

```
1  -----
2  -----
3  -----
4  -- This package provides transactions
5  --
6  -- $Author: jkienzle $
7  -- $Date: 2001/04/23 11:19:13 $
8  -- $RCSfile: transactions.adb,v $
9  -- $Revision: 1.9 $
10 -----
11 -----
12 -----
13 with Ada.Unchecked_Deallocation;
14 with Debug; use Debug;
15
16 with Recovery_Managers; use Recovery_Managers;
17 with Recovery_Support; use Recovery_Support;
18
19 package body Transactions is
20
21 -- We can have concurrent calls to Get_New_Transaction_Id
22 -- We therefore need a protected distributor
23
24 protected Distributor is
25
26   procedure Get_Next_Simple_Transaction
27     (Simple : out Simple_Transaction_Identifier_Type);
28
29   private
30
31     Next_Free_Number : Positive := 1;
32
33     -- This must be changed if we include distribution
34     Node_Number : Positive := 123;
35
36   end Distributor;
37
38 protected body Distributor is
39
40   procedure Get_Next_Simple_Transaction
41     (Simple : out Simple_Transaction_Identifier_Type) is
42   begin
43     Simple.Number := Next_Free_Number;
44     Next_Free_Number := Next_Free_Number + 1;
45     Simple.Node.Number := Node_Number;
46     end Get_Next_Simple_Transaction;
47
48   end Distributor;
49
50   procedure Check_And_Then_Add (Transaction : in out Transaction_Type;
51     Object : in Memory_Object_Ref) is
52   begin
53     Transaction.Protection.Check_And_Then_Add (Object);
54   end Check_And_Then_Add;
55
56   procedure Check_And_Then_Add_Modified
57     (Transaction : in out Transaction_Type;
58     Object : in Memory_Object_Ref) is
59   begin
60     Transaction.Protection.Check_And_Then_Add_Modified (Object);
61   end Check_And_Then_Add_Modified;
62
63   procedure Decrease_Reference_Count (T : in Transaction_Ref) is
64   begin
65     T.Protection.Decrement_Reference_Count;
66     end Decrease_Reference_Count;
```

```
67
68   procedure Increase_Reference_Count (T : in Transaction_Ref) is
69   begin
70     T.Protection.Increment_Reference_Count;
71     end Increase_Reference_Count;
72
73   procedure Free is new Ada.Unchecked_Deallocation
74     (Transaction_Type'Class, Transaction_Ref);
75
76   procedure Check_Reference_Count (T : in out Transaction_Ref) is
77   begin
78     if T.Protection.Get_Reference_Count = 0 then
79       Put_Line ("Deleted transaction");
80       Free (T);
81     end if;
82   end Check_Reference_Count;
83
84   protected body Concurrency_Protection_Type is
85
86   procedure Decrement_Participants is
87   begin
88     Number_Of_Participants := Number_Of_Participants - 1;
89     end Decrement_Participants;
90
91   procedure Increment_Participants is
92   begin
93     Number_Of_Participants := Number_Of_Participants + 1;
94     end Increment_Participants;
95
96   procedure Set_Number_Of_Participants (Number : in Natural) is
97   begin
98     Number_Of_Participants := Number;
99     end Set_Number_Of_Participants;
100
101   function Get_Number_Of_Participants return Natural is
102   begin
103     return Number_Of_Participants;
104   end Get_Number_Of_Participants;
105
106   procedure Decrement_Reference_Count is
107   begin
108     Reference_Count := Reference_Count - 1;
109     if Children /= null then
110       Put_Line ("Topref : " & Integer'Image (Reference_Count));
111     else
112       Put_Line ("Ref : " & Integer'Image (Reference_Count));
113     end if;
114     end Decrement_Reference_Count;
115
116   procedure Increment_Reference_Count is
117   begin
118     Reference_Count := Reference_Count + 1;
119     if Children /= null then
120       Put_Line ("Topref : " & Integer'Image (Reference_Count));
121     else
122       Put_Line ("Ref : " & Integer'Image (Reference_Count));
123     end if;
124     end Increment_Reference_Count;
125
126   procedure Set_Reference_Count (Number : in Natural) is
127   begin
128     Reference_Count := Number;
129   end Set_Reference_Count;
130
131   function Get_Reference_Count return Natural is
132   begin
```

```
133     return Reference_Count;
134 end Get_Reference_Count;
135
136 procedure Set_Transaction_Status (Status : in Transaction_Status_Type) is
137 begin
138   Transaction_Status := Status;
139 end Set_Transaction_Status;
140
141 function Get_Transaction_Status return Transaction_Status_Type is
142 begin
143   return Transaction_Status;
144 end Get_Transaction_Status;
145
146 entry Suspend_Task when Transaction_Status = Aborted or
147   Transaction_Status = Committed is
148 begin
149   null;
150 end Suspend_Task;
151
152 procedure Add_Child (Child : Transaction_Ref) is
153 begin
154   Child_Lists.Insert_Element (Child_List, Child);
155   Increment_Reference_Count;
156 end Add_Child;
157
158 procedure Remove_Child (Child : Transaction_Ref) is
159 begin
160   Child_Lists.Delete_Element (Child_List, Child);
161   Decrement_Reference_Count;
162 end Remove_Child;
163
164 procedure Check_And_Then_Add (Object : in Memory_Object_Ref) is
165 begin
166   Object_Lists.Insert_If_Not_Already (Accessed_Object_List, Object);
167 end Check_And_Then_Add;
168
169 procedure Check_And_Then_Add_Modified (Object : in Memory_Object_Ref) is
170 begin
171   Object_Lists.Insert_If_Not_Already (Accessed_Object_List, Object);
172   Object_Lists.Insert_If_Not_Already (Modified_Object_List, Object);
173 end Check_And_Then_Add_Modified;
174
175 procedure Append_Objects (Objects : Object_Lists.List_Type) is
176 begin
177   Object_Lists.Merge_Lists (Accessed_Object_List, Objects);
178 end Append_Objects;
179
180 procedure Append_Objects_Modified (Objects : Object_Lists.List_Type) is
181 begin
182   Object_Lists.Merge_Lists (Modified_Object_List, Objects);
183 end Append_Objects_Modified;
184
185 procedure Delete_Objects is
186 begin
187   Object_Lists.Delete_List (Accessed_Object_List);
188   Delete_Objects;
189
190 procedure Delete_Objects_Modified is
191 begin
192   Object_Lists.Delete_List (Modified_Object_List);
193 end Delete_Objects_Modified;
194
195 function Get_Object_List return Object_Lists.List_Type is
196 begin
197   return Accessed_Object_List;
198 end Get_Object_List;
```

```
199
200 function Get_Object_List return Accessed_Objects_Type is
201 N : Natural := Object_Lists.Count_Elements (Accessed_Object_List);
202 Result : Accessed_Objects_Type (1 .. N);
203 Tmp : Object_Lists.List_Type := Accessed_Object_List;
204 begin
205   for I in Result..Range loop
206     Result (I) := Tmp.Element;
207     Tmp := Tmp.Next;
208   end loop;
209   return Result;
210 end Get_Object_List;
211
212 function Get_Object_List_Modified return Object_Lists.List_Type
213 is
214   return Modified_Object_List;
215 end Get_Object_List_Modified;
216
217 function Get_Object_List_Modified return Accessed_Objects_Type is
218 N : Natural := Object_Lists.Count_Elements (Modified_Object_List);
219 Result : Accessed_Objects_Type (1 .. N);
220 Tmp : Object_Lists.List_Type := Modified_Object_List;
221 begin
222   for I in Result..Range loop
223     Result (I) := Tmp.Element;
224     Tmp := Tmp.Next;
225   end loop;
226   return Result;
227 end Get_Object_List_Modified;
228
229 end Concurrency_Protection_Type;
230
231 function Create_New_Toplevel_Transaction return Transaction_Ref is
232 Temp_Ref : Transaction_Ref := new Transaction_Type;
233 begin
234   Temp_Ref.Protection.Set_Transaction_Status (Open);
235   Temp_Ref.Protection.Set_Reference_Count (1);
236   Temp_Ref.Protection.Set_Number_Of_Participants (1);
237   Distributor.Get_Next_Simple_Transaction (Temp_Ref.Identifier);
238   return Temp_Ref;
239 end Create_New_Toplevel_Transaction;
240
241 procedure Create_Nested_Transaction
242 (Parent : in out Transaction_Type;
243  New_Transaction : out Transaction_Ref) is
244   Temp_Ref : Transaction_Ref := new Transaction_Type;
245 begin
246   Temp_Ref.Parent := Self (Parent);
247   Parent.Protection.Add_Child (Temp_Ref);
248   Temp_Ref.Protection.Set_Transaction_Status (Open);
249   Temp_Ref.Protection.Set_Reference_Count (1);
250   Temp_Ref.Protection.Set_Number_Of_Participants (1);
251   Distributor.Get_Next_Simple_Transaction (Temp_Ref.Identifier);
252   New_Transaction := Temp_Ref;
253 end Create_Nested_Transaction;
254
255 procedure Join_Transaction (Transaction : in out Transaction_Type)
256 is
257 begin
258   case Transaction.Protection.Get_Transaction_Status is
259     when Open =>
260       Transaction.Protection.Increment_Participants;
261     when Aborted =>
262       raise Transaction_Aborted;
263     when Committed =>
264       raise Transaction_Already_Committed;
265     when Closed =>
```

```

265         raise Transaction_Closed;
266     end case;
267 end Join_Transaction;
268
269 procedure Close_Transaction (Transaction : in out Transaction_Type) is
270 begin
271     case Transaction.Protection.Get_Transaction_Status is
272     when Open =>
273         Transaction.Protection.Set_Transaction_Status (Closed);
274     when Aborted =>
275         raise Transaction_Aborted;
276     when Committed =>
277         raise Transaction_Already_Committed;
278     when Closed =>
279         null;
280     end case;
281 end Close_Transaction;
282
283 procedure Commit_Transaction (Transaction : in out Transaction_Type) is
284 begin
285     Transaction.Protection.Decrement_Participants;
286     if Transaction.Protection.Get_Transaction_Status = Aborted then
287         raise Transaction_Aborted;
288     end if;
289     if Transaction.Protection.Get_Number_Of_Participants = 0 then
290         Commit_Transaction (Get_Recovery_Manager.all, Self (Transaction));
291     Transaction.Protection.Set_Transaction_Status (Committed);
292     if Transaction.Parent /= null then
293         -- pass accessed objects to parent
294         Transaction.Parent.Protection.Append_Objects
295             (Transaction.Protection.Delete_Object_List);
296     Transaction.Protection.Delete_Objects;
297     Transaction.Parent.Protection.Append_Objects;
298     Transaction.Protection.Get_Object_List_Modified;
299     Transaction.Protection.Delete_Objects_Modified;
300     Transaction.Parent.Protection.Remove_Child (Self (Transaction));
301     else
302         -- delete accessed objects
303         Transaction.Protection.Delete_Objects;
304     Transaction.Protection.Delete_Objects_Modified;
305     end if;
306 else
307     Transaction.Protection.Suspend_Task;
308     if Transaction.Protection.Get_Transaction_Status = Aborted then
309         raise Transaction_Aborted;
310     end if;
311     end if;
312 end Commit_Transaction;
313
314 procedure Abort_Transaction (Transaction : in out Transaction_Type) is
315 begin
316     Transaction.Protection.Decrement_Participants;
317     if Transaction.Protection.Get_Transaction_Status /= Aborted then
318         Abort_Transaction (Get_Recovery_Manager.all, Self (Transaction));
319     Transaction.Protection.Set_Transaction_Status (Aborted);
320     -- delete accessed objects
321     Transaction.Protection.Delete_Objects;
322     Transaction.Protection.Delete_Objects_Modified;
323     if Transaction.Parent /= null then
324         Transaction.Parent.Protection.Remove_Child (Self (Transaction));
325     end if;
326     end if;
327 end Abort_Transaction;
328
329 procedure Set_Final_Transaction_Status
330     (Transaction : in out Transaction_Type;

```

```

331     Status : in Transaction_Status_Type) is
332 begin
333     Transaction.Protection.Set_Transaction_Status (Status);
334     end Set_Final_Transaction_Status;
335
336     function Self (Transaction : in Transaction_Type) return
337         Transaction_Ref is
338 begin
339     return Transaction.Self.Obj.all'Unchecked_Access;
340 end Self;
341
342 procedure Finalize (Transaction : in out Transaction_Type) is
343 begin
344     Transaction.Protection.Delete_Objects;
345     end Finalize;
346
347     function Is_Ancesor_Or_Same (Transaction : in Transaction_Type;
348         Parent : in Transaction_Ref) return Boolean is
349     Tmp : Transaction_Ref := Self (Transaction);
350 begin
351     while Tmp /= null loop
352         if Tmp = Parent then
353             return True;
354         end if;
355         Tmp := Tmp.Parent;
356     end loop;
357     return False;
358 end Is_Ancesor_Or_Same;
359
360     function Is_Toplevel (Transaction : Transaction_Type) return Boolean is
361 begin
362     return Transaction.Parent = null;
363 end Is_Toplevel;
364
365     function Get_Parent (Transaction : Transaction_Type)
366     return Transaction_Ref is
367 begin
368     return Transaction.Parent;
369 end Get_Parent;
370
371     function Get_Simple_Transaction_Identifier (Transaction : Transaction_Type)
372     return Simple_Transaction_Identifier_Type is
373 begin
374     return Transaction.Identifier;
375 end Get_Simple_Transaction_Identifier;
376
377     function Transaction_Text (Transaction : Transaction_Type)
378     return String is
379 begin
380     return Natural'Image (Transaction.Identifier.Number);
381 end Transaction_Text;
382
383     function Transaction_Text (Transaction : Simple_Transaction_Identifier_Type)
384     return String is
385 begin
386     return Natural'Image (Transaction.Number);
387 end Transaction_Text;
388
389     function Get_Accessed_Objects (Transaction : Transaction_Type)
390     return Accessed_Objects_Type is
391 begin
392     return Transaction.Protection.Get_Object_List;
393 end Get_Accessed_Objects;
394
395     function Get_Modified_Objects (Transaction : Transaction_Type)
396     return Accessed_Objects_Type is

```

```
397   begin
398       return Transaction.Protection.Get_Object_List_Modified;
399   end Get_Modified_Objects;
400
401   end Transactions;
```