A COMPARISON OF VISHNU AND OPL STUDIO

David Montana
BBN Technologies
10 Moulton Street, Cambridge, MA 02138
dmontana@bbn.com

Abstract A reconfigurable scheduler is one that can be configured to solve a wide range of different scheduling problems without changing the software. Reconfigurable scheduling holds the promise of drastically reducing the cost of developing automated, optimized scheduling solutions and hence making optimized scheduling available for many real-world problems for which the cost has so far been prohibitive. While this promise is still largely unrealized, some reconfigurable scheduling applications are making progress towards this goal. The best known and the standard setter is OPL Studio, a commercial product from ILOG. A second is one that we have developed, called Vishnu. By comparing these two reconfigurable schedulers, we gain insight into both of them, as well as potential future developments in the area.

1. Introduction

While there have been many real-world success stories for automatic optimized scheduling, the vast majority of scheduling applications are still performed either manually or using simple dispatch rules. A challenge for the coming years is to make optimized scheduling accessible to the applications for which it has until this point been impractical due to costs and complexities. A key part of addressing this challenge is likely to be what we refer to as "reconfigurable" scheduling [Montana, 2001a].

Most optimizing schedulers have targeted a single problem or narrow class of problems. Changing such a scheduler to handle a new problem or domain has required redesigning the scheduler and rewriting portions of its software. This is expensive and time-consuming and therefore has limited the applicability of this technology.

In contrast, a reconfigurable scheduler can handle a wide range of different scheduling applications and domains without modification of its software. The user specifies the problem to solve (data formats, scheduling logic, etc.) as configuration data, and a solver automatically produces schedules. Reconfigurable scheduling offers the promise of making optimized scheduling more of a commodity item (like a spreadsheet).

While there has been significant progress in the development of reconfigurable scheduling, the promise remains largely unfulfilled. In this paper, we examine two reconfigurable schedulers, OPL Studio and Vishnu. We enumerate their similarities and differences, strengths and weaknesses. Based on these comparisons, as well as our experiences using Vishnu, we summarize with some ideas on what it will take for reconfigurable scheduling to realize its promise.

OPL Studio

The Optimization Programming Language (OPL) [Van Hentenryck, 1999] and its associated solver [Van Hentenryck et al., 2000] were originally a synthesis of previous work in constraint programming (from the artificial intelligence community) and mathematical programming (from the operation research community). On the constraint programming side, languages such as CHIP [Dincbas et al., 1988] and ODO [Davis and Fox, 1994] allowed the expression as "programs" of optimization problems using symbolic constraints. Constraint-based solvers could then perform the optimization specified in such a program, usually using tree search methods. On the operations research side, languages such as AMPL [Fourer et al., 1993] and GAMS [Bisschop and Meeraus, 1982] allowed the expression as "programs" of optimization problems using numerical constraints, i.e. mathematical programming problems. Analogously, solvers could perform the optimization specified by such a program. OPL combined the ability to handle numerical and symbolic constraints, as well as combining the best of both solution techniques. OPL also added a variety of scheduling-specific concepts to make scheduling as much a primary emphasis as other types of combinatorial optimization. One more important feature that helps set OPL apart is that it is released as a commercial product, OPL Studio, with all the benefits that such a product (support, documentation, stability, etc.) provides.

Vishnu

In constrast, Vishnu grew out of work on scheduling using evolutionary algorithms and, more generally, metaheuristics. Most work on evolutionary algorithms and metaheuristics for scheduling has focused on special-purpose solutions tuned to particular problems. While there are some others in this area investigating schedulers with some amount of reconfigurability, such as [McIlhagga, 1997], [Raggl and Slany, 1998] and [Cowling et al., 2000], Vishnu was the first to incorporate reconfigurability in the same broad sense as OPL. The original version of Vishnu, which was developed under government funding, is available open source for free download at the web page [Montana, 2001c]. Since then, we have added some critical new features, and while we have not yet made them publicly available we will discuss them below in the comparison. Since Vishnu was developed originally without any knowledge of the already existing OPL and since it grew out of a different body of work, it provides a good counterpoint for comparison. A more detailed, though still incomplete, discussion of Vishnu is available in [Montana, 2001b] and at the web page [Montana, 2001c].

2. Comparisons

We divide our comparisons of the two reconfigurable schedulers into three sections corresponding to their three major components: (i) problem/project specification, (ii) optimization/solving, and (iii) the surrounding system.

Specifying a Problem/Project

In comparing the two approaches to problem specification, the first difference to note is that of terminology: what Vishnu calls a *problem* OPL Studio calls a *project*. Both recognize that a problem (we will use Vishnu terminology) is divided into two parts. One part, which Vishnu calls the *specifications* and OPL Studio calls *model*, includes the data formats and the scheduling logic and is what stays constant over time. The second part is the *data*, which is often specified separately because it is large, it can vary, and it often comes from an external source.

Another immediately apparent difference is that they provide different formats for specifying a problem. OPL Studio represents a problem essentially as a computer program (with OPL being the language in which the programs are written). The data formats are specified via struct commands similar to those in the C programming language. In addition to standard constructs for standard operations such as arithmetic and looping, there are special keywords in the language for scheduling-related concepts. (For example, *requires* specifies that a certain task requires a certain resource). Data is included by leaving placeholders, including indefinite-sized arrays, in the model and then specifying the values in another place, usually another file.

An example OPL program for the Muth-Thompson job-shop scheduling problem is

```
int nbJobs = ...;
range JOBS 1..nbJobs;
int nbRes = ...;
range RESOURCES 1..nbRes;
int resource[JOBS,RESOURCES] = ...;
int duration[JOBS,RESOURCES] = ...;
```

```
Activity task[i in JOBS, j in RESOURCES](duration[i,j]);
var int Makespan in 0..totalDuration;
UnaryResource tool[RESOURCES];
minimize
   Makespan
subject to {
   forall(i in JOBS)
      Makespan >= task[i,nbRes].end;
   forall(i in JOBS)
      forall(j in 1..nbRes-1)
         task[i,j] precedes task[i,j+1];
   forall(i in JOBS)
      forall(j in RESOURCES)
         task[i,j] requires tool[resource[i,j]];
};
```

Vishnu has a different approach. The data formats are represented (i) as a table when viewed in the GUI, (ii) as XML when written to a file, or (iii) as database table entries when stored in a database. Despite their different representation, Vishnu data formats contain essentially the same information as the OPL structs, which is the names of the data fields plus their associated data types. The data itself is represented essentially the same way as the data formats.

The scheduling logic in Vishnu is specified by a set of formulas, each one associated with a particular hook. A hook is a place where the scheduling algorithm can look for a particular piece of information. For example, the Capability hook contains the information whether a particular resource is capable of performing a particular task. Similarly, the Task Unavailabile Times hook determines for what intervals of time a particular task is not available to be scheduled. The user (application developer) specifies the information on a hook as a formula that the scheduling algorithm can evaluate within an appropriate context. For example, the formula resource.skillLevel >= task.requiredSkill placed on the Capability hook would mean that a resource is capable of performing a task if the resource's skillLevel field is at least as large as the task's requiredSkill field.

An example Vishnu problem specification showing the same information as is expressed above using OPL is:

step (task)

Field	*Type*
id	string
duration	number
machine	string
preceedingStep	string

machine (resource)

Field	*Type*
id	string

Hook	Formula
Optimization Criterion	maxover (resources, “res”, complete (res)) - starttime
Capability Criterion	task.machine = resource.id
Task Duration	task.duration
Prerequisites	if (task.preceedingStep != “”, list (preceedingStep))
Task Unavailable Times	mapover (prerequisites, “task2”, interval (starttime, task-endtime (task2)))

[The above scheduling logic is likely to be hard to decipher for those not familiar with the purpose of each Vishnu hook and the functions available for the formulas. The documentation at [Montana, 2001c] will provide all the information necessary for the interested reader to rapidly decode all this logic. We provide an explanation only for the first hook and formula. The function *maxover* loops over a list finding the maximum value of an expression involving a named element of the list. The function *complete* returns the end time of the last task assigned to a resource. The variable *starttime* is the earliest time that a task can be scheduled, i.e. the start of the scheduling window. Hence, placing that formula on the Optimization Criterion hook makes the optimization criterion be the makespan.]

Each of these two different approaches has its advantages. OPL Studio provides a compact representation of a problem that is intuitively clear to those users who have experience with computer programming and scheduling. In contrast, Vishnu's use of formulas similar to those used in Excel can be more intuitive to non-programmers. Additionally, the way that Vishnu sets up a table of hooks with their associated formulas provides a type of checklist for what types of information can be specified that can quickly bring novices up to speed. Vishnu's use of XML makes for a verbose textual representation, but the dual use of XML and database tables for representing data enables the multi-user system with real-time data feeds from external systems described in Section 1.2.0. One other important difference is that the way constraints are expressed as individual statements in OPL is well suited to feeding a constraint-based scheduler, while the way constraints are potentially aggregated into formulas in Vishnu is better suited for the genetic-plus-greedy scheduler that Vishnu uses (described in Section 1.2.0).

While the two reconfigurable schedulers use different formats for specifying problems, much of the same types of information can be expressed (as one might expect). Some of the key types of information that they both can express are:

- **Tasks/Activities and Resources -** Both OPL and Vishnu allow the definition of *tasks* (or *activities* in OPL) to be scheduled and resources to perform the tasks. In Vishnu, in addition to tasks there are also activities, which are fixed times of unavailability for resources (such as doctor's appointments or scheduled maintenance).

- **Duration -** Both schedulers allow specification of how long a task will take to perform that is dependent on which resource is performing it.
- **Wrapup/Setup/Transition Times -** Both schedulers allow for the times that resources must potentially spend between tasks, with OPL calling this *transition* times and Vishnu referring to *setup* and *wrapup* times. In Vishnu, these times can be separately color coded on the Gantt charts.
- **Capability -** Both schedulers allow specification of which resources can perform which tasks. In Vishnu, this can be a dynamic concept dependent on the current state of the resource.
- **Multitasking and Multiple Resources -** Both schedulers allow specification of whether a resource can handle only a single task at a time or multiple tasks at a time. They also allow a task to require multiple resources.
- **Capacity -** Both schedulers permit specification of limits on the number of tasks that a resource can perform over a period of time or (in the case of multitasking) at one time.
- **Task Unavailability -** Both schedulers provide a way to specify when a task is not available to be scheduled, usually due to constraints on precedence, release dates, or due dates.
- **Scheduling Horizon -** Both schedulers allow specification of an earliest and latest time at which tasks can be scheduled.
- **Optimization Criterion -** Both schedulers allow specification of the single evaluation function (potentially a combination of multiple criteria) that the scheduler is trying to optimize. Vishnu also provides a way to specify to the greedy scheduler the marginal cost of assigning a particular task to a particular resource.
- **Search Directives -** Vishnu allows the user to select the parameters of the genetic algorithm (population size, convergence rate, etc.). OPL Studio not only provides the means for a user to select between search heuristics but also allows users to define search heuristics of their own.

Some types of constraints that are in OPL but not yet in Vishnu are the following. (Note that all of these are consistent with the Vishnu scheduling algorithm. We have just not yet needed this functionality for any problem to which we have applied Vishnu yet.)

- **Breakable activities -** OPL provides the ability to allow tasks to be performed over multiple disconnected periods of time.

- **Discrete energy resources -** OPL allows scheduling of tasks at various levels of time resolution. This allows scheduling of near-term tasks to a high level of detail and scheduling further in the future more coarsely.

Some types of constraints that are in Vishnu but not in OPL are the following.

- **Grouped multitasking -** Vishnu allows specification that two tasks can be performed simultaneously by the same resource but only if they start and end at the same time. This is useful for some transportation problems. For example, a boat can transport multiple items simultaneously from one port to another port but must pick them up and drop them off at the same time.
- **Auxilliary tasks -** There are times that completing a task requires also completing some supplementary (or auxilliary) tasks, and since these tasks are not done simultaneously this cannot be modeled by multiple resources. For example, flying an airlift mission is a task that would require a plane and a crew for its duration but can also require maintenance resources before the mission and at its stopovers, air refueling resources at certain positions, and runway resources at various stops. Vishnu provides the capability to specify auxilliary tasks.
- **Dynamic rescheduling -** Vishnu provides the support to allow definition of scheduling logic not just to build a schedule from scratch but also to modify an existing schedule. Past assignments can be frozen, i.e. fixed to the same time and resource as a hard constraint. More interestingly, maintaining past assignments can also be a soft constraint. Vishnu provides functions such as previousResource and previousStartTime to access previous assignment data in formulas. Furthermore, Vishnu provides access to the commitment levels that a human user has associated with particular assignments (where a commitment level specifies how important it is that the scheduler maintain the present assignment).
- **Display directivies -** As we discuss in Section 1.2.0, Vishnu allows the user to configure custom displays. It does this using the same type of formulas used to specify the scheduling logic to allow specification of (i) color coding of Gantt charts with legends, (ii) spreadsheet-like tables with computed quantities, (iii) filters, and more.

Optimization/Solving Techniques

The two reconfigurable schedulers use two different approaches to optimizing schedules. OPL Studio primarily uses constraint-based scheduling, where each constraint becomes a branch point in a tree. A variety of heuristics are used to determine how best to construct the tree and how best to search it. Many of these heuristics apply only in limited cases, and the optimization code examines the characteristics of the problem in deciding which heuristics to apply. The constraint programming scheduler can quickly produce a good feasible schedule, and with time it works to continually improve it.

Vishnu uses an approach that involves a genetic algorithm feeding task orderings to a greedy scheduler. The greedy scheduler does the main work in building the schedule one task at a time, but the schedule thus build generally depends on the order in which the tasks were scheduled. Hence, the genetic algorithm optimizes this ordering. We have not yet had the opportunity to refine this algorithm to recognize and use specialized algorithms for special cases. This leads to comparatively poor performance on standard scheduling benchmarks (such as the Muth-Thompson job-shop scheduling problem). However, the types of problems for which reconfigurable schedulers are most applicable are not the standard benchmarks, with their very simple constraints, but rather real-world problems with complex constraints for which there are no existing algorithms. For such problems, Vishnu very quickly devises a good solution (in comparison with human schedulers) with the first output from the greedy scheduler and improves it with time via the evolutionary process.

While complex, real-world problems generally do not have existing algorithms against which to compare, they do provide a good means of comparing the performance of reconfigurable schedulers. We therefore are in the process of selecting one or two such real-world scheduling problems to implement using both Vishnu and OPL Studio in order to compare the results. Unfortunately, we have not yet completed this process.

The Surrounding System: User Interfaces, Databases and Data Interfaces

In the real world, scheduling is usually not just about performing an optimization on a particular set of data. Instead, it is usually a dynamic process, involving both creation of an initial schedule and a continuous process of schedule repair, both prior to and during execution. Furthermore, one or more humans interact with the automated scheduler to help create the final schedules, making this a mixed-initiative process. In addition, the scheduling system interacts with other systems in real time, continually reading data updates and writing schedules back.

The usefulness of a reconfigurable scheduler for application to real-world problems usually depends at least as much (and often much more) on its ability to support this full process of scheduling as on its optimization performance. We now describe the capabilities of OPL Studio and Vishnu to support such a real-time, mixed-initiative scheduling system, focusing primarily on Vishnu because it provides much greater capabilities in this area.

For visual display of schedules, OPL Studio does provide Gantt charts but not much beyond this. Mechanisms for a human scheduler to cooperatively work with the automated scheduler are minimal. For external data interfaces, OPL Studio does provide the ability to read its data from and write its schedules to either a database of any standard variety or an Excel spreadsheet. (ILOG does sell software components outside of OPL Studio to provide additional display and interface capabilities, but these require the user writing code to utilize them and hence are not inside our definition of *reconfigurable*.)

Vishnu provides the features to create a user interface for the end user whose job it is to work with the automated scheduler to create schedules. One such feature is Gantt charts with problem-specific color coding (what one of our customers, the U.S. Air Force, calls *rainbow charts*). The color coding provides easier visual recognition of task and activity types. This is done using formulas that determine which tasks and activities use which color. A second feature is the ability to create custom-designed tabular displays of the data. Formulas compute what items go in which entry of the table (similar to Excel) as well as how to color the entries so as to provide visual indications of results of interest to the user. A third feature is manual assignment, whereby either by drag-and-drop or entry in a dialog box the user can override the automated scheduler's schedule, reassigning a task to a different resource and/or a different time. A fourth feature is the ability to lock assignments in place. Vishnu provides the capability for a user to specify varying levels of commitment to an assignment, penalizing the scheduler for changing this assignment, as well as the ability to freeze an assignment, specifying to the automated scheduler that the assignment must remain fixed. A fifth feature is custom-designed filtering. Formulas provide the mechanism for users to develop their own logic for filtering the data in visual displays.

Vishnu also provides the capability to automatically create a local database for storing the problem specification, data, and schedules. One situation in which this is important is when there are multiple humans working on a schedule. With the information stored in a database, the different users can all be working simultaneously and sharing results, still allowing the automated scheduler and real-time data feeds to also update the data. Vishnu furthermore provides a web-based interface to this database that allows users to access and modify the data using a browser, even if they are outside the local network.

A third important aspect of a full scheduling system that Vishnu provides is the ability to integrate with external systems, particularly for the purpose of accepting real-time data feeds. Because all data objects must be specifically named, Vishnu can accept updates to specific data objects (e.g., a change in the value of a field of a particular task object). The XML format provides a standardized format for communicating the data. Plus, Vishnu provides all the mechanisms required to send such XML updates to update the database via the web server or to update a Java process via RMI.

3. Future Directions

Having compared the current states of the two reconfigurable schedulers, we now discuss issues that need to be addressed for these (or other) reconfigurable schedulers to fulfill their promise.

Cost - As more people use a product, the cost tends to fall. As reconfigurable schedulers prove themselves in real-world applications and start to address the other issues below, they can find more users and hence lower the cost. However, it is not clear that this process will drive the cost low enough for 20-person shops scheduling daily production or towns scheduling athletic fields. Such users may require free software (via open source development).

Optimization Performance - In the traditional settings for optimized scheduling (large organizations with large financial consequences depending on schedules), the optimality of the schedules is what matters. For the potential customers for reconfigurable scheduling (smaller, cost-conscious organizations), schedule optimality is much less important although definitely not inconsequential. The issue is cost vs. benefits, and the potential contribution of reconfigurability is massively reducing the life cycle costs while sacrificing as little as possible of the benefits of optimized schedules. So, reconfigurable schedulers need to continue improving the scheduling algorithms, though not at the expense of the other issues discussed. The approach that OPL Studio has taken of recognizing special cases and applying specialized algorithms while still maintaining the general-purpose capability has been successful and is what we should also apply in Vishnu.

System Construction - As discussed in Section 1.2.0, an important functionality of a reconfigurable scheduler is its ability to create a full scheduling system rather than just a standalone optimizing scheduler. While OPL Studio and especially Vishnu do provide some such support, there are potential improvements. These include (i) better mechanisms to support coordination between multiple human schedulers and an automated scheduler and (ii) better support for integration with common data sources (e.g., Vishnu should integrate with Excel and they both could support common integration platforms such as IBM WebSphere).

Usability - Nomatter how powerful an application may be, if it is hard to use then usually it will not be used. In the case of reconfigurable scheduling, there are two types of users, application developers and end users, and hence two types of usability. Application developers require a GUI designed to allow problem specification (which both schedulers provide), good documentation and easy installation (which OPL Studio does provide but which needs improvement in Vishnu), debugging and diagnosis tools, and tools for constructing the scheduling system including the end user GUI. End users need intuitive displays for viewing and analyzing the data and schedules, as well as mechanisms and displays for interacting with the automated scheduler. While Vishnu in particular does provide some capabilities for the end user, in both schedulers the application developer is the primary focus.

Range of Applicability - This is probably the most important and overriding issue. For both OPL Studio and Vishnu, despite their flexibility, there are still too many problems that cannot be expressed and solved. This is potentially fixable by a continual process of expanding capabilities. Our philosophy with Vishnu is that each time we encounter a new scheduling problem to solve, if Vishnu in its present form cannot solve it, then we expand its capabilities so that it can, thereby continually making it more powerful. The trick in this process is to remain general enough that the new functionality added for a specific problem will apply to a variety of other problems, because otherwise we are adding complexity without adding power. In the case of ILOG and OPL Studio, there have been some enhancements to the language and solver that extend the range of applicability, but these have been slow in being released (partly due to the existence of the ILOG Scheduler product, a set of scheduling routines on which a user can develop a custom scheduling application, that provides a fallback for problems not solvable via OPL Studio). Reconfigurable scheduling will fall short of its promise until reconfigurable schedulers are nearly universal in their ability to capture the logic of different scheduling problems and solve these problems.

References

Bisschop, J. and Meeraus, A. (1982). On the development of a general algebraic modeling system in a strategic planning environment. *Mathematical Programming Study*, 20:1–29.

Cowling, P., Kendall, G., and Soubiega, E. (2000). A hyperheuristic approach to scheduling a sales conference. In *Selected Papers of the 3rd PATAT Conference*, pages 176–190.

Davis, G. and Fox, M. (1994). ODO: A constraint-based architecture for representing and reasoning about scheduling problems. In *Proceedings of the 3rd Industrial Engineering Research Conference*.

Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., and Berthier, F. (1988). The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 693–702.

Fourer, R., Gay, D., and Kernighan, B. (1993). *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, Belmont, CA.

McIlhagga, M. (1997). Solving generic scheduling problems with a distributed genetic algorithm. In *Proceedings of the AISB Workshop on Evolutionary Computing*, pages 85–90.

Montana, D. (2001a). Optimized scheduling for the masses. In *Genetic and Evolutionary Computation Conference Workshop Program*, pages 132–136.

Montana, D. (2001b). A reconfigurable optimizing scheduler. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1159–1166.

Montana, D. (2001c). Vishnu reconfigurable scheduler home page. http://vishnu.bbn.com.

Raggl, A. and Slany, W. (1998). A reusable iterative optimization library to solve combinatorial problems with approximate reasoning. *International Journal of Approximate Reasoning*, 19(1-2):161–191.

Van Hentenryck, P. (1999). *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA.

Van Hentenryck, P., Perron, L., and Puget, J.-F. (2000). Search and strategies in OPL. *ACM Transactions on Computational Logic*, 1(2):285–320.