International Journal of
Computer & Software Engineering

Original Article Open Access

# Design Support Based on Evaluation of Descriptive Quality of Hierarchical State Machines

**Tsuyoshi Nakajima[1*], Yoshihisa Takayama[1] and Shuichi Tokumoto[2]**

*[1]Department of Computer Science and Engineering, Shibaura Institute of Technology, Tokyo, Japan*

*[2]Information Technology R & D Center, Mitsubishi Electric Corporation, Kamakura, Japan*

**Abstract**

In designing the behaviour of systems and software using hierarchical state machines, the issue is whether the designers can properly layer the states or not. This is because improper layering easily makes hierarchical state machines complicated to hinder its readability and testability. To settle the issue, this paper proposes a method and tool to evaluate the descriptive quality of hierarchical state machines. The method defines two complexity measures of evaluating the hierarchy from the two perspectives of state and state transition, enabling to evaluate the descriptive quality of a single hierarchical state machine individually. Based on the method, the tool has been developed to support designing the diagram. The results of experiments to apply this method and tool shows their effectiveness in designing hierarchical state machines in a quality way.

**Publication History:**

Received: April 30, 2022
Accepted: May 13, 2022
Published: May 16, 2022

**Keywords:**

State machines, Hierarchy, Complexity measures, Design support

## 1. Introduction

The state transition diagram is suitable for describing the dynamic behaviour of the system, and has several advantages for systems and software design. One is the human comprehensible graphic representation, and the other is the mathematical background for performing formal verification and code / test case generation.

The original state transition diagram does not have a hierarchical structure of states (hereinafter referred to as a simple state transition diagram). When the system to be described becomes complicated, the number of states and the number of state transitions explosively increase in the simple state transition diagram, and therefore it is difficult to use it in a realistic design. To solve this problem, Harel devised an enhanced state transition diagram, Statecharts, which introduces hierarchy and parallelism into a simple state transition diagram [1].

Statecharts are used in various systems and software development since they are highly practical. Statecharts are used as state machine in UML (Unified Modelling Languages) [2] and SysML (System Modelling Languages). In this paper, Statecharts and UML state machine are collectively referred to as hierarchical state machines.

In behavioural design using a state machine, the issue is how to properly layer states. Improper layering makes the state machine complex to reduce its quality, such as readability and testability.

This paper proposes a method and tool to evaluate the descriptive quality of a hierarchical state machine to solve this problem. This method defines two measures to evaluate the hierarchy of the machine from two perspectives: states and state transitions, both of which are normalized using evaluate scale measures to evaluate the machine independently. The tool was developed to demonstrate how the method can be used to help designers to create quality state machine diagrams.

In this paper, Section 2 explains hierarchical state machines and the problems to use them for systems and software design, and Section 3 describes existing measures for evaluating the descriptive quality of hierarchical state machines and their issues. Section 4 defines the requirements for evaluating the descriptive quality of the diagram and Section 5 proposes a method to satisfy them. Section 6 describes an experimental evaluation of the method and its results. Section 7 shows a prototype of design support tool based on the proposed method. Section 8 provides related works, and Section 9 concludes this paper.

## 2. Hierarchical State Machine and Its Issues

### Hierarchical state machines

The hierarchical state machine is a graphical notation that enables compact description of systems and software by giving hierarchy and parallelism to the simple state transition diagram. The state created by exclusive division (OR decomposition) of a certain state is called the OR state, and the state created by parallel division (AND decomposition) of a certain state is called the AND state.

Hierarchy is repeatedly to introduce a common state to aggregate multiple OR states, which is also an OR state. This common state is called abstract state. Abstract states can have shared entry and exit actions and exit transitions, which greatly simplifies the diagram while retaining the same semantics. The abstract state also has the effect of modularity, dividing its interior and exterior.

Parallelism is to introduce parallel states each of which represents an independent state variable. A state can have many regions representing parallel states (AND states).

[*]**Corresponding Author:** Prof. Tsuyoshi Nakajima, Department of Computer Science and Engineering, Shibaura Institute of Technology, Tokyo, Japan, Tel: +81-35859-8514; E-mail: tsnaka@shibaura-it.ac.jp

**Citation:** Nakajima T, Takayama Y, Tokumoto S (2022) Design Support Based on Evaluation of Descriptive Quality of Hierarchical State Machines. Int J Comput Softw Eng 7: 177. doi: https://doi.org/10.15344/2456-4451/2022/177

The AND state allow us to describe the parallel operation of independent devices as they are. In the simple state transition diagram, it can be expressed only by multiplying the states, in which the number of states and the number of state transitions increase in combination. In contrast, the hierarchical state machine can reduce the number of both states and state transitions.

**Descriptive quality of hierarchical state machines and the issues on measuring it**

Hierarchical state machines make it possible to describe large-scale and complex systems compactly, and to create highly readable requirements specifications and design specifications with rigorous syntax and semantics. As a result, implementing and testing them in the subsequent processes can proceed smoothly, and maintainability of requirements specifications and design specifications can be improved.

On the other hand, the introduction of hierarchy and parallelism gives the designer some increased freedom in where and how much hierarchy and parallelism should be used. This is one of the main reasons why different state machines with different descriptive quality are created even for the same behaviour depending on the designer's skill.

We define the descriptive quality of hierarchical state machines as the following two characteristics:

Modularity: degree to which the specification can be divided into independent parts with no / little interactions between them: i.e., the hierarchy separates the lower levels (inside) from upper (outside) levels of an abstract state, and the parallelism decomposes a state into the AND states. Modularity enhances the readability and reusability of descriptions, reduces implementation errors, improves the effectiveness and efficiency of design inspections and testing.

Ease of implementation: degree to which the specifications are easy to implement, which is because the firing conditions for state transitions are integrated to reduce the number of actions. Ease of implementation enhances readability and testability as well as reduces the code size to be implemented.

Using examples of air conditioner operation modes shown in Figure 1 and 2, the descriptive quality of the hierarchical state machine is explained.

Figure 2a and Figure 2b are two examples of hierarchical state machines that have the same behaviour as the simple state transition diagram in Figure 1. Figure 2a is a bad example, in which the abstract state "Compressor ON" is introduced. Compared to the simple state transition diagram, number of transitions is reduced by one (improving ease of implementation), but five cross-hierarchical transitions have been created. This causes its internal components to directly affect the external components (decreasing modularity).

This shows that the descriptive quality differs greatly depending on how abstract states are introduced. Therefore, a means to quantitatively evaluate the hierarchy and point out the bad points will help designers to write more quality hierarchical state machines.

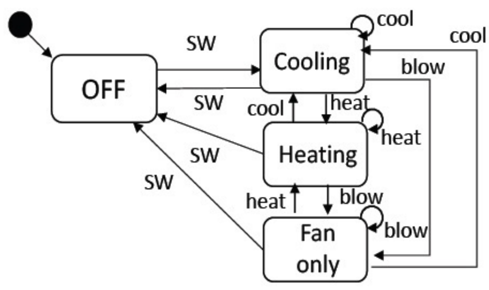


Figure 1: Original simple state transition diagram.

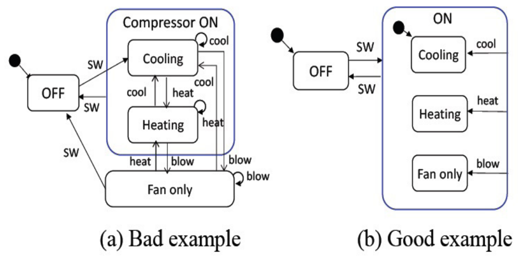


Figure 2: Two examples of hierarchical state machine equivalent to Figure 1.

## 3. Existing Measures

This section describes existing measures of the descriptive quality of hierarchical state machines and their problems.

**Basic measures of hierarchical state machine**

The following basic measures are used for the node (state) and edge (state transition) of hierarchical state machines [3].

- State-related: Number of states, number of abstract states, number of leaf states
- Transition related: Number of transitions, number of cross-hierarchy transitions

These measures are measured directly by counting elements appearing in the diagram, and easy to understand what they are.

Table 1 shows the results of measuring the base measures for the examples in Figure 3a and Figure 3b, where number of crosshierarchy transition is the total number of times across the boundary of the OR states through which the state transition passes.

| Base measure | | (a) Bad | (b)Good |
|---|---|---|---|
| State related | Number of states | 5 | 5 |
| | Number of abstract states | 1 | 1 |
| | Number of leaf states | 4 | 4 |
| Transition related | Number of transitions | 12 | 5 |
| | Cross-hierarchy transition | 5 | 0 |

Table 1: Base measures and their values for Figure 3a and 3b.

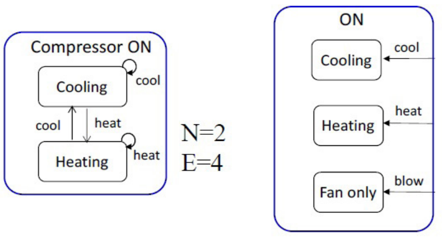


Figure 3: The second layer of the two hierarchical state machines in Figure 2.

In Table 1, the state-related measures have the same values in (a) and (b), whereas the state transition related measures have larger values in (a) than in (b). From this comparison, the hierarchical state machine (b) has the smaller number of transitions and cross-hierarchy transition, is considered to make better use of the hierarchy.

This example shows that the evaluation of the descriptive quality of the hierarchical state machines using these base measures needs to be performed by comparing multiple descriptions. This is because these base measures depend on not only the descriptive complexity but also that of the problem. Such evaluation by comparison is suitable for evaluating the effect of refactoring, but it cannot be used for evaluating a single hierarchical state machine.

**Cyclomatic complexity of hierarchical state machines**

(1) Cyclomatic complexity

Cyclomatic complexity is a measure for the complexity of a graph. The cyclomatic complexity (CC) can be calculated for a single graph by the following equation.

$$CC = E - N + 2$$

E: the number of edges
N: the number of nodes

Cyclomatic complexity is a measure for the complexity of a graph. The cyclomatic complexity (CC) can be calculated for a single graph by the following equation.

- Calculate CC as a branch point (IF statement, etc.) for a node and as a control flow for an edge.
- Evaluate CC for each program unit such as a function.

C strongly depends on the complexity of the problem itself as branches represent some kinds of decisions.

By evaluating CC per module, it can be used as an index showing its quality. For example, the module has good quality if the CC per module is 6 or less and is judged too complicated if it is 10 or more. CC has been widely used as an index for module complexity showing the ease of unit testing (and code inspection).

(2) Application of cyclomatic complexity to hierarchical state machines

Hall proposes to apply cyclomatic complexity to hierarchical state machines as follows [5].

1. Count states as nodes and state transitions as edges.
2. Evaluate CC for each hierarchy level.

To calculate the CC of the hierarchical state machines in Figure 2a and 2b, we extract the state machines at the first are shown in Figure 4, and those at the second layer level is shown in Figure 3.

In Figure 4, transitions across hierarchies are counted at the outermost hierarchy level. The initial state and history are not counted as nodes, and the state transitions from the initial state and history are not counted as edges.

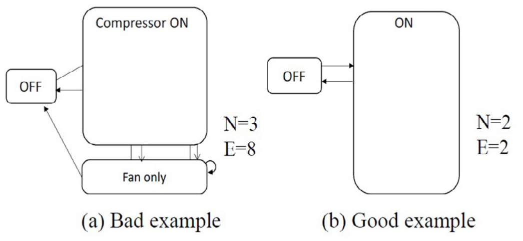


Figure 4: The first layer of the two hierarchical state machines in Figure 2.

| Hierarchy level | Cyclomatic complexity (CC) | |
|---|---|---|
| First level | (a) | (b) |
| Second level | 7 | 2 |
| Cross-hierarchy transition | 4 | 2 |

Table 2: Values of cyclomatic complexity for Figure 3 and Figure 4.

Table 2 shows the values of CC for the state machines in Figure 3 and Figure 4.

In addition, since the evaluation is performed for each hierarchical level, it seems that we can use the CC by layer as an index value similarly to the CC by module: i.e., it can be used for keeping the CC value of each layer under some criterion to incorporate hierarchy in a well-balanced manner.

However, there are the following problems when applying it.

- Introducing many abstract states can simply reduce the complexity of each layer. Since there is no penalty for introducing them in vain, the quality of introducing the abstract states themselves cannot be evaluated properly.
- The number of cross-hierarchy transitions that have an adverse effect on modularity has not been evaluated.
- There is no consideration for AND states.

## 4. Requirements and Approaches

**Requirements for the evaluation methods**

To solve the problems of the existing measures, we set the following four requirements for the method of evaluating the descriptive quality of the hierarchical state machines.

1. Goodness of the hierarchy for the hierarchical state machine with AND states can be evaluated.
2. A single machine can be evaluated.
3. Only a small set of measures are used.
4. Information to improve them are given.

**Approaches to meeting the requirements**

As a set of measures for the descriptive quality of hierarchical state machines to meet the above four requirements, we propose the following two measures.

*Es* = Evaluation of states / State size (1)
*Et* = Evaluation of transitions / Transition size (2)

To satisfy requirement 1), we define two independent measures to quantify hierarchy, "Evaluation of states" and " Evaluation of transitions." To satisfy requirement 2), we define two scale measures, "State size" and "Transition size" to normalize the above two measures into Es and Et so that we can evaluate them regardless of complexity of the problem.

The fact that only two measures (*Es* and *Et*) are used satisfies requirement 3). In fact, plotting the measured values on a scatter diagram allows us to evaluate them as positions on a two-dimensional plane. In addition, when *Es* is outside the proper region, meaning that the hierarchy of states is bad, you can see what is wrong using the state-related base measures in Table 1. The same is true for *Et*. This means that requirement 4) will be satisfied.

## 5. Evaluation of Hierarchy

In this section, we propose two measures for evaluating the hierarchy and describe the rationale for them.

**Complexity on state**

(1) Preparation

The hierarchical level L (X) of a certain state X is defined as follows:

- When X = U, L (U) = 0
- When X is an OR state, L (X) = L (SP (X)) + 1
- When X is an AND state, L (X) = L (SP (X))

where SP (X) is the parent state of X, and U is the universe state consisting of all the states in the target machine. In this paper, U is treated as the single top-level state, i.e., not enclosed by any state on the state machine. When L(X)>L(Y), we say that the state hierarchy level of X is lower than that of Y.

The state weight w (X) of a certain state X is defined as follows.

- When X = U, w (U) = 1
- When X is an OR state, w (X) = w (SP (X)) / noc (SP (X))
- When X is an AND state, w (X) = w (SP (X))

where noc (X) is the number of OR states directly under state X.

The state weight represents the importance of each state in the hierarchy. The OR decomposition of each hierarchical level is considered to divide the space of the parent state into equal parts by the number of the subordinate OR states. Therefore, the deeper the hierarchy, the less the weight.

On the other hand, the weights of the AND states are set to be the same as that of the parent state, considering that it is inherited from the parent state. Figure 5 shows an example of the weight calculation results.

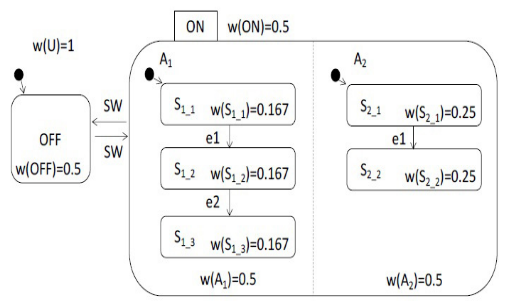


Figure 5: Example of state weights.

(2) Definition

The state hierarchical complexity *Cs* of a certain hierarchical state machine is recursively defined using the hierarchical complexity *Cs (X)* of the state *X*.

- *Cs* = *Cs* (U)
- For state X, if the set of child states Sub (X) = {X1, X2, ..., XN}, then the state hierarchy complexity Cs (X) of X is:
  - When Sub (X) = ø, *Cs* (X) = 0
  - When N>0 and the elements of Sub(X) are OR states,
    $$C_S(X) = (N\text{-}1)w(X) + \sum_{i=1}^{N} C_S(Xi)$$
  - When N>0 and the elements of Sub(X) are AND states,
    $$C_S(X) = \sum_{i=1}^{N} C_S(Xi)$$

In other words, the state hierarchical complexity *Cs* (X) is calculated by multiplying the number of divisions (OR decomposition) of the state X by the state weight and summing them up.

(3) Design rationales

The state hierarchy complexity *Cs*(X) is defined as the state complexity for measuring the modularity of the hierarchical state machine. The design rationales are shown below.

1. The complexity of state X is considered proportional to the number of OR states of X. Therefore, *Cs* (X) increases when adding OR states directly under X.
2. By adding the OR state to the state X, the complexity of the state X increases by the weight of X. As a result, the complexity increment at deeper hierarchies is smaller. This is because the introduction of the abstract state is considered to have the effect of limiting the scope of attention and increasing the modularity. State weights are used to reflect this point.

3. Unlike the OR decomposition, because the AND decomposition does not create a subset group of the parent states, AND states directly under the state X inherits the same weight of the state X and simply adding an AND state to state X does not increase its complexity

(4) Verification by examples

The state hierarchy complexities for three examples (in Figure 1 and Figure 2a and Figure 2b, which have the same behavior, are:

$Cs$(Fig.1)=(noc(U)-1)·w(U)=(4-1)·1=3
$Cs$(Fig.2(a))=(noc(U)-1)·w(U) + $Cs$(compressor operating)
=(3-1)·1 + (noc(compressor operating)-1)·

w(compressor operating)
= 2+(2-1)·0.33=2.33
$Cs$(Fig.2(b))=(noc(U)-1)·w(U)+$Cs$(ON)=(2-1)·1+
(noc(ON)-1)·w(ON)=1+(3-1)·0.5=2.0

The example in Figure 1, which is a simple state transition diagram, has the largest $Cs$ value of 3. That in Figure 2a, which is a bad example of hierarchal state machine, has the second largest of 2.33. That in Figure 2b, which is a good example of hierarchal state machine, has the smallest of 2.0. It indicates that $Cs$ evaluates that the smaller the number of states in the upper layer, the lower the complexity.

(5) Analysis on range of $Cs$

Consider a simple state transition diagram with N states, assuming that there are two or more OR states at the highest level of the hierarchy. In this case, the minimum value of the state hierarchy complexity $Cs$ is 1. Therefore, the range of the state hierarchy complexity $Cs$ is as follows:

$$1 \leq Cs < \text{Number of states when reduced} \qquad (3)$$

In other words, the lower limit of the state transition complexity is a fixed value of 1, and the upper limit depends on the specification size.

**Complexity on transition**

(1) Definition

State transition complexity $Ct$ is defined as follows:

$$Ct = \text{Number of transitions} + \text{Number of cross-hierarchy transitions} \qquad (4)$$

where number of transitions is counted only for state transitions whose transition source is an OR state, not including the transitions whose source is the initial state or history.

(2) Design rationales

State transition complexity is the complexity on transition to measure the balance between ease of implementation and modularity in hierarchical state design. It quantifies the following positive and negative effects caused by hierarchy:

- Reduction of number of transitions (positive effect on ease of implementation)
- Increase in number of cross-hierarchy transitions (negative effect on modularity)

Modularity in OR decomposition is the degree to which abstract states are structured so that changes does not affect beyond their boundaries. This is achieved by localization of events, variables, and state transitions within the hierarchy level. The higher the modularity, the less the dependency between layers. In addition, it becomes easier to verify the entire specification by stepwise inspection on each level. Cross-hierarchy transitions impair this modularity.

(3) Verification by examples

The state transition complexities for three examples (in Figure 1 and Figure 2 (a) and (b)), which have the same behaviour, are:

$Ct$(Fig.1) = 13+0=13
$Ct$(Fig.2(a)) = 12+5=17
$Ct$(Fig.2(b)) = 5+0=5

The state transition complexity $Ct$ for Figure 2a is 17, which is an increase of 4 points from 13 for Figure 1. This is because the increase of 5 in number of cross-hierarchy transitions exceeds the decrease of 1 in number of transitions.

On the other hand, the state transition complexity Ct of hierarchy Figure 2b is 5, which is 8 points less than 13 for Figure 1. Since the increase in number of cross-hierarchy transitions is 0, the decrease in number of transitions of 8 is simply evaluated as a decrease in the complexity.

This shows that number of transitions decreases because many of them are integrated when the hierarchy is good, and number of cross-hierarchy transitions increases when it is bad. In other words, the state transition complexity $Ct$ is a measure that balances positive effect on ease of implementation and negative effect on modularity due to introduction of abstract states.

In addition, state hierarchy complexity $Cs$ of a machine can be reduced to some smallest value without considering occurrence of cross-hierarchy transitions, but it may cause the increase of state transition complexity $Ct$. State transition complexity allows us to evaluate such negative aspects of introducing abstract states.

(4) Analysis on range of $Ct$

Since $Ct$ is the sum of number of transitions and number of cross-hierarchy transitions. The number of transitions does not exceed that in the equivalent simple state transition diagram, and the maximum of the number of cross-hierarchy transitions is proportional to it. In other words, the upper and lower limits of $Ct$ depend on the size of the specification.

**Scale measures for normalization**

As mentioned in the previous subsections, defined measures: state hierarchy complexity $Cs$ and state transition complexity $Ct$ increase with the scale of the specification. Therefore, they cannot be used for evaluating the descriptive quality of a single hierarchical state machine yet.

To solve this problem, we propose two scale measures for normalization of state hierarchy complexity and state transition complexity, state size *Ns* and transition size *Nt*.

*Ns*: number of OR states without hierarchy
*Nt*: number of transitions without hierarchy

*Ns* and *Nt* are the number of states and the number of transitions, respectively, for the equivalent simple state transition diagram excluding abstract states.

Transforming from a state machine to the one without hierarchy is equivalent to unfolding abstract states and AND states in the machine. The methods for them are described below. They are applied repeatedly until all the abstract states are taken away and all the AND state reaches the highest level.

(1) Method for unfolding abstract states

The method for removing an abstract state is shown below:

1. For all the transitions outgoing from the abstract state, expand each transition as transitions outgoing from all the subordinate states.
2. For all the transitions incoming to the abstract state, change the destination of each transition to the initial state.
3. Remove the abstract state.

The state machine on the right in Figure 6 is the result of deleting abstract state ON from the one on the left. First, the state transition from ON to OFF is duplicated and expanded to each of the subordinate states (Cooling, Heating, and Fan only), and the state transitions transitioning from ON to the subordinate states are duplicated and expanded to each of the subordinate states (S1). Next, since the state transition from OFF transitions to the initial state, in this case the cooling state (S2). After that, ON is deleted (S3).

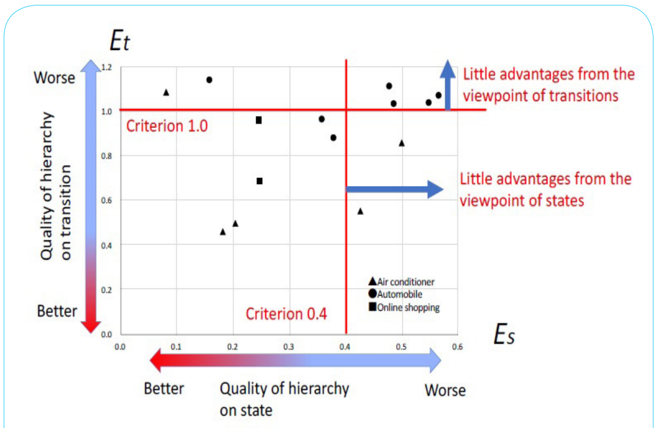


Figure 6: Scatter plot f *Es* and *Et* for the three problems with criteria.

The figure on the right of Figure 7 is the state machine without hierarchy, and its number of states *Ns* is 4 and its number of transitions *Nt* is 13.

(2) Method for unfolding AND states

The method for moving an abstract state which have a set of AND states to one level higher layer of the OR hierarchy as follows:

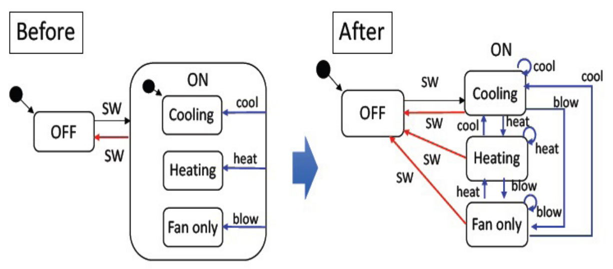


Figure 7: Example of unfolding an abstract state.

S1) For all the transitions outgoing from the abstract state, expand each transition as transitions outgoing from all the subordinate states.

S2) For all the transitions incoming to the abstract state, change the destination of each transition to the initial state.

S3) Expand the transitions from the abstract state to the OR states (originally outside of the abstract state) to ones from all the OR states in each AND state to the duplicated OR state (also in there).

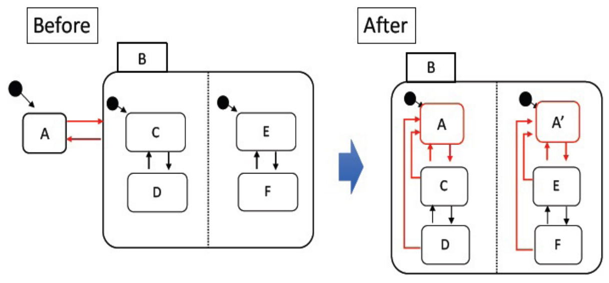


Figure 8: Example of unfolding AND states.

In the example of Figure 8, S1) state A, which is outside abstract state B (having two AND states) is duplicated into the two AND states as A1 and A2, and the original A is deleted. Next, S2) the state transition from A to B expands to the state transition from A1 to C and from A2 to E as certain states respectively. Finally, S3) the state transition from B to A expands to the state transition from C and D to A1 and from E and F to A2.

Like the method for unfolding abstract states, the method for unfolding AND states also increases number of transitions. Furthermore, since the OR states are duplicated and expanded to each AND state, number of states also increases. In the example shown in Figure 9, *Ns* = 6 and *Nt* = 10.

As a method of unfolding AND states, it is common to replace them with the result of the product of a set of OR states included in each AND state. However, the proposed measures focus on evaluating the hierarchy of a hierarchical state machine. Therefore, this method is adopted to treat the effect of parallelism equally on all machines. The results of this method, which aggregates AND divisions into the highest layer, are the same as object divisions, which means that it can be applied to object-oriented modelling.

Based on the state hierarchy complexity *Cs*, state transition complexity *Ct*, state size *Ns*, and transition size *Nt* defined so far, two measures for the descriptive quality of hierarchical state machines Es and *Et* are defined as follows.

$$Es = (Cs - 1) / Ns \qquad (5)$$
$$Et = Ct / Nt \qquad (6)$$

where *Cs* is subtracted by 1 to make the minimum value of Es 0 because the minimum value of *Cs* is 1.

Table 3 shows the measured values of *Es* and *Et* of state machines in Figure 1 and Figure 2a and 2b.

| Measure | Figure 1 | Figure 2a | Figure 4b |
|---|---|---|---|
| *Es* | 0.5 (= (3-1)/4) | 0.33 (= (2.33-1)/4) | 0.25 (= (2-1)/4) |
| *Et* | 1.0 (= 13/13) | 1.3 (= 17/13) | 0.71 (= 5/13) |

Table 3: Values of *Es* and *Et* of state machines in Figure 1 and Figure 2a and 2b.

## 6. Experimental Evaluations

### Purpose of experiment

In order to confirm whether the proposed method satisfies the requirement for descriptive quality evaluation methods 2) of "a single machine can be evaluated," it is necessary to check whether quality criteria that do not depend on the size and type of the problem can be set for the two measures. Therefore, the following research questions are set and evaluated experimentally.

### Approaches to satisfying the requirements

To satisfy R1 and R5, we have implemented the tool as a C# add-in for Enterprise architect (EA) [6], a UML tool with hierarchical state transition diagram editing capabilities. This allows us to check the descriptive quality of the target diagram during its editing process, to improve it, and to see the effect of the improvement immediately.

RQ1. Do the description samples of different problems have an unbiased distribution?

RQ2. Is it possible to set effective quality criteria based on past data?

### Experimental method

For the above purposes, we prepared three problems and multiple subjects to solve each of them. Table 4 shows problems used in the experiment, subjects of the experiment, problem size, and number of samples.

The description experiments on air conditioner, automobile, and online shopping were conducted in the classes of graduate students in different years. In these classes, we first taught the grammar and description examples of the hierarchical state machine and then the problems were given to the students as their assignments. To reduce misinterpretations risks about the problems as much as possible, the corresponding test cases are distributed, and their machines were reviewed in classes, which the students corrected resubmitted them.

Even so, those that did not meet the specifications and/or had grammatical mistakes were corrected by hand, trying not to change any hierarchical structure of the machines. Those that absolutely need to change the hierarchical structure due to the modification are excluded from the samples.

| Problem | Subject | Problem size (Average) | | # of samples |
|---|---|---|---|---|
| | | NOS | NOT | |
| Air conditioner | graduate students, instructor | 22.2 | 31.0 | 5 |
| Automobile | graduate students, instructor | 40.1 | 62.7 | 7 |
| Online shopping | student (who makes the problem), instructor | 33.0 | 60.5 | 2 |

Table 4: Subjects of the experiments.

### Experiment results

Figure 6 plots *Es* and *Et* as a scatter plot for the three problems."

The following results are obtained.

- All data groups are plotted in a modest area of the graph with little scale bias.
- For air conditioner and automobile, which have five to seven samples, *Es* is 0.08 to 0.50 and 0.16 to 0.57, and *Et* is 0.46 to 1.09 and 0.88 to 1.14, respectively.
- Automobile, which have a larger size of specifications (measured by number of sates and number of transitions) than air conditioner, has a narrower range of Et variation. On the other hand, online shopping, which has almost the same size of specifications as automobile has a wider range of Et of 0.69 to 0.96. This means that the difference in the distribution of the data groups is not due to the size of the problem. For example, automobile is the problem of highly independent object decomposition: i.e., subjects tend to select parallelism in their modelling. It can be inferred that the distribution of the data group depends on the degrees of freedom in modelling on the problems.

From the above results, we find that there is a possibility that the quality of the hierarchy of a single hierarchical state machine can be evaluated as the absolute position on the two dimensional plane by the proposed measures, regardless of its problem size. This allows us to evaluate it without comparison.

### Setting specific criteria

In order to get the answer to RQ2 " Is it possible to set effective quality criteria based on past data?", We examined to set criteria for the two proposed measures to properly evaluate the sample data shown in 6.2.

(1) Criteria for the descriptive quality measure on transitions

The following criterion can be used for t the descriptive quality measure on transitions *Et*.

$Et \leq 1.0$

The rationale is that since *Et* has number of transitions for the "state machine without hierarchy" as its denominator, the value of 1.0 is the situation in which a merit of reduction in number of transitions and a demerit of emerging number of cross-hierarchy transitions are offset. If the measured value is more than 1.0, the state machine is considered to rather have worse quality than the one without hierarchy.

(2) Criteria for the descriptive quality measure on states

The following criterion can be used for t the descriptive quality measure on states $Es$.

$Es \leq 0.4$

Unlike $Et$, there is no theoretical basis for this criterion, and the value of 0.4 is a tentative value based on the sample data. The rationale comes from only the fact that the samples with $Es$ of 0.4 or more have a characteristic that AND states existside by side at the highest level and almost no hierarchy is used, which is not seen in the samples with $Es$ less than 0.4.

It is thought that the application of this criterion has the effect of discouraging the easy use of AND decomposition and encouraging the application of OR decomposition to first extract commonality such as ON / OFF. As the number of data increases, it can be expected that a more appropriate criterion for judging the quality of hierarchy can be obtained.

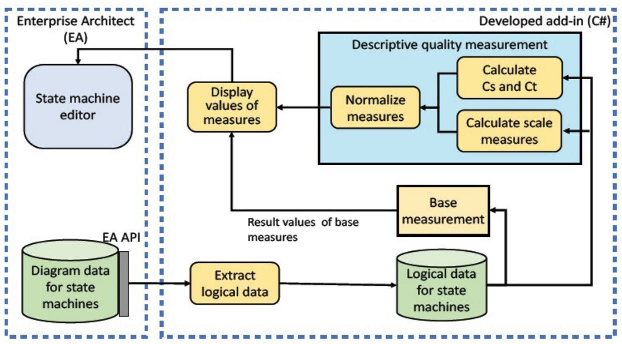
Figure 6 shows that the scatter plot is divided into four regions by the above two criteria. If the criteria are appropriate, data in each region can be interpreted in the followings:

- $Et \leq 1.0$ and $Es \leq 0.4$, good quality
- $Et \leq 1.0$ and $Es > 0.4$, bad quality in hierarchy on states
- $Et > 1.0$ and $Es \leq 0.4$, bad quality in hierarchy on transitions
- Et > 1.0 and Es > 0.4, bad quality in hierarchy on both states and transitions

## Analysis on the proposed measures

Since $Ns \geqq Cs$ and $Nt \geqq Ct$, the proposed method uses $Ns$ and $Nt$ as the denominator for normalization. $Es$ and $Et$ differ in what they can evaluate by the normalization.

$Es = (Cs\text{-}1) / Ns$: Since $Cs$ and $Ns$ are based on the same specifications, this normalization makes $Es$ a numerical value in the range of 0 to 1 regardless of the problem size. If an appropriate criterion can be set for $Es$, it can be used to judge the quality of the hierarchy on states of an individual state machine.


Enterprise Architect (EA)
State machine editor
EA API
Diagram data for state machines
Developed add-in (C#)
Descriptive quality measurement
Calculate Cs and Ct
Normalize measures
Calculate scale measures
Display values of measures
Base measurement
Result values of base measures
Extract logical data
Logical data for state machines
Figure 9: System structure of the design support tool.

$Et = Ct / Nt$: Since $Ct$ is "number of transitions + number of cross-hierarchy transitions", the normalization enables judging whether and how much the decrease in number of transitions or the increase in number of cross-hierarchy transitions is larger by using the criteria of 1.0. This is because the distance between $Et$ and the criterion of 1.0 does not depend on the problem size.

As described above, the proposed method can evaluate the descriptive quality of hierarchical state machines with an appropriate set of criteria for $Es$ and $Et$.

## 7. A Design Tool Using the Proposed Measures

### Requirements for the design support tool

With the aim of supporting better design of hierarchical state machines, we developed a design support tool that implements the two proposed measures. We defined the following requirements for the tool:

R1 The tool shall be able to run seamlessly while editing hierarchical state transition diagrams.

R2 The syntactic correctness of the output is not too strictly required.

R3 The descriptive quality of the target diagram shall be judged pass/fail.

R4 Guidance for the improvement shall be provided.

R5 The descriptive quality shall be able to be evaluated immediately after the improvement.

### Approaches to satisfying the requirements

To satisfy R1 and R5, we have implemented the tool as a C# add-in for Enterprise architect (EA) [6], a UML tool with hierarchical state transition diagram editing capabilities. This allows us to check the descriptive quality of the target diagram during its editing process, to improve it, and to see the effect of the improvement immediately.

As shown in Figure 9, the tool consists of the following six major functional parts.

1. Extract logical data: Logical information (inclusion relations among states and graph structure by states and state transitions) is extracted from the hierarchical state machine data in the EA using the EA API.

   The proposed measures (and the other basic measures) can be computed from this information alone and are therefore insensitive to errors other than inclusion relations and graph structure. This satisfies R2.

2. Calculate $Cs$ and $Ct$: $Cs$ and $Ct$ are calculated based on 5.1 and 5.2, respectively.
3. Calculate scale measures: A non-hierarchical state machine is created to calculate two size measures $Ns$ and $Nt$ from it based on 5.3.
4. Normalize measures: $Es$ and $Et$ are calculated based on equation 4 and 5, respectively.
5. Base measurement: Base measures for states and transitions are calculated.

6. Display values of measures: The results of 5) and 6) are summarized and presented to the designer. Figure 10 shows an example of the information provided by the tool.

## 8. Related Works

There are various factors in the complexity of hierarchical state machines. Beldjehem [7] provides three categories of complexity in hierarchical state machines: computational, psychological, and expressive, and further enhances psychological complexity into problem-specific, cognitive, and structural ones. They recommend that these complexities be treated separately. Our proposed method defines a measure that makes it possible to measure the structural complexity and separates it from the effects of problemspecific one by introducing scale normalization.

There have also been several studies to relate the readability of hierarchical state machines to a set of base measures of their descriptive elements. Miranda et al. [8] experimentally show that there is a correlation between number of actions, number of states, and number of transitions in a hierarchical state machine and how long it takes for readers to understand it. Cruz-Lemus et al. [9] evaluated the effect of abstract states on readers' comprehension in hierarchical state machines by large-scale experiments, whose results show that the use of abstract states improves readability. Fonte et al. [10] propose a prediction formula for the complexity of hierarchical state machines using a linear equation of number of transitions, number of state transitions divided by number of states, and depth of states, and they conducted an experiment to determine its parameters. However, these studies do not separate and assess problem-specific complexities.

Abadi et al. [11] have proposed refactoring patterns for five types of methods: grouping, merging, extraction, pulling, and composition, as methods for improving the description quality of hierarchical state machines. It can be used to improve hierarchical state machines diagnosed as having poor descriptive quality.

## 9. Conclusion and Future Issues

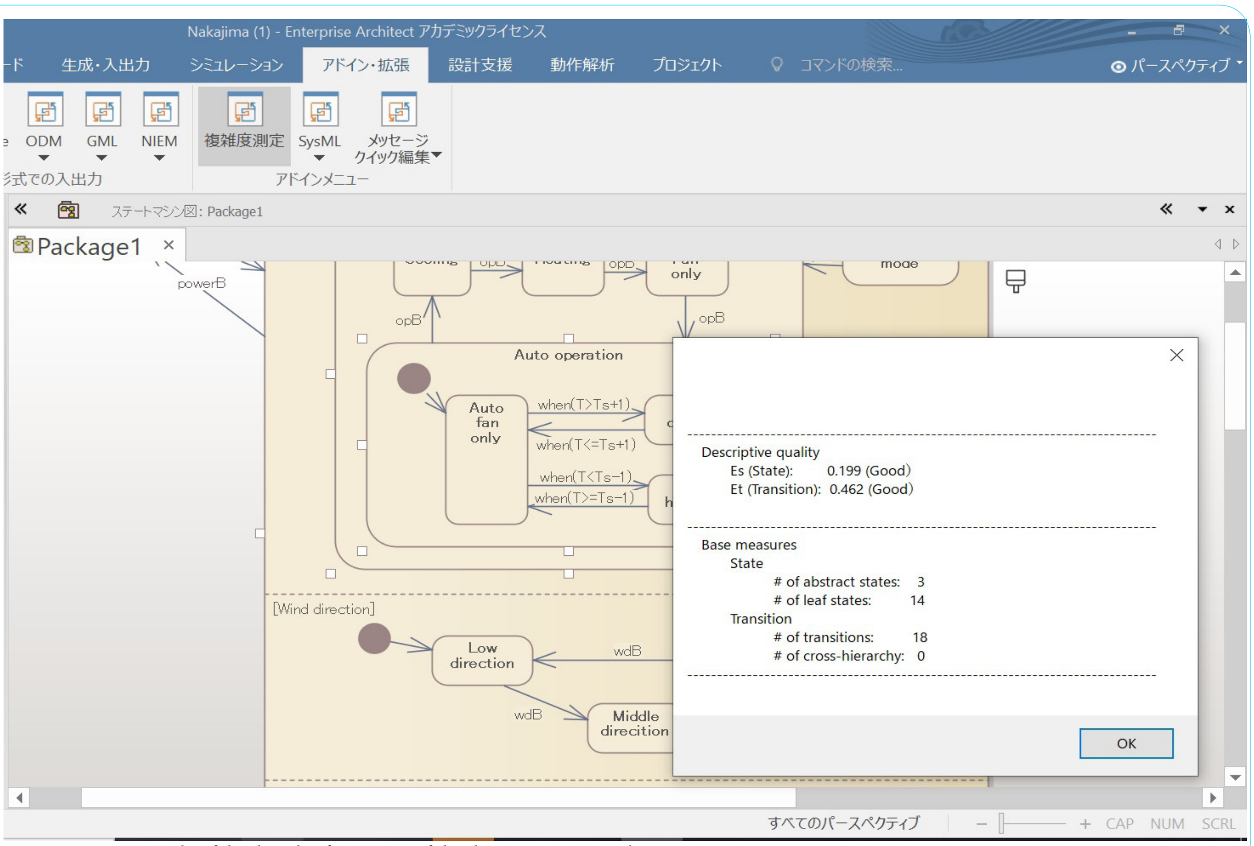
In this paper, we proposed a method and tool to evaluate the descriptive quality of hierarchical state machines. This method defines a set of two measures, which are normalized using complexity measures and scale measures from two viewpoints: state and transition. This enables us to evaluate that of an individual machine independently. The tool support designer to create quality hierarchical state machines.

From descriptive experiments, we found that the proposed method may have a small bias in the distribution of measured values for problems of different types and sizes and certain criteria can be set to evaluate a single hierarchical state machine individually.

As a future task, since the number of samples for each of the problems is not large enough, more samples need to be collected to evaluate the effectiveness of this method and tool, and to determine a set of appropriate criteria for the two measures. It is also necessary to apply this method to many actual developments to show that it is useful for improving the descriptive quality of hierarchical state machines.

## Competing Interests

The authors declare that they have no competing interests.



```
Descriptive quality
    Es (State):        0.199 (Good)
    Et (Transition):  0.462 (Good)

Base measures
    State
        # of abstract states:   3
        # of leaf states:       14
    Transition
        # of transitions:       18
        # of cross-hierarchy:   0

                                        OK
```

Figure 10: An example of displayed information of the design support tool.

## References

1. Harel D (1987) Statecharts: A visual formalism for complex systems, Science of Computer Programming 8: 231-274.
2. OMG® Unified Modeling Language (OMG UML) Version 2.5.1.
3. Genero M, Miranda D, Piattini M (2002) Defining and validating metrics for UML statechart diagrams, Proceedings of QAOOSE 2002.
4. McCabe TJ (1976) A Complexity Measure, IEEE Transactions on Software Engineering 2:308-320.
5. Hall, M.: Complexity Metrics for Hierarchical State Machines, SSBSE 2011, LNCS-6956, pp. 76–81 (2011).
6. Beldjehem M (2013) A granular hierarchical multiview metrics suite for statecharts quality, Advances in Software Engineering 2013: 952178.
7. Enterprise architect: full lifecycle modelling for business, software and system: (referenced 2022-4-29).
8. Miranda D, Genero M, Piattini M (2005) Empirical validation of metrics for UML statechart diagrams, Enterprise Information Systems V, Springer, Dordrecht, pp. 101-108 .
9. Cruz-Lemus JA, Genero M, Manso ME, Piattini M (2005) Evaluating the effect of composite states on the understandability of UML statechart diagrams, International Conference on Model Driven Engineering Languages and Systems, Springer, Berlin, Heidelberg, pp. 113-125.
10. Fonte D, Boas IV, e Azevedo J, ́e Jo ̃ao Peixoto J, Faria P, et al. (2012) Modeling Languages: metrics and assessing tools.
11. Abadi M, Feldman YA (2009) Refactoring of Statecharts, Next Generation Information Technologies and Systems, pp.50-62.