Explaining Complex Scheduling Decisions

Jeremy Ludwig, Annaka Kalton, & Richard Stottler
Stottler Henke Associates, Inc.
San Mateo, CA, USA
ludwig; kalton; stottler @ stottlerhenke.com

ABSTRACT
The work presented in this paper describes the explanation facility of an intelligent scheduling software framework that has been customized and deployed in a variety of domains. The customizability of the framework allows the software to develop a valid schedule that reflects each domain's specific preferences and constraints. In all domains, the software quickly solves a complex scheduling problem (generally in less than five minutes) and produces a schedule that is significantly better than those reached by previous methods. The primary contribution of this paper is to describe the explanation facility used by schedulers across a variety of real-world domains to answer the ever-present question: why was a task was scheduled *here* and not *there?* The transparency to see *why* inspires greater confidence in the results and facilitates understanding of how constraints affect the schedule, enabling the user to further improve the schedule by assessing specific constraints. This paper also outlines future work that will improve upon the existing explanations.

Author Keywords
scheduling framework; heuristics; explanations

ACM Classification Keywords
• Human-centered computing~Interactive systems and tools.

INTRODUCTION
Scheduling, at its most basic, is the process of assigning tasks to resources over time, with the goal of optimizing the result according to one or more objectives [5]. Scheduling is heavily used in aerospace, manufacturing, defense, and service industries to minimize the time and cost associated with the completion or production of large and complex projects.

The Aurora scheduling framework is one example of a general-purpose scheduler that has been successfully applied in a variety of domains, including all of those listed above [3,4,6]. Aurora combines graph analysis with heuristic scheduling techniques to quickly produce an effective schedule based on a defined set of *tasks* and *constraints*. Tasks are the actions that need to be completed, e.g. *InstallCockpitDoor.* Constraints define limits on when a task could be completed. While constraints vary across domains, they generally include:

- **Temporal:** Tasks must be scheduled between the project start and end dates. Each task has a duration and an optional start date and optional end date. For example, *InstallCockpitDoor* must be completed by 5:00 PM on March 1, 2018.
- **Ordering:** Tasks can optionally be assigned to follow either immediately after/before another task or sometime after/before another task; optionally with a specific lag time in between. For example, *PaintCockpitDoor* must happen sometime before *InstallCockpitDoor* with a lag of at least 48 hours in between to allow the paint to cure.
- **Resource:** Each task can require that resources be available for the task to be scheduled. A task might require a specific resource (e.g. *CockpitArea*) or might select from a pool of resources (e.g. three mechanics from the *Mechanics* pool).
- **Calendar:** Tasks can only be scheduled during working shifts; tasks cannot be scheduled on holidays. Resources may also have calendar constraints as well. For example, individual *Mechanics* work one of two shifts M-F while the *CockpitArea* is generally available at any time.

The framework distills the various operations involved in creating a schedule that respects all of these constraints into reconfigurable modules that can be exchanged, substituted, adapted, and extended. This framework acts as a foundation for creating scheduling tools that respect domain-specific constraints and use heuristics tuned to each domain to ensure a high-quality schedule.

The remainder of this paper will describe related work, followed by an overview of the explanation facility. The explanation facility relies on the scheduling engine to generate explanations and the user interface to visualize these explanations. The goal of the combined system is to help users understand the decisions that were made during the process of creating large and complex schedules. Following this is a discussion of explanations in Aurora and next steps for improving the explanation facility.

Related Work
There is a body of prior work outside of Aurora towards creating a general-purpose scheduling framework that forms the basis of domain-specific scheduling tools. The OZONE Scheduling Framework [7] is one example. [1] describes the validation of the OZONE concept through its application to a diverse set of real-world problems, such as transportation logistics and resource-constrained project scheduling. [2] presents a design for a general scheduling framework for

ExSS '18, March 11, Tokyo, Japan.

manufacturing. [5] presents an overview of several modern general-purpose scheduling systems such as the SAP Production Planning and Detailed Scheduling System, ASPROVA Advanced Planning and Scheduling, Preactor Planning and Scheduling Systems, and ORTEM Agile Manufacturing Suite. Each of these modern systems has a distinct feature set while sharing some aspects in common with Aurora and each other.

The primary contribution of this paper is to describe the explanation generation facility in the Aurora framework, how the resulting explanations are used by schedulers to understand the decisions that drive complex schedules, and to outline future work that would improve the utility of the explanations.

EXPLANATION FACILITY

The scheduling framework consists of two primary components: the engine and the user interface. The scheduling engine is responsible for creating an explanation for each task, describing why the task was placed in its particular position (which includes both time and resources) as the schedule is created. The user interface is responsible for presenting the explanation to the user (Figure 1) and helping them understand why a task was scheduled *here* and not *there*. This information helps direct the user's attention to the driving constraints and supports carrying out *what-if* analysis on how changing these specific constraints might improve the schedule.

For example, perhaps the user would like to see the *InstallCockpitDoor* task happen earlier in the schedule to avoid conflicting with the *InstallCockpit* task. The explanation shows that *PaintCockpitDoor* is driving the date of *InstallCockpit*. Digging deeper, the driver for *PaintCockpitDoor* is the delivery date of the door itself. The scheduler can perform a *what-if* analysis by changing the availability date of the part and rescheduling. If this has the desired effect, the next step is talking to the supplier to see if the date can be moved up.

Generation

Explanations are created by the scheduling engine as it works to generate a schedule. In order to understand the explanation, it is helpful to provide a brief overview of the scheduling engine first, followed by how explanations are created in this process.

Scheduling Engine

The scheduling engine runs through three distinct phases: initialization, scheduling, and finalization.

Initialization Phase

First, Aurora applies the **Preprocessor** to the tasks to prepare for scheduling. Examples of preprocessor tasks are setting the schedule direction for the tasks and marking resource constrained tasks for special handling. Second, the **Prioritizer** is applied to determine the order of the tasks in the scheduling queue. The Prioritizer may be re-applied within the scheduling loop.

Scheduling Phase

First, the **Scheduler** calls constraint propagation on the highest priority schedulable element to be sure that all its requirements and restrictions are up to date. Second, the Scheduler assigns the task to a time window and resources such that all constraints are satisfied. It also returns a list of the conflicts resulting from the given assignment, if any. Third, the Scheduler calls constraint propagation on the task (again) to update all the neighbors so that they are appropriately restricted by the newly scheduled element. This process may result in additional conflicts; if so, these are added to the list of conflicts from scheduling. Fourth, the Scheduler asks the **Conflict Manager** to resolve those conflicts. This process is repeated until every task in the scheduling queue is scheduled.

Finalization Phase

When the queue is empty, Aurora goes through a final conflict management step, this time at the global level. Aurora calls the **Postprocessor** on the schedule, so that any additional analysis may be done before Aurora returns the schedule results.

Generating Explanations

Explanations are generated for each task and are constructed from decisions made in each of the scheduling phases. The cumulative decisions work together to narrow the window in which the task can be scheduled by adjusting its early start and late end dates. The scheduling order determines when a task is given a chance to schedule relative to the other tasks, where it will be placed in the first available time slot in which all of its constraints can be satisfied. An explanation is composed of multiple text descriptions that describe the decisions made and why they were made leading up to the assignment of a task to a particular time and set of resources.

In the initialization phase, The Prioritizer assigns the schedule order as shown in Figure 1, which is an integral part in understanding why scheduling decisions are made. Initial constraints are also applied to each task by the Preprocessor

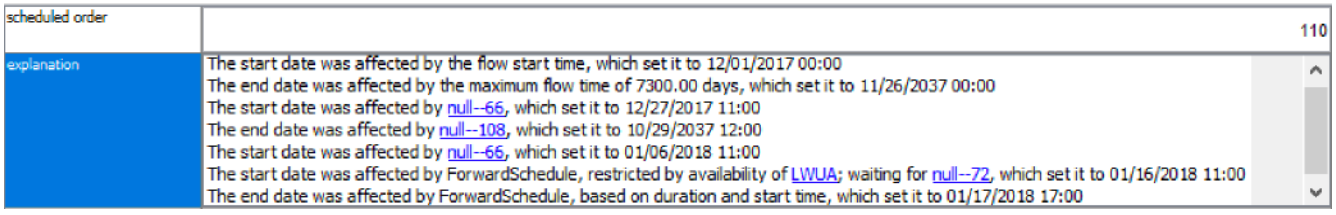

Figure 1. A scheduling explanation for a task in Aurora. Task and resource names are obfuscated to protect client data.

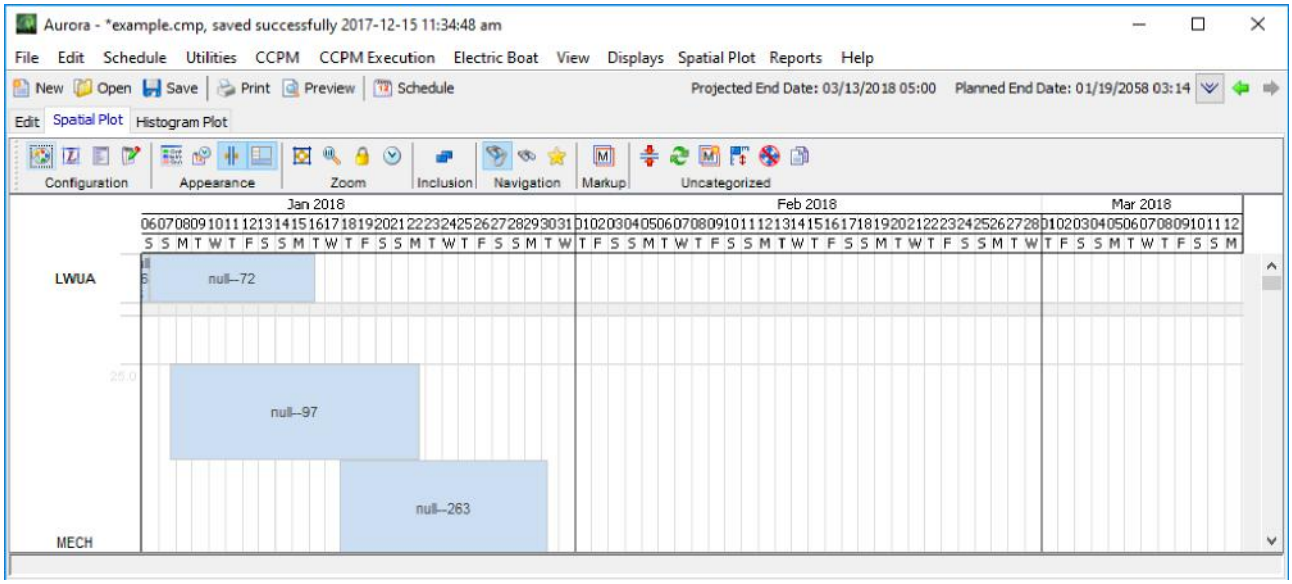

Figure 2. A plot of tasks scheduled before the explained task. The x-axis is time; the y-axis are resource constraints. The plot starts on 01/06/2018, which is the first time the task could be scheduled due to constraint propagation as shown in Figure 1. LWUA is the constraining resource, pushing out the start of the task to 01/16/2018. A mechanic from the pool of MECH is also required, but as seen in the image this is not a constraining resource.

in this phase. For example, one constraint is that a task's late end date (the date it must be finished by) can be no later than the project end date. This constraint narrows the schedule window and is recorded in the explanation. Preprocessor results are seen in lines 1 and 2 in the explanation field of Figure 1.

The scheduling phase adds to the explanation in two ways. First, constraint propagation is applied before any tasks are scheduled and again after each task is scheduled. The temporal, calendar, and resource constraints (and their interactions) will further narrow the early start dates and/or late end dates of each task. Any time the scheduling window is narrowed an explanation is recorded, as seen in lines 3-5 in the explanation field. Second, when a task reaches the front of the scheduling queue it will be placed at the first available time that meets all of the constraints. If this time does not match the early start date, then the constraints responsible for the delay are recorded in the explanation. Lines 6 and 7 in the explanation field were generated when the task was actually scheduled.

Finally, in the finalization phase the Postprocessor may move scheduled tasks and record the reason in the explanation. This supports domain-specific finalization of schedules. The example in Figure 1 does not include any explanation information from this phase.

Visualization

The user interface is responsible for presenting the explanation to the user in an understandable form. As shown in Figure 1, the scheduling explanation is part of the Scheduling Results display. The explanation portion of the result lists the constraints that have driven the scheduling of this particular task, along with its schedule order. Where other tasks are referenced, hyperlinks provide browsing support so that the user may follow the cascade of inter-related explanations through the dependent series.

The explanations displayed in the UI are a filtered view of all of the individual decisions contained in the explanation. As generated, the explanation would be a complex tree of decisions – not a simple list. The explanation presented to the user only includes the most specific path through the tree leading up to the task being scheduled. This represents the decisions that led to the final schedule but does not show all of the other possibilities that were examined.

Additionally, there are cases where the same constraint would be responsible for a number of changes. For example, TaskP precedes TaskQ. Every time the early start of TaskP gets changed, TaskQ gets changed as well. In this case, only the most recent change to TaskQ caused by TaskP is presented to the user.

The collapsing of constraints produces an unintuitive result (although still very useful) for scheduled tasks with resource constraints. The primary issue is cascading unavailability of resources. For example, first ZoneA is unavailable. Then a Crane is unavailable. Finally, Labor is unavailable. In this case, only Labor will be shown in the explanation, as it is the most recent cause of delay. This can be confusing to the user if they only look at the Labor requirement because it may appear that Labor had availability earlier; the user misses the fact that either ZoneA or the Crane was also needed but not available. The alternative of adding a line for each of the possible resources responsible for the schedule delay over

time generally results in too much information and relatively little utility for the user.

Instead, the schedule order and explanation are paired with graphic visualization to help the user understand the scheduling decisions. The graph shown in Figure 2 provides the context in which the scheduling decision is made. The x-axis is time and the y-axis represents the possible resources that would be needed by this task. The graph starts on the first available date that the task could be scheduled. Only the tasks that were scheduled before this task (based on schedule order) are plotted on the resources they use and the time they use them. This graph allows the user to visually confirm the task's final scheduling decision given the preceding decisions.

DISCUSSION

Much of this paper so far has focused on what an explanation in the Aurora scheduling framework is and how users interact with it. In order to discuss the utility of explanations, we need to examine why a scheduler wants an explanation to begin with. Generally, the users of Aurora are intimately familiar with the schedule they are creating. Whether it is running a machine shop, assembling an airplane, or managing a pharmaceutical packing line, a user is attuned to their schedule and have an intuitive sense of what the schedule should look like. When all is working correctly, explanations are rarely examined by the user.

However, when the schedule does not look 'as expected' is exactly when explanations are brought into play. The explanation facility fills one of two roles in this case. If it demonstrates that the correct scheduling decision was made, it builds trust in the overall system. On the other hand, if it demonstrates that the right decision was *not* made, the explanation facility usually points to problems in the domain specific Prioritizer or, more often, reveals an error in the scheduling model itself (tasks, resources, and constraints). For both of these reasons, we have found that scheduling explanations are an invaluable part of the Aurora framework.

Additionally, the explanation provides another perspective into the schedule, facilitating understanding. For example, if many tasks of interest have an explanation that shows the same resource causing the delay, the user would review the histogram for that resource. At this point, the user would discover that while the resource is never overloaded according to the schedule, based on the user's experience it is likely that during execution there will be times when that resource will be not as available as modeled (due to unplanned maintenance or PTO). The user then might run some scenarios to determine the effect of higher than normal absences or downtime on this critical resource. With the results of these scenarios, changes to the scheduling model or resource pool may be made.

FUTURE WORK

While the current explanation facility has demonstrated its utility for numerous deployments, there is still room for improvement. First, as pointed out constraint collapsing for scheduled tasks with resource constraints produces confusing results, such as showing that Labor is causing the delay because it was the most recent unavailable resource. It would be more informative and more accurate to report the multiple resource delays in a way that doesn't overwhelm the user. For example, collapsing all the resource delays into a single line where each resource or set of resources is included only once might provide a better at-a-glance summary. Second, it would be a significant improvement to have a one-click button that produces a graph like the one in Figure 2 for a given scheduling explanation. Currently it is tedious to configure such a graph manually; it also requires significant knowledge on how to use the Aurora filtering mechanisms. Third, it would be helpful to improve the visual representation of the explanation. A small improvement would be to number the lines and use icons and colors to note which part of the scheduling process generated each explanation line (Preprocessor, Constraint Propagation, Forward/Backward Scheduler, PostProcessor). A larger improvement might visualize the explanation list as a horizontal timeline, with markers for the changes that adjust the early start and late end dates. We plan to address all three changes as part of future work.

REFERENCES

1. Marcel Antoine Becker. 1998. Reconfigurable Architectures for Mixed-initiative Planning and Scheduling. Carnegie Mellon University, Pittsburgh, PA, USA.
2. Jose M. Framinan and Rubén Ruiz. 2010. Architecture of manufacturing scheduling systems: Literature review and an integrated proposal. *European Journal of Operational Research* 205, 2: 237–246. https://doi.org/10.1016/j.ejor.2009.09.026
3. Annaka Kalton. 2006. Applying an Intelligent Reconfigurable Scheduling System to Large-Scale Production Scheduling. In *International Conference on Automated Planning & Scheduling (ICAPS) 2006*.
4. J. Ludwig, R. Richards, A. Kalton, and D. Stottler. 2017. Applying a heuristic-based scheduling framework in manufacturing, service, and communication domains. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 1–4. https://doi.org/10.1109/SMC.2017.8122568
5. Michael L. Pinedo. 2016. *Scheduling*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-26580-3
6. Robert Richards. 2010. Critical Chain: Short-Duration Tasks & Intelligent Scheduling in e.g., Medical, Manufacturing & Maintenance. In *2010 Continuous Process Improvement (CPI) Symposium*.
7. S.F. Smith, O. Lassila, and M. Becker. 1996. Configurable, Mixed-Initiative Systems for Planning and Scheduling. In *Advanced Planning Technology*, A. Tate (ed.). AAAI Press, Menlo Park, CA.