17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID # 561

Toward Flexible and Powerful Scheduling Rule Setup using CodeDOM for .NET Framework

Soojeon Lee[a]*, In Jun Kim[a], Byoung-Sun Lee[b c]

[a] *Artificial Intelligence Research Lab., ETRI, 218 Gajeong-ro, Yuseong-gu, Daejeon, 34129, KOREA, {soojeonlee, ijkim}@etri.re.kr*
[b] *Telecommunications & Media Research Lab., ETRI, 218 Gajeong-ro, Yuseong-gu, Daejeon, 34129, KOREA, lbs@etri.re.kr*
[c] *Artificial Intelligence Major, University of Science and Technology, 217 Gajeong-ro, Yuseong-gu, Daejeon, 34113, KOREA, lbs@ust.ac.kr*

* Corresponding Author

Abstract

Modern Mission Planning and Scheduling Systems (MPS) generally provide autonomous scheduling capabilities such as conflict detection, conflict resolution and optimization. To perform the scheduling efficiently, operators need to build proper scheduling rules *a priori*. However, as the missions are getting complex, traditional rule DBs containing only static information (e.g., relations among mission entities) cannot satisfy various user requirements. Thus, dynamic rule setup methods (e.g., based on a script language such as Python) have been adopted for various space missions. But in many cases, the programming language (e.g., .NET/C#) used for developing the MPS software itself is different from the rule setup script language (e.g., Python). This language discrepancy sometimes brings development burdens to developers since various interfaces are required for adaptation. In this paper, we introduce a Code Document Object Model (CodeDom) based flexible and powerful rule setup methodology for .NET framework and show that the C# based rule setup method reduces the software development complexity.
Keywords: Mission Planning, Scheduling, Rule

Acronyms/Abbreviations

Mission Planning and Scheduling System (MPS), Code Document Object Model (CodeDom), Commercial off-the-shelf (COTS)

1. Introduction

From the viewpoint of space operations, scheduling rules have become more complex as user demands become increasingly sophisticated [1-6]. In this paper, we briefly introduce the basic elements and concepts of mission scheduling and explain how to easily and effectively implement complex scheduling rules in real systems using CodeDom.

1.1 Scheduling Unit

The basic unit subject to scheduling is an *entity*; events, missions, tasks, etc. can be the types of an entity. In general, entities have start time, end time, and other properties (e.g., duration, type, ID, name, etc.) as attributes. Because of the structural characteristics of entities, they are usually saved in .xml or .json file formats. Furthermore, for the user convenience of mission operators, entities can be displayed on the computer screen in the form of a chart. Fig. 1 shows an example where entities are reflected in a chart so that users can easily manage and schedule them.

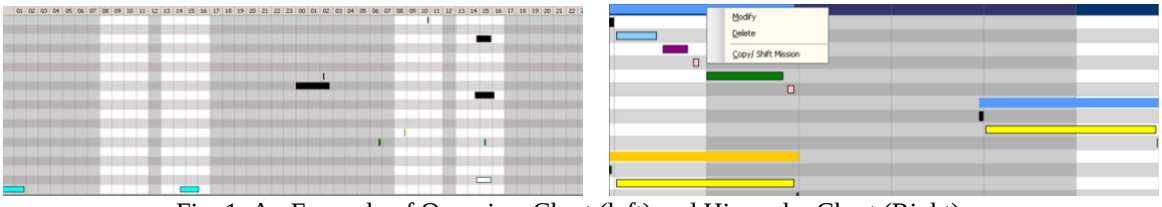

Fig. 1. An Example of Overview Chart (left) and Hierarchy Chart (Right)

1.2 Relationships between Entities

There are various types of relationships depending on the user requirements. In this paper, we only introduce some representative ones due to space limitation. They include

- **Hierarchy**: Entities can have hierarchies with each other (e.g., *Mission_A = Task_B + Task_C*). Moreover, certain properties of an ancestor entity may be inherited by its descendant entities.
- **Priority**: Entities can have a different priority. When a collision between an entity with a higher priority and an entity with a lower priority occurs, it can be handled (e.g., the entity with a lower priority is discarded) by a predefined scheduling rule.
- **Overlap**: It is required to check whether execution times of entities overlap in part or in total. Depending on the result, proper rules (e.g., exclusion/inclusion) can be applied.
- **Sequence**: The order of executions between entities can be configured (e.g., *Mission_B* must be followed by *Event_D* while *Event_D* must precede *Task_E*).
- **Spacing**: (Min or max) time interval between entities can be configured (e.g., Time gap between *Mission_C* and *Mission_D* must be larger than 60 seconds).

1.3 Constraints

Entities may be constrained by certain conditions. For example, it may be necessary to determine whether a particular property of a particular entity is within an allowed range. For example, a constraint (e.g., the duration of *Mission_B* must be less than 10 seconds) can be defined as a scheduling rule in advance.

2. Considerations on Scheduling

In general, a scheduler takes inputs, performs scheduling and feedbacks the result to users. Fig. 2 shows an example of scheduling workflow for satellite control & monitoring.

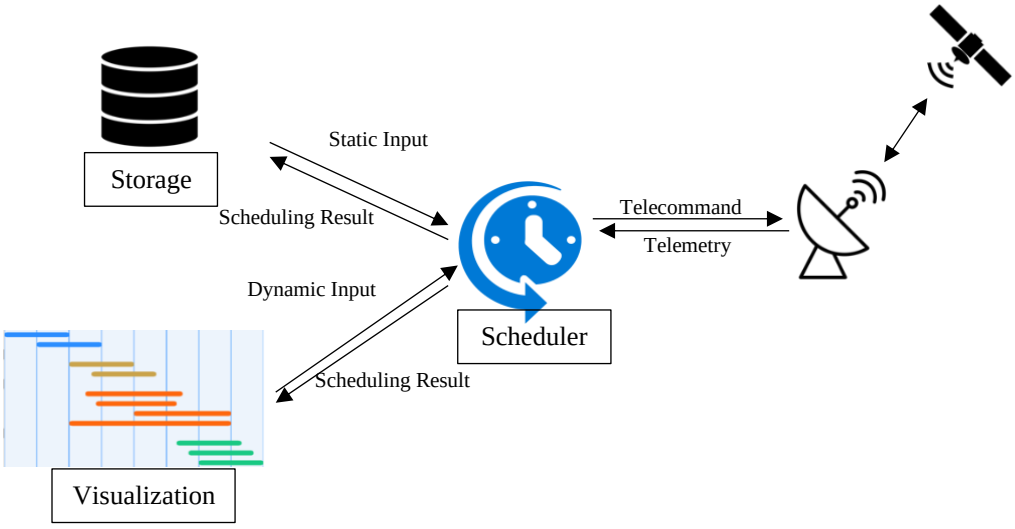

Fig. 2. An Example of Scheduling Workflow

2.1 Inputs

Scheduling rules take two types of inputs as follows.

- **Static inputs**: The definitions of "entities", "relationships between entities", "constraints" are not updated frequently thus rather static. Once they are properly setup in MPS, they do not change for a long period of time (e.g., several months).
- **Dynamic inputs**: The user requests such as for payload operations (e.g., imaging) or maneuver (e.g., station keeping and wheel-offloading) change frequently (e.g., daily basis) thus they are rather dynamic. In general, dynamic inputs are visually managed with chart (e.g., Gantt chart).

2.2 Scheduler

A scheduler runs its scheduling algorithms using the static and dynamic inputs. The scheduling algorithms are usually pre-defined in scheduling rules.

- **Typical (Basic) algorithms**: The algorithms widely used for various space missions can be categorized into typical ones. For instance, they simply use the relationships between entities (i.e., hierarchy, priority, overlap, sequence and spacing) and constraints described in Section 1. Due to their generality, the same algorithm can be used for other space missions without any modification.
- **Non-typical (Specific) algorithms**: Since they are dedicated to a specific space mission in general, non-typical algorithms cannot be used for other space missions. Furthermore, many of them are complex (for a special purpose) and even *need to be modified from time to tim*e during the mission lifetime.

The purpose of scheduling algorithms includes but is not limited to conflict detection, conflict resolution, optimization, and so on.

2.3 Outputs

The scheduling result is usually feedbacked to operators in the form of visualization (e.g., through Gantt chart) and data itself (e.g., .xml/.json files). If successful, for instance, the result is converted into telecommands and sent to a satellite.

3. Effect of Programming Languages for MPS Software

In many cases, the programming languages used for implementing the MPS software in industry require high performance, flexible UI interface, efficiency and plenty of COTS supports. Thus, powerful programming languages such as C/C++, .NET/C#, and JAVA are widely adopted.

Note that before the MPS software is deployed to a real system, the source codes must be compiled *a priori*. That means the source code modification is difficult after deployment. Due to this nature, typical (basic) scheduling algorithms mentioned in Section 2.2 are suitable for being implemented by those programming languages (i.e., C/C++, .NET/C#, and JAVA) since the typical scheduling algorithms are too common to change during mission lifetime; thus source code modification is not required.

3.1 Script-based Implementation for Scheduling Algorithms

3.1.1 Pros.

However, non-typical (specific) scheduling algorithms explained in Section 2.2 are more mission specific, often incomplete (or unfinalized) and may change or be newly added even during an operation phase. The difficulty comes for instance, when a non-typical scheduling algorithm must be modified after deployment. That means the source code must be compiled and validated again and then the MPS software needs to be re-distributed. This is time consuming and risky. Furthermore, *it cannot be done on a running system*; i.e., the MPS software must be shut down before update.

To cope with this problem, script languages (e.g., Python) which focus on simplicity and fast development can be considered for implementing non-typical scheduling algorithms. The most obvious advantage of using script languages in this case is easy deployment; re-compilation of source code is not required and the changes can be applied immediately to the running system without shutting down it.

3.1.2 Cons.

As discussed above, script-based approach brings a lot of flexibility in implementing and deploying scheduling algorithms. However, the language discrepancy between the scheduling algorithms (e.g., via Python) and the other parts of the MPS software (e.g., via .NET/C#) often causes some development burdens to developers since various interfaces are required for adaptation.

One example is the decoupling with GUI. The script language used for describing scheduling algorithms cannot directly handle various functions provided by other programming languages used for MPS implementation due to lack of APIs. For instance, to be used for an input of a scheduling algorithm, the entities being manipulated by a user through Gantt chart must be collected (e.g., as an array) and converted (e.g., to a .xml file) for adaptation. Furthermore, after scheduling, if some conflicts are detected, the conflict information needs to be reflected back to the Gantt chart (e.g., displaying conflicted entities red). However, this also cannot be done directly and requires adaption due to the language discrepancy.

3.2 CodeDom based Implementation for Scheduling Algorithms

We introduce a Code Document Object Model (CodeDom) based flexible and powerful methodology in describing scheduling algorithms for .NET/C# framework and explain that the C# based description of scheduling algorithms reduces the software development complexity. Fig. 3 briefly shows how to enable this. A scheduling algorithm developer writes codes in C#. Usually, a C# class is the unit of representing a scheduling algorithm. After that, the developer validates the codes.

- **Delivery of a scheduling algorithm to MPS**: If the validation is successful, the developer delivers the scheduling algorithm to MPS through Algorithm Editor in Fig.3. The delivery can be done in the form of a file (i.e., .cs, an entire C# source code file format itself) or a string (i.e., containing the C# class source code as a character array).
- **Applying the (new) scheduling algorithm to MPS**: The newly delivered algorithm can be integrated into MPS in the form of executable files (e.g., .dll or .exe) or through memory. But the former requires rebooting the MPS software to apply the scheduling algorithm hence the latter is more preferable for continuous operation of MPS.

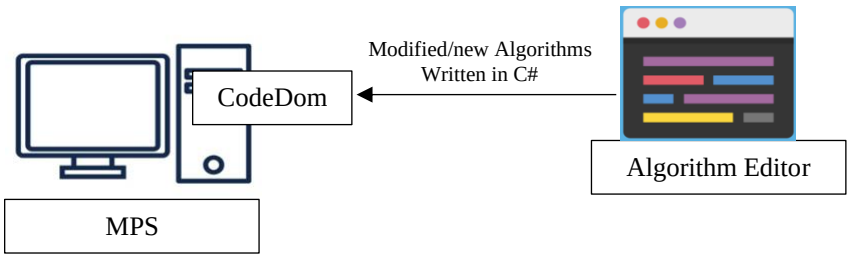

Fig. 3. How a Scheduling Algorithm written in C# is integrated to MPS

Table 1 shows an example how to enable this in the source code level. In *Scheduler.SchedulingAlgorithmManager* class, *CompileAndApplyAlgorithm* (line 15) method exists. This method gets the algorithm source code written in C# as a string. Note that *GetAlgorithmClassText* (line 31) method is an example reading the scheduling algorithm class through Algorithm Editor in Fig.3. By combining headerCodes/tailCodes (i.e., template of head/tail of a scheduling algorithm) and bodyCodes, the source code can be provided as a string (line 34). The new algorithm can be compiled and saved to an executable file (e.g., .exe or .dll). However, in that case, to apply the new algorithm to MPS, we must stop and restart MPS. To avoid that situation, it is proper to manage the new algorithm in memory so that MPS can use the new algorithm immediately without shutdown; corresponding setup is done in line 39. Finally, the algorithm is compiled and ready to be used (line 42~43).

Table 1. Management of a Scheduling Algorithm *directly* from the MPS software using CodeDom

```
1   /* ... */
2       using System;
3       using System.CodeDom.Compiler;
4       /* ... */
5
6       namespace Scheduler
7       {
8           public class SchedulingAlgorithmManager
9           {
10              /* ... */
11              string headerCodes = string.Empty;
12              string tailCodes = string.Empty;
13              string bodyCodes = string.Empty;
```

```
14
15          private void CompileAndApplyAlgorithm(object sender, RoutedEventArgs e)
16          {
17              /* Template of head/tail of a scheduling algorithm */
18              headerCodes =
19                  @"
20                  using System;
21                  using System.Collections.Generic;
22                  using System.Text;
23                  using System.Windows;
24
25                  namespace SchedulingAlgorithm
26                  {";
27              tailCodes = @"
28                  }";
29
30              /* Main body of a scheduling algorithm */
31              bodyCodes = GetAlgorithmClassText();
32
33              /* Construct CSharp source code containing a scheduling algorithm */
34              string codes = headerCodes + bodyCodes + tailCodes;
35
36              /* Apply the new (modified) algorithm on the fly */
37              CodeDomProvider cdp = new CSharpCodeProvider();
38              CompilerParameters parameters = new CompilerParameters(referenceAssemblies);
39              parameters.GenerateInMemory = true;
40
41              /* Compile from code string */
42              CompilerResults cr = cdp.CompileAssemblyFromSource(parameters, codes);
43              Assembly asm = cr.CompiledAssembly;
44              /* ... */
45          }
46          /* ... */
47      }
48  }
49
50
```

4. Conclusion

We introduce a CodeDom-based scheduling algorithm implementation methodology with .NET/C# framework. It is flexible like Python and powerful like C# in describing, distributing and executing scheduling algorithms in MPS.

Furthermore, describing scheduling algorithms with the same programming language used for MPS implementation reduces the software development complexity (such as adaption/interface between languages) usually caused by programming language discrepancy. Addition or modification of scheduling algorithms to a running MPS is also possible since CodeDom-based approach makes the algorithms deployed without shutting down the MPS software.

In the near future, MPS needs to manage a large heterogeneous satellite constellation. That situation requires high complexity in mission planning and frequent update of scheduling algorithms. We expect that our approach introduced in this paper shall be widely used for massive scale mission design for future MPS.

References

[1] S. Lee, W.C. Jung, J. Kim, Task Scheduling Algorithm for the Communication, Ocean, and Meteorological Satellite, ETRI Journal. 30 (2008) 1–12.
[2] S. Lee, B. Lee, J. Kim, A Strategy to Determine Whether to Use GPU for a Satellite Mission Scheduling Algorithm, TRANSACTIONS OF THE JAPAN SOCIETY FOR AERONAUTICAL AND SPACE SCIENCES. 55 (2012) 166–174.
[3] L. Barbulescu, J.-P. Watson, L.D. Whitley, A.E. Howe, Scheduling Space-Ground Communications for the Air Force Satellite Control Network, Journal of Scheduling. 7 (2004) 7–34.
[4] Y. Caseau, F. Laburthe, "Cumulative Scheduling with Task Intervals," Proc. Joint International Conference on Logic Programming, 1996, 363-377.
[5] L.A. Kramer, S.F. Smith, Task Swapping for Schedule Improvement: A Broader Analysis, Proc. International Conference on Automated Planning and Scheduling, 2004, 235-243.
[6] A. Bar-Noy, S. Guha, J.S. Naor, B. Schieber, Approximating the Throughput of Multiple Machines in Real-Time Scheduling, SIAM Journal on Computing, 31 (2002), 331-352.