A coupling strategy and its software implementation for waves propagation and their impact on coast

C. Kassiotis[1,2], B. Rogers[2] and P. Stansby[2]

February 5, 2010

1 Introduction

The complexity of flows, and especially flows at large scales such as the ones of wave propagation through ocean make the introduction of simplified model natural [20, 29, 30]. Since the XIX$^{\text{th}}$ century, and through the XX$^{\text{th}}$, models such as Saint-Venant [8], Boussinesq [4, 27] and fully Non-Linear Shallow Water equations (NLSW) [12, 10] propose satisfying results in their range of application (from deep to shallow water).

They are however, by definition, unable to represent accurately the complexity of the flow near the coast, when waves are sloshing. The sloshing of the wave is often handle by energy dissipation models in the near coast area (sponge layer), with often heuristic and barely physical, whose coefficient are tuned on simple case. In [11], a study compares for instance the results from analytical, NLSWE software and two-phase slightly compressible flows solved by VOF strategy for the classical dam break problem, and show the necessity of advanced models for this kind of application.

Concerning these advanced models, the difficulty encountered is to represent complex free-surface, evolving in time, with possible multi-connected domain. Among all the strategy proposed to over-come this difficulty, one can cite the Volume-Of-Fluid (VOF) approach that propose to introduce a characteristic function describing high and low density fluid domains (here water and air) and rely on a Finite Volume discretization [14, 28, 9]. Another discretisation widely used is the Smooth Particle Hydrodynamics method (SPH) that inherently handle the complexity of some free surface flows hence gridless by definition. If the wave propagation is possible with these models, the high computation cost made same possible only on 2D domains. Furthermore, they often suffer of over damping, and the waves are dissipated before reaching the coast if no proper treatment is apply.

For these reasons the coupling between any of the wave propagation models and complex free-surface flow strategies seems the right way to takle this problem [17, 21]. One of the idea behind our work will be to re-use existing codes

in order to avoid the long development and validation phase. We propose to couple a Boussinesq code with a SPH software.

The outline of this paper is the following. In the first part we present the algorithms employed in the code re-used for the coupling. In the second part, the coupling algorithm as well as the software implementation is given.

2 Coupled subproblems

	name	licence	language	2d	3d	val
waves	`BSQ_V2P3` (EDF R&D)		Fortran	✓		✓
	funwave	GPL	Fortran	✓	✓	✓
	shallowWaterFoam	GPL	`C++`	✓	✓	
SPH	Spartacus		Fortran	✓	✓	✓
	SPHysics	GPL	Fortran	✓	✓	✓

Table 1: Software suitable for waves propagation and SPH

Remarks:

- shallowWaterFoam is part of OpenFoam 1.6
- GPU-SPHysics is programmed in `C++`
- Spartacus will be realed as GPL for Linux in a near future (already available for Windows)
- `BSQ_V2P3` is a prototype code

2.1 `BSQ_V2P3`: equations, discretization and solving algorithm

In this section we present the main algorithm of the code and some results of a benchmark given by M. Benoit.

The results given in Fig. 1 are obtained using the plot method of the component.

2.2 `SPHysics`: equations, discretization and solving algorithm

3 Coupling strategy and its software realisation

3.1 Coupling algorithm

In [21], an explicit staggered coupling algorithm between Boussinesq wave and a SPH model is proposed. It requires a coupling domain. The Boussinesq solver advance in time with a given time step Δt_{Bsq}. From the Boussinesq solver you

Algorithm 1 `BSQ_V2P3`

Given:
for $n = 1 \dots N_{\max}$ **do**
  initialize iteration counter $(k) = 0$
  predictor (explicit, order 3):

$$\begin{cases} \eta_{i,n+1}^{(0)} &= \eta_{i,n} + \dfrac{\Delta t}{12}(23E_{i,n} - 16E_{i,n-1} + 5E_{i,n-2}) \\ U_{i,n+1}^{(0)} &= U_{i,n} + \dfrac{\Delta t}{12}(23F_{i,n} - 16F_{i,n-1} + 5F_{i,n-2}) \end{cases}$$

  compute velocity $u_{i,n+1}^{(0)}$ from $\hat{U}_{i,n+1}^{(0)}$
  compute $E_{n+1}^{(0)} = E(\eta_{i,n+1}^{(0)}, u_{i,n+1}^{(0)})$ and $F_{n+1}^{(0)} = F(\eta_{i,n+1}^{(0)}, u_{i,n+1}^{(0)})$
  while $\Delta\eta > 0.0001$ or $\Delta u > 0.0001$ **do**
    corrector (explicit, order 4):

$$\begin{cases} \eta_{i,n+1}^{(k+1)} &= \eta_{i,n} + \dfrac{\Delta t}{24}(9E_{i,n+1}^{(k)} + 19E_{i,n} - 5E_{i,n-1} + E_{i,n-2}) \\ U_{i,n+1}^{(k+1)} &= U_{i,n} + \dfrac{\Delta t}{24}(9E_{i,n+1}^{(k)} + 19F_{i,n} - 5F_{i,n-1} + F_{i,n-2}) \end{cases}$$

    compute velocity $u_{i,n+1}^{(k)}$ from $\hat{U}_{i,n+1}^{(k)}$
    compute $E_{n+1}^{(k)} = E(\eta_{i,n+1}^{(k)}, u_{i,n+1}^{(k)})$ and $F_{n+1}^{(k)} = F(\eta_{i,n+1}^{(k)}, u_{i,n+1}^{(k)})$
    compute error indicator:

$$\Delta\eta = \frac{\sum_i |\eta_{i,n+1}^{(k)} - \eta_{i,n+1}^{(0)}|}{\sum_i |\eta_{i,n+1}^{(k)}|} \quad \text{and} \quad \Delta u = \frac{\sum_i |u_{i,n+1}^{(k)} - u_{i,n+1}^{(0)}|}{\sum_i |u_{i,n+1}^{(k)}|}$$

    $(k) \longleftarrow (k+1)$
  end while
end for

can get the velocity profile, that is imposed at one of the boundary of the SPH solver. For stability reasons, the SPH solver advances in time with smaller time step, denoted Δt_{SPH}. From the SPH solver you can obtain the velocity at the reference depth and the wave height.

This algorithm is described in Alg. **??**

3.2 Communication between software – Component Oriented Programming paradigm

The need to re-use existing software products and tools in a more general context is a major trend of software engineering. In the famous Garmish (Germany) NATO conference that took place in 1968, the first stones of modern software

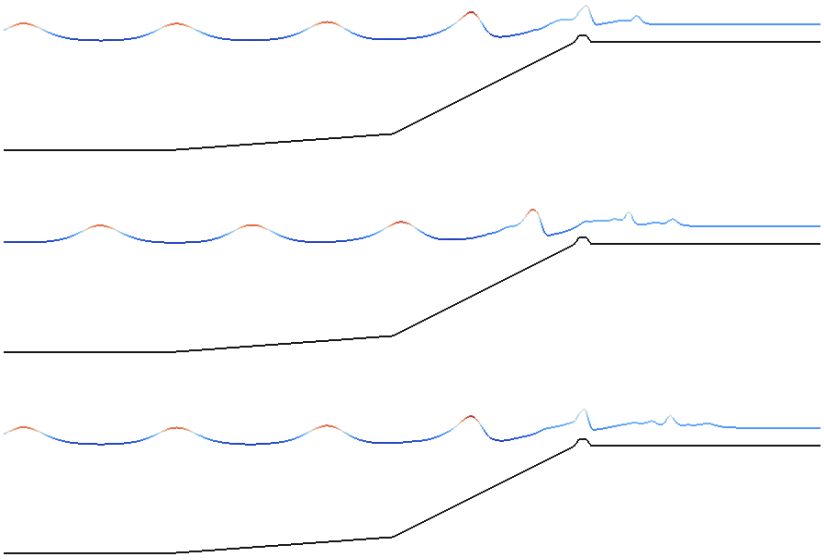

Figure 1: Free surface flows modeled `BSQ_V2P3`: bathymetrie and wave height are scaled by 10. Snapshot at time steps 100, 125 and 150.

engineering where laid, and some answers to the problems described previously, and more generally, to the so-called Software Crisis were given [25]. In [19] for instance, the notion of components and their role in software re-use was introduced for the first time, but a proper definition is yet to be accepted by everybody. For instance, a loose definition of components as entities which can be used, possibly with modifications, in a new software development can be found in [7, 3]. With a more specific meaning, in [31] ones defines a component as a piece of software capable of performing certain tasks prescribed within its interface and which to that it is able to communicate with other components. In the present case, latter definition is considered as the more stringent.

This paradigm somehow extends the concept of object in object-oriented programming with the notion of communication between components. The generalization of the definition of class methods – for the oriented-object programming – equals here an interface listing the tasks it can execute on input data or on its attribute. This interface can be seen as the contract that links the two parties: components and clients. Another important feature of components is that they are deployed on a system. Or as statted in [31]:

> Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system

Thus, in order to be considered as a component, binaries should obey the five following criteria as specified in [22]:

Algorithm 2 Explicit coupling between Boussinesq and SPH solvers (2D)

Given:
for $n = 1 \dots n_{\max}$ **do**
  $t = n\Delta t_{\text{Bsq}}$
  impose wave height and velocity at reference height on the Boussinesq boundary $(\eta_b, v_{\alpha,b})$
  solve fluid problem with time step Δt_{Bsq}
  get velocity from Boussinesq solver at points (x_i, z_i) on the SPH boundary:

$$\begin{aligned} u(x_i, z_i) &= u_\alpha(x_i) + \partial_x z_\alpha \partial_x h u_\alpha + (z_\alpha - z_i)\partial_x^2 h u_\alpha + \\ & \quad z_\alpha \partial_x z_\alpha \partial_x u_\alpha + \tfrac{1}{2}\left(z_\alpha^2 - z_i^2\right)\partial_x^2 u_\alpha \end{aligned}$$

  solve SPH problem from time $t - \frac{\Delta t_{\text{Bsq}}}{2}$ to $t + \frac{\Delta t_{\text{Bsq}}}{2}$
  for $k = 1 \dots k_{\max}$ **do**
    $t = (n - \frac{1}{2})\Delta t_{\text{Bsq}} + k\Delta t_{\text{SPH}}$
    interpolate velocity u_i^k(linear) and imposed SPH boundary particule displacement as: $x_i^{k+1} = x_i^k + \Delta t_{\text{SPH}} u_i^k$
    solve SPH problem with time step Δt_{SPH}
  end for
  get wave height (particle with largest z value on the boundary) and velocity at the reference height (smoothed particle velocity at z_α) of the Boussinesq boundary.
end for

Multi-usability: several components should be instantiated on the same CPU or on different machines; this feature allows parallel processing.

Non-context specific implementation: two components that fit the same interface can be used by a service without any modification.

Composability: it is possible to make components out of components.

Encapsulation: it is not possible to access the inner structure or details of a component via an interface. Hence, the implementation of the algorithm defined by the interface and the implementation of the communication method are strictly separated.

Development and version independent: a component defined through its interface should be independent from its version, programming language and even of the compiler used. It means that once deployed on a machine, it can be called by any service knowing only its interface and the middleware until it is erased.

More details can be found in the following conversation [5].

3.2.1 Component-based implementation and its specificities

From a scientific computing point of view, the implementation based on component technology relies on the definitions of API (Application Program Interface). Indeed, both clients – those that call a method – and services – those that execute the task – side have to share the knowledge of classes, functions and methods available. The API allows the Remote Procedure Call (RPC), or yet called Remote Method Invocation (RMI), to access libraries, procedures or stored object throughout a network or on the same machine.

In a component-based implementation, a strict separation is made between the algorithm and the method implementation on one side, and the communication handled by the middleware on the other side. Ideally, the client side is totally independent from the way the clients are called, and it does not matter whether they are available locally or remotely called.

In scientific computing, RPC are rivaled by explicit Message Passing API. The main drawback of Explicit Message based programs is that they melt what concerns the algorithm with what concerns the communication between pieces of software. However, from a historical point of view, they were the first to be used in this domain. Thus, MPI [15] or PVM [13] initially only allowed Explicit Message Passing. Nevertheless, the advantages of RPC is now generally admitted, as confirmed by the growing number of types of middleware products available for scientific computing.

3.2.2 The middleware CTL

A simple way to make software communicate can be to make one writes file with data and to provide to another the adapted function to read these data. This method suffer of poor time efficiency, and is therefore not easily generalisable. Component based development requires a middleware layer between clients and services. More precisely, in [26], a middleware is defined as:

> Middleware works by providing a standardized, API-like interface that can allow applications on different platforms or written in different languages to interoperate.

Many free and non-free middleware are currently available on the market; The most well known are, CORBA (Common Object Request Broker Architecture), Java™RMI or Microsoft® .NET. However, the field of scientific computing requires very high performance in communication, which implies that only a few of the available middleware are of interest for extensive computation. In fact, according to information on performance computing between different types of middleware in [22], only CORBA – among the quoted environments – fulfills the cost requirement, but is known for its complicated syntax.

In the last ten years, new components like CCA [16], Charm++ [18] or CTL where specifically developed for the need of scientific computing and with the aim of simplifying the syntax. In this work following in the steps of earlier development [23], we will use the CTL as middleware.

Initially a part of `ParaFEP` [24], the Component Template Library (CTL) was developed by Dr. R. Niekamp at the Institute für Wissenschaftliches Rechnen (TU-Braunschweig). It is a `C++` template library, like the STL, that builds a wrapper or a communication layer around a software, and thus allows so to build components from existing piece of code. This layer ensures a serialization of the data to be exchanged over a network, and implements, via code generation, an interface defined in a particular header file. Unlike complicated CORBA syntax, the API is here written in C-preprocessor language, often in a `*.ci` (for Component Interface) file.

The two main advantages of CTL are [6]:

- providing a lightweight that can be used on top of several local (library and thread) or remote communication methods (TCP/IP, MPI and others).
- making the process of writing an application or a service which uses the CTL protocol as transparent as possible. Developers of a service can write its implementation like they would write a normal local class, with the exception that they need to give the CTL a method to serialize the contained data. Developers of a client only needs then to choose a service within the CTL API and how it starts. They can use the objects provided by CTL services as if they were standard local objects.

3.3 Components

3.3.1 Wave component

First a Component Interface (`.ci` file) has to be declared. It is done in a `SimuWave.ci` file that contains method declaration in a C pre-processor format:

```
/*****************************************************************************

  Component Interfaces (CI) for wave simulators

  Author: Christophe KASSIOTIS
          Laboratoire d'Hydraulique Saint-Venant
          UniversitÃ© Paris-Est (EDF R&D, Ecole des Ponts ParisTech, CETMEF)
  Email:  christophe.kassiotis@enpc.fr

  Copyright (c) 2010 Christophe KASSIOTIS.
  All rights reserved. No warranty. No liability.

*****************************************************************************/
#ifndef __SIMUWAVE__CI
#define __SIMUWAVE__CI

#include <ctl.h>

#define CTL_ClassTmpl SimuWaveCI, ( scalar1 ), 1
#include CTL_ClassBegin
  #define CTL_Constructor1 ( const string /*init control file*/ ), 1
  #define CTL_Method1 void, getnodes,
    ( int4 /*dimension*/, array<scalar1> /*nodes*/ ) const, 2
  #define CTL_Method2 void, set,
    ( const string /*fieldName*/, const array<scalar1> /*field*/ ), 2
  #define CTL_Method3 void, get, ( const string /*fieldName*/,
```

```
      array<scalar1> /*nodes*/, array<scalar1> /*field*/ ), 3
  #define CTL_Method4 int4, solve, ( const scalar1 /*timeStep*/ ), 1
  #define CTL_Method5 int4, goback, (), 0
  #define CTL_Method6 int4, plot, (), 0
#include CTL_ClassEnd

#endif
```

The name of the method is quite explicit. For `BSQ_V2P3` based component, the method marked with ✓have been implemented, whereas the one with × are either not implemented or checked.

- ✓ `Constructor1` instantiate the component with a given input file specifying options.
- ✓ `getnodes` get the nodes and the dimension of problem.
- ✓ `solve` solve the problem for a time step of size Δt.
- ✓ `plot` plot the results with a given format
- `get` get a field described by its name at given points: *e.g.* gives velocity field at a certain number of points
- `set` set the velocity and wave height at given nodes.
- `goback` required for implicit coupling schemes with different time steps for subproblems

3.3.2 SPH Component

References

[1] C. Austruy. Approches multi-échelles et multi-physiques pour quantifier l'amortissement d'une vague par des obstacles. Mémoire de Master 2-R MIS, parcours Génie-civil, École Normale Supérieure de Cachan, Cachan, France, June 2008. Encadrants : J.-B. Colliat, C. Kassiotis. Mention TB, Rang 1.

[2] C. Austruy, C. Kassiotis, J.-B. Colliat, A. Ibrahimbegović, H. G. Matthies, and F. Dias. A multiscale and multiphysic approach to quantify waves damping by structures. In A. Ibrahimbegović and M. Zlatar, editors, *NATO–ARW 983112 Damage assessments and reconstruction after natural disasters and previous military activities*, Sarajevo, Bosnia-Herzegovina, October 2008.

[3] G. Berti. *Generic Software Components for Scientific Computing*. Ph.D. Thesis, Technische Universität Cottbus, Germany, 2002. 5

[4] J. Boussinesq. Théorie des ondes et des remous qui se propagent le long d'un canal rectangulaire horizontal, en communiquant au liquide contenu dans ce canal des vitesses sensiblement pareilles de la surface au fond. *Journal de Mathématiques Pures et Appliquées*, 17(2):55–108, 1872. 1

[5] M. Broy, A. Deimel, J. Henn, K. Koskimies, F. Plášil, G. Pomberger, W. Pree, M. Stal, and C. Szyperski. What characterizes a (software) component? *Software – Concept & Tools*, 19:49–56, 1998. 5

[6] B. Bügling. *The Component Template Library Protocol and its Java Implementation*. Master Thesis, Technische Universität Braunschweig, Institut for Scientific Computing, 2006. http://www.wire.tu-bs.de/forschung/projekte/ctl/files/ctl_spec.pdf. 7

[7] B. Coulange. *Software reuse*. Springer Verlag, London, 1998. 5

[8] B. de Saint-Venant. Théorie du mouvement non-permanent des eaux, avec application aux crues des rivières et à l'introduction des marées dans leur lit. *Compte Rendu de l'Académie des Sciences*, 73:147–154, 1871. 1

[9] F. Dias, D. Dutykh, and J.-M. Ghidaglia. A two-fluid model for violent aerated flows. *Computers and Fluids*, In Press, Accepted Manuscript, 2009. 1

[10] D. Dutykh. *Modélisation Mathématique des Tsunamis*. Thèse de Doctorat, Centre de Mathématiques et Leurs Applications, École Normale Supérieure de Cachan, France, 2008. 1

[11] D. Dutykh and D. Mitsotakis. On the relevance of the dam break problem in the context of nonlinear shallow water equations. *Discrete and Continous Dynamical System – Series A*, Accepted, 2009. 1

[12] C. Fochesato, S. Grilli, and F. Dias. Numerical modeling of extreme rogue waves generated by directional energy focusing. *Wave Motion*, 44:395–416, 2007. 1

[13] a. Geist. Pvm official home page, 2007. http://www.csm.ornl.gov/pvm. 6

[14] J.-M. Ghidaglia, A. Kumbaro, and G. Le Coq. On the numerical solution to two fluid models via a cell centered finite volume method. *European Journal of Mechanics/B Fluids*, 20(6):841–867, 2001. 1

[15] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1994. 6

[16] J. A. Kohl and D. E. Bernholdt. CCA home page, 2002. http://www.csm.ornl.gov/cca/. 6

[17] C. Lachaume, B. Biausser, S. T. Grilli, P. Fraunie, and S. Guignard. Modeling of breaking and post-breaking waves on slopes by coupling of bem and vof methods. In *13th Offshore and Polar Engineering Conference (ISOPE)*, pages 353–359, 2003. 1

[18] O. S. Lawlor and G. Zheng. Charm++ home page, 1999. http://charm.cs.uiuc.edu/research/charm/. 6

[19] M. D. Mac Ilroy. Mass produced software components. NATO Software engineering conference, Garmish, Germany, 1968. 5

[20] C. C. Mei. *The Applied Dynamics of Ocean Surface Waves*. World Scientific, Singapore, 1989. 1

[21] M. S. Narayanaswamy. *A hybrid Boussinesq-SPH wave propagation model with applications to forced waves in rectangular tanks*. Ph. d. thesis, The Johns Hopkins University, USA, 2009. 1, 2

[22] R. Niekamp. Software component architecture. Lecture note, Institut für Wissenschaftliches Rechnen, Germany, 2005. http://www.wire.tu-bs.de/forschung/projekte/ctl/files/ctl_course.pdf. 5, 6

[23] R. Niekamp, D. Markovič, A. Ibrahimbegović, H. G. Matthies, and R. L. Taylor. Multi-scale modelling of heterogeneous structures with inelastic constitutive behavior: Part II–software coupling implementation aspects. *Engineering Computations*, 26:6–28, 2009. 6

[24] R. Niekamp and E. Stein. An object-oriented approach for parallel two- and three-dimensional adaptive finite element computations. *Computers and structures*, 80:317–328, 2002. 7

[25] G. O'Regan. *A Brief History of Computing*. Springer-Verlag, London, Great-Britain, 2008. 5

[26] D. Orenstein. Quickstudy: Application Programming Interface (API). *http://www.computerworld.com*, 2000. 6

[27] D. H. Peregrine. Long waves on a beach. *Journal of Fluid Mechanics*, 27(4):815–827, 1967. 1

[28] H. Rusche. *Computational Fluid Dynamics of Dispersed Two-Phase Flows at High Phase Fractions*. Ph.D. Thesis, Department of Mechanical Engineering, Imperial College of Science, Technology and Medicine, London, G.-B., 2002. 1

[29] R. J. Sobey. Linear and Nonlinear Wave Theory. Lecture note, Leichtweiß-Institut für Wasserbau, Braunschweig, 1998. 1

[30] J. J. Stoker. *Water Waves: The Mathematical Theory and Applications*. Wiley-Interscience, New-York, 1992. 1

[31] C. Szyperski. *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley and ACM Press, Reading, Massachusetts, 1998. 5

[32] R. L. Taylor. *FEAP – A Finite Element Analysis Program Version 8.2 User Manual*. Departement of Civil and Environemental Engineering – University of California at Berkeley, March 2008. http://www.ce.berkeley.edu/~rlt/feap/manual.pdf.