Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXIII, págs. 2939-2955 (artículo completo)
Graciela Bertolino, Mariano Cantero, Mario Storti y Federico Teruel (Eds.)
San Carlos de Bariloche, 23-26 Setiembre 2014

A COUPLED SCHEME FOR THE DETERMINISTIC SAFETY TRANSIENT ANALYSIS OF THE ATUCHA I NUCLEAR POWER PLANT

Theler, Germán[a], Gómez Omil, Juan Pablo[a] and Mazzantini, Oscar[b]

[a]*TECNA Estudios y Proyectos de Ingeniería S.A.*
Grupo de Cálculo y Análisis de Sistemas Nucleares, Buenos Aires, Argentina

[b]*Nucleoeléctrica Argentina S.A.*
Grupo de Licenciamiento, Seguridad y Cálculo del Núcleo, Lima, Argentina

Keywords: Atucha, coupled calculations, RELAP, FSAR, safety analysis

Abstract. Every nuclear power plant in the world has to solidly prove that its reactor safety systems are able to cope with any design-basis accident situation in a satisfactory way using sound mathematical models and computational tools. On the one hand, more accurate models are developed as new experimental results are obtained using innovative test loops. On the other hand, computer hardware and software tools are being continuously improved including remarkable breakthroughs such as massive parallelization methods or GPU-based computations. Therefore, deterministic safety analysis of nuclear power plants ought to be regularly updated using state-of-the-art techniques. This article summarizes how different branches of nuclear engineering—namely neutronics, thermalhydraulics, control systems and computational fluid-dynamics—have to be merged in order to perform coupled transient calculations considering the rather different phenomena that take place within a nuclear reactor core, together with the actuation of the associated safety systems. In particular, the work is focused on the case of a loss-of-coolant accident that requires the fast injection of a boric acid solution into the moderator tank. This computation involves different computer codes, each solving a particular engineering problem. The space and time-dependent evolution of the boron plume is computed by a CFD code and fed into a neutron spatial kinetic code, that takes into account the thermalhydraulic conditions of the core—which are computed by another code—evaluates the instantaneous power distribution. Additionally, the plant status is monitored by a computational implementation of the reactor control and protection systems which command the actuation of the appropriate safety mechanisms that drive the plant to a safe condition.

1 INTRODUCTION

Atucha I is a pressure-vessel heavy-water reactor with online full-power refueling, both cooled and moderated by heavy water. The plant is owned and operated by Nucleoeléctrica Argentina S.A, and started commercial operation on March 1974 rendering it as the first Nuclear Power Plant that has been commissioned in Latin America. Due to licensing requirements, the Final Safety Analysis Report (FSAR) and in particular, chapter 15 "Accident Analyses" has to be updated. The analysis of postulated events that trigger certain plant dynamics that correspond to incident and accidental scenarios have to be computed using state-of-the-art mathematical models and computer codes. These methods include up-to-date experimentally-validated two-phase flow models and correlations, CFD-based calculations in complex geometries, spatial neutron kinetics and a detailed representation of the electronics that implement the reactor control and protection systems.

The reactor core has 250 vertical cooling channels, each one containing one fuel element. The complete fuel column has a height of approximately 6 meters, whereas the active length is 5.30 m long, consisting of 37 rods with a Zy-4 cladding arranged to form an array of concentric circles. Within the reactor pressure vessel, the moderator and the coolant are separated. Due to reactivity reasons, in average, the moderator is maintained approximately 100°C colder than the coolant. The moderator tank provide small nozzles that communicate with the upper plenum in such a way that both the moderator and the coolant are kept at the same pressure of 115 bar. The original gross electric design power was 330 MWe with a thermal reactor power of 1100 MW. Two changes were implemented afterward: in 1977 the gross power was increased up to 357 MWe (8%), generating a net power of 335 MWe and a thermal power of 1179 MW. From 1995 up to 2000, a progressive conversion from natural uranium to slightly enriched uranium (0.85% of enrichment) was performed.

The plant code RELAP provides both state-of-the-art mathematical models and experimental correlations of two-phase heat transfer and flow. The reactor control and protection systems involve many common logic components such as threshold comparators with hysteresis, sliding limiters, dead bands, etc. that can be implemented using RELAP's control elementary components and logic trips. However, this implementation results in a cumbersome input file that is hard to debug, to maintain and that takes up almost all of the available component slots provided by RELAP which could lead to the inability to implement some in-built models like setting the power dissipated in a heat structure using control variables for a detailed representation of the core. Even more, for the case of Atucha II (which includes also an intermediate layer of limitation besides the control and protection systems) it was impossible to implement the control, limitation and protection systems using RELAP's components to the required level of detail for transient operational and safety analysis. It was then decided that in order to perform the calculations needed for the Accident Analyses in Chapter 15 of the FSAR a coupling mechanism between RELAP and an external code that implemented the control logic was needed.

With the objective of updating the FSAR of Atucha I, besides the implementation of the reactor's control and protection logic as a set of Fortran routines that interact with the plant model in RELAP, the introduction of a three-dimensional neutronic kinetics is desired. In this work, a brief review of the previous works about coupling RELAP with external codes is performed. Then, the proposed method for the update of Atucha I's FSAR is introduced and compared to the previous schemes. The main features of the proposed coupling scheme are illustrated with excerpts of input files and short examples of application.

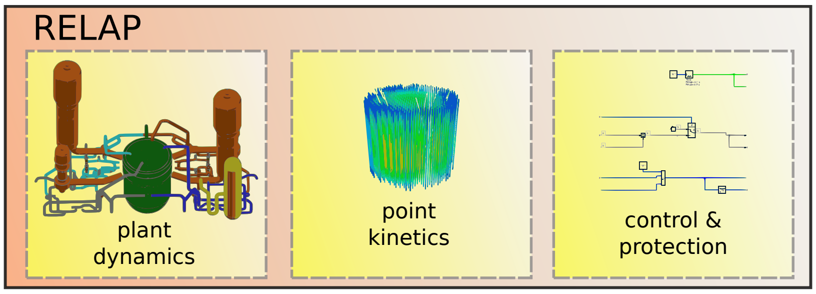


Figure 1: RELAP provides means to compute the plant dynamics, the neutron point kinetics and the control logic. However, the control components are not flexible enough to model the control and protection systems of Atucha-like reactors up to the required level of detail.

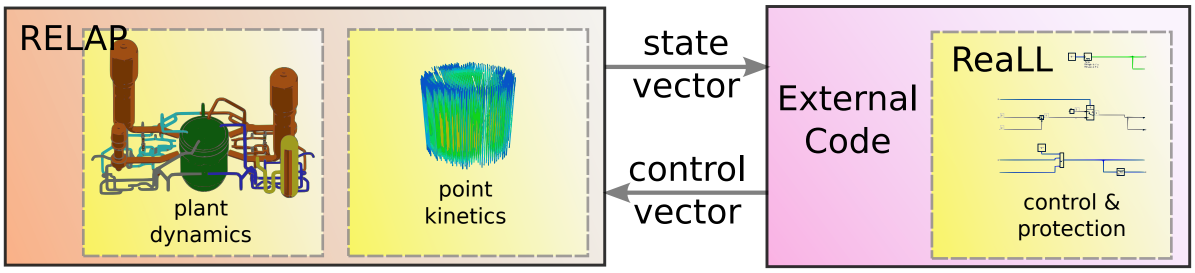


Figure 2: The main objective of the previous works was to be able to couple RELAP to an external code that emulates the control and protection systems (Reaktor Leistung Leittechnik).

2 PREVIOUS WORKS

In principle, the plant code RELAP is able to model the plant dynamics, the neutron kinetics using the point reactor equations and control logic (figure 1). However, as discussed in the introduction, the facilities provided by RELAP to model the control and protection system of an Atucha-like reactor are not satisfactory for the required level of detail. The main original objective was thus to be able to exchange information between the plant code RELAP and a generic external code that should emulate the reactor control, limitation (in the case of Atucha II) and protection systems. In the German slang, these routines are known as ReaLL that stand for Reaktor Leistung Leittechnik (figure 2) and due to historical reasons are written in FORTRAN 77. as a preliminary approach, RELAP should generate a status vector composed of a certain number of properties (temperatures, mass flows, levels, valve positions, etc.) which ought to be read by ReaLL, which in turn generates a control vector composed of a number of actions (movement of control rods, signals to open or close valves, etc.). This control vector is to be read back by RELAP so the state vector in the next discrete time can be computed.

A brief description of the coupling schemes between RELAP and generic external codes that were developed *ad-hoc* for the Atucha II Nuclear Power Plant follows.

2.1 Coupling through RELAP's restart and input files

This scheme was first proposed by the University of Pisa to perform safety transient analysis of Atucha II. It is based on information exchange using plain-text files. The main idea can be resumed with the following steps:

1. Execute RELAP with an input file that starts a new problem and advances exactly one single time step
2. Execute again RELAP in strip mode to extract the values of the control variables contained in the state vector
3. Execute a program that reads the ASCII file with the state vector, advances one single step of the ReaLL routines and writes the control vector into an ASCII file
4. Execute a program that reads the ASCII file with the control vector and generates a RELAP input file where the control variables that belong to the control vector are re-defined as contants with the values computed by ReaLL
5. Execute RELAP with such input file that advances exactly one single time step loading the state from a restart-plot file
6. If $t < t_{\text{final}}$ go to step 2, else quit

The main advantage of this method is that there is no need to modify RELAP's source code. However, even though the file-access time can be reduced by using RAM disks, the overhead of starting and executing different codes in each time step renders this method very inefficient. Besides, the scheme is fairly rigid and the conversion between the binary data and the ASCII representation loses precision. Finally, debugging the ReaLL routines to track back why a certain signal was triggered is tricky because a new session has to be started whenever the time advances one step. Nevertheless, this method was used to some extent to perform coupled transient analysis.

2.2 Coupling through RELAP's control variables

The Argentinian National Atomic Energy Commission (CNEA) developed a modified version of RELAP to allow coupled calculations with an external code (Maciel, 2011). This modified version is called RELAP_MEM and incorporates two new control variable types, namely "ivar" and "ovar". RELAP_MEM thus parses the input file as usual but detects these new types of control variables and, at each time step, when the routine convar that updates the control variables is executed, first a certain pre-defined segment of shared memory (the state vector) is written with the content of the ovar-variables following its numerical order. Afterward, the values of the ivar-variables are overwritten with the information read from another pre-defined segment of shared memory (the control vector). The synchronization is performed by sending and receiving a single dummy byte through a shared pipe.

The coupling is effectively achieved when there exists another code that performs the exact opposite operations, i.e. reads the status vector from the shared segment written by RELAP_MEM and writes the control vector into the segment that RELAP_MEM reads, synchronizing accordingly through the shared pipe. Not only should this code read and write data from shared memory, but it also should pass the information from the state vector to the appropriate variables of the ReaLL routines and then build the control vector that is to be written into the shared memory object out of the resulting variables computed by ReaLL. The flow of information is depicted in figure 3.

This scheme is of course faster than the Pisa method, and was successfully used to perform coupled safety transient analysis of Atucha II, which was the main objective. Nevertheless, this

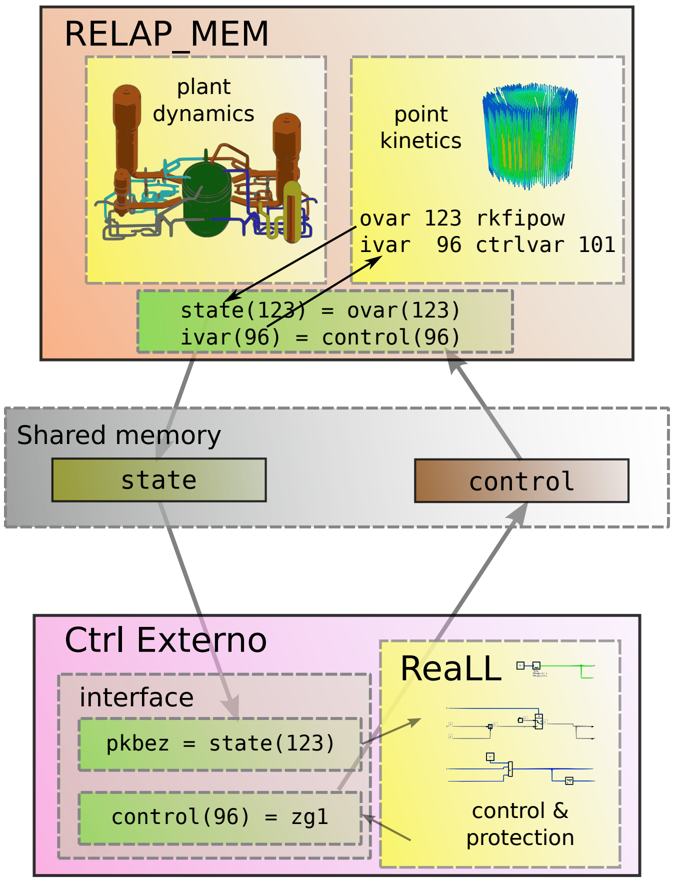


Figure 3: The coupling scheme proposed by CNEA. A modified version of RELAP called RELAP_MEM incorporates two new types of control variables “ivar” (for example the position of the G10 group of control rods zg1) and “ovar” (for example the instantaneous fission power pkbez) which are read and written into two pre-defined shared memory objects. An external code reads the state vector, passes the information into the ReaLL routines, advances one time step, builds the control vector out of the resulting Fortran variables and writes it back to the shared memory object which is finally read by RELAP_MEM. The interface routines of the external control code have to be re-written each time the elements of the status or control vector change.

approach has some limitations that, if overcame, may improve the usage of the code. For example, RELAP provides a limited number of control variables and therefore there is an inherent limit to the size of the data exchanged. Another issue is that the proposed scheme is based on pre-defined shared memory segments and synchronizes using a shared pipe, which implies that only one external code can be coupled with RELAP. Moreover, should the order of appearance of individual variables in both the state and control vector change, a manual intervention to update the interface routines in the external code is needed. A minor con is that RELAP_MEM only works for a definite version of RELAP (namely version 5 mod 3.3) and runs only under Windows.

2.3 Coupling through RELAP's fast array

All the disadvantages of the RELAP_MEM method are minor when compared to the first approach. However, this layout lacks the possibility of addding another code to the scheme, which is needed to include a spatial neutronic kinetics code. Therefore, TECNA S.A. acceded to develop a new coupling layout for NA-SA with the ability to couple an arbitrary number of external codes to RELAP.

In order to accomplish this objective, the information exchange and synchronization mechanisms should be flexible and general, instead of fixed and pre-defined. Moreover, the access to RELAP data should not be based on control variables because the already-discussed imposed limit on the number of variables prevents the temperature, densities and power distributions within the reactor core—which are needed for spatial neutron kinetics with thermalhydraulic feedback—to be successfully exchanged between the codes. In this regard, TECNA developed a modified version called RELAPCPL that besides reading the input file as usual, also reads another file known as the "coupling file." This file contains alphanumeric keywords that instruct RELAPCPL to export to (import from) an arbitrary number of shared memory objects—whose names are defined in the coupling file—an arbitrary number of volume properties (i.e. tempf, voidg, etc.), junction properties (i.e. mflowj, velgj, etc.), heat structure properties (i.e. htpown, htvatp, etc.), component properties (i.e. vliq, przlvl, etc.), control variables properties (i.e. cnvarn, cnvsan, etc.), general table properties (i.e. gtbl) and/or neutron kinetic properties (i.e. rktpow, rkrn, etc.).[1] The instantaneous value of these properties are directly taken from the internal memory of the RELAP solver, which is known as the "fast" array. When a coupling file is given using the commandline argument "-a" (which is present but not used in the original RELAP code), the modified version RELAPCPL locates the offset within the fast array where RELAP stores each property asked for in the coupling file. Then, each time step (or every k time steps, or every Δt seconds, as defined by the user in the coupling file) RELAPCPL synchronizes with the external codes using shared semaphores and exchanges information using shared memory segments as instructed in the coupling file.

This extension is implemented as a number of routines written in C that parse the coupling file at start-up and perform the actual synchronization and data exchange at each time step or as set in the coupling file. The original Fortran routines of RELAP are also slightly modified so the C functions are called when needed. The actual information exchange is performed inside the tran routine after calling the RELAP's convar subroutine that processes the control variables. The extension was applied to various versions of RELAP, namely version 5 mod 3.3, version 5 3D v2.4.2, SCDAP mod 3.3 and SCDAP mod 3.4. RELAPCPL can be compiled with a variety of compilers and run both under Windows and GNU/Linux architectures. Some details of the rationale behind the scheme can be found in Theler (2013). Figure 4 shows an excerpt from a valid coupling file for RELAPCPL.

With the extended code, complex coupled calculations involving neutronic codes and even a RELAP-RELAP coupling can be obtained, as shown in Mazzantini et al. (2011). The simple ivar-ovar coupling scheme is a particular case that RELAPCPL is able to reproduce. In effect, except for the introduction of semaphores for the synchronization, almost the same external interface code of figure 3 could be coupled to RELAPCPL, provided the appropriate coupling

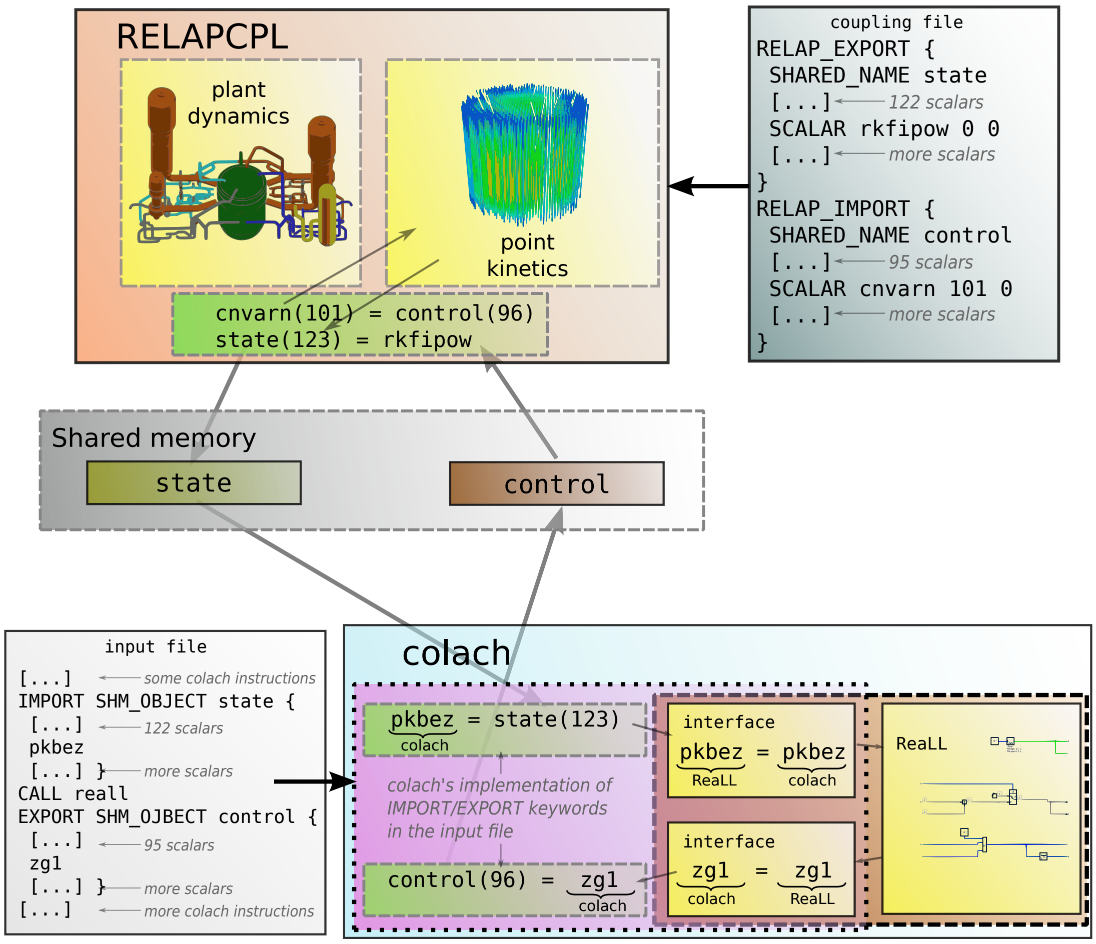
[1]The proposed nomenclature of referring to properties is taken from RELAP Programmers Manual and may be different to the usual nomenclature used when requesting minor edits. The reason is that there are some properties that may be useful to import or export—such as the scaling value of a control variable—that are not available as minor-edit requests.

```
RELAP_EXPORT {
 SHARE_NAME state
 SEMAPHORE_READY sem_state

 SCALAR cnvarn 3000 0       * estacio

 SCALAR tempf 205 04        * tka1
 SCALAR tempf 245 04        * tka2
[...]
 SCALAR cnvarn 2306 0      * l_diesel3
 SCALAR cnvarn 2307 0      * l_diesel4
}


RELAP_IMPORT {
 SHARE_NAME control
 SEMAPHORE_WAIT sem_control

 SCALAR cnvarn 9200 0     * ipkag_1
 SCALAR cnvarn 9201 0     * yk1a01_1
 SCALAR cnvarn 9202 0     * yk1a02_1
[...]
 SCALAR cnvarn 9507 0    * rises
 SCALAR cnvarn 9508 0    * mwtr1i
 SCALAR cnvarn 9509 0    * mstm2i
}
```

Figure 4: Excerpt from a RELAPCPL coupling file. An arbitrary number of exports and/or imports can be defined, each one containing an arbitrary number of RELAP properties. For exports, the properties asked for (the value of control variable 3000, the fluid temperature of the fourth cell of pipe 205, etc.) are written into the POSIX shared memory object ("state") in the given order. For imports, the properties asked for (the value of control variables 9200, 9201, etc.) are read from the POSIX shared memory object ("control"). It should be taken into account that the internal RELAP fast array is accessed. Exports will result in copies of the properties computed by RELAP. Imports will directly write the data contained in the shared memory object to the fast array, which may be overwritten again by RELAP if the imported property is a derived one. For example, to import control variables, they should be declared as "constant" in the RELAP input file, otherwise their content will be rewritten. In the same sense, to import thermalhydraulic information, the internal energy of a cell has to be read instead of the fluid temperature, as this last property is computed from the former by RELAP whenever the time step is advanced.

files are prepared. Nevertheless, in the spirit of flexibility regarding data manipulation and execution control, a different approach was chosen. Instead of writing a set of dummy interface routines that just passed information back and forth from shared memory and Fortran routines, an attempt to write a control code able to compute algebraic and differential operations to signals (such as time integrals or first-order lags) and to output them to ASCII files in an user-controlled way using plain text files with alphanumeric keywords (in the same sense of coupling files) was performed. This control code should be able, on the one hand, to exchange information with shared memory segments and to synchronize the processes involved using shared semaphores in a way completely defined by keyword written by the user in the input file. On the other hand, it should be able to execute arbitrary binary instructions given in the form of binary objects (i.e. the ReaLL routines).

Such external control code was called colach, and it provided some added flexibility to the user with respect to the external control code of figure 3. For example, it allowed the user to



```
coupling file
RELAP_EXPORT {
 SHARED_NAME state
 [...]←— 122 scalars
 SCALAR rkfipow 0 0
 [...]←— more scalars
}
RELAP_IMPORT {
 SHARED_NAME control
 [...]←— 95 scalars
 SCALAR cnvarn 101 0
 [...]←— more scalars
}
```

```
input file
[...]          ←— some colach instructions
IMPORT SHM_OBJECT state {
 [...]         ←— 122 scalars
 pkbez
 [...] }       ←— more scalars
CALL reall
EXPORT SHM_OJBECT control {
 [...]         ←— 95 scalars
 zgl
 [...] }       ←— more scalars
[...]          ←— more colach instructions
```

```
cnvarn(101) = control(96)
state(123) = rkfipow
```

Figure 5: The colach-based approach to coupling RELAP with the ReaLL routines. The possibility of defining how the shared memory objects are accessed and how the output is generated using an input file instead of hard-coding instructions in the executable gives the user more flexibility than the “ivar-ovar” approach shown in figure 3. The colach’s input file can be edited by the user and is expected to mirror RELAPCPL’s coupling file. However, the interface routines that transfer values between colach and ReaLL have to be manually updated by the user should the state or control vectors change. The ReaLL routines coded in FORTRAN 77 are statically linked into the colach executable, so a complete re-compilation is needed also if some parameter that is not contained into the exchange vector is to be changed for a certain computation or a change in the ReaLL routines is needed for a given transient.

control the access to shared objects and the output text files from a keyword-based input file that was read at run-time instead of having to re-compile the code each time a modification was needed. The colach approach is depicted in figure 5. Apart from performing safety transient analysis for FSAR Chapter 15, the coupling scheme can be used for other applications. Indeed, the University of Pisa employed the colach-based approach to allow RELAP 3D’s embedded neutronic code NESTLE to efficiently read a time-dependent distribution of boron inside the moderator tank of Atucha II computed by CFD calculations. The resulting reactivity was then used to compute the actual safety transients using point kinetics.

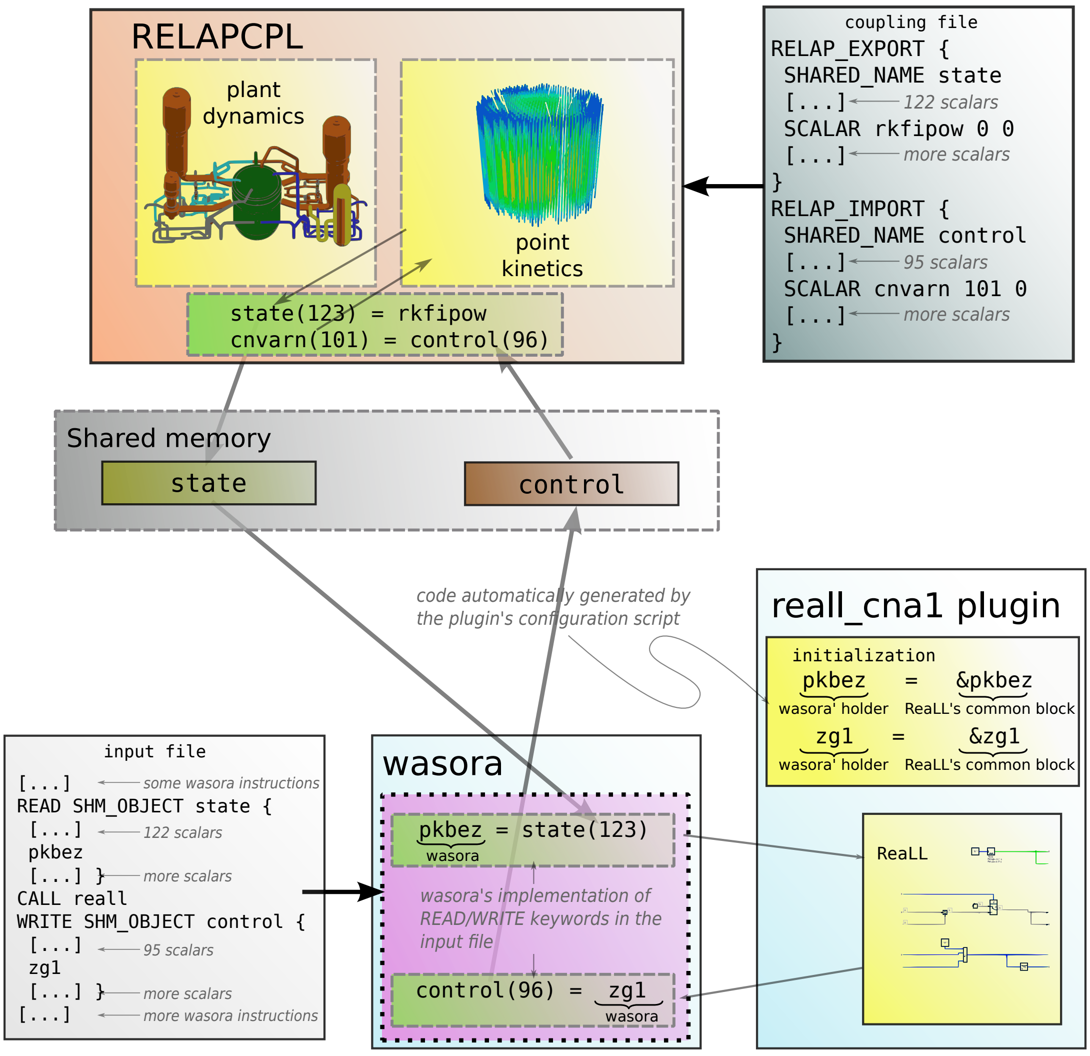
The ReaLL routines that emulate the limitation and protection systems are compiled into object files which are statically embedded into the colach executable. These routines are based on the code DYNETZ developed by KWU for the Atucha II project, which was itself a continuation of the code NLOOP, applicable to German Convoy reactor design. In this layout, the perturbations that trigger a certain transient situation (e.g. a pump failure) are supposed to be hard-coded into the Fortran source code (e.g. by changing the status of a flag) and the corresponding executable should be re-compiled. Thus, some computations are performed with one executable and some others with another one. The colach-ReaLL solution developed by TECNA included a simple difference-tracking system that reported either to the screen or to the output files which changes a particular binary contained with respect to a certain base source tree. In any case, this coupling scheme was successfully utilized by NA-SA and PISA to study the postulated transients in Chapter 15 of Atucha II's FSAR using RELAP 3D as the plant code.

3 PROPOSED COUPLED CALCULATION LAYOUT

As stated in the introduction, NA-SA is updating the Atucha I's FSAR using state-of-the-art codes, models and methods. With the objective of designing a robust coupled layout that could be used for a wide variety of applications, TECNA decided to tackle the design flaws of the colach approach and proposed a similar but slightly different scheme. First, the code wasora replaced colach. The new code follows the same basic ideas but it was re-written almost from scratch shifting complexity from algorithms to data structures, as recommended by the rule of representation of the UNIX Philosophy (Raymond, 2003), which instructs to "fold knowledge into data so code logic can be stupid and robust." These features include pointers to structures, linked lists, hashed tables, numerical libraries, a debugger-like interactive interface and the possibility to load dynamic plugins at run-time.

It is this last feature that introduces most of the novelty of the scheme with respect to the previous approach taken by NA-SA. In this layout, the Fortran ReaLL routines are compiled along with a few C functions that act as fixed entry points of a shared library which can be dynamically loaded at runtime from wasora. Apart from this fact, the main difference with the colach-based approach discussed in section 2.3 is that the plugin's configuration script that builds the makefiles is able to parse the Fortran routines (provided they follow the DYNETZ standards) and detect the storage location of all the global variables, vectors and matrices—i.e. the ones stored inside common blocks—which are involved in the computation of the ReaLL routines. Therefore, after loading the plugin, the initialization entry point function dynamically defines equivalent variables, vectors and matrices that are directly accessible from the wasora input file and whose value holder points directly to the location of the corresponding Fortran symbols within the common blocks. Therefore, any symbol that is relevant for the ReaLL computation—whether it belongs to the state or control vector or not—can be accessed either from wasora or from Fortran without needing to explicitly copy the values when advancing the time step. Any variable can be initialized from a Fortran block data, be assigned from the wasora input—optionally getting its value from an algebraic expression, an ASCII or binary file or, of course, a shared-memory object—and be transparently used inside a Fortran routine. Every variable can be an input and every variable can be an output. The scheme is depicted in figure 6.

With this approach, the details of the coupling scheme are completely defined by instructions contained both in RELAPCPL's coupling file and in wasora's input file. Besides, the content of any global symbol can be accessed from the wasora input, its value overwritten using wasora's facilities (which include algebraic expressions, one and multi-dimensional interpolation of point-wise defined functions, numerical integration and differentiation, etc.) and/or translated into an ASCII representation via a user-defined C's printf-compatible format string and written


RELAPCPL
plant dynamics
point kinetics
state(123) = rkfipow
cnvarn(101) = control(96)
Shared memory
state
control
code automatically generated by the plugin's configuration script
reall_cna1 plugin
initialization
pkbez = &pkbez
wasora' holder ReaLL's common block
zgl = &zgl
wasora' holder ReaLL's common block
ReaLL
wasora
pkbez = state(123)
wasora
wasora's implementation of READ/WRITE keywords in the input file
control(96) = zg1
wasora

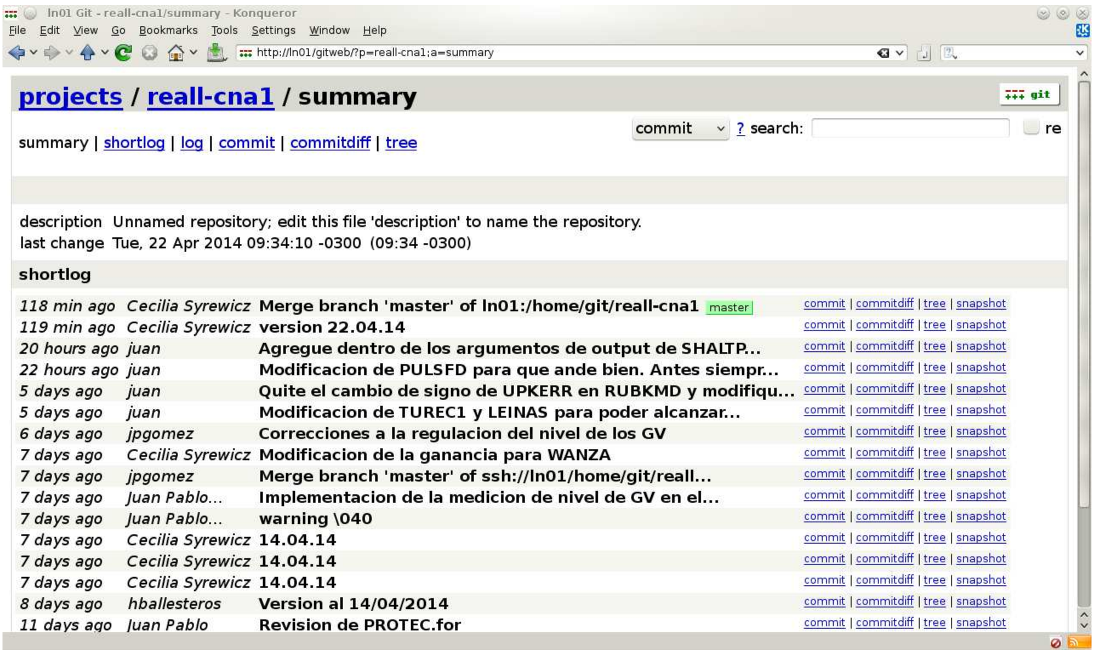
```
coupling file
RELAP_EXPORT {
 SHARED_NAME state
 [...]← 122 scalars
 SCALAR rkfipow 0 0
 [...]← more scalars
}
RELAP_IMPORT {
 SHARED_NAME control
 [...]← 95 scalars
 SCALAR cnvarn 101 0
 [...]← more scalars
}
```

```
input file
[...]      ← some wasora instructions
READ SHM_OBJECT state {
 [...]     ← 122 scalars
 pkbez
 [...] }   ← more scalars
CALL reall
WRITE SHM_OBJECT control {
 [...]     ← 95 scalars
 zg1
 [...] }   ← more scalars
[...]      ← more wasora instructions
```

Figure 6: The new proposed method to couple RELAP with the ReaLL routines. The RELAPCPL side is identical to the colach case (figure 5). The improvement resides in the way the ReaLL routines are incorporated into the control code wasora, which reads the pkbez variable from the shared-memory object as before, but does not need to explicitly pass the value to ReaLL because the holder of wasora's pkbez variable points to the memory address of the Fortran's pkbez variable, and conversely for zg1. The need of recompilation is thus reduced. The ReaLL plugin also takes care of the low-level implementation of the restart mechanisms. Information about history and changes in the Fortran routines is included into the binary and can be reported at run-time (see figure 8).

into a file, whose name can also contain evaluated expressions in its name. Under these considerations, the cases where re-compiling the ReaLL plugin is mandatory are highly reduced. These few situations include fictitious (but sometimes needed) cases where a ReaLL routine has to be bypassed for example to simulate failure of the actuation of the control rods; sometimes a limitation signal has to be reset before and after calling a ReaLL routine and a change in the Fortran source is needed. Nevertheless, further improvements may even remove this drawback.

In the proposed layout, the C functions of the plugin use Bazaar as a version control system and the Fortran source tree of the ReaLL routines use the Git version control system. Not only


ln01 Git - reall-cna1/summary - Konqueror
File Edit View Go Bookmarks Tools Settings Window Help
http://ln01/gitweb/?p=reall-cna1;a=summary
git
commit ? search: re

projects / reall-cna1 / summary

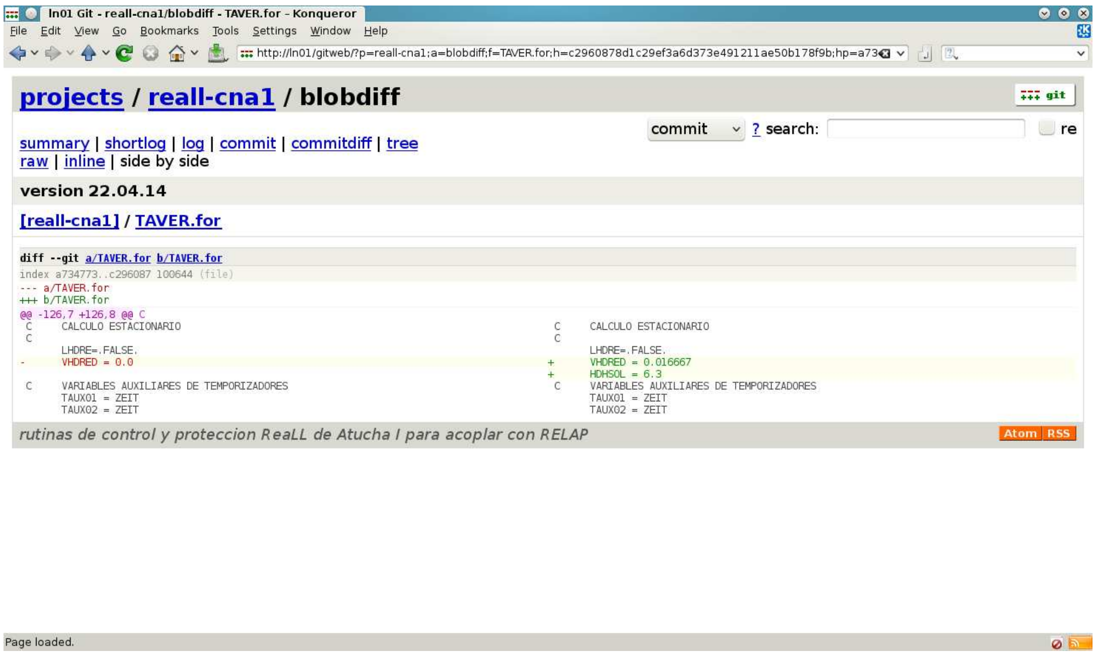
summary | shortlog | log | commit | commitdiff | tree

description Unnamed repository; edit this file 'description' to name the repository.
last change Tue, 22 Apr 2014 09:34:10 -0300 (09:34 -0300)

shortlog

118 min ago	*Cecilia Syrewicz*	**Merge branch 'master' of ln01:/home/git/reall-cna1** master	commit | commitdiff | tree | snapshot
119 min ago	*Cecilia Syrewicz*	**version 22.04.14**	commit | commitdiff | tree | snapshot
20 hours ago	*juan*	**Agregue dentro de los argumentos de output de SHALTP...**	commit | commitdiff | tree | snapshot
22 hours ago	*juan*	**Modificacion de PULSFD para que ande bien. Antes siempr...**	commit | commitdiff | tree | snapshot
5 days ago	*juan*	**Quite el cambio de signo de UPKERR en RUBKMD y modifiqu...**	commit | commitdiff | tree | snapshot
5 days ago	*juan*	**Modificacion de TUREC1 y LEINAS para poder alcanzar...**	commit | commitdiff | tree | snapshot
6 days ago	*jpgomez*	**Correcciones a la regulacion del nivel de los GV**	commit | commitdiff | tree | snapshot
7 days ago	*Cecilia Syrewicz*	**Modificacion de la ganancia para WANZA**	commit | commitdiff | tree | snapshot
7 days ago	*jpgomez*	**Merge branch 'master' of ssh://ln01/home/git/reall...**	commit | commitdiff | tree | snapshot
7 days ago	*Juan Pablo...*	**Implementacion de la medicion de nivel de GV en el...**	commit | commitdiff | tree | snapshot
7 days ago	*Juan Pablo...*	**warning \040**	commit | commitdiff | tree | snapshot
7 days ago	*Cecilia Syrewicz*	**14.04.14**	commit | commitdiff | tree | snapshot
7 days ago	*Cecilia Syrewicz*	**14.04.14**	commit | commitdiff | tree | snapshot
7 days ago	*Cecilia Syrewicz*	**14.04.14**	commit | commitdiff | tree | snapshot
8 days ago	*hballesteros*	**Version al 14/04/2014**	commit | commitdiff | tree | snapshot
11 days ago	*Juan Pablo*	**Revision de PROTEC.for**	commit | commitdiff | tree | snapshot

(a)


ln01 Git - reall-cna1/blobdiff - TAVER.for - Konqueror
File Edit View Go Bookmarks Tools Settings Window Help
http://ln01/gitweb/?p=reall-cna1;a=blobdiff;f=TAVER.for;h=c2960878d1c29ef3a6d373e491211ae50b178f9b;hp=a73
git
commit ? search: re
Atom RSS
Page loaded.

projects / reall-cna1 / blobdiff

summary | shortlog | log | commit | commitdiff | tree
raw | inline | side by side

version 22.04.14

[reall-cna1] / TAVER.for

```
diff --git a/TAVER.for b/TAVER.for
index a734773..c296087 100644 (file)
--- a/TAVER.for
+++ b/TAVER.for
@@ -126,7 +126,8 @@ C
 C     CALCULO ESTACIONARIO                              C     CALCULO ESTACIONARIO
 C                                                      C
       LHDRE=.FALSE.                                          LHDRE=.FALSE.
-      VHDRED = 0.0                                     +      VHDRED = 0.016667
                                                       +      HDHSOL = 6.3
 C     VARIABLES AUXILIARES DE TEMPORIZADORES            C     VARIABLES AUXILIARES DE TEMPORIZADORES
       TAUX01 = ZEIT                                          TAUX01 = ZEIT
       TAUX02 = ZEIT                                          TAUX02 = ZEIT
```

rutinas de control y proteccion ReaLL de Atucha I para acoplar con RELAP

(b)

Figure 7: Web-based interface of the Git distributed version control system showing (a) the history of changes of Atucha I's ReaLL routines and (b) the differential change of a particular routine. The system is designed at coping with collaborative development and eases the management and tracking of changes in the code. For example, all the developers get an e-mail notification each time a new commit is pushed into the repository.

```
$ wasora -p reall_cna1 --version
reall_cna1 0.2.58 reall (2014-06-23 15:28:48 -0300 clean)
CNA1 ReaLL routines

 plugin bzr branch gtheler@tecna.com-20140623182848-nq7wrsjispzb2vtz
 plugin last bzr commit on 2014-06-23 15:28:48 -0300 (rev 58 clean)
 plugin last build on  2014-07-04 14:14:59 -0300

 reall_cna1 last git commit log message:
commit 97e368b3e452047b247503f03fe0e9a6d08bb8dc
Author: Juan Pablo Gomez Omil <jpgomezomil@debian>
Date:   Thu Jul 3 17:20:37 2014 -0300

    Incorporacion de la memoria del estado de las bombas previo a NOTSTR



--------8<------- git diff output --------8<-------
diff --git a/BKIN.for b/BKIN.for
index c6f01f7..aeb2349 100644
--- a/BKIN.for
+++ b/BKIN.for
@@ -235,7 +235,7 @@ C
      &  0.412588,0.204799,0.000000,-1.055599/
 C
 !Temperature reactivity coefficient (coolant and moderator)
-      DATA ALFA_COOLT, ALFA_MODT /2.73E-5, 8.57E-5/
+      DATA ALFA_COOLT, ALFA_MODT /2.51E-5, 8.42E-5/
 C
 !Density reactivity coefficient (only moderator)
       DATA ALFA_MODD /13.633E-5/
diff --git a/POTRE1.for b/POTRE1.for
index 0bba233..35ff2ab 100644
--- a/POTRE1.for
+++ b/POTRE1.for
@@ -162,6 +162,10 @@ C
       DIF3   = DIFF2 - (DIFF2 - DIF3)*EXP(-DZEIT/TAUPHI)
       DIF33  = DIF3*V3
       DIFF3  = DIF1 + DIF2 - DIF33
+C&&&&&&&&&
+! gth: eliminamos el control de potencia temporalmente
+      DIFF3 = 0.0
+C&&&&&&&&&
 C
 C     LA POTENCIA ES ALTA
       LEIH1  = .FALSE.
--------8<--------------8<----------------8<-------



 compiled on 2014-07-04 14:16:02 by gtheler@barnie (linux-gnu x86_64)
 with gcc (Debian 4.7.2-5) 4.7.2 using -g -O0
  and GNU Fortran (Debian 4.7.2-5) 4.7.2 using -g -O0 -fdefault-real-8





---------        -------       -----     ----    ----
wasora 0.2.141 trunk (2014-07-03 20:50:36 -0300 clean)
wasora’s an advanced suite for optimization & reactor analysis

 branch jeremy@tom-20140703235036-62ttgmvalw4k99pc
 last commit on 2014-07-03 20:50:36 -0300 (rev 141 clean)
 last build on  2014-07-04 14:15:00 -0300

 compiled on 2014-07-04 14:16:02 by gtheler@barnie (linux-gnu x86_64)
 with gcc (Debian 4.7.2-5) 4.7.2 using -g -O0 and linked against
  GNU Scientific Library version 1.15
  GNU Readline version 6.2

 wasora is copyright (C) 2009-2014 jeremy theler
 licensed under GNU GPL version 3 or later.
 wasora is free software: you are free to change and redistribute it.
 There is NO WARRANTY, to the extent permitted by law.
```

Figure 8: The information about the branch history of the Fortran routines used to compile the ReaLL plugin and the changes of the local working tree with respect to the last commit are shown when run with the “-v” (show version) commandline option. The diff output is highlighted with colors in GNU/Linux architectures. Using the reported hashes of the base code (wasora), the plugin (reall_cna1) and the Fortran source (ReaLL), the actual source tree and its history can be completely tracked back.

does this approach ease the concurrent development of both the plugin and the Reall routines performed by a different number of experts of TECNA and NA-SA (figure 7), but it also allows to incorporate information about the history of the Fortran source and its modification with respect to the master repository into the binary object. This way, when a particular instance of the plugin is loaded, the user can track back the history of both the plugin and the ReaLL routines, locate the actual snapshot of source code that was used to generate it through the hashes reported by the version control system and quickly identify the uncommitted changes that correspond to the particular physical disturbance introduced into the code to model a certain transient. All this information is embedded into the binary object, and is shown in the standard output when wasora is instructed to load the plugin and called using the "-v" command-line option used to report its version (figure 8). This feature represents a great improvement in terms of code traceability over the simple diff-based solution of the colach approach, not to mention the previous works in which changes in the Fortran source had to be manually merged and updated by the user that performed the transient analysis run.

Moreover, as the Fortran source code is parsed and the actual composition of the global common blocks can be known, a robust restart system can be implemented. In effect, the plugin provides new keywords that instruct the code to read or write restart records from files in a flexible way, as illustrated in figure 9. In the example, whether a new problem has to be started or a previous state has to be loaded is defined by RELAP's cards 100 and 103, whose contents is exported by the coupling extension into shared memory, read by wasora and stored in the variables called run_num, problemtype05 and restartnum (these instructions are given in the file input.was). During the transient calculation, whenever RELAP writes a restart record itself,

```
end_time = infinite   # run until RELAP says done = true
NUMBER restart = 2    # numerical code for restart-type problems in RELAP

INCLUDE input.was     # read data from RELAP

# define some wasora file identifiers
INPUT_FILE restart_last      "cna1.%02.0f.rst" run_num-1
INPUT_FILE restart_current   "cna1.%02.0f.rst" run_num

# if RELAP says the problem type is restart, read the asked restart record
IF problemtype05=restart
  DYNETZ_RESTART READ FILE restart_last REGISTER restartnum RESET_TIME run_num<3.1
ENDIF

# if RELAP writes a restart record, so should we
IF iscallrstrec0
  PRINT TEXT "\#␣Saving␣restart␣number" %02.0f count
  DYNETZ_RESTART WRITE FILE restart_current REGISTER count
ENDIF

DYNETZ_STEP           # advance one step of ReaLL

INCLUDE output.was    # write data from RELAP

PRINT t dt pbez surho
```

Figure 9: Example of a valid wasora input file (provided the reall_cna1 plugin was loaded with the "-p" commandline option) that illustrates how the restart records can be flexibly accessed using high-level plain-text keywords. The actual information exchange is performed in the included files input.was and output.was. The keywords that start with "DYNETZ_" are provided and interpreted by the plugin.

the flag iscallrstrec0 is set to true and ReaLL is instructed to save a restart record too. Not only is the proposed restart handling far more flexible than the previous works (up to and including the colach approach), where the user had to manually change a reference to the size of the common blocks if a variable was added or deleted, but also a run-time check is automatically performed when loading restart records. This way, if the plugin is instructed to load a record from a file which was generated with a different number of variables contained in the common blocks, the inconsistency will be detected and the execution will stop with an error message. Previous implementation would not detect such mismatch and proceed to load a binary blob of data that would produce a shift of the memory contents resulting in corrupted values for the individual variables.

To sum up, in the current coupling method, the plant is modeled by RELAP which is extended to export a state vector plus other administrative information (current time, time step, flags that indicate the problem type, flag to write restart records, etc.) into a shared memory object. If a proper wasora code mirroring the shared object definition of RELAPCPL's coupling file is to be constructed, then the details of how the information is exchanged are not relevant for the final user, who has to pay attention to what information is to be printed into the standard output and what is to be written into other output files. The execution can be stopped at any point either manually or automatically by setting conditional breakpoints, to enter an interactive debugger-like interface provided by wasora which can be used to watch the instantaneous values of variables, vectors and matrices. If a more detailed debugging session is needed, an external debugger can be attached to the wasora process so the individual instructions of the ReaLL routines can be advanced and inspected step-by-step at the Fortran-instruction level.

3.1 Spatial neutron kinetics

For those transient cases where it is important to take into account the spatial dependance of either the power distribution or the liquid absorbers (e.g. diluted boron injected by the second shutdown system) within the core, the introduction of a neutronic code capable of computing three-dimensional core-level transient problems is needed. Previous works performed by TECNA used an approach similar to the coupling-file RELAP extension and existing neutronic codes were modified to be able to exchange information through shared memory objects. However, following the plugin-based approach of incorporating particular calculation codes into the general wasora framework, the coupling mechanisms of a neutronic code can be generalized by implementing it as a wasora plugin. This way, a file-based (i.e PISA's) or a shared-object-based coupled scheme (i.e. RELAP5CPL) can be obtained by importing and exporting the plugin's internal variable using wasora's READ/WRITE keywords with the FILE or SHM_OBJECT arguments (and optionally synchronizing with the SEM keyword), respectively. If the other code that needs neutronic information is implemented also as a wasora plugin, then the information can be exchanged by explicitly assigning the variables of one plugin to the content of the other one.

In particular, for some cases contained in Chapter 15 of the Atucha I FSAR, the second shutdown system is demanded. This system injects a solution of deuteroboric acid into the moderator tank from high-pressure accumulators through three nozzles. The resulting spatial and temporal distribution of boron concentration within the core is not trivial, and a detailed calculation using three-dimensional CFD techniques with a spatial discretization mesh is needed (figure 10). To incorporate this information into the neutronic code, a condensation of such fine grid into the coarser grid of the core-level neutronic code is required. In the case of Atucha I, this

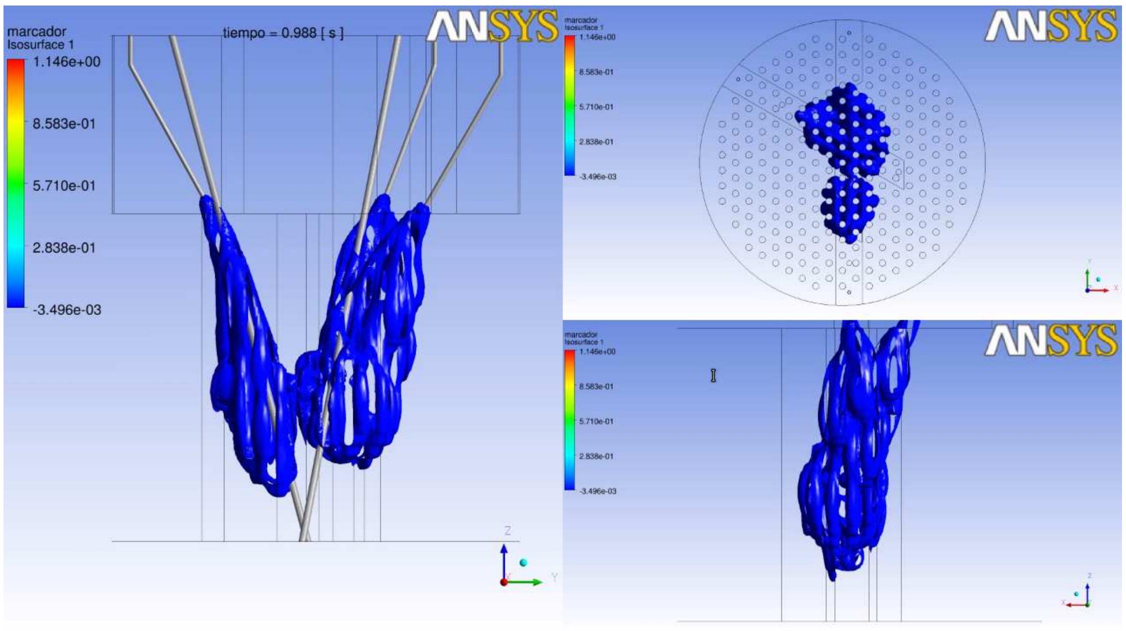


Figure 10: Computation of the transient spatial distribution of boron within the moderator tank after the demand of the reactor second shutdown system using computational fluid-dynamic techniques with the finite volume method with three million cells.

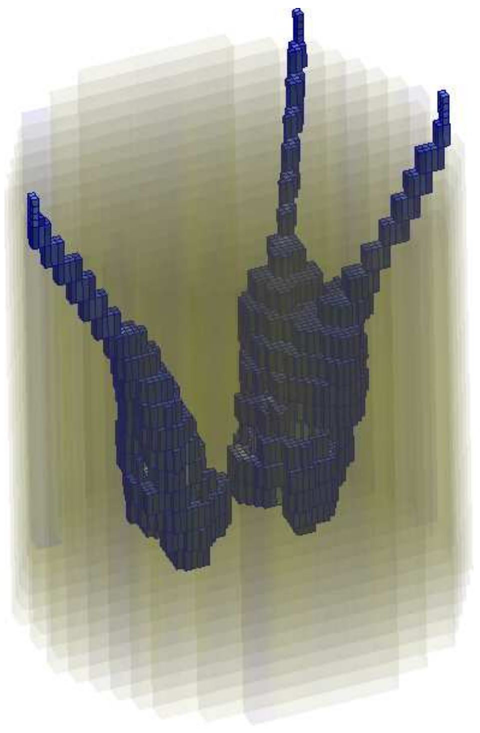


Figure 11: Incorporation of the CFD results of figure 10 into the calculation grid of the neutronic code with approximately two hundred thousand cells. A set of scripts condense the results in the fine grid into the coarse one, generating a set of ASCII files which can be read and interpolated using the wasora framework as shown in figure 12.

```
# number of files with the boron distribution (one per time step)
NUMBER ntimes 271
# these two vectors contain the spatial distribution of boron
# in the neutronic code's calculation mesh of size nx*ny*nz
VECTOR cloud_last    SIZE nx*ny*nz
VECTOR cloud_current SIZE nx*ny*nz
# this vector contains the instantaneous times at which the
# boron distribution was computed
VECTOR times         SIZE ntimes

# nsub and ncore are a mesh refinement factor
# $3 is read from commandline and indicates the id of the case
INPUT_FILE times  cloud/times-$3-%gx%g-%g.dat         nsub nsub ncore
INPUT_FILE cloud  cloud/cloud-$3-%gx%g-%g-%04.0f.dat  nsub nsub ncore times(k+1)

# read the first distribution file
IF in_static
 READ ASCII_FILE cloud  cloud_last
 READ ASCII_FILE times  times
ENDIF

end_time = 2.5
dt = 1e-2
k_0 = 1

# if time advanced enough, read the next file
IF 1000*t>times(k+1)
 CLOSE cloud
 k = k + 1
 cloud_last(i) = cloud_current(i)
 READ ASCII_FILE cloud  cloud_current
ENDIF

# linearly interpolate between the current and the last files
xi = (1000*t-times(k)) / (times(k+1)-times(k))
t_interp = (1-xi) * times(k) + xi * times(k+1)
vec_calcboron(i) = 3800 * ((1-xi) * cloud_last(i) + xi * cloud_current(i))

# advance one step of the spatial kinetics neutronic computation
PLUMITA_STEP
```

Figure 12: Excerpt from a wasora input with the plumita neutronic plugin that reads the instantaneous boron distribution from a series of discrete ASCII files containing a vector of size $n_x \times n_y \times n_z$ (order of magnitude 2×10^5). The actual boron distribution used in the transient neutronic calculation is linearly interpolated between the current and the previous file as the time step advances. The neutronic plugin provides the PLUMITA_STEP instruction and maps the wasora's vector named vec_calcboron to the internal vector that holds the boron distribution for the neutronic computation. The rest of the instructions are provided by the wasora framework.

step is performed by using a set of awk scripts that write the boron distribution within the neutronic mesh as one ASCII file for a number of sampling time steps (figure 11). If the neutronic code is implemented as a wasora plugin, this information can be easily read and incorporated into the low-level vectors needed by the calculation code with the high-level instructions provided by the wasora code. Figure 12 illustrates one way of reading the discrete boron clouds contained in the ASCII files and interpolating them so they can be evaluated at any arbitrary time t, which in a coupled calculation is to be determined by RELAP according to the transient termalhydraulic conditions. If, on the other hand, a classical point kinetics calculation is needed instead of a full three-dimensional coupled computation, the proposed method of incorporating the transient boron distribution into the neutronic code can be still be used to flexibly perform an off-line computation of the boron reactivity-worth reactivity curve as a function of time to be incorporated into RELAP's point kinetics equations.

4 CONCLUSIONS

From the authors' point of view, the proposed coupling scheme based on both shared-memory objects and run-time dynamically loadable plugins gives far more flexibility and traceability than the previous methods provided. The advantages overcome the added complexity to the methodology of calculation and analysis of accidents. The collaborative development of the ReaLL routines of Atucha I amongst several experts from NA-SA and TECNA was greatly simplified by the implementation of a distributed version control system, that also allowed to track full histories and particular modifications for the execution of transient problems. Writing the low-level implementation of the restart mechanism in C from a parsed representation of the Fortran common blocks allows to shift the focus of the end user away from the programming details toward the analysis and debugging of results. The automatic checking of the format of the binary restart records prevents the execution of ill-conditioned problems which may result in unfruitful engineering time. Finally, the flexibility that the wasora framework provides for reading data from assorted source not only allows coupled calculations to be completely defined by plain-text input files but also complex computations such as the evaluation of the reactivity worth of a transient boron cloud as computed with CFD techniques can be performed with relatively low effort. As flexible as the RELAPCPL extension is as compared with previous works, the plugin-based approach can be further exploited by re-writting RELAP as a wasora plugin so fluid properties can be directly mapped to wasora variables, which can then be further accessed by other calculation codes. For example, there exist a wasora plugin that is able to execute arbitrary Python code, providing even further flexibility and extensibility, which was the main objective of the proposed coupling scheme for deterministic safety assessment of nuclear power plants.

REFERENCES

Maciel F. Coupling the RELAP code with external calculation programs (shared memory version). Technical Report NUREG/IA-0405, U.S. Nuclear Regulatory Commission, 2011.

Mazzantini O., Schivo M., Di Cesare J., Garbero R., Rivero M., and Theler G. A coupled calculation suite for Atucha II operational transients analysis. *Science and Technology of Nuclear Installations*, 2011:785304, 2011.

Raymond E.S. *The Art of UNIX Programming*. Addison-Wesley, 2003.

Theler G. A shared-memory-based coupling scheme for modeling the behavior of a nuclear power plant core. *Mecanica Computacional*, XXXII(18), 2013.