```
 1 //#include "RTI/RTI1516.h"
 2
 3 #include <RTI/RTIambassadorFactory.h>
 4 #include <memory>
 5 #include <iostream>
 6 #include <cstdlib>
 7 #include <cerrno>
 8 #include <cstring>
 9 #ifndef _WIN32
10 #include <csignal>
11 #include <unistd.h>
12 #endif
13
14 #include "PrettyDebug.hh"
15 #include "RTI1516ambassador.h"
16
17 #include "M_Classes.hh"
18
19 #include "config.h"
20
21 #ifndef dbg
22 #define dbg \
23     { \
24       FILE *logfiletmp = fopen("out2.txt","at"); \
25       fprintf (logfiletmp, "DBG: %s::%s::%d\n", __FILE__, __FUNCTION__, __LINE__);
   \
26       fclose (logfiletmp); \
27     }
28 #endif
29 #ifndef dbg2
30 #define dbg2(str) \
31     { \
32       FILE *logfiletmp = fopen("out2.txt","at"); \
33       fprintf (logfiletmp, "DBG: %s::%s::%d --> %s\n", __FILE__, __FUNCTION__,
   __LINE__, str); \
34       fclose (logfiletmp); \
35     }
36 #endif
37
38 rti1516::RTIambassadorFactory::RTIambassadorFactory () {
39   dbg
40 }
41
42 rti1516::RTIambassadorFactory::~RTIambassadorFactory () throw () {
43 }
44
45 namespace
46 {
47   static PrettyDebug D1516("LIBRTI1516", __FILE__);
48   static PrettyDebug G1516("GENDOC1516", __FILE__);
49 }
50
51 std::auto_ptr<rti1516::RTIambassador>
   rti1516::RTIambassadorFactory::createRTIambassador (std::vector<
52     std::wstring> & args) throw (BadInitializationParameter, RTIinternalError) {
53   dbg
54   rti1516::RTI1516ambassador* p_ambassador(new rti1516::RTI1516ambassador());
55   dbg
56   std::auto_ptr<rti1516::RTIambassador> ap_ambassador(p_ambassador);
57   dbg
58   G1516.Out(pdGendoc, "enter RTIambassador::RTIambassador");
59   PrettyDebug::setFederateName("LibRTI::UnjoinedFederate");
```

```
  std::wstringstream msg;
  dbg
  p_ambassador->privateRefs = new RTI1516ambPrivateRefs();
  dbg
  p_ambassador->privateRefs->socketUn = new SocketUN(stIgnoreSignal);
  dbg
  p_ambassador->privateRefs->is_reentrant = false;
  dbg
  std::vector<std::string> rtiaList;
  const char* env = getenv("CERTI_RTIA");
  if (env && strlen(env))
    rtiaList.push_back(std::string(env));
  env = getenv("CERTI_HOME");
  if (env && strlen(env))
    rtiaList.push_back(std::string(env) + "/bin/rtia");
  rtiaList.push_back(PACKAGE_INSTALL_PREFIX "/bin/rtia");
  rtiaList.push_back("rtia");
  dbg
#if defined(RTIA_USE_TCP)
  int port = p_ambassador->privateRefs->socketUn->listenUN();
  if (port == -1) {
    D1516.Out( pdError, "Cannot listen to RTIA connection. Abort." );
    throw rti1516::RTIinternalError(L"Cannot listen to RTIA connection" );
  }
#else
  int pipeFd = p_ambassador->privateRefs->socketUn->socketpair();
  dbg
  if (pipeFd == -1) {
    dbg
    D1516.Out(pdError, "Cannot get socketpair to RTIA connection. Abort.");
    throw rti1516::RTIinternalError(L"Cannot get socketpair to RTIA connection");
  }
#endif

#ifdef _WIN32
  STARTUPINFO si;
  PROCESS_INFORMATION pi;
  dbg
  ZeroMemory(&si, sizeof(si));
  si.cb = sizeof(si);
  ZeroMemory(&pi, sizeof(pi));
  /*
   * Avoid displaying console window
   * when running RTIA.
   */
  si.dwFlags = STARTF_USESHOWWINDOW;
  si.wShowWindow = SW_HIDE;
  dbg
#if !defined(RTIA_USE_TCP)
  SOCKET newPipeFd;
  dbg
  if (!DuplicateHandle(GetCurrentProcess(),
                       (HANDLE) pipeFd,
                       GetCurrentProcess(),
                       (HANDLE*) &newPipeFd,
                       0,
                       TRUE, // Inheritable
                       DUPLICATE_SAME_ACCESS)) {
    dbg
    D1516.Out(pdError, "Cannot duplicate socket for RTIA connection. Abort.");
    throw rti1516::RTIinternalError(L"Cannot duplicate socket for RTIA connection.
Abort.");
```

```
121   }
122 #endif
123   dbg
124   bool success = false;
125   for (unsigned i = 0; i < rtiaList.size(); ++i) {
126     std::stringstream stream;
127 #if defined(RTIA_USE_TCP)
128     stream << rtiaList[i] << ".exe -p " << port;
129 #else
130     stream << rtiaList[i] << ".exe -f " << newPipeFd;
131 #endif
132
133     dbg2((char*) stream.str().c_str());
134     // Start the child process.
135     if (CreateProcess(NULL, // No module name (use command line).
136                       (char*) stream.str().c_str(), // Command line.
137                       NULL, // Process handle not inheritable.
138                       NULL, // Thread handle not inheritable.
139                       TRUE, // Set handle inheritance to TRUE.
140                       0, // No creation flags.
141                       NULL, // Use parent's environment block.
142                       NULL, // Use parent's starting directory.
143                       &si, // Pointer to STARTUPINFO structure.
144                       &pi)) // Pointer to PROCESS_INFORMATION structure.
145     {
146       dbg
147       success = true;
148       break;
149     }
150     dbg
151   }
152   dbg
153   if (!success) {
154     dbg
155     msg << "CreateProcess - GetLastError()=<" << GetLastError() << "> " << "Cannot
    connect to RTIA.exe";
156     throw rti1516::RTIinternalError(msg.str());
157   }
158   dbg
159   p_ambassador->privateRefs->handle_RTIA = pi.hProcess;
160   dbg
161 #if !defined(RTIA_USE_TCP)
162   dbg
163   closesocket(pipeFd);
164   closesocket(newPipeFd);
165 #endif
166   dbg
167 #else
168   dbg
169   sigset_t nset, oset;
170   // temporarily block termination signals
171   // note: this is to prevent child processes from receiving termination signals
172   sigemptyset(&nset);
173   sigaddset(&nset, SIGINT);
174   sigprocmask(SIG_BLOCK, &nset, &oset);
175
176   switch((p_ambassador->privateRefs->pid_RTIA = fork())) {
177     case -1: // fork failed.
178       dbg
179     perror("fork");
180     // unbock the above blocked signals
181     sigprocmask(SIG_SETMASK, &oset, NULL);
```

```
182 #if !defined(RTIA_USE_TCP)
183     close(pipeFd);
184 #endif
185     throw rti1516::RTIinternalError(wstringize() << "fork failed in RTIambassador
    constructor");
186     break;
187
188     case 0: // child process (RTIA).
189       dbg
190     // close all open filedescriptors except the pipe one
191     for (int fdmax = sysconf(_SC_OPEN_MAX), fd = 3; fd < fdmax; ++fd) {
192 #if !defined(RTIA_USE_TCP)
193       if (fd == pipeFd)
194       continue;
195 #endif
196       close(fd);
197       dbg
198     }
199     for (unsigned i = 0; i < rtiaList.size(); ++i)
200     {
201       dbg
202       std::stringstream stream;
203 #if defined(RTIA_USE_TCP)
204       stream << port;
205       execlp(rtiaList[i].c_str(), rtiaList[i].c_str(), "-p", stream.str().c_str(),
    NULL);
206 #else
207       stream << pipeFd;
208       execlp(rtiaList[i].c_str(), rtiaList[i].c_str(), "-f", stream.str().c_str(),
    NULL);
209 #endif
210     }
211     dbg
212     // unbock the above blocked signals
213     sigprocmask(SIG_SETMASK, &oset, NULL);
214     msg << "Could not launch RTIA process (execlp): "
215     << strerror(errno)
216     << std::endl
217     << "Maybe RTIA is not in search PATH environment.";
218     throw rti1516::RTIinternalError(msg.str().c_str());
219
220     default: // father process (Federe).
221     // unbock the above blocked signals
222       dbg
223     sigprocmask(SIG_SETMASK, &oset, NULL);
224 #if !defined(RTIA_USE_TCP)
225     close(pipeFd);
226 #endif
227     break;
228   }
229 #endif
230   dbg
231 #if defined(RTIA_USE_TCP)
232   if (p_ambassador->privateRefs->socketUn->acceptUN(10*1000) == -1) {
233 #ifdef _WIN32
234     TerminateProcess(p_ambassador->privateRefs->handle_RTIA, 0);
235 #else
236     kill(p_ambassador->privateRefs->pid_RTIA, SIGINT );
237 #endif
238     dbg
239     throw rti1516::RTIinternalError( wstringize() << "Cannot connect to RTIA" );
240   }
```

```
241 #endif
242   dbg
243   certi::M_Open_Connexion req, rep;
244   dbg
245   req.setVersionMajor(CERTI_Message::versionMajor);
246   dbg
247   req.setVersionMinor(CERTI_Message::versionMinor);
248   dbg
249   G1516.Out(pdGendoc, "        ====>executeService OPEN_CONNEXION");
250   dbg
251   p_ambassador->privateRefs->executeService(&req, &rep);
252   dbg
253   G1516.Out(pdGendoc, "exit  RTIambassador::RTIambassador");
254   dbg
255   return ap_ambassador;
256 }
257
258 //} // end namespace rti1516
259
```