Journal of Innovative Technology Convergence
Vol. 3, No. 1 June 2021, pp. 9-22

CORBA-Based Class Scheduling Framework Using Decorator and Facade Design Patterns

John Jowil D. Orquia [1], Jason P. Sermeno [2]

Abstract: The class scheduling problem has been regarded as one of the most difficult yet important challenges for developers developing timetabling applications. Different algorithms or schemes were devised to address such problems. Yet, considering the growing complexity of the input factors, it contributes to difficulties in maintaining the said system. In order to aid in the development and maintenance of such a system, this paper presents an object-oriented approach to designing a class scheduling paradigm using decorator and facade design patterns with CORBA architecture. This technique was based on the concepts of several existing object-oriented components of Windows GUI platforms. The paradigm designs are expressed using Unified Modeling Language (UML) and illustrated in a Java programming environment. The result of the study is a simplified method for designing an efficient and reusable component for the class scheduling model.

Keywords: Design Patterns, Class Scheduling Problems, Timetabling, Decorator and Façade

1. Introduction

The design of a class schedule is a tedious and troublesome task when resources are limited [1]. Yet this is an important process that must be done before enrollment starts at a university [2]. It is considered a complex combinatorial optimization problem or a non-polynomial complete type problem, which involves determining the most desirable schedule of lecture times and classes taught for each faculty member under constraints [3]. Such constraints as classroom requirements, teacher conflicts, course conflicts, time restrictions, time preferences, and classroom preferences are some of the input factors that contribute to the complexity of solving the problem [4][5]. Since many factors and situations have to be considered, this work would be time-consuming and often leads to inadequate solutions. Additionally, redesigning an existing system would be costly and difficult to maintain, especially when the applications are distributed on a locally hosted network [6].

[1] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: johnjowil.orquia@antiquespride.edu.ph
[2] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: jason.sermeno@antiquespride.edu.ph

Received [December 12, 2020]; Revised [February 15, 2021]; Accepted [March 25, 2021]

Published by InnoCon Publishing
ISSN 2704-4440

A possible inexpensive solution to ease the method of redesigning the system is to consider its design patterns [7]. Nowadays, design patterns have been used in various applications and are becoming prevalent in providing better designs. It also simplifies the data structure and corresponding operations, which lead to numerous benefits [8]. With several different patterns to deal with, it can be used to build effective frameworks from security mechanisms [9][10], detection and recognition processes [11][12] or complicated content retrieval procedures [13]. Furthermore, the design patterns jive with the choice of known algorithms such as the heuristic approach [14], graph search [15], or genetic algorithms [16]. So, basically, it suits any algorithm that could be applied in constructing various applications.

To resolve such client-server issues in a distributed system, CORBA could be integrated. Developed by OMG, its significant purpose is now being realized by most industries, such as the US Army and CNN, and it is frequently used for their servers that handle large numbers of clients [17]. CORBA is middleware and it jives along with an object-oriented programming approach [18].

There have been numerous studies done to compensate for solving class scheduling problems using devised schemes. Some have applied classroom scheduling rather than faculty scheduling [19][20], some are tailored to student schedules rather than those of the department [21]. Others try to address it using time patterns such as timeslots, time grains, or chained-through time patterns, depending on the use cases [22]. But most of the existing approaches, however, were focused on the design of an algorithm, such as the use of genetic algorithms [23][24], multi-agent-based [25][26], ant colony [27][28], neurocomputing [29], *etc*. However, these solutions are so complex that they don't provide us with a basic and clear view of structural design at a level of abstraction, thus making the system difficult to maintain.

Up to this date, there is no literature yet that synthesizes the design of a class scheduling model utilizing a CORBA architecture and design patterns. This paper is now geared towards the development of a CORBA-based class scheduling model using decorator and facade design patterns.

The significance of conducting this research is to provide designers with an efficient, reliable, robust, flexible, and ultimately reusable method of designing class schedule-related applications.

2. Related Works

2.1 Scheduling and Timetabling

There has been widely conducted classroom-based research over the past decades. Of this state-of-the-art literature, many have been proposed to solve traditional classroom scheduling problems. One particular study presents a flat-fading-based method for classroom instruction [30]. However, the computational complexity could dramatically drag down the performance as the authors simulate the scheme using MATLAB. Meanwhile, a considerable number of algorithms are based on local search techniques, such as hill climbing, simulated annealing, and tabu search [31][32]. But these algorithms cannot guarantee an optimal solution. In other words, it cannot guarantee that if a solution is found, it has the best possible optimization cost.

Another study [33] proposes a linear programming approach to solve the scheduling problem. Using generic algorithms and Boolean Satisfiability techniques [34], the authors were able to develop and solve the ILP model using the three advanced ILP solvers.

Simultaneously, graph theory-based approaches are still focused on the scheduling problem. For example, [35] proposed a heuristic approach to promote uniform distribution of courses over the colors and balance the course load for each time slot over the available class rooms. Similarly, in [36] and [37], a meta-heuristic algorithm based on the principles of particle swarm optimization (or PSO) is proposed for the course scheduling problem.

However, most of the related works mentioned have several shortcomings. First, most of the scheduling approaches are performed with high-complex algorithms, which are not even a practical solution for NP-Complete scheduling. Second, the approaches to the scheduling schemes lack modularity, which may cause excessive and inefficient workloads for different objectives of scheduling methods. And finally, when some factors are to be modified, the algorithm must be reconsidered from the beginning, which brings significant redesign and simulation overheads.

2.2 The Decorator and Façade Design Patterns

A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. It simply identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities [38].

Although there have been no known related studies pertaining to the design patterns for the class scheduling system, this would be the first of its kind to be explored and should be taken advantage of. Fortunately, there have been related studies that focus on the design patterns in other aspects of a certain application. For example, [39] describes the application of decorator and observer design patterns in testbench environments. The author was able to show how the decorator pattern was able to self-maintain re-configuration in a chip-level testbench when registers are accessed at runtime. Another study [40] involves an application of the decorator pattern for non-intrusive profiling of object-oriented applications. The authors of this study were able to provide a case study to compare the cost trade-offs of validating invariants at different points in a program.

Recently, one study focused on the use of the decorator design pattern to work on the forensic model for logical traffic isolation services [41]. They use the pattern to randomly wrap any traffic generated with either normal or suspicious behavior. While [42] used the decorator to partially enhance service designs in an ERP system by handling the service implementation change.

Similar to the decorator, the facade pattern is another structural design pattern that helps conceal communication and dependencies between subsystems. Currently, there have been many studies that utilize the said pattern for image processing and analysis. One such study [43] proposed a sequential optimization technique for segmenting a rectified image of a facade into semantic categories. Similarly, [44] used the facade pattern together with the block matrix model to repeatedly extract images. Others worked on the recognition aspect of the image by using the facade pattern to model objects and object structures in a unified Marked Point Process framework [45]. In terms of software engineering, one study was devoted to the design and implementation of a software and hardware infrastructure to enable multiple interactive applications to run on a facade using different types of devices for interaction [46]. Meanwhile, [47] proposed a similar approach but implemented a flexible distributed system with a launcher, website, and different configurable output devices to specify and run applications on a media facade.

Merely most studies concerning the design patterns are very effective in optimizing the processes and help designers or developers achieve better results in their specific findings. The impact of utilizing such design patterns in software development is huge, as it opens up best practices for designing complex and yet reusable components of a certain project.

2.3 CORBA

A Common Object Request Broker Architecture, or simply CORBA, is a standard that enables interaction between two different objects written in different languages and running on different platforms. Its specification and implementation are most widely used in the IBM and Netscape Open Network Environment architectures.

The CORBA architecture has been used recently in several studies. One such study [48] presents the design and implementation of a Pluggable Fault Tolerant CORBA infrastructure that provides fault tolerance for CORBA applications by utilizing a pluggable protocol framework. Another study [49] proposed a technique called CORBA-as-needed to improve the performance of distributed CORBA applications. It used three design techniques to incorporate CORBA-as-needed in the current CORBA architecture. Similarly, [50] came up with a convenient way of adding distributed computing capability to the OSGi framework and interoperation support between OSGi and legacy CORBA applications. While [51] proposed a resource management framework using CORBA. The CORBA architecture was also used in the development of an Intelligent Humanoid Robot (IHR) [52].

3. The Proposed Class Scheduling Model

3.1 The Conceptual and Operational Framework

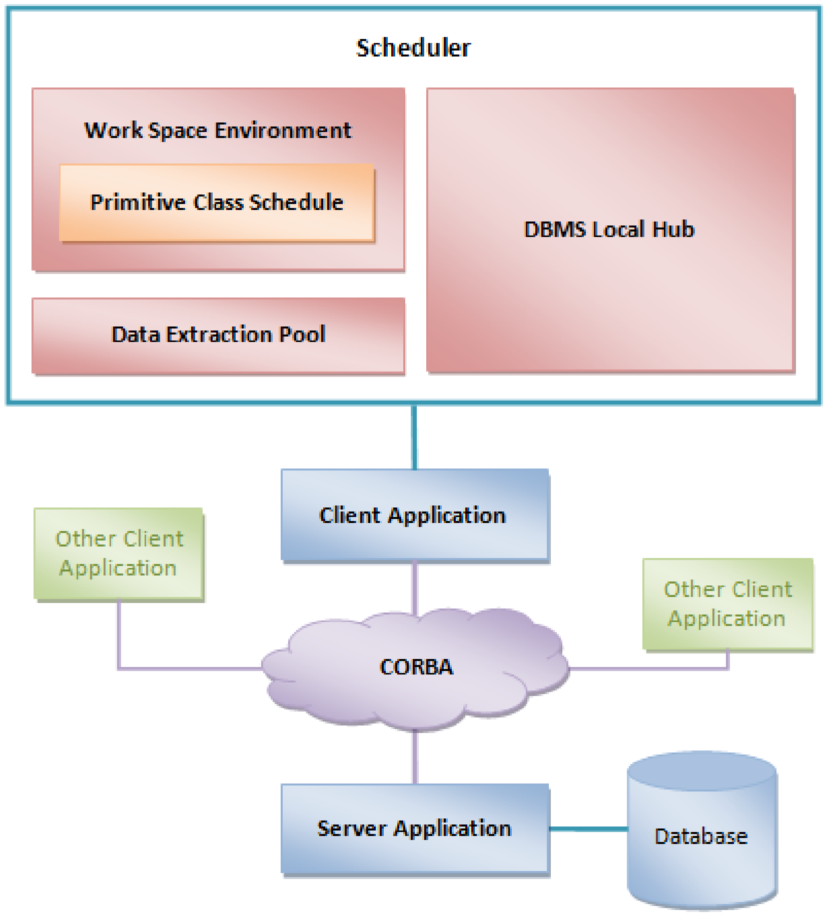


Figure 1. A Conceptual Diagram of the Proposed Class Scheduling Paradigm

The study is focused on the object-oriented design of the class scheduling model. Figure 1 illustrates a conceptual model of the proposed scheduling paradigm. The objective of this model is to generate a conflict-free class schedule while being implemented on a CORBA architecture. The model is composed of the following components: (a) a DBMS local hub; (b) a data extraction pool; (c) a primitive class schedule; (d) a work space environment; (e) a server application; (f) a client application; and (g) an underlying CORBA framework. The database local hub serves as the controller for retrieving and updating data from a database on a local host server. The data extraction pool serves as an interface for extracting related basic information pertaining to faculty, sections, rooms, and curricular courses. The primitive class schedule is a raw collection of schedules that outlines the individual workload of a certain faculty, section, or room. The work space environment serves as a work area for constructing a schedule. It simply uses three layers of the primitive class schedules to extract available slots for a certain course of a section to be handled by the faculty in a specific room. The server application performs the necessary operations in initializing the CORBA objects so that clients could use it to retrieve or update necessary data in the system through ORBs. Basically, it does not contain instructions pertaining to local execution, such as SQL instructions or retrieving certain information in the database. The client application, on the other hand, runs the necessary operations dealing with the scheduling algorithms in addition to its initialization and access to the ORBs.

In order to achieve this notion, the proponent applied two design patterns to address the functionalities and relationships between components. This provides a formal specification of the components and how it could be used as a program transformation without altering the external, observable behavior of the system. Such a transformation is referred to as a correctness preserving transformation (CPT) [53].

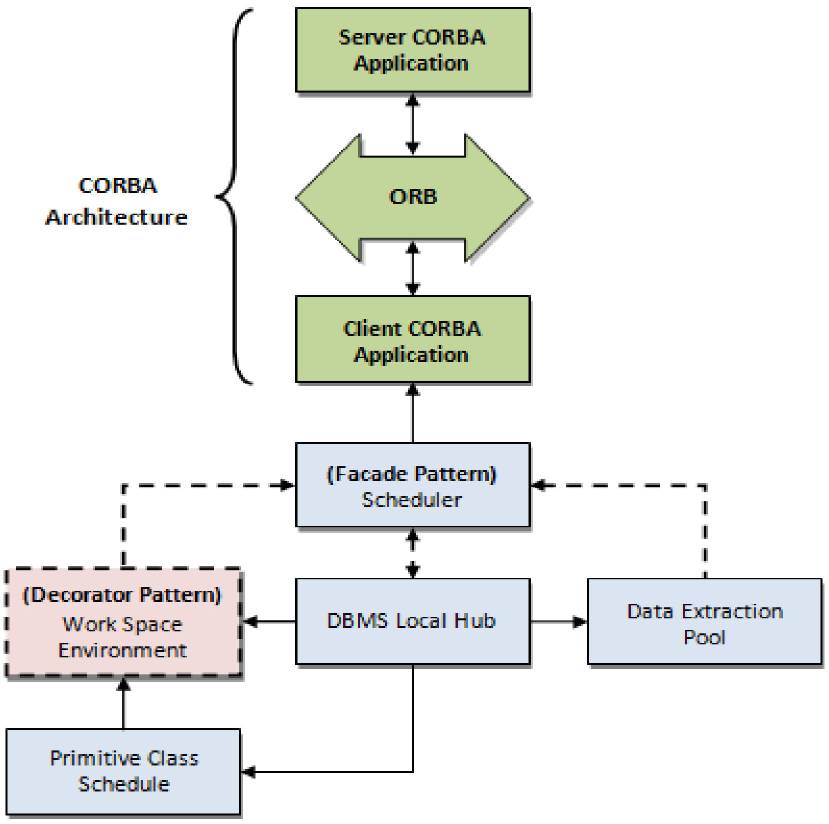


Figure 2. The Operational Framework of the Proposed Model

Figure 2 illustrates a more detailed framework where the design patterns and the CORBA framework are revealed. The work space environment is designed using the decorator pattern to allow responsibilities or functionalities to be added. All accessible functions of the components are interfaced in a single component (scheduler) to simplify all necessary operations to be done in a schedule. An implementation of the CORBA framework lies between the communication between the server and client applications.

3.2 Using the Decorator Design Pattern

Design patterns are very popular among software developers, and in fact, they are a well-described solution to a common software problem [54]. A decorator is one example that falls under structural design patterns and is used to modify the functionality of an object at runtime.

The basic idea of employing this pattern in the proposed model is to generate an elegant work space environment, as shown in Figure 3. As the raw schedules of selected faculty, sections, and rooms are fetched, the system should be able to virtually decorate a layer-controlled schedule.

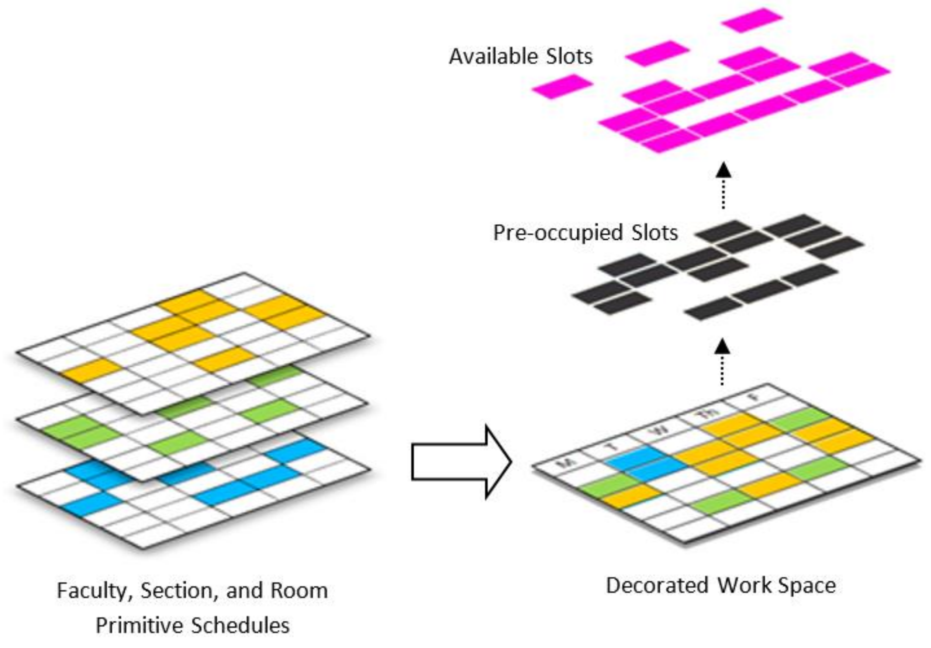


Figure 3. Basic setup for the Work Space Environment

The ability to add responsibilities to a working model will definitely ease the system’s ability to capture sets of pre-occupied slots, compute, and return available slots in a table. Regardless of the algorithm to be used in the model, the system would not worry about the occurrence of conflicting schedules as it has already mapped the available slots for creating new or modifying schedules.

As shown in Figure 4, the WorkSpace class is the component interface that defines the common methods such as noOfVacantSlots() and getVacantSlots() to be implemented. The DefaultWorkSpace is the basic implementation of the WorkSpace interface. The model uses a WorkSpaceDecorator to implement the component interface, and it has a HAS-A relationship with its component interface.

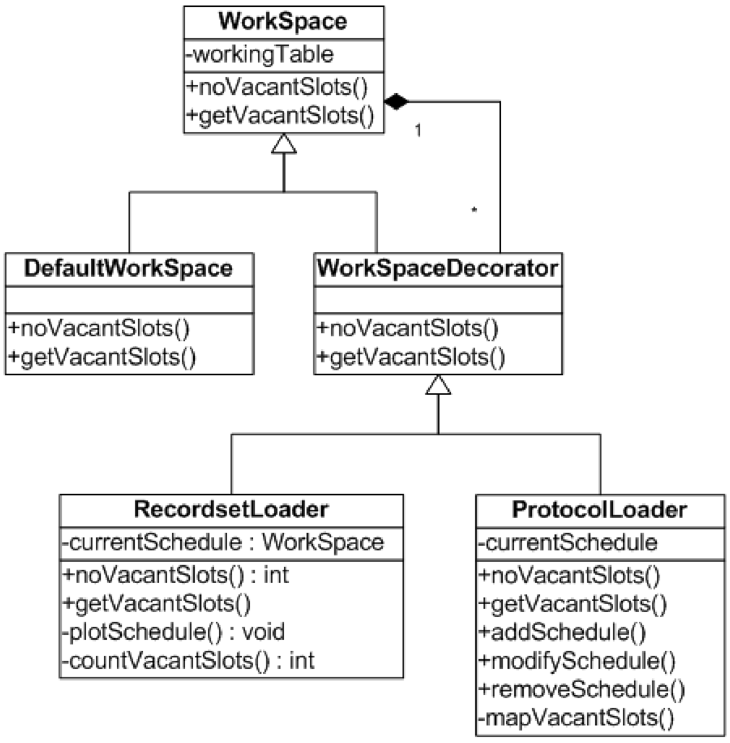
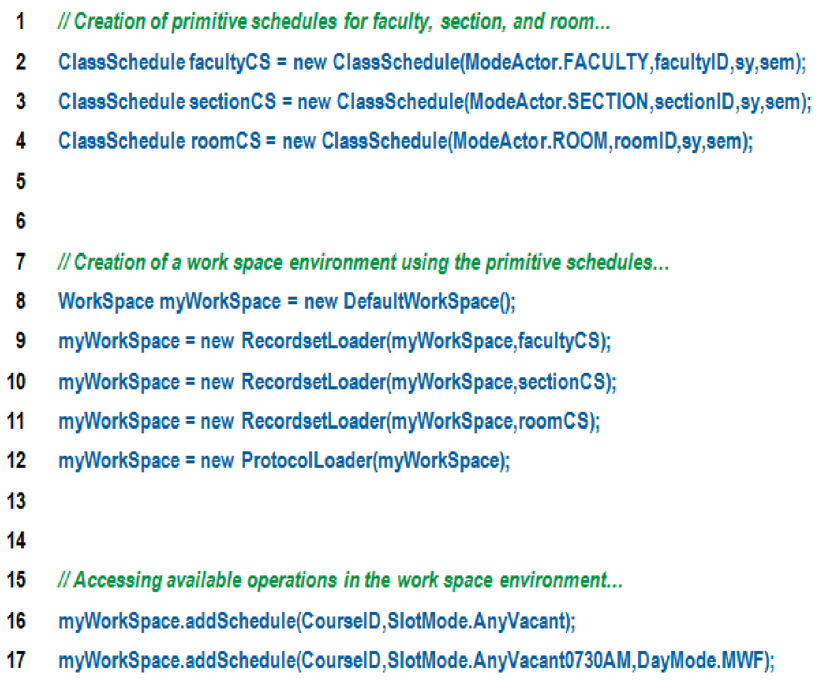


Figure 4. A Class Diagram of a Work Space Environment using the Decorator Design Pattern

To utilize the decorator, a concrete decorator must be defined – in this case, the RecordsetLoader and ProtocolLoader classes. These concrete decorators extend the base decorator's functionality and modify the component's behavior accordingly. The RecordsetLoader handles a collection of primitive schedules, while the ProtocolLoader provides additional functions for the work space environment, such as adding or modifying a schedule or mapping vacant slots in a time table.



```
// Creation of primitive schedules for faculty, section, and room...
ClassSchedule facultyCS = new ClassSchedule(ModeActor.FACULTY,facultyID,sy,sem);
ClassSchedule sectionCS = new ClassSchedule(ModeActor.SECTION,sectionID,sy,sem);
ClassSchedule roomCS = new ClassSchedule(ModeActor.ROOM,roomID,sy,sem);


// Creation of a work space environment using the primitive schedules...
WorkSpace myWorkSpace = new DefaultWorkSpace();
myWorkSpace = new RecordsetLoader(myWorkSpace,facultyCS);
myWorkSpace = new RecordsetLoader(myWorkSpace,sectionCS);
myWorkSpace = new RecordsetLoader(myWorkSpace,roomCS);
myWorkSpace = new ProtocolLoader(myWorkSpace);


// Accessing available operations in the work space environment...
myWorkSpace.addSchedule(CourseID,SlotMode.AnyVacant);
myWorkSpace.addSchedule(CourseID,SlotMode.AnyVacant0730AM,DayMode.MWF);
```

Figure 5. Sample Code Fragment for Decorator Class

To clearly illustrate how the decorated work space is utilized in the code, Figure 5 demonstrates a code fragment written in the Java programming language.

Before a work space is established, the program needs three primitive class schedule objects to be instantiated. These objects will be used as one of the parameters to instantiate the myWorkSpace object. Notice that the myWorkSpace object is instantiated repeatedly. At line 8, a default workspace is created. Lines 9 to 11 load the corresponding primitive schedules for a selected faculty, section, and room. At line 12, the work space environment is added, along with additional operations to work with the schedule, such as in lines 16 and 17. This is possible because the decorator allows nesting of decorators recursively. Thereby, allowing an unlimited number of responsibilities to a decorated object.

As a result of the design, the system could be easily maintained, extend, and add operations to individual objects dynamically and transparently without affecting other objects in the system. It also aids the system in providing runtime modification abilities and flexibility.

3.3 Using the Façade Design Pattern

Structuring a system into subsystems helps reduce complexity and minimize communication and dependencies between subsystems [55]. One way to achieve this goal is to introduce a facade object that provides a single, simplified interface to the more general facilities of a system.

Clearly, the facade pattern provides a unified interface to a set of interfaces in a subsystem. It generally defines a higher-level interface that makes the subsystem easier to use.

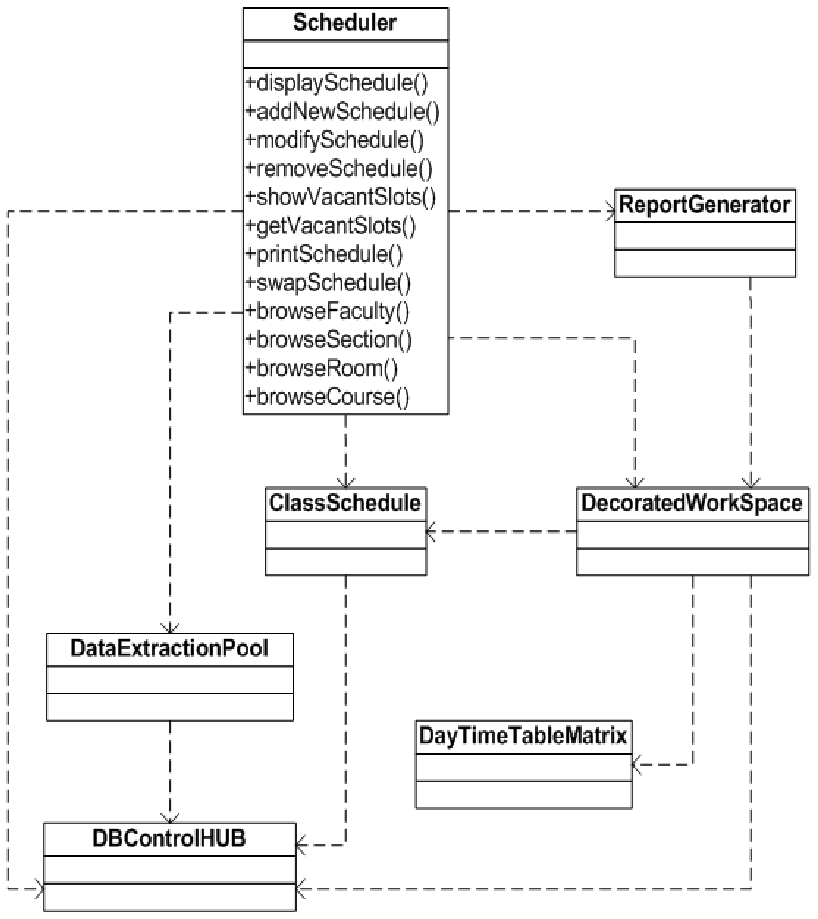
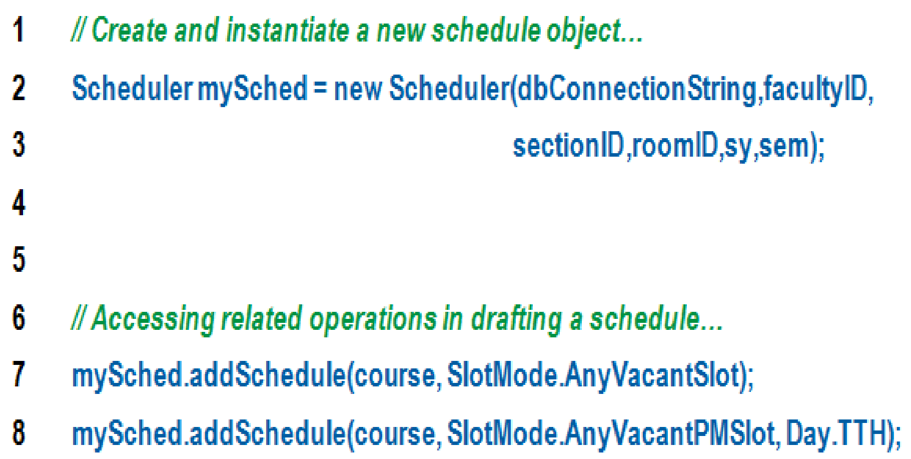


Figure 6. A Class Diagram of the Scheduler using the Façade Design Pattern

The proposed scheduling model is patterned in this manner. All relevant operations pertaining to drafting a schedule are defined in the Scheduler facade class, thus limiting redundant access to various operations such as connecting to a database, extracting numerous primitive schedules, instantiating a work space, *etc*.

Figure 6 illustrates the class diagram of the scheduler utilizing the said pattern. The design was based on the conceptual and architectural model that houses the functionalities of extracting data, loading primitive class schedules, controlling the database, and using the work space environment. It also defines additional operations such as the generation of reports and handling of the timetable matrix.

Figure 7 demonstrates a sample code implementation of how the facade interface is used in an easier and cleaner way.



```
// Create and instantiate a new schedule object...
Scheduler mySched = new Scheduler(dbConnectionString,facultyID,
                                   sectionID,roomID,sy,sem);


// Accessing related operations in drafting a schedule...
mySched.addSchedule(course, SlotMode.AnyVacantSlot);
mySched.addSchedule(course, SlotMode.AnyVacantPMSlot, Day.TTH);
```

Figure 7. Sample Code Fragment for the Facade Class

The application of the facade pattern in the proposed model is a profound solution, as it can be applied at any point of development and provides a better interface to client systems.

3.4 The Detailed Design of the Proposed Model

A detailed view of the design of the class scheduling model can be found in Figure 8. The overall structure of the class scheduling paradigm is a combination of design patterns and CORBA architecture.

The class diagram in Figure 8 shows a complete specification of the class scheduling paradigm. Using the scheduler will allow the system to utilize a set of simplified operations on the schedules as well as accessing and updating primitive data from the database. Since the design is an object-oriented approach, it could be easily used as a component for any object-oriented application.

The speed and accuracy of the system implementing this designed model depend clearly on the hardware requirements that support it. With the ability to add features to this model, it is important that the classes, especially the common component classes, be kept simple and lightweight; otherwise, it may increase the probability that the concrete subclasses will pay for features that they do not need.

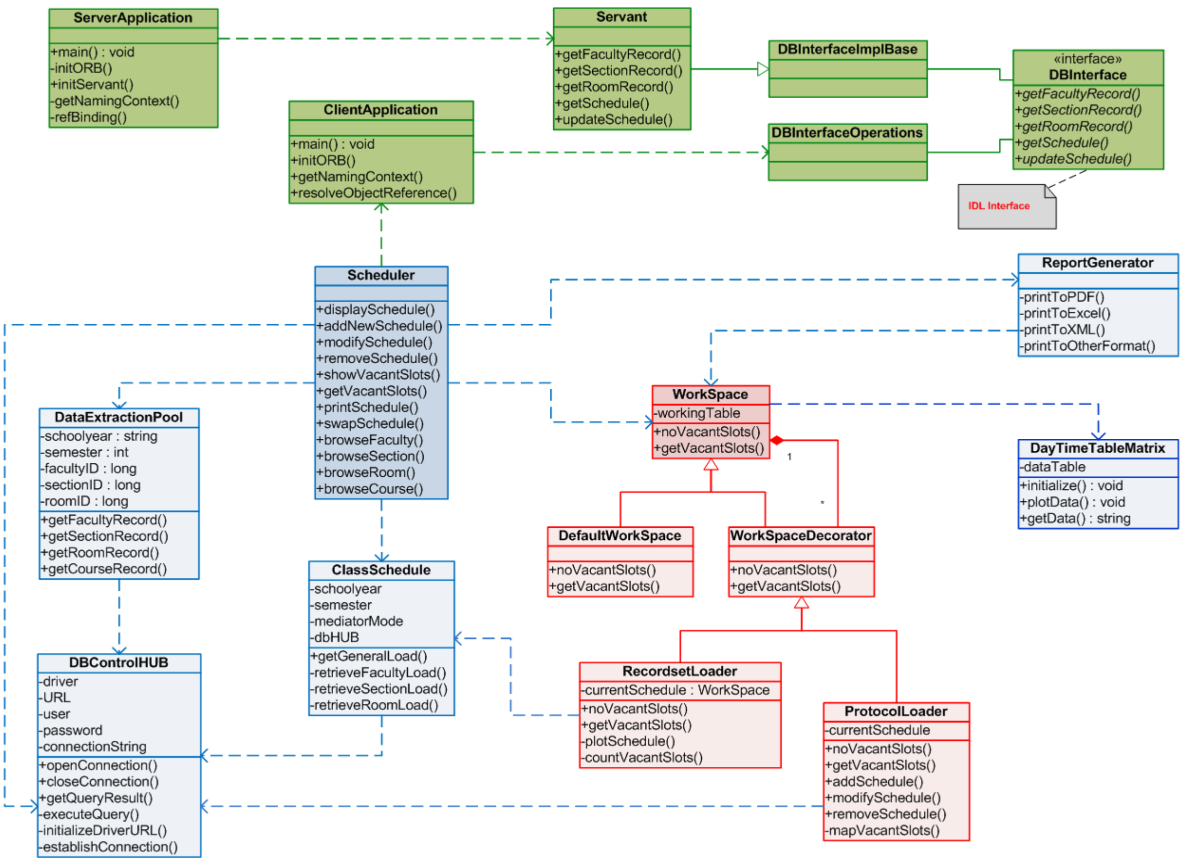


Figure 8. A Class Diagram of the Proposed CORBA-Based Class Scheduling Model

4. Conclusion and Future Extensions

In this paper, a CORBA-based class scheduling model using decorator and facade design patterns has been designed. The model will address the key constraint of handling conflicts in a class schedule. It will make it easier for developers to create or maintain an existing class schedule application.

Although the model could perform almost all necessary operations for working with the typical schedule, it is still open to many possible improvements for a simpler, more robust, and more flexible design. For example, one could provide different decorator classes for a specific component or restructure the scheme for handling specific operations. Another possible area in this study could involve the integration of encryption schemes to provide a level of security. Another area for refinement could be the use of different or hybrid design patterns, such as flyweight or strategy patterns, that could handle large quantities of objects or heavyweight operations.

References

[1] N. Pothitos, P. Stamatopoulos, and K. Zervoudakis, "*Course Scheduling in an Adjustable Constraint Propagation Schema*", in 2012 IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI), Athens, Greece, November 7-9, 2012, pp. 312-320, doi: 10.1109/ICTAI.2012.53.

[2] S. Chen and D. Zhang, "*Research on solution to course scheduling problems based on a hierarchical planning strategy*", in 2011 International Conference of Soft Computing and Pattern Recognition (SoCPaR), Dalian, China, October 14-16, 2011, pp. 263-267, doi: 10.1109/SoCPaR.2011.6089118.

[3] J. D. Ullman, "*NP-Complete Scheduling Problems*", Journal of Computer and System Sciences, vol. 10, no. 3, 1975, pp. 384-393, doi: 10.1016/S0022-0000(75)80008-0.

[4] C. Wang, X. Li, A. Wang, and X. Zhou, "*A Classroom Scheduling Service for Smart Classes*", IEEE Transactions on Services Computing, vol. 10, no. 2, June 2017, pp. 155-164, doi: 10.1109/TSC.2015.2444849.

[5] C. J. Hwang and C. Kao, "*A Knowledge-Based Approach to Class Scheduling Problems with a Developed System ACS*", in Eighth Annual International Phoenix Conference on Computers and Communications. 1989 Conference Proceedings, Scottsdale, AZ, USA, March 22-24, 1989, pp. 555-561, doi: 10.1109/PCCC.1989.37445.

[6] M. Fowler, K. Beck, J. Brandt, W. Opdyke, and D. Roberts, "*Debugging a Code*," in *Refactoring: Improving the Design of Existing Codes*, New York, USA: McGraw-Hill, 2009, ch. 1, pp. 12-13.

[7] A. Hunt and D. Thomas, "*System Walkthrough*," in *The Pragmatic Programmer: From Journeyman to Master*, California, USA: Addison-Wesley, 2012, ch. 1, pp. 25.

[8] V. Holmstedt and S. A. Mengiste, "*The importance of program Design Patterns training*," in 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), Klagenfurt, Austria, February 20-24, 2017, pp. 433-439, doi: 10.1109/SANER.2017.7884676.

[9] M. A. Sheta, M. Zaki, K. A. E. S. El Hada, and H. Aboelseoud, "*Anti-spyware Security Design Patterns*," in 2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC), Harbin, China, July 21-23, 2016, pp. 45-53, doi: 10.1109/IMCCC.2016.202.

[10] M. S. Haq, Z. Anwar, A. Ahsan, and H. Afzal, "*Design pattern for secure object oriented information systems development*," in 2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST), January 10-14, 2017, pp. 178-186, doi: 10.1109/IBCAST.2017.7868093.

[11] G. Luitel, M. Stephan, and D. Inclezan, "*Model Level Design Pattern Instance Detection Using Answer Set Programming*," in 2016 IEEE/ACM 8th International Workshop on Modeling in Software Engineering (MiSE), Austin, TX, January 9, 2017, pp. 1-8, doi: 10.1145/2896982.2896991.

[12] A. K. Dwivedi, A. Tirkey, R. B. Ray, and S. K. Rath, "*Software design pattern recognition using machine learning techniques*," in 2016 IEEE Region 10 Conference (TENCON), Singapore, Singapore, November 22-25, 2016, pp. 9-17, doi: 10.1109/TENCON.2016.7847994.

[13] S. Abid, Z. Qamar, N. Khan, M. Shayan, and H. A. Basit, "*Retrieving Design Pattern Usage Examples using Domain Matching*," in 2016 23rd Asia-Pacific Software Engineering Conference (APSEC), Hamilton, New Zealand, December 6-9, 2016, pp. 78-90, doi: 10.1109/APSEC.2016.016.

[14] G. Guizzo and S. R. Vergilio, "*Metaheuristic Design Pattern: Visitor for Genetic Operators,*" in 2016 5th Brazilian Conference on Intelligent Systems (BRACIS), Recife, Brazil, October 9-12, 2016, pp. 57-64, doi: 10.1109/BRACIS.2016.038.

[15] D. M. Le, D. H. Dang, and V. H. Nguyen, "*Domain-driven design patterns: A metadata-based approach*," in 2016 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), Hanoi, Vietnam, November 7-9, 2016, pp. 241-250, doi: 10.1109/RIVF.2016.7800302.

[16] M. E. Fayad and C. A. Flood, "*Unified Software Engineering Reuse (USER) using stable analysis, design and architectural patterns*," in Future Technologies Conference (FTC), San Francisco, CA, USA, January 19, 2017, pp. 300-312, doi: 10.1109/FTC.2016.7821682.

[17] O. H. Mahmoud, "*Distributed Programming with Java*", Manning Publications, September 1999, ISBN: 9781884777653.

[18] I. Calvo, M. Marcos, and D. Orive, "*A framework based on CORBA and OO technologies for remote access to industrial plants*," in 2003 European Control Conference (ECC), Cambridge, UK, UK, September 1-4, 2003, pp. 447-455, doi: 10.23919/ECC.2003.7085320.

[19] E. Humo and M. El-Zayat, "*Visualization of Heuristic Approach to Classroom-Period Schedule Optimization*," in 2010 International Conference on e-Education, e-Business, e-Management and e-Learning, Sanya, China, January 22-24, 2010, pp. 358-360, doi: 10.1109/IC4E.2010.82.

[20] E. Humo and Z. Vejzovic, "*A Mathematical Model for Classroom-Period Schedule Defragmentation*," in EUROCON 2007 - The International Conference on "Computer as a Tool", Warsaw, Poland, September 9-12, 2007, pp. 2030-2033, doi: 10.1109/EURCON.2007.4400293.

[21] B. C. Lih, S. S. Nah, and N. A. Bolhassan, "*A Study on Heuristic Timetabling Method for Faculty Course Timetable Problem*", in 2015 9th International Conference on IT in Asia (CITA), Kota Samarahan, Malaysia, August 4-5, 2015, pp. 1-3, doi: 10.1109/CITA.2015.7349832.

[22] G. De Smet, "*Time scheduling design patterns*", OptaPlanner, www.optaplanner.org/blog/2015/12/01/TimeSchedulingDesignPatterns.html (Accessed November 20, 2020).

[23] S. Parera, H. T. Sukmana, and L. K. Wardhani, "*Application of Genetic for Class Scheduling (Case Study: Faculty of Science and Technology UIN Jakarta*", in 2016 4th International Conference on Cyber and IT Service Management, Bandung, Indonesia, April 26-27, 2016, pp. 1-5, doi: 10.1109/CITSM.2016.7577525.

[24] A. Ansari and A. A. Bakar, "*A Comparative Study of Three Artificial Intelligence Techniques: Genetic Algorithm, Neural Network and Fuzzy Logic on Scheduling Problem*," in 2014 4th International Conference on Aritificial Intelligence with Applications in Engineering and Technology, Kota Kinabalu, Malaysia, December 3-5, 2014, pp. 31-36, doi: 10.1109/ICAIET.2014.15.

[25] D. Weili and S. M. Qui, "*Multi Agents Based for Humanistic Intelligent Class Scheduling System*", in 2010 Third International Symposium on Information Science and Engineering, Shanghai, China, December 24-26, 2010, pp. 476-480, doi: 10.1109/ISISE.2010.121.

[26] I. Pereira, A. Madureira, and P. de Moura Oliveira, "*Meta-heuristics Self-Parameterization in a Multi-Agent Scheduling System Using Case-Based Reasoning*," in *Computational Intelligence and Decision Making*, A. Madureira, C. Reis, V. Marques (Eds), Intelligent Systems, Control and Automation: Science and Engineering, Springer, Dordrecht, vol. 61, 2013, ch. 1, pp. 99-109, doi: 10.1007/978-94-007-4722-7_10.

[27] Al-Mahmud and M. A. H. Akhand, "*ACO with GA Operators for Solving University Class Scheduling Problem with Flexible Preference*," in 2014 International Conference on Informatics, Electronics & Vision (ICIEV), Dhaka, Bangladesh, May 23-24, 2014, pp. 1-6, doi: 10.1109/ICIEV.2014.6850742.

[28] X. Zhang, L. Dong, and Q. Bai, "*A Decision Support System with Ct_ACO Algorithm for the Hot Rolling Scheduling*," in 2010 International Conference on Intelligent Computation Technology and Automation, vol. 1, Changsa, China, May 11-12, 2010, pp. 65-68, doi: 10.1109/ICICTA.2010.701.

[29] S. Tsutsui and Y. Fujimoto, "*Class Scheduling by Neurocomputing: A Comparison with the Expert System Approach*," in 1990 IEEE International Conference on Systems, Man, and Cybernetics Conference Proceedings, Los Angeles, CA, USA, November 4-7, 1990, pp. 227-232, doi: 10.1109/ICSMC.1990.142098.

[30] G. S. Mason, T. R. Shuman, and K. E. Cook, "*Simulation of Flat Fading using MATLAB for Classroom Instruction*," IEEE Trans. Edu., vol. 45, no. 1, February 2002, pp. 19-25, doi: 10.1109/13.983217.

[31] E. Mooney, R. Dargen, and W. Parameter, "*Large-Scale Classroom Scheduling*," ITE Transactions, vol. 28, no. 5, 1996, pp. 369-378, doi: 10.1080/07408179608966284.

[32] V. Tam and D. Ting, "*Combining the Min-Conflicts and Look-Forward Heuristics to Effectively Solve a Set of Hard University Timetabling Problems*," in 15th IEEE International Conference Tools Artifacts Intelligence, Sacramento, CA, USA, November 5, 2003, pp. 492-496, doi: 10.1109/TAI.2003.1250230.

[34] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "*Generic ILP Versus Specialized 0-1 ILP: An Update*," in International Conference Computer Aided Designs, San Jose, CA, USA, November 10-14, 2002, pp. 450-457, doi: 10.1109/ICCAD.2002.1167571.

[35] C. M. Hsu and H. M. Chao, "*A Two-Stage Heuristic Based Class-Course-Faculty Assigning Model for Increasing Department-Education Performance*," in 2009 International Conference on New Trends in

Information and Service Science, Beijing, China, June 30-July 2, 2009, pp. 256-263, doi: 10.1109/NISS.2009.130.

[36] K. Hamdi, "*A Mathematical Model and a GRASP Metaheuristic for a Faculty-Course Assignment Problem for a University in Saudi Arabia*," in 2014 IEEE International Conference on Industrial Engineering and Engineering Management, Bandar Sunway, Selangor, Malaysia, December 9-12, 2014, pp. 672-676, doi: 10.1109/IEEM.2014.7058723.

[37] C. Oswald and C. A. D. Durai, "*Novel Hybrid PSO Algorithms with Search Optimization Strategies for a University Course Timetabling Problem*," in 2013 Fifth International Conference on Advanced Computing (ICoAC), Chennai, India, December 18-20, 2013, pp. 77-85, doi: 10.1109/ICoAC.2013.6921931.

[38] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "*Design Patterns. Elements of Reusable Object-Oriented Software*," Boston, USA: Addison-Wesley, 1995.

[39] F. Karimi, "*Applications of Decorator and Observer Design Patterns in Functional Verification*," 2008 IEEE International High Level Design Validation and Test Workshop, Incline Village, NV, USA, November 19-21, 2008, pp. 18-22, doi: 10.1109/HLDVT.2008.4695867.

[40] E. B. Duffy, J. P. Gibson, and B. Malloy, "*Applying the Decorator Pattern for Profiling Object-Oriented Software*," 11th IEEE International Workshop on Program Comprehension, Portland, OR, USA, May 10-11, 2003, pp. 84-93, doi: 10.1109/WPC.2003.1199192.

[41] I. Dlamini, M. Olivier, and S. Sibiya, "*Pattern-Based Approach for Logical Traffic Isolation Forensic Modelling*," 20th International Workshop on Database and Expert Systems Application, Linz, Austria, August 31-September 4, 2009, pp. 145-149, doi: 10.1109/DEXA.2009.8.

[42] Y. Z. Rabie and E. E. Hassanein, "*Enhancing Service Design in ERP Systems Using Patterns*," in The 7th International Conference on Informatics and Systems (INFOS), Cairo, Egypt, March 28-30, 2010, pp. 1-5, INSPEC Accession Number: 11292720.

[43] A. Cohen, A. G. Schwing, and M. Pollefeys, "*Efficient Structured Parsing of Facades Using Dynamic Programming*," in IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, June 23-28, 2014, pp. 3206-3213, doi: 10.1109/CVPR.2014.410.

[44] H. Xiao, G. Meng, L. Wang, S. Xiang, and C. Pan, "*Facade Repetition Extraction Using Block Matrix Based Model*," in IEEE International Conference on Image Processing (ICIP), Paris, France, October 27-30, 2014, pp. 1673-1677, doi: 10.1109/ICIP.2014.7025335.

[45] J. Wang, C. Liu, T. Shen, and L. Quan, "*Structure-Driven Facade Parsing with Irregular Patterns*," in 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Kuala Lumpur, Malaysia, November 3-6, 2015, pp. 41-45, doi: 10.1109/ACPR.2015.7486462.

[46] J. Letellier, J. Reinhardt, M. Thiele-Maas, and J. Sieck, "*A Modular Software Architecture to Interact with Media Facades*," in 2015 International Conference on Emerging Trends in Networks and Computer Communications, Windhoek, Namibia, May 17-20, 2015, pp. 95-98, doi: 10.1109/ETNCC.2015.7184815.

[47] S. Bergemann and J. Sieck, "*Application Infrastructure Management System for Media Facades*," in 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), Berlin, Germany, September 12-14, 2013, pp. 231-234, doi: 10.1109/IDAACS.2013.6662678.

[48] W. Zhao, L. E. Moser, and P. M. Melliar-Smith, "*Design and implementation of a pluggable fault tolerant CORBA infrastructure*," in 2002 International Parallel and Distributed Processing Symposium (IPDPS), Proceedings, Ft. Lauderdale, FL, USA, April 15-19, 2001, pp. 80-89, doi: 10.1109/IPDPS.2002.1015513.

[49] S. Mishra and Nija Shi, "*Improving the performance of distributed CORBA applications*," in 2002 International Parallel and Distributed Processing Symposium (IPDPS), Proceedings, Ft. Lauderdale, FL, USA, April 15-19, 2001, pp. 542-553, doi: 10.1109/IPDPS.2002.1015514.

[50] Y. Wu, J. Jia, and H. Wang, "*Using CORBA as a distributed extension solution for OSGi*," in 2014 IEEE Workshop on Electronics, Computer and Applications, Ottawa, ON, Canada, May 8-9, 2014, pp. 239-249, doi: 10.1109/IWECA.2014.6845587.

[51] M. Ravirala, S. Sivasubramanian, and S. Kothari, "*Resource-aware real-time CORBA in multi-server distributed environment*," in 2002 International Parallel and Distributed Processing Symposium (IPDPS), Proceedings, Ft. Lauderdale, FL, USA, April 15-19, 2001, pp. 297-308, doi: 10.1109/IPDPS.2002.1016481.

[52] D. W. Kim, "*Humanoid robot based on the middleware CORBA*," in 2015 15th International Conference on Control, Automation and Systems (ICCAS), Busan, South Korea, October 13-16, 2015, pp. 333-341, doi: 10.1109/ICCAS.2015.7364838.

[53] J. P. Gibson, T. F. Dowling, and B. A. Malloy, "*The Application of Correctness Preserving Transformations to Software Maintenance*," in International Conference on Software Maintenance (ICSM '2000), San Jose, CA, USA, October 11-14, 2000, pp. 108-119, doi: 10.1109/ICSM.2000.883025.

[54] E. Freeman, E. Robson, K. Sierra, and B. Bates, "*Preparing for a Head Start*," in *Head First Design Patterns: A Brain-Friendly Guide*, Chicago, USA: O'Reilly, 2010, ch. 1, pp. 14-13.

[55] Martin Fowler, *et al.*, "*Discovering Patterns*", in *Patterns of Enterprise Application Architecture*, New York, USA: Addison-Wesley, 2004, ch. 1, pp. 23.