Exploration of Modifiability in Logic Control Systems

Authors: Wang Sujuan, Zhu Hongfeng, Wu Xiaotao, Wang Sujuan

Date: 2025-01-07T00:00:00+00:00

Abstract

The purpose of this paper is to discuss the modifiability of logic control software systems from both theoretical and practical perspectives. From a philosophy of technology viewpoint, it proposes that to achieve system modifiability, the abstract narrative of the world in computer science should be: "The world consists of objects and the relationships between them." This paper demonstrates that the SEA (State Event Analysis) method, based on this abstraction, can achieve modifiability in logic control systems. Through an example of group elevator modeling, it illustrates how the SEA method enables modeling of complex logic control systems by leveraging and preserving system modifiability. The content of this paper contributes to exploring new pathways for modeling complex manufacturing systems.

Full Text

Discussion on the Modifiability of Logic Control Systems

Wang Sujuan[1], Zhu Hongfeng[1], Wu Xiaotao[1]
(1. School of Intelligent Manufacturing and Control Engineering, Shanghai Second Polytechnic University, Shanghai 201209, China)

Abstract

This paper examines the modifiability of logic control software systems from both theoretical and practical perspectives. From a philosophy of technology standpoint, we propose that to achieve system modifiability, computer science' s abstract description of the world should be: "The world is composed of objects and the relationships between objects." We demonstrate that the SEA (State Event Analysis) methodology, based on this abstraction, can realize the modifiability of logic control systems. Using elevator group modeling as an example, we illustrate how the SEA methodology models complex logic control systems by

utilizing and preserving their modifiability. This work contributes to exploring new pathways for modeling complex manufacturing systems.

Keywords: Complex Systems; Modifiability; Philosophy of Technology; Object Technology; Elevator Group Modeling

Logic control systems are becoming increasingly complex. Mobile phones, automobiles, robots, field production control systems, network applications, electronic games, and consumer systems are all evolving into complex intelligent systems—either as pure logic control systems or with logic control as their framework. In particular, current complex manufacturing system modeling faces numerous challenges [1], and system modifiability represents a fundamental issue in addressing these challenges.

This paper defines "system modifiability" as follows: (1) When modifying the framework structure between different stages/versions during system evolution, there should be clear and straightforward specifications and methods [2]; (2) Within the framework and structure of a single version, conventional programming practices should suffice to achieve version objectives [3]; (3) Most importantly, system modifiability must be sustainable and should not deteriorate or disappear as systems grow more complex [4]. Achieving such modifiability enables methods for complex system evolution, yielding traceable, analyzable, optimizable, and easily testable complex systems. For instance, widely-used development approaches like prototyping and agile development require specifications in the sense of systematic modifiability under methodological guidance; otherwise, both the methods and system scale become difficult to advance [5].

2.1 Do We Need Strict Object-Orientation?

This paper collectively refers to both object-oriented and object-based concepts as "object technology." We discuss the modifiability of logic control systems from the perspective of object technology. When emphasizing the modifiability of computer software systems, should modeling languages be strictly object-oriented or object-based? We argue that object-based, rather than strictly object-oriented, languages should be chosen.

We now examine a proposition in the philosophy of technology. Currently, computer technology abstracts the world as: "The world is composed of objects." This corresponds to strictly object-oriented languages. Through practice and reflection, however, we contend that the abstraction should be: "The world is composed of objects and the relationships between objects." This corresponds to object-based languages.

Software development requires more philosophical speculation [6]. We argue that in computer modeling, objects/entities are easily identifiable, but relationships between objects are often vague and difficult to determine, representing a primary challenge in the early modeling phase. Inter-object relationships are constrained by various factors: (1) External constraints—requirements for sys-

tem logic, functionality, and performance proposed by industry, market, or users, such as the elevator industry' s "maximum same-direction, en-route service" requirement, time interval requirements between process steps in chip production, or software system compatibility requirements; (2) Internal logical consistency requirements—how can systems crafted with if-statements or logic encapsulated in individual objects ensure overall system logic without conflicts or omissions? What methods or methodologies can guarantee the quality metrics of system logic? (3) Computational resource constraints—feasible algorithms and data are limited by resources measured in terms of system timing requirements and program/data space; (4) System modifiability requirements—do the relationships between objects in a built system allow for future modifications? (5) In logic control systems, real-time systems and systems with stringent logic requirements are particularly challenging, as modeling success often hinges on revealing and unraveling deep relationships between objects. To build systems that are both concise and highly modifiable, it is necessary to contemplate more general inter-object relationships and the structure of modeling languages.

Target systems must satisfy various constraints on inter-object relationships. These relationships are not subjectively or locally defined but possess objective limitations and mutual constraints, exhibiting chaotic characteristics in the early modeling phase. There is no reason to assume that inter-object relationships can be encapsulated into complex object specifications at the outset of modeling [7]. Modeling languages should preserve the possibility of recognizing and revealing inter-object relationships, allowing modelers to gradually advance their understanding of these relationships. The more language mechanisms prescribe objects and the more a priori information is encapsulated within objects, the more likely they are to fragment, sever, or disrupt the constrained relationships awaiting revelation in the system. Particularly when focusing on system modifiability, object-based rather than strictly object-oriented languages should be chosen in object technology.

2.2 Event-Driven Mechanism and System Modifiability

Object-based programs are event-driven. We now discuss how the event-driven mechanism affects program structure from the perspective of system modifiability. Discussions of modifiability often involve concepts of "loose coupling" and "decoupling." From a decoupling perspective, the event-driven mechanism disperses long-flow programs into numerous event handling processes—this may be called the first type of decoupling. Furthermore, strictly object-oriented languages involve excessive encapsulation and inheritance within objects, whereas object-based languages represent a decoupling from complex object definitions and specifications—this may be called the second type of decoupling. For system modifiability, modeling should employ languages with inherently low coupling to avoid introducing accidental complexity into the system, thereby facilitating the revelation of essential complexity [8].

Therefore, object-based languages are more appropriate for modeling. However,

the event-driven mechanism allows the timing, quantity, and interrelationships of events (sequences) to be random and uncertain, necessitating resolution of the "event sequence trap" problem [9]. Especially as programs grow larger and more complex with stronger modification requirements, addressing program disorderliness must precede achieving modifiability.

This paper discusses system modifiability based on our modeling practice over several years using the State Event Analysis (SEA) methodology. Examining the evolution from a single-elevator system to a group elevator system reveals how the SEA methodology consistently preserves modifiability throughout this evolution.

3 SEA Methodology and System Orderliness and Modifiability

After establishing a single-elevator model using the SEA methodology [10], we developed an N-elevator multi-agent "blackboard pattern"group elevator scheduling model [11]. We now describe the relationship and evolution between single and group elevator models from several perspectives: state transition diagrams, timer object usage, data structures, decision algorithms, and visualization interface technology, using this example to observe and discuss how the SEA methodology realizes, utilizes, and preserves the modifiability of logic control systems.

3.1 Use of State Transition Diagrams in the SEA Methodology

The SEA (State Event Analysis) methodology performs state transition analysis on event-driven systems. It employs universal computer visualization objects/entities for system modeling, providing a consistent approach for system analysis, design, programming, debugging, modification, and maintenance that yields excellent modifiability.

Although the target system' s state transition diagram is key to SEA methodology modeling, it is not necessarily the first step to draw the complete system state transition diagram. For instance, when a portion of the target system can be defined as one or several states within the overall state transition diagram while the remainder remains poorly understood, modeling can begin by defining partial states and then completing the target system through state expansion, ultimately obtaining the overall system state transition diagram through the modeling process. Single-elevator system modeling using the SEA methodology primarily achieves the system and its modifiability through state expansion [10]. For group elevator system modeling, this paper introduces another system expansion approach in the timer section that enables evolution from single to group elevators. In summary, the timing and manner of using state transition diagrams for target systems are predicated on facilitating system analysis and understanding—this represents an important characteristic of state transition diagram usage in the SEA methodology.

State transition diagrams are tools for describing systems from the outside in and serve as a framework for analyzing and understanding systems even in their infancy. Determining the timing and conditions for state transitions is a process from coarse to fine, from vague to precise. Because state transition diagrams have an implementation at the statement level—the select case statement, which is the unified, sole framework statement within each event handling function—the gradual clarification of system understanding primarily relies on the programming process guided by state transition diagrams. The state transition diagram, the global single state control variable "state," and its select case statement collectively form the SEA methodology' s outside-in, top-down system framework [12]. Although SEA methodology involves multiple modeling elements, because the state perspective controls the entire system workflow, every program statement except those in the select case statement and initialization program exists in and only in one definite state. "State" becomes the organizing principle that integrates and orders the system. State connects the static text of each event handling function while also establishing dynamic, organic state-dimensional relationships among all processing. State' s partitioning and unifying of the system resembles a butcher' s dissection and reassembly of an ox.

3.2 Two Phases of Single-Elevator Modeling

[Figure 1: see original paper] shows the state transition diagram of a single-elevator system. State-controlled single-elevator modeling is completed in two phases. The first phase uses a visualization interface to establish an unconditional reciprocating operation between floors, stopping at each floor with door opening/closing, creating a dynamic system environment. Critically, no command buttons are placed on the interface during this phase, avoiding programming that processes each button input as it occurs. The first phase' s modeling task is to obtain the logic for processing/servicing stored user requests when the system is in working states other than the two pending-service states. Without buttons on the interface, the assumption is that once the program runs, it cuts into a designated working state—the dashed line in Figure 1 indicates cutting into the upward state. Therefore, the system initialization program sets: (1) the state the system will cut into; (2) the current positions of the elevator car and doors; (3) assumed accepted requests in the user request array; and (4) corresponding values for key variables. Upon program startup, the elevator begins upward operation. Figure 1 does not show dashed lines for transitioning from the initialization program to the stopping or downward states, though such transitions can be configured when modeling requires them. These dashed lines only represent temporarily applied state transitions during the first modeling phase. Modelers configure various user request combinations, elevator positions, and directions in the initialization program to analyze, program, and debug the logic constrained by "maximum same-direction, en-route service" within the upward, downward, and stopping states (these three states are also called non-pending-service or non-idle states), thereby obtaining scheduling algorithms covering all

user request data [10]. When the elevator completes servicing user requests, it enters the first-floor or non-first-floor pending-service state (these two states are also called pending-service or idle states). In SEA methodology, events causing state transitions are mostly those defined in event-driven languages, such as user request input events that cause transitions from pending-service to non-pending-service states. However, some are system-level defined behavioral events, such as the elevator leveling event that causes transitions from upward/downward states to the stopping state, which is implemented using timer interrupt events. An important feature of the SEA methodology is using the dynamic, animated environment of visualization programs to analyze, clarify, and refine system logic.

The second phase of single-elevator modeling places floor and in-car command buttons on the visualization interface and completes programming for button input processing. This processing mainly includes: (1) input validity checking; (2) starting idle elevators; and (3) storing valid requests in the user request array [10]. Because state governs every statement and every animation action on the graphical user interface is programmed within event handling functions, there is no separate GUI development issue under the SEA methodology [13]. This phase completes all programming for the single-elevator system. Command buttons on the interface can be pressed at any time, and these inputs are processed logically and functionally appropriate in each state.

3.3 Timer Object

Both single and group elevator systems use only one timer object. The timer starts and exits simultaneously with the system and remains the core object of modeling.

First, consider the timer' s operation in the single-elevator system. Because the maximum structure in the timer interrupt routine is a select case statement for the state variable, each timer interrupt enters a definite state. The timer routine accomplishes: (1) displaying dynamic behaviors of visualization objects on the interface, such as displacement changes during elevator movement and door opening/closing; (2) controlling time durations for door opening/closing through time counting; (3) causing state transitions when detecting positions or moments for state migration; and (4) calling three decision algorithms at system-scheduled decision moments to make the system act according to decision results. The first step in transforming the single-elevator program into a group-elevator program is replicating the original single-elevator program N times within the timer, where N is the number of elevators. Correspondingly, the visualization interface increases to N elevators, and all variables in the single-elevator program become N-element variable arrays. Because each elevator has its own independent state control variable state(i), i=1~N, the group elevator system is an asynchronous parallel system. This evolution from single to multi-elevator system represents a non-state-expansion expansion approach, thereby increasing system complexity while preserving modifiability.

3.4 In-Car Requests

We define in-car requests as the destination floor numbers entered by users after entering the elevator—these are tasks the elevator must complete. The system stipulates that in-car requests are only valid when they are destination floor numbers ahead of (approachable by) the elevator' s current direction. For fault tolerance, in-car requests entered while the elevator is in pending-service state can also start the elevator to service that request.

3.5 Data Structure for User Requests in Group Elevator Systems

All user requests in an N-elevator group system are stored in a task table array button(j,i), j=1~MAXL, where MAXL is the highest building floor. The N elevators' in-car requests occupy columns i=1~N of the task table, denoted as stopi(j). Each cell in this MAXL×N array takes two values: 0 indicates no request, and 1 indicates an in-car request corresponding to that cell. Column N+1 of the task table stores upward floor requests button(j,N+1)=up(j), j=1~MAXL-1; column N+2 stores downward floor requests button(j,N+2)=down(j), j=2~MAXL. Cells in these two floor request columns take four values: (1) =0, indicating no floor request; (2) =i, i=1~N, indicating elevator i will service this floor request; (3) =N+i, indicating elevator i has an in-car request matching this floor request; (4) =2N+1, indicating the elevator to service this floor request is undetermined. The third value N+i represents a "match" where if elevator i has an in-car stop request at floor m and the system has a floor request at m in the same direction as elevator i, these two requests are considered matched. The system performs matching whenever any floor or in-car request arrives, and if matching succeeds, writes N+i (i=1~N, the matched elevator number) into the floor request cell. A floor request may match in-car requests from multiple elevators, and the system allows all matching situations to be written into that floor request cell in an overlay manner. The other two values indicating tasks, i or 2N+1, are dynamic and will be detailed later. Each individual elevator in the group system only faces its own data structure, identical to the single-elevator system' s data structure —composed of elevator i' s in-car requests stopi(j) and floor up(j) and down(j) columns. However, unlike the single-elevator system, the group system' s two floor request columns are shared and jointly serviced by N elevators. Therefore, the original single-elevator scheduling logic must be modified so that N elevators respond to randomly arriving in-car and floor requests without missing any request while minimizing the average response time caused by extended or frequent stops by any single elevator.

3.6 Starting Idle Elevators in Group Elevator Systems

Besides in-car requests that can cause idle elevators to leave idle state, the system' s response to newly input floor requests represents the primary opportunity to start idle elevators. For a new floor request, if it matches no in-car request, the system finds the idle elevator nearest to that floor request. When found (elevator i), the system writes elevator number i into that floor request cell in

the task table and simultaneously starts elevator i to service the floor request. If no idle elevator exists in the system, the value 2N+1 is stored in that floor request cell.

3.7 Brief Introduction to Single-Elevator Logic

The state transition diagram from the original single-elevator system (Figure 1) is still used to describe individual elevator operation in the group system. Each elevator' s decision algorithms in the group system are derived by modifying the single-elevator system' s three decision algorithms. Table 1 provides a brief introduction to the three decision algorithms called in the single-elevator system [10].

Table 1: Introduction to Three Decision Algorithms in Single-Elevator System

#	Decision Algorithm	Calling Moment	Decision Content
1	Stop Decision	Before reaching floor during up/down movement	Whether to stop at this floor
2	Priority Direction Decision	After floor stop, before door opening	Service direction for this door opening
3	Operation Decision	After stop layer service ends	Search for new service tasks; if none, transition to pending-service state

When a single elevator responds to a user request across floors, it proceeds through three decision algorithms in sequence. The stop decision is called before each floor, while the other two decisions are called during floor stops. Each individual elevator in the group system still uses calling these three decision algorithms as its main operational logic.

3.8 Scheduling Logic of Group Elevator Systems

In the task table, the elevator number i written for floor tasks is entered by the system only when starting idle elevators; otherwise, it is entirely entered proactively by elevator agent i onto the task table "blackboard."We now describe each individual elevator' s operational logic. During a given period, elevator i in the group system only works to complete either its own in-car request (matched or unmatched) or a floor request marked with i—one of these two exclusively. The purpose of marking elevator numbers in the task table is to enable multiple elevators to claim different floor tasks, thereby avoiding overlapping service. The stipulation is: for any elevator i, there is only one cell in the task table containing its elevator number i during necessary periods; during most periods

when elevator i's current operation target is an in-car request and during most of its stopping state, the task table contains no cells with value i. This stipulation minimizes the number of floor requests marked for any elevator at any moment, allowing other elevators to mark and respond to floor requests originally valued at 2N+1 as early as possible.

During stop decisions before each floor, elevator i checks for new targets. New, closer targets may be: (1) new in-car requests for elevator i; (2) other elevators returning their originally marked tasks to the system by changing the task value to 2N+1; or (3) the system writing 2N+1 for newly arrived floor requests when no idle elevator was started. If a randomly occurring situation is closer to elevator i, it replaces the original, farther target. If the original farther target was a floor task, its value i is changed to 2N+1, keeping elevator i' s marked tasks in the task table minimal. After leveling at a floor and before opening doors, elevator i calls the priority direction decision, whose purpose remains as listed in Table 1. Notably, if a task valued 2N+1 exists ahead of elevator i, the elevator should at least maintain its running direction. After the priority direction decision, the elevator opens doors for service, at which point the task table no longer contains floor tasks valued i. After completing stop layer service, elevator i performs the operation decision, searching for the nearest in-car request (matched or unmatched) and floor requests valued 2N+1. If the nearest is an in-car request, the task table still contains no cells valued i, and operation begins. If the nearest is a floor task valued 2N+1, its value is changed to i and operation begins. If no such tasks are found ahead, the elevator searches and services in the opposite direction until confirming no tasks require service, then enters the pending-service state.

The scheduling characteristic of this method is that balancing tasks among elevators is achieved dynamically through the independent, autonomous behavior of each elevator agent. The system only compares distances between idle elevators and floor signals when processing new floor signals that require starting idle elevators. Beyond this, neither the system nor individual elevators perform horizontal comparisons between elevators, nor does the system conduct top-down task allocation.

This represents a logically closed, complete foundational scheduling model for group elevator systems. "Logically closed"means that for both the system and its individual elevators, from the first user request to the last serviced request, each elevator experiences a closed loop of states from pending-service to non-pending-service and back to pending-service. Moreover, this system program can adapt to any elevator count N from 1 to 8. When programming, setting constant N=1 yields a single-elevator system. During early group elevator modeling, most editing and debugging of group elevator logic occurs with N=1. Then, values N$ $8 are set to run, debug, verify, and modify the system separately.

3.9 Further Modifications to This Group Elevator System

This is a highly adaptable and modifiable foundational group elevator model. Based on this model, future modifications can proceed in several main directions:

First, modifying the conditions for starting idle elevators. To balance system energy consumption and average response speed, a threshold for average tasks of working elevators can be added, called the idle elevator start decision factor. When a floor request arrives, the system calculates the average task load of working elevators. If this value exceeds the set threshold, an idle elevator is started; otherwise, the floor request is directly entered into the request task table with value 2N+1.

Second, limited by space, this paper only presents key points of top-down modeling for group elevator systems. For actual engineering needs, this model can be further refined. We have implemented a refined version of this model on PLC systems [11], demonstrating the advantages and feasibility of our method.

Third, implementing system peak scheduling functions. Real group elevator systems often have various specific peak scheduling requirements, such as up-peak/down-peak services (UPS/DPS). Generally, to meet specific requirements, the SEA methodology allows modifications at three levels: (1) Program level —adding new event lines to the state transition diagram in Figure 1 without adding new states, enabling more state transitions under additional events (conditions); (2) Modifying the state transition diagram in Figure 1 to add specific peak states to some individual elevators; (3) Treating the aforementioned group elevator foundational model as a normal state of the group elevator system. Adding a top-level state transition diagram and top-level state control variable top_{state} above this model creates a system containing peak states that can transfer with the normal state. Notably, even when adding another layer of states, the top-level top_{state} still controls states that partition the system down to the statement level—states remain extensible. These three modification schemes may need to be combined and used simultaneously. In practice, peak state switching generally occurs through timing. Because peak functions and requirements vary across specific systems, the SEA methodology and this model provide these modification possibilities. Providing such a foundational model with multiple modification possibilities is theoretically and engineering-wise necessary and meaningful.

4 Conclusion

The SEA methodology achieves and preserves system modifiability because it can always realize the framework value of its target systems [2]. Whether for simple systems, 雏形系统 (nascent systems), or complex systems like the group elevator model in this paper, the SEA methodology simultaneously achieves both the behavioral value [2] and framework value of systems. As real-world entities increasingly transform into computer-based systems, revealing the essential complexity [8] of such entities becomes an inevitable mission of object

technology. Object technology is not obsolete, nor are its technical problems all solved [14]; rather, it is at a stage of fulfilling its mission while developing its own theory, technology, applications, and experience. We should acknowledge the uniqueness of object technology in revealing the essential complexity of logic control systems and should now further develop its specifications and applications. Simultaneously, we should develop China' s own object-based modeling (meta-model) language [15] and its theory based on the philosophy of technology proposition discussed in this paper.

References

[1] Yu Qingyun, Zhao Hui, Xu Jia, et al. Research status and prospects of complex manufacturing system modeling and optimization [J]. Information and Control, 2023, 52(1): 1-17.
[2] [US] Robert C. Martin. Clean Architecture [M]. Translated by Sun Yucong. Beijing: Publishing House of Electronics Industry, September 2018: 3-11.
[3] [US] Robert C. Martin. Clean Code [M]. Translated by Han Lei. Beijing: Posts & Telecom Press, February 2020: 1-13.
[4] [US] Martin Fowler. Refactoring: Improving the Design of Existing Code (2nd Edition) [M]. Translated by Xiong Jie. Beijing: Posts & Telecom Press, April 2019: 45-68.
[5] [US] Robert C. Martin. Agile Development [M]. Translated by Jian Fangda. Beijing: Tsinghua University Press, August 2021: 1-21.
[6] [US] John Ousterhout. A Philosophy of Software Design (2nd Edition) [M]. Translated by Ru Bingsheng. Beijing: Posts & Telecom Press, November 2024: 191-198.
[7] [CA] Ilya Suzdalnitski. Object-Oriented Programming: A Trillion Dollar Disaster [EB/OL]. Henan: Cloud Headlines, August 23, 2019. https://medium.com/better-programming/object-oriented-programming-the-trillion-dollar-disaster-92a4b666c7c7
[8] [US] Frederick P. Brooks Jr. The Mythical Man-Month (Commemorative Edition) [M]. Translated by UMLChina. Beijing: Tsinghua University Press, June 2023: 1-7.
[9] [IL] Natan Silnitsky. 5 Pitfalls to Avoid in Event-Driven Architecture [EB/OL]. Translated by Ming Zhishan. Liaoning: InfoQ, January 20, 2023. https://natansil.medium.com/event-driven-architecture-5-pitfalls-to-avoid-b3ebf885bdb1
[10] Wu Xiaotao, Wang Sujuan, Tang Guochun. A modeling method for logic control systems [J]. Computer Simulation, 2006, 23(08): 49-54.
[11] Wang Sujuan, Wu Xiaotao. Research on scheduling algorithm for elevator group control system [J]. Modern Electronics Technique, 2022, 45(11): 104-107.
[12] Wu Xiaotao, Chen Guanling, Yang Meihua, et al. Implementation of modeling for logic control systems [J]. Computer Engineering, 2005, 31(21): 195-197, 200.
[13] Liu Guicong, Zhou Gan, Du Chao, et al. Research on topology modeling method for visualization applications oriented to aerospace missions [J]. Net-

work Security and Data Governance, 2022, 41(2): 99-105.
[14] [US] Rhea Moutafis. Is Object-Oriented Programming Dead? [EB/OL]. Translated by Sambodhi. Beijing: AI Frontline, October 12, 2020. https://towardsdatascience.com/object-oriented-programming-is-dead-wait-really-db1f1f05cc44
[15] Wu Xiaotao, Tang Guochun, et al. Preliminary exploration of computer modeling for complex systems [J]. Journal of System Simulation, 2004, 16(8): 1779-1784.

(Corresponding author: Wang Sujuan E-mail: sjwang@sspu.edu.cn)

Author Contribution Statement:

Wang Sujuan: Designed the research plan and implemented the research plan;
Zhu Hongfeng: Data collection and analysis;
Wang Sujuan, Zhu Hongfeng: Drafted the paper;
Wu Xiaotao: Proposed the research idea, designed the research plan, revised the final version of the paper.

Note: Figure translations are in progress. See original paper for figures.