a95list.htm
  - Real-Time Programming: Real-Time with Ada, Andreas Polze http://www.dcl.hpi.uni-potsdam.de/teaching/EmbeddedOS/Slides/05_Ada.pdf
  - Taschenbuch: Programmiersprachen, Peter A. Henning und Holger Vogelsang, 2007, Kapitel 11: Ada

- **GNAT** : https://libre.adacore.com/gps/
- **Ada#** : http://asharp.martincarlisle.com/