Implementing Agent-based Resource Management in Tsunami Modeling – Preliminary Considerations

Michał Drozdowicz[1], Kensaku Hayashi[2], Maria Ganzha[1], Marcin Paprzycki[1], Alexander Vazhenin[2], Yutaka Watanobe[2]

[1] Systems Research Institute Polish Academy of Science, Warsaw, Poland
`(name.surname)@ibspan.waw.pl`

[2] Graduate School Department, University of Aizu, Aizu-Wakamatsu, Japan
`{vazhenin, yutaka, m5161111}@u-aizu.ac.jp`

Abstract. Recently, work has started to apply the agent-semantic infrastructure, developed within the scope of the *Agents in Grid* project, to the resource management needed in tsunami modeling. The original proposal was based on the perceived simplicity, versatility and flexibility of the agent-based approach that makes it easier to deploy than the standard grid middlewares. The aim of this paper is to report on the progress in implementing and deploying the proposed system at the University of Aizu.

1 Introduction

One of the key effects of the Great Japanese Earthquake and Tsunami was restatement of the importance of studying the impact of such events at different time scales. Here, the two main issues that have to be addressed are: (1) real-time tsunami warning, and (2) long-term hazard assessment. To respond to these needs, it is necessary to: (a) facilitate use of distributed clients, providing access to computational services, (b) address scalability, to allow an arbitrary number of users and computational resources to interact in a customizable working environment, and (c) since different applications and services may be developed for a variety of hardware/software platforms, reusability and interoperability are important aspects of applying (and, possibly, combining) computational resources and services [1, 2].

Here, observe that personal computers (with, or without, additional enhancements, such as GPU processors) can be used for high-demand computing applications, e.g. consider the Folding@home project that involves distributed PC-based simulations of protein folding and other molecular dynamics simulations [3, 4]. Unfortunately, while most volunteer projects (e.g. projects based on the BOINC infrastructure) do note require human supervision – results are collected / accumulated during a long-term execution, and only in the final stage they are inspected by the humans, this is not the case with tsunami modeling. Here, the typical research scenario requires human interactions during the process, e.g.

checking/understanding the results of the current test(s) is often required to instantiate the next round of experiments. Furthermore, different tsunami models often need to be combined to study various phases of the tsunami phenomenon. This makes a direct application of the BOINC-like approach difficult, if not impossible.

As a result, in [20] it was stipulated that an agent-semantic system, designed for resource management in the grid (developed in the *Agents in Grid* project; *AiG*), can be successfully applied in tsunami modeling research. The aim of this paper is to discuss how the *AiG* approach can be used to instantiate a distributed tsunami modeling laboratory. To this effect, we start with a brief summary of the state-of-the-art in tsunami modeling. Next, we outline the MOST algorithm designed for tsunami simulation, and provide details of the implementation developed and used at the University of Aizu. We follow with an overview of the Agents in Grid system and discuss its key aspects related to the process of its customization to the requirements of tsunami modeling. Finally, we provide a description of the process of specifying and submitting a job within the distributed tsunami modeling laboratory. This process is used to illustrate key features of the system under development.

2 Tsunami modeling state-of-the-art

Let us start form briefly discussing the state-of-the-art in tsunami modeling. In [5], authors suggested that complex mathematical models and high mesh resolution should only be used when necessary. They have developed a "parallel hybrid tsunami simulator," based on mixing different models, methods and meshes. This simulator was implemented using object-oriented techniques, allowing for easy reuse of existing codes. Here, high performance was not the main goal. Instead, research was focused on combining various approaches to develop high quality hybrid tsunami models.

Authors of [6], experimented with eight different parallel tsunami propagation simulators. Each of them used a mixed-mode programming model, consisting of a thread-based shared memory part, a distributed memory part and, finally, a virtual shared memory-based part. Obtained results have illustrated various problems with scalability of the investigated software artifacts. Furthermore, it was shown that if sufficient node memory is not available, threading becomes the bottleneck.

The TsunamiClaw is a software package based on a finite volume method [7]. It solves the shallow water equations in the, physically relevant, conservative form. Thus, the obtained solution is represented as water depth and momentum. Currently, this project is no longer actively pursued. Instead it has been generalized into the GeoClaw software.

The TUNAMI-N2 software [8], is a tsunami simulation, which uses separate models for the deep sea and shallow water. Interestingly, it uses constant grid size in the entire domain. The TUNAMI was originally authored by by Imamura (in 1993) and later applied to the real tsunami events in many countries. The package was written in FORTRAN and has a standard GUI.

Finally, the MOST (Method of Splitting Tsunami) software allows for real-time tsunami inundation forecasting, by incorporating real-time data from actual detection buoys [9,10]. Furthermore, in the US, the MOST model is used for developing inundation maps [11]. To use in computational practice, a web enabled interface, named ComMIT, has been implemented.

3 Tsunami Modeling Environment and Processes

3.1 Basic Model and Software Tools

As discussed, there exist multiple tsunami models and algorithms implemented to realize them. These models deal with: origins of tsunamigenic earthquakes (estimation of magnitude and epicenter location), determination of the initial displacement at the tsunami source, wave propagation, inundation into the dry land, etc. Overall, the tsunami modeling environment is typically used to simulate three phases of the tsunami evolution: (1) estimation of residual displacement area, resulting from an earthquake and causing the tsunami, (2) transoceanic propagation of the tsunami through the deep water, and (3) contact with the land (run-up and inundation). Out of available choices, the University of Aizu team selected the MOST package [9,10]. The MOST approach was initially developed in the Tsunami Laboratory of the Computing Center of the USSR Academy of Sciences in Novosibirsk. Subsequently, the method was updated in the National Center for Tsunami Research (NCTR, Seattle, USA) and adapted to the standards accepted by tsunami watch services in the US, as well as other countries. It was then used in tsunami research in many countries around the world. According to this approach, propagation of the wave in the ocean is governed by shallow-water differential equations:

$$\begin{aligned} H_t + (uH)_x + (vH)_y &= 0, \\ u_t + uu_x + vu_y + gH_x &= gD_x, \\ v_t + uv_x + vv_y + gH_y &= gD_y, \end{aligned} \tag{1}$$

where $H(x, y, t) = h(x, y, t) + D(x, y, t)$; h - is the water surface displacement, D - depth, $u(x, y, t)$ and $v(x, y, t)$ - are the velocity components along the x and y axis', g - is the gravity. The initial conditions should confirm the presence of water in all grid points, except for the tsunami source, where the surface displacement is not equal to zero.

The numerical algorithm splits the difference scheme, which approximates equations (1) in the spatial directions. A finite difference algorithm, based on the splitting method, reduces the solution of equations with two space variables to the solution of two one-dimensional equations. As a result, effective finite difference schemes, developed for the one-dimensional problems, can be applied. Moreover, this method permits to set boundary conditions for a finite-difference boundary value problem, using a characteristic line method.

3.2 General Calculation Process

Figure 1 shows the block-diagram illustrating the overall structure of calculations. To run the program, it is necessary to specify:

- bottom topography or bathymetry data;
- initial and boundary conditions;
- modeling parameters such as time-steps and length of the model run.

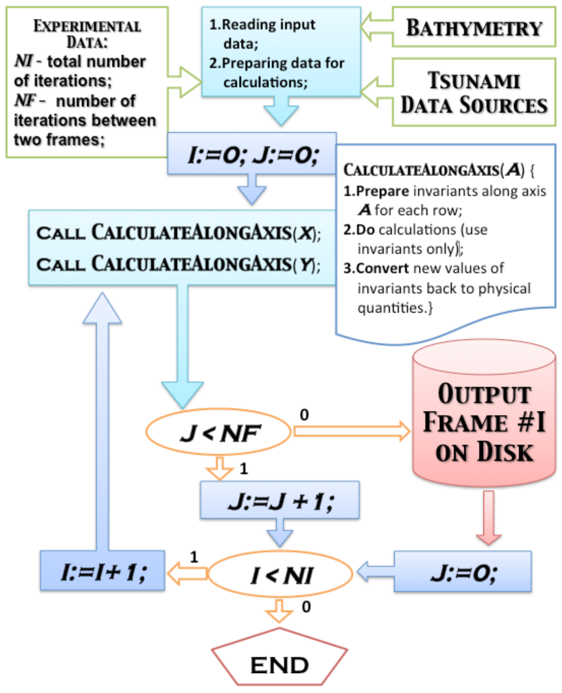

EXPERIMENTAL DATA:
NI - total number of iterations;
NF - number of iterations between two frames;
1.Reading input data;
2.Preparing data for calculations;
BATHYMETRY
TSUNAMI DATA SOURCES
I:=0; J:=0;
CALCULATEALONGAXIS(A) {
1.Prepare invariants along axis A for each row;
2.Do calculations (use invariants only);
3.Convert new values of invariants back to physical quantities.}
CALL CALCULATEALONGAXIS(X);
CALL CALCULATEALONGAXIS(Y);
J < NF
0
1
OUTPUT FRAME #I ON DISK
J:=J+1;
I:=I+1;
I < NI
J:=0;
END

Fig. 1. General computational scheme of the MOST software

The necessary parameters are passed to the modeling program as a *scenario file* that is composed of the following elements:

- Area Information – BathymetryArea. Containing:
  - Grid Name – Name of the Sea Area under consideration
  - Grid Axes Version – Version Number
  - Grid File Name – Link/Path to the Bathymetry Data/File
- Computational Parameters – CalculationInformation. Containing:
  - Minimum Depth – Minimum depth for the offshore area
  - Time step – Time between two modeling iterations (in seconds)
  - Total number of steps – The total number of iterations

  - Number of steps between snapshots – Number of output frames (NF)
  - Modeling time passed – The time elapsed since the beginning of modeling
  - Save output every n-th grid point – Specification of results saving structure
  - global b.c.s – 1=global, 0=non-reentrant
- Naming Rules – NameOfOutputResult. Containing:
  - Filename – <prefix_ha.nc>, or "auto"
  - Source Zone Name – Name of domain inside the bathymetry area
  - Source Zone Code – Code of domain
  - Source Column – Code of Grid (Column)
  - Source Row – Code of Grid (Row)
  - Source Version – Version of source
- Fault Plane Information – InformationAboutEarthquakeData. Containing:
  - Number of Fault Planes – Information concerning number of fault planes
  - x-integration – X value for integration
  - y-integration – Y value for integration
  - Vp – P-wave velocity
  - Vs – S-wave velocity
  - Deform Area X – Value of the Deform Area (x-axis)
  - Deform Area Y – Value of the Deform Area (y-axis)
  - Longitude (deg) – Center of the initial Wave (Longitude)
  - Latitude (deg) – Center of te initial Wave (Latitude)
  - Length (km) – Size of the initial Wave (Length)
  - Width (km) – Size of the initial Wave (Width)
  - DIP (deg) – DIP of the Wave
  - RAKE (deg) – RAKE of the Wave Form
  - STRIKE (deg) – Strike of the Wave
  - SLIP (m) – Slip of the Wave
  - DEPTH (km) – Depth of the Wave

The complete model of this information is depicted in Figure 2. As will be seen, this information was used to develop the initial version of tsunami modeling ontology.

After launching, the program implements calculations and stores results as a series of frames representing tsunami propagation process in time. Parameter NF defines the time interval, during which the results of computations are persisted in the computer memory. After this time expires, results are stored on the secondary storage devices in the NetCDF format ([11]).

It is important to note that it took about 3.31 seconds to complete a single time step of the original (Fortran 90) program on a computer with 4 dual-core, Intel Xeon 2.8GHz, CPUs. After the program was ported to C/C++, it takes about 3.00 seconds for a single time step [12]. Since a typical simulation, consists of about 10000 time steps, it requires about 8 hours to complete. Therefore, the tsunami modeling needs to be significantly accelerated (e.g. through parallel processing); especially for real-time tsunami warning generation. However, speeding up modeling is also crucial for repetitive tsunami simulations; e.g. in the artificial island modeling scenario.

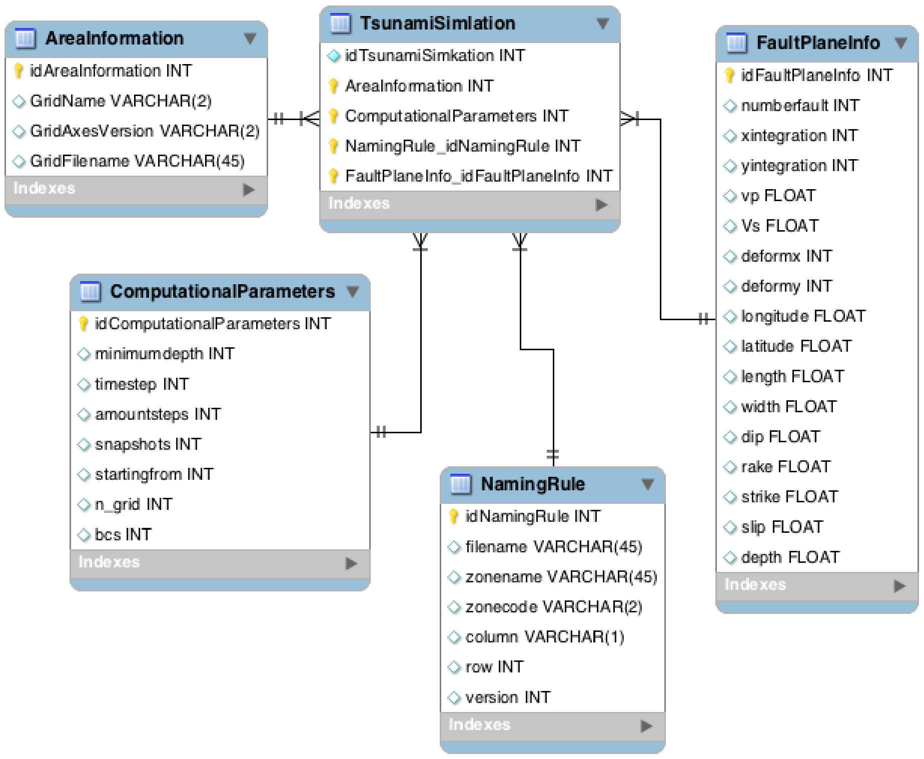


Fig. 2. Scenario parameters

3.3 Hybrid Tsunami Modeling Combining Natural and Artificial Bathymetry Objects

Lessons from the Great Japanese Tsunami stress importance of: (i) being able to provide real-time tsunami warning, (ii) long-term hazard assessment (e.g. running detailed inundation models across the Japanese sea-line, and (iii) studies of the well-known "Matsushima effect." The later concerns the influence of geographical objects, like islands, on the wave height and/or speed. This observation finds its foundations in the research reported in [14]. Here, results of an experimental study on effects of submarine barriers on tsunami wave propagation indicated capability of reducing tsunami run-up through strategic placement of artificial objects interacting with the tsunami waves. Specifically, it may be possible to design and build a set of artificial objects (islands) that can be used to protect the coastal areas. In particular, such protection could be of extreme value in highly populated areas (e.g. coastal cities, such as Sendai, that was affected by the tsunami of 2011), as well as in industrial areas (e.g. nuclear plants, factories, airports, etc.). Note that such effect, caused by local islands, already exist in the Matsushima area, while absent on the Fukushima coast. It can be conjectured that adding a small number of appropriately placed artificial objects "in front of" the Fukushima Nuclear Plant could have mitigated the effect of the tsunami and prevent the disaster. For more details, see also [19, 20].

4 AiG for Tsunami Modeling

Let us now discuss how the *Agents in Grid; AiG* project can be used to support tsunami modeling. We will start from an overview of these parts of the *AiG* approach that are pertinent in the current context. The *AiG* project aims at providing a flexible agent-based infrastructure for managing resources in the grid ([15, 16]). Application of software agents and semantic technologies makes it well-suited for open, dynamic and heterogeneous environments. The *AiG* architecture is based on the premise of an open Grid – a network of heterogeneous resources, owned and managed by different organizations. It allows for users to either provide a new resource to the Grid in order to earn money, or to use the Grid to execute a task.

In the original *AiG* approach, each resource is governed by a *WorkerAgent* and performs its tasks as part of a team, managed by an *LMaster* agent. Teams are registered in a yellow-pages-like directory service, represented by the *Client Information Center* (*CIC*) agent, which handles the initial matchmaking of users to teams. Users interact with the system through their (dedicated) *UserAgents*. The decision, which team to choose to execute the job is a result of autonomous negotiations between the *UserAgent* and the *LMasters* of appropriate teams. In a similar way, adding a resource to the system involves negotiations with teams looking for new members. The main features of the system are shown in Figure 3, in the form of a use case diagram.

An important aspect of the project is the fact that all data and information in the system are represented in ontological format, using the OWL language. The

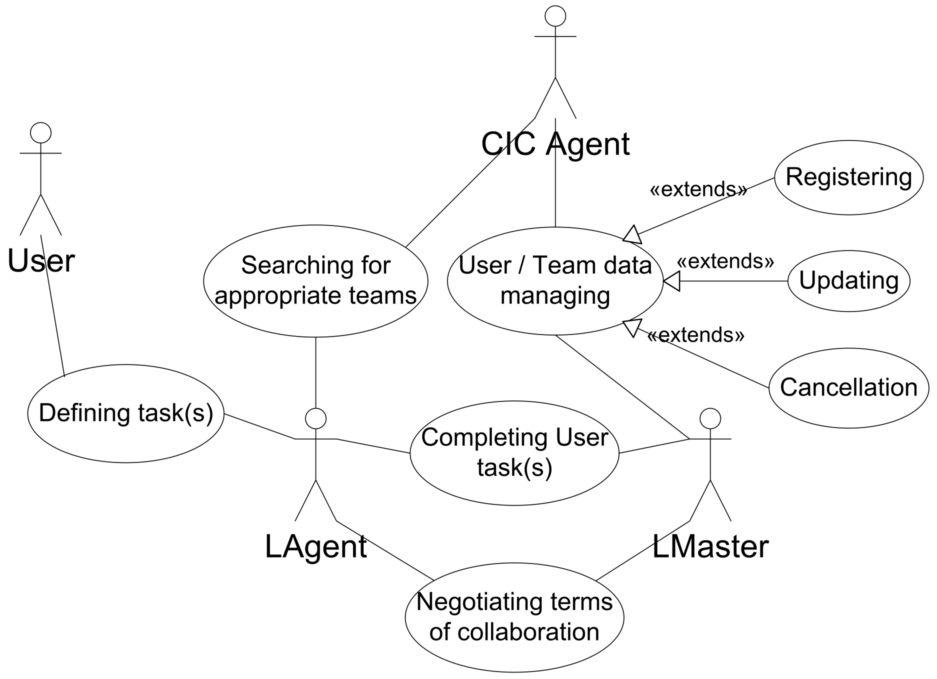


Fig. 3. Use Case diagram of the AiG system

usage of ontologies enables to describe jobs, resources and their relationships in a structured, yet flexible way (for more details, see [17]). Thanks to application of ontologies, providing support for new (added to the system) types of hardware and augmented software configuration (such as new software libraries or programs) involves only modification of the ontology terms and does not require any additional customization. As the job descriptions are also defined using the OWL language, it is possible to specify different parameter sets and requirements towards computing resources depending on the type of the task to be performed. For instance, as part of the initial implementation of the AiG system at the University of Aizu, we have extended our core ontology to be able to accurately describe the hardware configuration of the machines used in the experiments, as well as terms related to the tsunami modeling (see, Section 3.2).

Putting the AiG system to work for tsunami modeling we have made a few observations. First, computers made available to the tsunami research at the University of Aizu do not constitute an open environment where resources join and leave "dynamically." Second, there is no economic aspect – we can safely assume that if a resource matches the requirements of the job and is available for use, there is no need for negotiations concerning its price. However, what is still required are negotiations concerning availability of resources (e.g. when a given laboratory is in use during certain class periods and machines cannot be used for other purposes then instruction). To address these points we have modified the *AiG* system. First, we have resigned from the notion of resource team, and placed a *WorkerAgent* (playing the role of the *LMaster*) on each computing node. We have also eliminated the scenario in which the *WorkerAgent* is joining a team. Instead, adding a new resource means registering it with the *CIC Agent* as a standalone node (one-member team). Finally, in the next phase of the implementation, we will re-focus the negotiations. Their role will be to provide information about current and planned utilization of the resources. This will allow the *UserAgent* to decide where to run, which job, and when.

4.1 Job submission process – summary

Let us now go through the entire process of conducting an experiment, using resources at the University of Aizu and use it to illustrate the details of the modified *AiG* system.

The user starts by accessing a web based interface, which is the entry point to the communication with the *UserAgent*. The next step is to specify the hardware requirements for the job in the form of constraints on the ontological terms describing the resources. This task is done using the interface based on the OntoPlay module [18] (its Condition Builder component), giving the user complete freedom in describing the needed resources, while guiding her through the contents of the ontology without the need for deep knowledge of its structure (knowledge of semantic technologies, in general). As shown in Figure 4, the Condition Builder is composed of a series of condition boxes used to create constraints on class-property relationships. Depending on the chosen class, the user can select, which class property she wishes to restrict. For example, having selected the *GPUMemory* class, the expanded property box will contain properties such as *hasTotalSize* and *hasAvailableSize* (see, Figure 4).

After selecting the class and property the user can choose the required operator and value. Here, she sees only the operators applicable to the given type of the property. Specifically, this means that for value properties (such as amount of available GPU memory) it would be operators such as *equalTo*, *lessThan* or *greaterThan*, while for object properties (e.g. the CPU installed on the node) the user would be allowed to select, e.g. is equal to individual or is constrained by. Note that the selection of available operators is performed by the front-end, on the basis of the ontology and was *not* hand-coded. Should a user wish to restrict the value of a particular property to a fixed individual from the ontology, the Condition Builder lists all available individuals that can be used in the context (see, Figure 5).

Let us now assume that an object property is selected choosing the "is constrained by" operator. This enables the user to specify the type of object for which the value should be constrained, and to create additional constraints on that class (see, Figure 6).

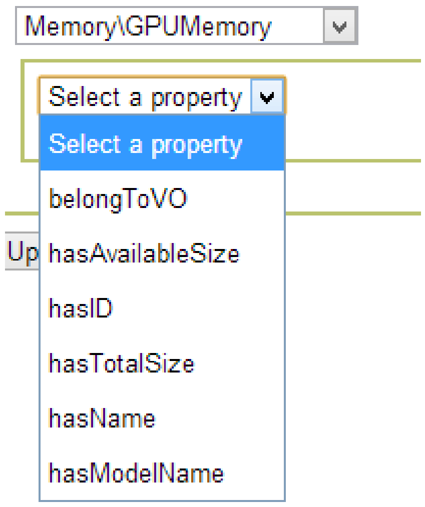


Fig. 4. Example of choosing a property for constraining

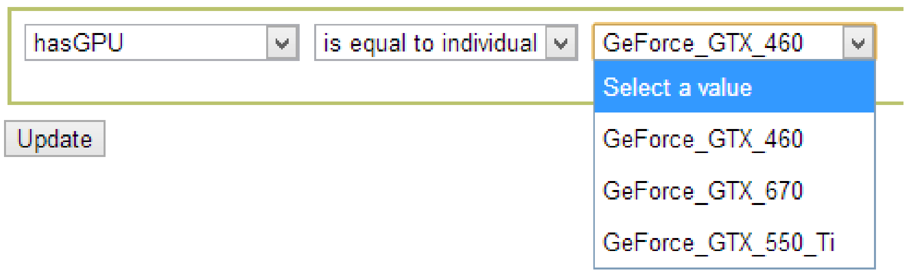


Fig. 5. Example of choosing an individual

4.2 Job submission process – detailed example

Let us now look at a practical example that will illustrate the entire process. As described in [9, 10, 12, 13], the implementation of the MOST code, used at the University of Aizu, is most effective when run on a CUDA-based GPU, with a sufficient amount of available GPU memory. Therefore, let us assume that the user wishes to schedule a job on a resource that has a GPU with at least 512 MB of available GPU memory (which is one of the machines available for our experiments). In this case the user starts with an empty *ComputingComponent* specification. First, she would constrain the property *hasGPU* of the class *ComputingComponent* to contain a value of type *GPU_CUDA*. Let us assume that it does not need to be any particular GPU model, so we do not add additional conditions on this class (though such specification already exists in the com-

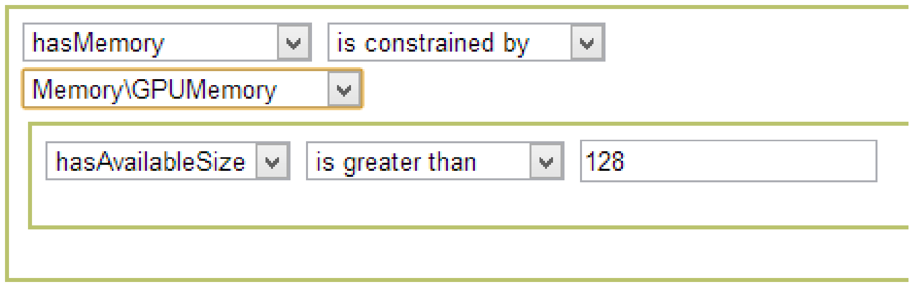
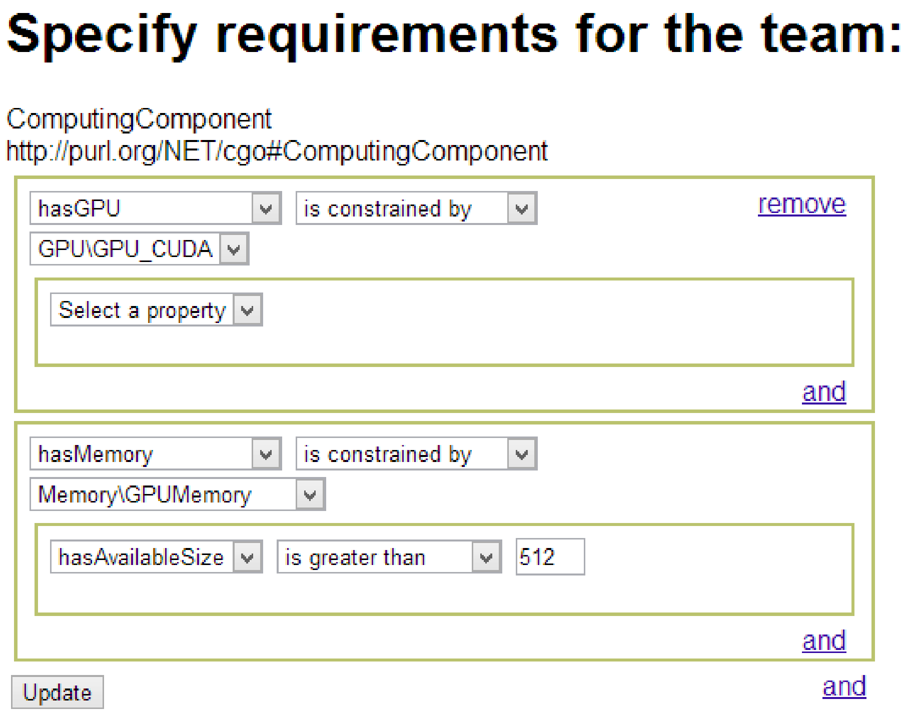


Fig. 6. Example of choosing a nested condition

pleted representation of available machines). Second, the user adds a condition on the property *hasMemory*, constraining it to the *GPUMemory* subclass and adding a nested condition specifying that the *hasAvailableSize* property should have a value greater than 512 MB. Figure 7 represents the completed condition. Once more, note that during the selection process, only these properties and individuals are "shown to the user" that have been specified within the ontology; and that no hand-coding of these terms and conditions was required. All "work" is done by the front-end on the basis of the ontology.

After the user submits the resource requirements, the *UserAgent* passes this description to the *CIC Agent*, which performs semantic reasoning on its knowledge base, to find resources satisfying the given criteria and returns a list of matching nodes, including the information on how to contact the *LMasters* (in our case the *WorkerAgents*) at each node.

Note that in a dynamic environment, such as the university laboratory, there is no guarantee that the resources found by the *CIC Agent* are, at the moment, available for use. The machine might be offline, used for other purposes, or the agent process (and/or container) might not be running. Therefore, there is a need for additional verification of the availability of the resources. This is handled using the mechanism of multi-agent negotiations, albeit in a very simplified form. When the *UserAgents* receives the list of *LMaster* addresses, it issues a *Call For Proposal; CFP* message to gain confirmation of whether the resources are able to perform the task. The *LMasters* confirm that this is the case (or reject the proposal), and provide information when they could start executing the job. This helps to handle the case of temporarily occupied nodes. Once the *UserAgent* receives offers from the agents (here, note that we assume their benevolence), it presents the list to the user, who can choose the node(s) on the basis of their availability and other parameters. Here, the resource selection may be also passed to the *UserAgent*, but this will require further considerations and will be approached in the next phase of the project.


Specify requirements for the team:

ComputingComponent
http://purl.org/NET/cgo#ComputingComponent

hasGPU — is constrained by — GPU\GPU_CUDA

Select a property

hasMemory — is constrained by — Memory\GPUMemory

hasAvailableSize — is greater than — 512

Fig. 7. Complete requirements specification

4.3 Specifying the scenario description

The final step of submitting jobs is the specification of the executable code / library an of the necessary parameters. As described in the previous sections, for the tsunami simulations it is crucial to be able to run different kinds of algorithms on different data sets and variables to come up with collections of results (particularly in the case of tsunami modeling, rather than generating tsunami warnings). Consequently, in this case, the user is going to provide multiple job descriptions (one for each model / parameter set in the simulation). The job description is provided using *the same* Condition Builder mechanism, although using a different ontology.

As part of the implementation of the *AiG* system at the University of Aizu, a new ontology – the *MOSTOntology* – has been created to represent the entities forming the simulation scenario (recall that these parameters have been described in Section 3.2). This ontology is an extension of the *AigGridOntology* in a way that a newly introduced class `TsunamiSimulation` is introduced, which is a sub-class of the `JobDescription` class. Other classes contained in the *MOSTOntology* correspond directly to the entities from the scenario file:

– `AreaInformation`

– `ComputationalParameters`
– `NamingRule`
– `FaultPlaneInfo`

The ontology also contains object properties linking `TsunamiSimulation` with the above mentioned classes, as well as all data properties describing them (as specified in Section 3.2).

The introduction of the *MOSTOntology* into the *AigGridOntology* enables the user to specify the job using the same Condition Builder interface. It also makes it possible for the *WorkerAgent* to generate the scenario file from the ontological information, thus removing the need to deploy the scenario files onto each (potential) grid node. Of course, the system will still support running jobs using scenario files accessible locally on the nodes, or at a network location accessible to them, but the goal of achieving simplified access of users to distributed, heterogeneous resources has been achieved.

To give an example of how the contents of the scenario file corresponds to the contents of the *MOSTOntology* and to illustrate the usage of the Condition Builder for the job specification, Listing 1.1 contains a sample scenario file, while Fig. 8 depicts a part of the same scenario represented in the *AiG* user interface.

Listing 1.1. Sample scenario file

```
#  MOST Propagation test input file
#
#  This is the format for an input file for running the
#  MOST Propagation program version 1.3
#
#  Comments are prefixed with a hash "#", and can appear
#  on their own line, or after a parameter.
#   The only important thing is order of parameters
#
#  If there are multiple fault-planes, user must provide
#  all deformation parameters repeated for each fault-plane
#  (repeat from "x-integration" to "Depth" for each fault-plane)
#
#  If number of fault-planes is 0, MOST expects to read the
#  deformation from a file in the MOST grid (ASCII Grid) format
#  If number of fault-planes is < 0, MOST reads deform.dat file
#  created from the previous run of MOST (hint, use this to
#  keep from re-running deformation)
#
#  Beginning of test input file:
#       Grid Name
Pacific
#       Grid Axes Version
20060823
#       Grid Filename
/home/tsunamiagent/MOST/Bathymetry/pacific_4m_nocaribbean.corr
#       Computational parameters
20  # Input minimum depth for offshore (m)
10  # Input time step (sec)
1000    # Input amount of steps
6   # Input number of steps between snapshots
6   # ...Starting from timestep
4       # Save output every n-th grid point
0   # Input global b.c.s (1=global, 0=non-reentrant)
#       Output filename (<prefix>_ha.nc, or "auto")
auto
#       Source naming info
#   Source Zone Name
```

```
Aleutian-Cascadia
#    Source Zone Code (two characters)
ac
#    Source Column (one character)
b
#    Source Row (integer)
13
#    Source Version (integer)
0
#        Fault plane info
1               # Number of fault-planes
41              # x-integration
21              # y-integration
8.11            # Vp - P-wave velocity
4.49            # Vs - S-wave velocity
200             # Deform Area X
200             # Deform Area Y
179.842         # Longitude (deg)
51.085          # Latitude (deg)
100.0           # Length (km)
50.0            # Width (km)
15.0            # DIP (deg)
90.0            # RAKE (deg)
271.0           #STRIKE (deg)
1.0             # SLIP (m)
5.0             # DEPTH (km)
```

4.4 Job execution

Once the user completes the job description, it is sent by the *UserAgent* to the respective *LMaster* (the one that was selected as a result of the, above described, negotiations), which then starts task execution. The information that is passed from the *UserAgent* to the *LMaster* is the ontology fragment, containing information needed to generate the scenario for the MOST software. When the computation is finished, the *LMaster* creates a JobResult message, which contains information about the job execution, the outcome and links to the result data and (any resources created by the simulation algorithm). The *UserAgent*, on the other hand, is responsible for gathering all responses from the nodes taking part in the experiment (in the case, when multiple simulations have been submitted to multiple nodes).

After specifying the scenario description, the user is redirected to a page presenting the status of the scheduled job(s). The information shown therein is periodically retrieved using a Query message, sent to the *UserAgent*. The job(s), for which the result(s) has/have already been received by the *UserAgent* are represented along with a list of output files generated by the executed process.

The complete sequence of actions and messages for a typical job execution is depicted in Figure 9.

5 Concluding remarks

The aim of this paper was discuss issues involved in applying the approach based on the *Agents in Grid* project to the tsunami research. We have presented how the *AiG* system has been modified (simplified) to instantiate the computational infrastructure of the tsunami research laboratory at the University of Aizu. With the initial setup in place for three machines, we will now proceed to increase their number. This will require stretching the system across multiple sub-networks within the University and possibly stretching it to the computers available at the SRIPAS. Furthermore, as mentioned above, this is going to result in the need for more complex negotiations and task scheduling. We will report on our progress in subsequent publications.

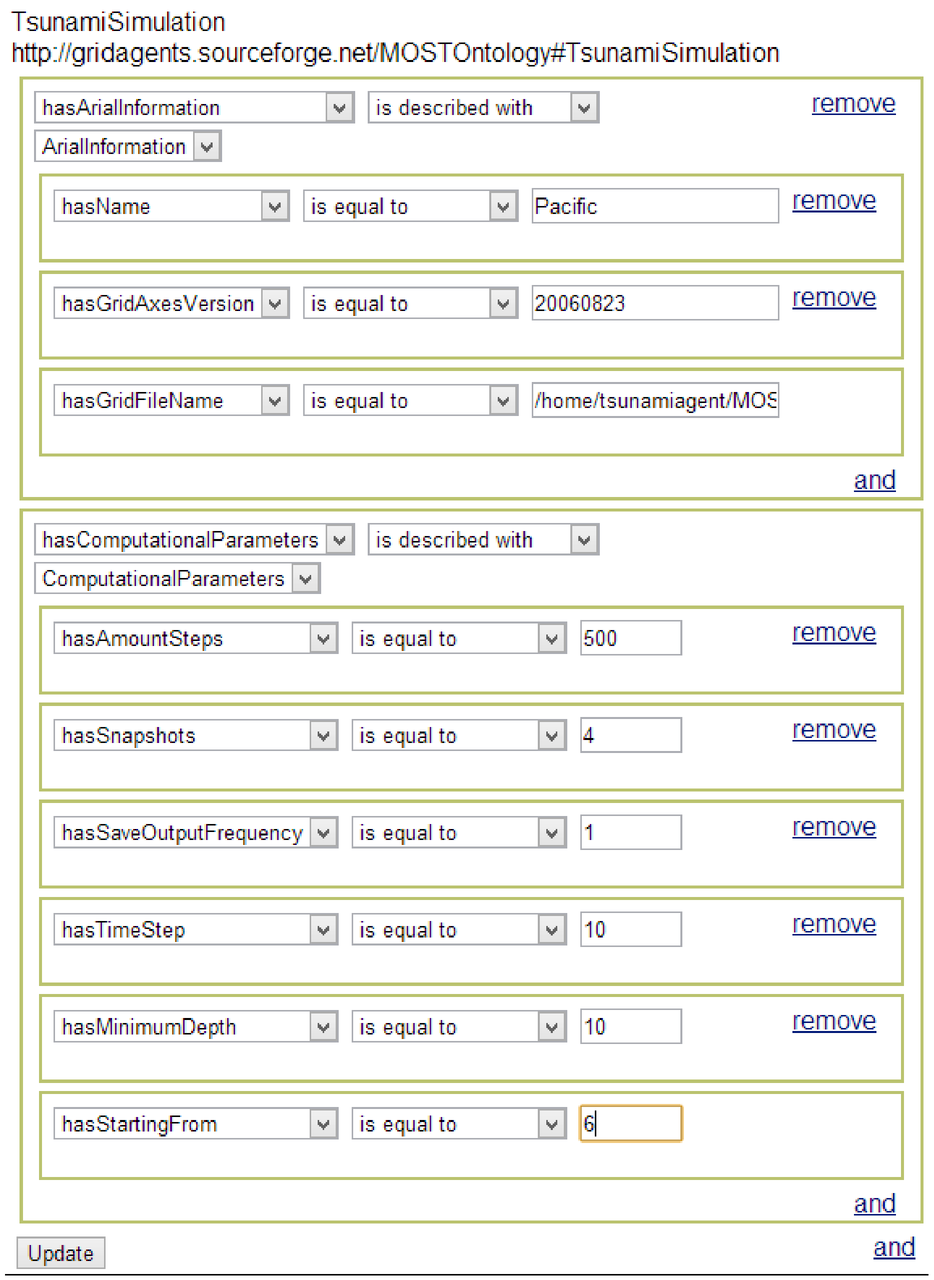
Acknowledgment

Work of Marcin Paprzycki was completed while visiting the University of Aizu.

References

1. Th. Erl, *SOA Design Patterns,* Prentice Hall, 2010.
2. M. Kuniavsky, *Smart Things: Ubiquitous Computing User Experience Design,* Elsevier, 2009.
3. Folding@home Distributed Computing, `http://folding.stanford.edu/`.
4. A. Beberg, D. Ensign, G. Jayachandran, S. Khaliq, Folding@home: Lessons from eight years of volunteer distributed computing, *in Proc. of IEEE International Symposium on Parallel & Distributed Processing (IPDPS),* Rome, Italy, 1–8, 2009.
5. X.Caiand, and P.Langtangen, Making Hybrid Tsunami Simulators in a Parallel Software Framework, *LNCS,* vol. 4699, pp. 686–693, Springer–Verlag. 2008.
6. K.Ganeshamoorthy, D. Ranasinghe, K.Silva, and R.Wait, Performance of Shallow Water Equations Model on the Computational Grid with Overlay Memory Architectures, *Proc. of the Second International Conference on Industrial and Information Systems (ICIIS 2007), IEEE Press,* Sri Lanka, 415–420, 2007.
7. D. George, TsunamiClaw User's Guide, `http://faculty.washington.edu/rjl/pubs/icm06/TsunamiClawDoc.pdf/pubs/icm06/TsunamiClawDoc.pdf`
8. N. Shuto, F. Imamura, A. C. Yalciner, G. Ozyurt, TUNAMI N2; Tsunami modelling manual, `http://tunamin2.ce.metu.edu.tr/`
9. V. Titov. Numerical Modeling of Tsunami Propagation by using Variable Grid'. *Proc. of the IUGG/IOC International Tsunami Symposium, Computing Center Siberian Division USSR Academy of Sciences, Novosibirsk,* USSR, 46–51, 1989
10. V. Titov and F. Gonzalez, Implementation and Testing of the Method of Splitting Tsunami (MOST). *Technical Memorandum ERL PMEL-112,* National Oceanic and Atmospheric Administration, Washington DC, 1997.
11. J.C. Borrero, K. Sieh, M. Chlieh, and C.E. Synolakis, Tsunami Inundation Modeling for Western Sumatra, *Proc. of the National Academy of Sciences of the USA,* Vol. 103, N 52, http://www.pnas.org/content/103/52/19673.full, 2006.
12. A. Vazhenin, K. Hayashi, Al. Romanenko, Service-oriented tsunami wave propagation modeling tools, *in Proc. of the Joint International Conference on Human-Centered Computer Environments (HCCE '12),* Aizu-Wakamatsu, Japan, ACM Publisher, 131–136, 2012.
13. A. Vazhenin, M. Lavrentiev, A. Romanenko, An. Marchuk, Acceleration of Tsunami Wave Propagation Modeling based on Re-engineering of Computational Components, *International Journal of Computer Science and Network Security,* vol.13, no. 3, 24–31, 2013.

14. `iisee.kenken.go.jp/staff/fujii/OffTohokuPacific2011/tsunami.html`
15. Katarzyna Wasielewska, Michal Drozdowicz, Maria Ganzha, Marcin Paprzycki, Naoual Attaui, Dana Petcu, Costin Badica, Richard Olejnik, and Ivan Lirkov. Negotiations in an Agent-based Grid Resource Brokering Systems. Trends in Parallel, Distributed, Grid and Cloud Computing for Engineering. Saxe-Coburg Publications, Stirlingshire, UK, 2011.
16. Wojciech Kuranowski, Maria Ganzha, Maciej Gawinecki, Marcin Paprzycki, Ivan Lirkov, and Svetozar Margenov. Forming and managing agent teams acting as resource brokers in the grid–preliminary considerations. International Journal of Computational Intelligence Research, 4(1):9–16, 2008.
17. Michal Drozdowicz, Maria Ganzha, Katarzyna Wasielewska, Marcin Paprzycki, and Pawel Szmeja. Using ontologies to manage resources in grid computing: Practical aspects. In Sascha Ossowski, editor, Agreement Technologies, volume 8 of Law,Governance and Technology Series, Springer, 149–168, 2013.
18. Michal Drozdowicz, Maria Ganzha, Marcin Paprzycki, Pawel Szmeja, Katarzyna Wasielewska, OntoPlay – a flexible user-interface for ontology-based systems, `http://ceur-ws.org/Vol-918/111110086.pdf`
19. A. M. Fridman, L. S. Alperovich, L. Shemer, L. A. Pustilnik, D. Shtivelman, An. G. Marchuk,and D. Liberzon, Tsunami wave suppression using submarine barriers, Physics–Uspekhi, vol. 53, no. 8, pp. 809–816, 2010.
20. A. Vazhenin, Y. Watanobe, K. Hayashi, M. Drozdowicz, M. Ganzha, M. Paprzycki, K. Wasielewska, P. Gepner. Agent-based resource management in Tsunami modeling.Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on, 1047-1052, 2013



TsunamiSimulation
http://gridagents.sourceforge.net/MOSTOntology#TsunamiSimulation

hasArialInformation | is described with | remove
ArialInformation

hasName	is equal to	Pacific	remove
hasGridAxesVersion	is equal to	20060823	remove
hasGridFileName	is equal to	/home/tsunamiagent/MOS	

and

hasComputationalParameters | is described with
ComputationalParameters

hasAmountSteps	is equal to	500	remove
hasSnapshots	is equal to	4	remove
hasSaveOutputFrequency	is equal to	1	remove
hasTimeStep	is equal to	10	remove
hasMinimumDepth	is equal to	10	remove
hasStartingFrom	is equal to	6	

and

Update | and

Fig. 8. Simulation scenario

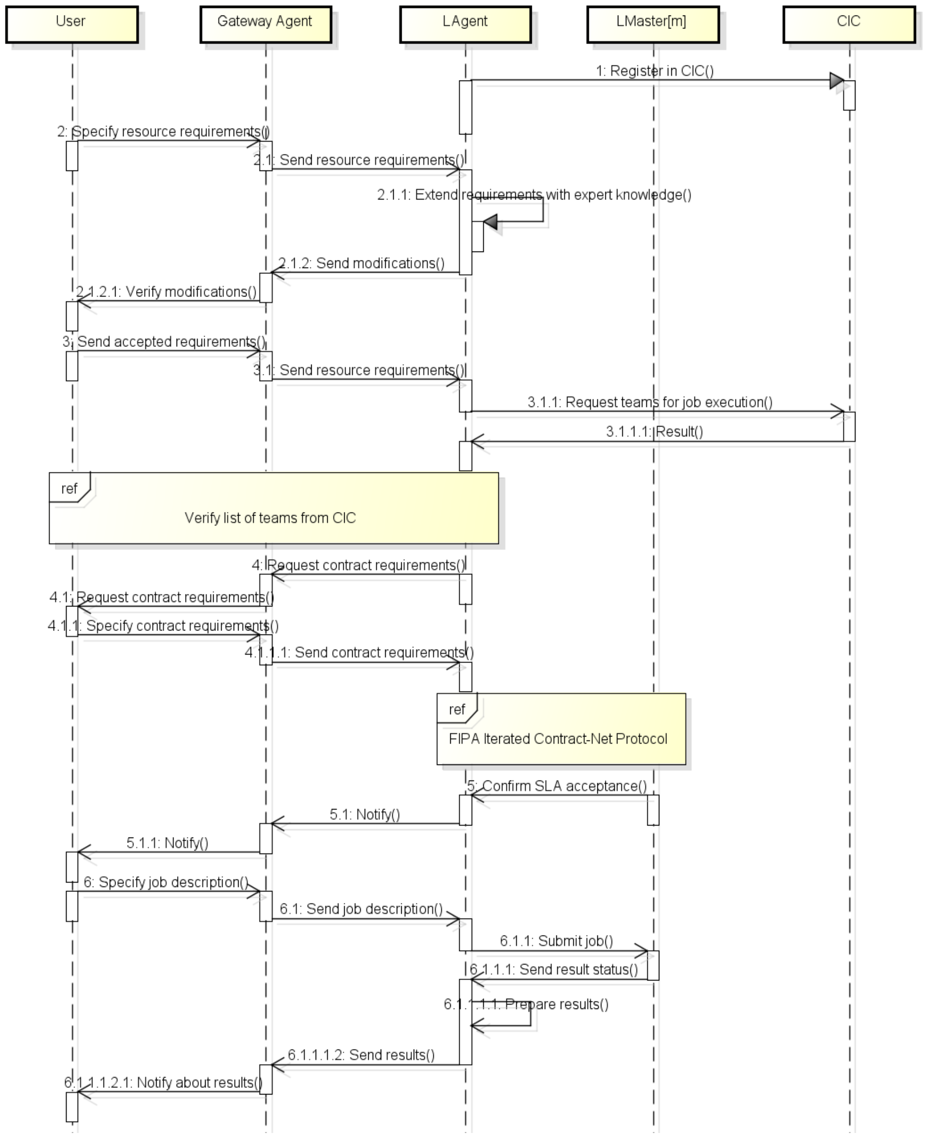

User
Gateway Agent
LAgent
LMaster[m]
CIC
1: Register in CIC()
2: Specify resource requirements()
2.1: Send resource requirements()
2.1.1: Extend requirements with expert knowledge()
2.1.2: Send modifications()
2.1.2.1: Verify modifications()
3: Send accepted requirements()
3.1: Send resource requirements()
3.1.1: Request teams for job execution()
3.1.1.1: Result()
ref
Verify list of teams from CIC
4: Request contract requirements()
4.1: Request contract requirements()
4.1.1: Specify contract requirements()
4.1.1.1: Send contract requirements()
ref
FIPA Iterated Contract-Net Protocol
5: Confirm SLA acceptance()
5.1: Notify()
5.1.1: Notify()
6: Specify job description()
6.1: Send job description()
6.1.1: Submit job()
6.1.1.1: Send result status()
6.1.1.1.1: Prepare results()
6.1.1.1.2: Send results()
6.1.1.1.2.1: Notify about results()

Fig. 9. AiG job execution sequence