Testing Configurable Component-Based Software
- Configuration Test Modeling and Complexity Analysis

Jerry Gao, Jing Guan, Alex Ma, Chuanqi Tao
San Jose State University, USA
Xiaoying Bai
Tsinghua University, China
David C. Kung
University of Texas at Arlington, USA

Abstract: As the advance of software component technology, engineers encountered different issues and challenges in testing and automation of configurable components and component-based programs. One of them is how to validate configurable components and programs to achieve adequate test criteria and support test automation. This paper uses a test model, known as a semantic tree, to assist engineers to model and analyze diverse composite components and configurable software in terms of configurable environments, organization structures and functions. Based on this model, well-defined test criteria are presented to address the adequate testing issues. In addition, the paper discusses two test complexity evaluation methods for configurable components and software. Furthermore, some case study results are reported to demonstrate the testing complexity of diverse configurations.

KEYWORDS: test modeling and analysis, configurable software testing, test complexity, test criteria, and configuration testing.

1. Introduction

In the last two decades, software component reuse has been a popular concept and approach in software construction. To assure software product quality, engineers need to assure the quality of reused components and programs before deployment. Hence, testing component-based systems has been a very hot research subject in the past decade.

As the advance of component technology, the complexity of components increased from functional component box to frameworks/middleware and configurable/customizable components. Software users (or clients) are allowed to configure, select, and customize components and software based on their functional requirements, desirable environments, and selected organization structures. In the past two decades, many published papers focus on software design, construction and management issues concerning configurable components and programs in production lines [2][4][5][6][7][8]. Although there are numerous published papers discussing how to construct configurable software and components in different aspects, only a few of them focus on testing and automation of configurable components and software [21-26].

In 2003, the authors have pointed that there is an emergent need in testing configurable and customizable components [1][27]. In the past decade, there are numerous published papers addressing issues on testing components and component-based systems. However, only a few of them addressed testing issues and challenges for configurable components, frameworks, and configurable component-based software [21-26]. From 2008 to 2009, we collaborated with a software QA group in SUN Microsystems to set up several test projects in the software testing class at San Jose State University to validate the selected open-source components and middleware developed by SUN Microsystems. Typical examples are SocialSite, SailFin, and Glassfish. After went through the projects, students not only found numerous bugs, but also discovered some other problems in testing component-based software with configurable features and structures. These issues and challenges are summarized below.

- How to identify and present diverse configurations in terms of component/software composition structures, functions, and environments?
- How to define adequate test criteria and evaluate the test coverage for configurable components and software?
- How to analyze and measure the test complexity of configurable components and software?

Today engineers lack well-defined test process, adequate test models and criteria, as well as test automation solutions for configurable components and systems. According to QA and test engineers in the real world, there are the following needs.

- Effective test models to assist test modeling and analysis of components and systems with configurable structures and features.
- Well-defined test criteria and coverage analysis for components with diverse configurations in environments, functions, and structures.
- Cost-effective solutions to evaluate configuration-oriented test complexity and cost.
- Test automation solution to support auto-testing of configurable components and software.

This paper focuses on the first three needs. The paper uses a model-based approach to address testing issues. A new model, known as a semantic tree, is used to assist engineers to perform test modeling and analysis for configurable component-based systems. This model can be used to present diverse component configurations statically or dynamically, including their configurable environments, organizational structures, and functions. Based on the given model, a set of configuration-based test criteria is defined, and a test complexity evaluation method is provided.

The paper has three primary contributions in software component testing. Firstly, it uses a model-based approach to modeling, presenting and analysis of diverse configurations in components. Unlike the existing work, the proposed model allows engineers to present and analyze both static and dynamic configurable structures and features of component-based systems. Secondly, it presents well-defined test criteria based on the proposed model to address configuration-oriented test adequacy. Thirdly, it provides a systematic method to evaluate test complexity of diverse configurations in components and systems.

The paper is structured as follows. The next section discusses the background and related work. Section 3 discusses the essential issues in testing configurable components. Section 4 presents a test model, known as a semantic tree, and related properties and
spanning trees to model the configurable structures for software components. Section 5 discusses a set of test criteria for configuration features in components and software. Finally, Section 6 reports some case study results, and the conclusion remarks and future work are given in Section 7.

2. Related Work

There are numerous published books and papers addressing different topics and issues on component testing in the past decade.

- **Component testing techniques** - For example, the authors in [13] present flow-graph based test model and a technique to support API-based component black-box testing by focusing on a component’s accessible functional sequences. In addition, the paper in [17] discusses testing of state-based dynamic behaviors for a component.

- **Design for component testability** – Roy Freedman [10] discusses how to measure component function testability. Gao et al. in [11] proposed a component testability model to evaluate software components from five different factors. In addition, there are a number of papers discussing how to design for component testability using different approaches.
  - Built-in-test components [9][20], where component tests are coded inside components.
  - Framework-based testable components [14][15], where test-driven handlers are built inside components based on well-specified component APIs.
  - Systematic wrapping for testable components [19], where test-oriented component wrappers are created to support tests based on well-defined API artifacts.

- **Component test adequacy and coverage** – As pointed out in [1][27], most existing test criteria and test methods can be used in component testing to achieve function-oriented test criteria and state-based test adequacy. However, there are some open questions. One of them has been addressed by [12], which is API-based access sequences and related test criteria. The other open issues are relating to component reuse contexts, component interactions, and configurations.

Based on our recent literature survey, there are only a few of papers addressing testing issues and challenges in configurable components. The existing work can be classified into three groups:

- **Testing configurable system constraints using combinatorial interaction testing (CIT)** [24, 25, 26] – CIT is a method to sample configurations of a software system systematically for testing. Many algorithms have been developed to create CIT samples. A general constraint representation and the related solving technique are presented in [26]. It focuses on this problem by examining two highly configurable software systems to quantify the nature of constraints in real systems. CIT can provide an effective way to sample configurations for testing. Cohen et al. in [25] focused on CIT-based test coverage and fault coverage for crossing system configurable parameters and their constraints. Based on CIT, Qu et al. in [21] introduced an approach to regression testing of configurable software using prioritization, which aims at earlier detection of defects.

- **Regression testing of system with configurable features** – For instance, Robinson et al. proposed a firewall method for regression testing of user-configurable software [22]. This paper focused on user-centered tests for system configuration.

- **Component compatibility testing** – Yoon et al. in [23] presented some configuration space models, known as CDG (component dependency graph) and ACDM (annotated component dependency model), to present the relationships among components, their versions, inter-component dependency, and constraints. Moreover, they also provided three test strategies based on the proposed models and experimental results.

Unlike the existing research, this paper provides a configuration model to present three-dimension configuration space for components and systems, which includes configurable environments, compositional architectures, and selective functions. The model is known as a semantic tree model, which can be used to present both static and dynamic configurations for component-based software and its parts. Instead of focusing on configurable system combinational constraints based on parameters, we focus on diverse configurations in environments, architectural structures, and selective functional features. We use a model-based approach to address the testing issues in three-dimension configuration space for component-based systems, including test modeling, test adequacy, and test complexity analysis. Some case studies are reported.

3. Testing Configurable Software

What is a configurable component? It refers to a compostable component, which not only provides a specified contracted interface, domain-specific service functions, and deployment solution with necessary artifacts, but also is equipped with the configuration capability that allows its users to make configuration selections in its environments, functions, and structures statically or dynamically. Similarly, configurable software is a program supported with a capability that allows users to make various configuration decisions statically or dynamically to generate different deployment instances based on their desires. Component-based software usually supports configurations in two different ways: a) static configuration, in which decisions and selections are made statically, and b) dynamic configuration, in which decisions and selections are made during runtime. Based on our recent experience, many modern frameworks and component-based software allow users to make following types of configuration decisions.

- **Environment configuration capability** – which allows users to configure and select different deployment environments.
- **Organization configuration capability** – which allows users to select different organization and composition structures based on available components and building parts.
- **Function configuration capability** – which allows users to select different functional features based on their needs.

Figure 1 presents the configuration test space as a 3-dimension space for configurable software and components, in which the X-axis presents all possible environment configurations, the Y-axis presents all of configurable structures/organizations, and the Z-axis presents all of configurable functions.
If $P_1$ has been tested in a deployment environment $P_E$, then there is no need to validate another configured instance $P_2$ of $P$ on the same deployment environment $P_E$.

**Reality:**
We found that $P_2$ may have problems to perform its quality function services on $P_E$ even though $P_1$ functions well on the same deployment environment $P_E$.

**Myth #2:**
When Component A is a configurable part of a software P, if it has been tested as a part of one instance of P, then there is no need to test it again in another instance of P when it is included as its part.

**Reality:**
We found that as a configurable part, Component A may work well in one deployed instance of P, but it may not work properly inside another deployed instance of P, because both instances may include component A with different composition structures (or relations). This leads to two different reused contexts for Component A. Therefore, it should be tested in both reused contexts.

**Myth #3:**
When a function (F) is a configurable part of a configurable component P, if it has been tested in one of its configured instance $P'$, then there is no need to validate the function F again in another configured instance $P''$.

**Reality:**
We found that a configurable function F in a configurable component P may work well in one configured instance, but may not work properly in another instance of component P although both instances provide function F as one of their service functions. This leads to two different reused contexts for Component A. Therefore, it should be tested in both instances since they are different reused contexts.

4. **Test Modeling for Configurable Software**

Although there are numerous useful models in software testing, very few are suitable to model and present diverse configurations and their mappings in the 3-dimension configuration space for configurable component-based software. As shown in Figure 1, we need a well-defined test model to assist engineers to analyze and present each configurable structure (or compositional architecture) (say CS-i) under a specified configurable environment (say CE-j) as well as the corresponding configuration function (say CF-k), so that model-based test criteria, test generation methods, test complexity analysis techniques can be developed.

This section first discusses an updated test model to serve this purpose. It is known as a semantic tree model, which has been proposed as a test model to address problems and needs in software installation testing [16]. Here, the semantics tree model is used as a basis to model to address the needs in configuration testing of the configurable component-based software in the 3-dimension space.

4.1. **Test Model – A Semantic Tree Model**

Intuitively, a semantic tree is a special tree model for configurable component-based software. Each tree can be used to model and present the configuration elements and their relations in one of three dimensions in the configuration space. The tree nodes present configurable parts (or elements), for example, configurable
components. The links present different semantic relations between

nodes. A semantic tree model can be formally defined as 3-tuple =

\((N, E, R)\), where

- \(N\) is a set of tree nodes. There are three types of nodes: a) a
  single root node, b) intermediate nodes (or parent nodes), and
  c) leaf nodes.
- \(E\) is a set of links between nodes. Each link connects a parent
  node and one of its child nodes in a tree. Each link show a
  part of a semantic relation between a parent node and its child
  nodes.
- \(R\) is a set of relations, and each item in \(R\) has a semantic label
  that presents a semantic relation between a parent node and its
  child nodes. There are four types of semantic relations with
  labels: EOR, AND, SELECT-1, and SELECT-M. Their
detailed semantics are given in Table 1. Figure 3 shows their
notations, where \(P\)-Node is a parent node, and \(C\)-nodes are its
child nodes.

To support the model-based analysis, we introduce a concept of

semantic spanning trees based on the semantic tree model.

### Semantic Spanning Tree:

A semantic spanning tree \(G_{SPT}\) is a sub-tree of a given semantic
tree \(G_{ST}\). Unlike common spanning trees, a semantic spanning tree
\(G_{SPT}\) for \(G_{ST}\) only can be derived based on the following
properties:

- For each parent node (\(N_p\)) with an AND relation in \(G_{SPT}\), it
  must include all of its child nodes and its links.
- For \(N_p\) with an EOR relation in \(G_{SPT}\), it must include only one
  of its child nodes and the corresponding link.
- For \(N_p\) with a SELECT-1 relation in \(G_{SPT}\), it must include
  only one of its child nodes and the corresponding link.
- For \(N_p\) with a SELECT-M relation in \(G_{SPT}\), it must include
  only M child nodes and the related links.

<table>
<thead>
<tr>
<th>Relations</th>
<th>Relation description (where (P) is a parent node, and (C_i) is its child node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOR(P, (&lt;C_1, C_2&gt;))</td>
<td>EOR relation indicates that (P)-Node has two child nodes (C_1) and (C_2). They only can be configured exclusively.</td>
</tr>
<tr>
<td>AND(P, (&lt;C_1, ..., C_n&gt;))</td>
<td>AND relation indicates that (P)-Node must be configured with all of its child nodes (C_1, ..., C_n).</td>
</tr>
<tr>
<td>SELECT-1(P, (&lt;C_1, ..., C_n&gt;))</td>
<td>SELECT-1 relation indicates that (P)-Node must be configured with one of its selective child nodes (C_1, ..., C_n).</td>
</tr>
<tr>
<td>SELECT-M(P, (&lt;C_1, ..., C_n&gt;))</td>
<td>SELECT-M relation indicates that (P)-Node must be configured with (M) selective nodes from its child nodes (C_1, ..., C_n).</td>
</tr>
</tbody>
</table>

Table 1 Different Semantic Relations in a Semantic Tree Model

Figure 3 The Notation of Semantic Relations in a Semantic Tree

As shown in Figure 4(a), a simple semantic tree example is
presented with one root node and three parent nodes (A, B, and C).
Each has one relation. One corresponding semantic spanning trees
are given in Figure 4(b). An algorithm is given in Figure 6 to find
out a semantic spanning tree for a given semantic tree model.

`Algorithm Figure 6: An Algorithm for Finding A Semantic Spanning Tree`

```plaintext
Semantic-Spanning-Tree(GST-Node, NSTART)
{
    if GST-Node is a leaf node, then GST-Node Æ NSTART; // add this leaf node into NSTART
    return
    else add GST-Node into NSTART

    switch (GST-Node’s relation) {
        case “EOR”: pick a GST-Node’s child node (say Ci);
                    Ci Æ NSTART // add into NSTART
                    Add GST-Node’s link to Ci Æ ESTART
                    Semantic-Spanning-Tree(Ci, NSTART);
                    break;
        case “SELECT-1”: pick a GST-Node’s child node (say Ci)
                         Add GST-Node’s link to Ci Æ ESTART
                         Semantic-Spanning-Tree(Ci, NSTART);
                         break;
        case “SELECT-M”: GST-Node’s child nodes Æ NSTART
                         Add all its links to its child nodes Æ ESTART
                         Loop for each child node (say Ci) and do:
                         Semantic-Spanning-Tree(Ci, NSTART);
                         break;
        default “NOT”: pick a GST-Node’s child node (say Ci)
                      Ci Æ NSTART // add into NSTART
                      Add GST-Node’s link to Ci Æ ESTART
                      Semantic-Spanning-Tree(Ci, NSTART);
                      break;
    }
}
```
4.2. Modeling for Configuration Testing in Configurable Component-Based Software

Since 2009, we used two software testing classes and two master project teams to apply the semantic tree model and spanning tree concept onto one in-house-built component-based system and several open-source configurable frameworks and platform servers in San Jose State University. Here, we provide some examples to demonstrate its effectiveness in modeling various environment configurations, diverse configurable organizations, and configuration functions.

4.2.1 Modeling Diverse Configurable Environments

All commercial software and components must be executed in a certain operation environment Figure 5 shows a simple semantic tree example which presents different configurable operating system environments for software. In the real world, we can use this model to consider all required configurable hardware and software elements (or entities) in a product’s operation environment. They include different configuration selections in network protocols, device drivers, diverse operating systems and their versions, multimedia and third-party dependent technologies. As shown in [16], it is very important and necessary to have a model like semantic tree to present diverse configurable environments in software installation testing. Similarly, in a component-based software production line, test engineers also need a semantic tree model to perform test modeling and test complexity of these diverse configuration environments. Since each configurable environment usually requires a set of environment-oriented test scripts to set up so that system function and performance testing can be conducted properly.

4.2.2. Modeling Configurable Structures/Organizations

In a configurable component-based system, the program can be configured by using selective components and parts to form different organizational structures. In this case, software testing must address and cover these diverse structures to make they are adequately covered whenever they are required. In 2009, we have used the semantic tree model to present the different configurable organization structures for a configurable component-based elevator simulation system, which is developed by SJSU students using Java-based component technology. This system provides a set of configurable components, including Floor Panel, Door, User Panel, and Elevator Car components. A user interface is provided to its customer to support a user to select and build diverse elevator system instance based on their need. For example, a user can configure a Door component with two models: a) Single-Door and b) Double Door. Similarly, the user can configure other parts of an elevator system. For example, Figure 7 shows the semantic tree model for the Lift component (known as a ICar) of the elevator system.

4.2.3. Modeling Diverse Configuration Functions

Today most configurable programs (or frameworks) provide rich configuration service functions that allow users to make many configuration decisions to configure different product instances. All of them provide an Administration Console with a graphic user interface to support user’s selections in setting diverse configurations. To test the GUI-based configuration service functions, we can apply many existing black-box test methods, such as decision table, category-partition, and boundary value analysis. However, based on our testing experience, students entered the difficulty in dealing with the configuration-oriented test coverage for the provided configuration service functions; because these methods are not designed to easily address the diverse configuration selections and decisions. They found that using the semantic tree model is more effective to address and model the test problems in software configuration functions in configurable software. Due to the space limitation, we report the detailed finding in our future publications.

4.3. Configuration Model Identification and Generation

It is important to have some systematic way to specify diverse configurations to support test modeling, complexity and coverage analysis. Although there are well-established software analysis and design models, such as UML, they are not suitable to present the diverse configurations in configurable software in terms of environments, organizational structures, and configuration functions. The proposed semantic tree model provides an effective modeling tool to support engineers to perform software configuration analysis and specification for configurable component-based system in the 3-dimensional configuration space. The first approach is a static specification-based approach, in which engineers use the semantic tree model to specify and model the configurations in three steps:

1. Identifying and modeling configuration environments.
2. Identifying and modeling configurable component-based organizational structures (or architectures) based on the customer needs.
3. Identifying and modeling configuration functions, including their sectional options.

Clearly, when the given software supports complicated configurations, this approach becomes tedious. Therefore, the second approach is more dynamic and systematic one, in which some built-in dynamic configuration discovery and tracking capability will be provided in configurable software. With this capability, dynamic configuration decisions in environments, organization structures, and configuration functions can be tracked and analyzed for the purpose of test modeling, test complexity analysis, and test coverage measurement.

5. Configuration Test Criteria and Complexity

To adequately test configurable component-based software \((P)\), engineers must answer the following questions:

- Have we adequately checked \(P\) under each configurable environment?
- Have we adequately tested \(P\) with each of configurable organizational structures?
- Have we adequately tested \(P\) for each of its provided configuration service functions?

To answer these questions, engineers need well-defined test criteria and an effective test coverage analysis solution. This section defines the test criteria based on the semantic tree model given in Section 3. In addition, two configuration test complexity evaluation approaches are presented.
5.1. Test Criteria and Complexity for Configuration Environments

When configurable software (say $P$) can be operated under different configurable environments, it must be validated under diverse configured environments to achieve certain environment test criteria for vendors and customers. For example, if a customer requires $P$ to be functioned properly in a set of specified operation environments, then $P$ must be validated under these environments.

Let’s use the semantic tree model ($G_E$) to present the diverse configurable environments of software $P$. Since each configured operation environment for $P$ can be modeled as a semantic spanning tree, we can define the test criteria for configuration environments of $P$ below.

**Test criterion for Single-Operation-Environment:**
- For any configured operation environment, this test criterion can be achieved only when software $P$ has been tested under this configured environment.

**Test criterion for All-Operation-Environment:**
- This criterion can be achieved only when software $P$ has been validated under each of the configurable environments.

Clearly, setting up these various configurable environments require engineers efforts to prepare and execute the set-up scripts. Using the semantic tree model $G_E$ for $P$, engineers can easily prepare the environment set-up scripts based on its semantic spanning trees in $G_{SGP}$. Hence, the test environment complexity, represented as $T$-complexity($G_E$) of $P$ can be computed below.

$$T\text{-complexity}_E(G_E) = \text{No. of semantic spanning trees in } G_{SGP}$$

This complexity formula enables to engineers to figure out the required number of pre-test scripts for environment configuration and set-up. This will be useful for test planning in test cost and complexity analysis.

5.2. Test Criteria and Complexity for Configurable Organization Structures

For a configurable component-based software $P$, each configurable organizational structure directly corresponds to one of its semantic spanning trees of $G_S$. For a software vendor of $P$, it is very important to validate its instances to cover its diverse organizational structures.

Since each semantic spanning tree in $G_S$ represents one of its configured organization structures, it is necessary for us to understand and define the adequate test criteria for validating its various organizational (compositional) structures. Here, we
provide test criteria for configurable organization structures of \( P \) based on a semantic tree model. Let’s use \( G_S = (N_S, E_S, R_S) \) as a test model to present its configurable organizations. \( G_{SPT} = \{ G_{SPT_i} \mid i = 1,...,n \} \), where \( G_{SPT_i} \) is a semantic spanning tree of \( G_S \). Let \( T_S \) be the test set for \( P \).

**Adequate test criterion for Single-Organization-Structure:**
- For a single organization structure modeled as a semantic spanning \( G_{SPT} \) of \( G_S \), this test criterion is achieved if and only if \( T_S \) of \( P \) has been exercised onto at least one deployed instance \( P_i \) with \( G_{SPT} \) as its organization structure.

**Adequate test criterion for All-Organization-Structures:**
- This test criterion is achieved if and only if \( T_S \) of \( P \) has been exercised onto the deployed instances of \( P \) configured with each spanning tree in \( G_S \).

To support the evaluation of test complexity of diverse configurable organization structures, we provide a hierarchical computation method below. For any parent node \( N_{Pi} \) in \( N_S \) of \( G_S \), its organizational configuration complexity can be computed based on its semantic relation with child nodes. Let \( T\text{-complexity}(N_{Pi}) \) be the configuration complexity for its organization structures. It can be computed below.

- If the node \( N_{Pi} \) has the EOR semantic relation with its child nodes, then its configuration complexity can be computed below.
  \[
  T\text{-complexity}(N_{Pi}) = T\text{-complexity}(N_{C1}) + T\text{-complexity}(N_{C2})
  \]
  Where \( N_{C1} \) and \( N_{C2} \) are the two child nodes of \( N_{Pi} \).

- If the node \( N_{Pi} \) has the AND semantic relation with its child nodes, then its configuration complexity can be computed below.
  \[
  T\text{-complexity}(N_{Pi}) = \prod (T\text{-complexity}(N_{Cj}))
  \]
  Where \( j = 1,...,n \), and \( N_{Cj} \) is a child node of \( N_{Pi} \).

- If the node \( N_{Pi} \) has the SELECT-1 semantic relation with its child nodes, then its configuration complexity can be computed below.
  \[
  T\text{-complexity}(N_{Pi}) = \sum (T\text{-complexity}(N_{Cj}))
  \]
  Where \( j = 1,...,n \), \( n \) is the number of its child nodes, and \( N_{Cj} \) is a child node of \( N_{Pi} \).

- If the node \( N_{Pi} \) has the SELECT-M semantic relation with its child nodes, then its configuration complexity can be computed below.
  \[
  T\text{-complexity}(N_{Pi}) = \sum (T\text{-complexity}(N_{Cj})) \ast n!/(m!(n-m)!) \]
  Where \( j = 1,...,M \), \( N_{Cj} \) is one of \( M \) selected child nodes of \( N_{Pi} \), from its \( N \) child nodes.

Following these formula, engineers can easily implement an automatic solution to compute the test complexity for diverse configurable organization structures in any given configurable component-based software. Figure 7 presents the detailed test complexity of different configurable organization structures for Elevator Controller (ICar), which is a composite component in the elevator simulation system. For example, a Door component with an EOR relation to its two child nodes (Single Door and Double Door) has to be checked for two different settings. Hence, its test complexity is 2. The iDoor component has an AND relation with its three child nodes, including Door component. Its test complexity for different compositions is 2 based on the formula in (3). The test complexity for the Elevator Controller (ICar) is 16.

Similarly, the test complexity computation approach can be used for a semantic tree model to present diverse configuration functions in the elevator simulation system. Figure 8 shows the detailed results of test complexity for testing function configurations for the Elevator Controller (ICar) based on the same approach. As shown in Figure 8, its test complexity is 320.

**6. Case Study Results on Configuration Testing**

In 2010, we used our students in software quality testing class (CMPE287) to conduct a case study, where 10 student groups are required to conduct a term-project to validate the given component-based elevator systems. One of the tasks is to use the semantic tree model to support configuration-based testing, so they are able to understand the testing complexity and challenges in testing configurable component-based software.

In this case study, we focus on the following items:
- Discover the semantic tree model for the selected configurable software to see the effectiveness of the model in presenting different configurations, including environments, organization structures, and configuration functions.
- Identify, understand, and analyze the test complexity of configuration-based testing in configurable component-based systems.

| Table 2. The Semantic Tree for Organization Structures and Its Complexity |
| --- | --- | --- | --- | --- | --- |
| Semantic Tree for Organization Structures | # of Spanning Trees | # of Nodes | # of Leaves | Max Height | # of EOR | # of AND |
| Elevator Semantic Tree | 64 | 54/36 | 53 | 4 | 6 | 12 |
| ICar Semantic Tree | 16 | 33/22 | 32 | 3 | 4 | 7 |

| Table 3. The Semantic Tree for Configuration Functions and Its Complexity |
| --- | --- | --- | --- | --- | --- | --- |
| Semantic Tree for Configuration Functions | # of Spanning Trees | # of Nodes | # of Leaves | # of Links/Max Height | # of EOR | # of AND | # of SELECT-T-1 |
| Elevator Configuration | 640*256 | 47/37 | 46/3 | 4 | 3 | 6 |
| ICar Configuration | 320 | 23/17 | 22/2 | 2 | 1 | 3 |

As shown in Figure 7, the semantic tree model for ICar has 33 nodes. Among of them, there are 22 leaf nodes, and 10 parent nodes. In addition, there are two different types of semantic relations, including EOR and AND. And its test complexity of various configurable organization structures is 16 since the semantic tree has 16 different spanning trees. Table 2 shows the detailed complexity of the semantic tree model of the Elevator Simulation System. Its total test complexity is 64, which presents the total number of different configured structures for the elevator system. Hence, while validating this software, a vendor’s engineers must test its deployed instances to cover its configurable organization structures. In practice, they can achieve the defined adequate test criteria (discussed in Section 5) in an incremental approach. For example, whenever a customer is deployed one instance, its configured system organization structure (or component composition structure) will be recorded. This idea has been implemented in one of our master project. For this system, we
found that we need to develop 64 scripts to set up and cover different organization structures so that the deployed system instance can be tested with certain adequate function test set using the existing test methods. Table 3 shows the related semantic tree model for configuration functions in the system based on the provided system user interface and configuration service functions. Clearly, its complexity is much higher due to more choices are given in the semantic tree. This suggests the configuration testing could be very complicated. More test automation research work for configurable software testing is needed.

7. Conclusions and Future Work

Although there are numerous papers addressing how to construct configurable software and components, only a few papers discussed how to test configuration features and structures of component-based software. This paper uses a model-based approach to discussing the relating issues, challenges, and test process. It applied a semantic tree model as a test model to present and analyze the diverse configuration environments and configurable structures in component-based systems. In addition, a set of adequate test criteria configurations has been defined. Furthermore, some case study results are reported demonstrate its effectiveness and application in test modeling and test complexity analysis. Currently, we are developing a test automation solution to support automatic testing of configurable component-based software. The future extension of this research is to study how to use a model-based approach to addressing regression testing issues and challenges in configurable features and services in SOA applications.

Acknowledgement

This research project was funded by Sun Microsystems in 2009, and Dr. Gao is supported by Tsinghua National Laboratory for Information Science and Technology (TNLIST) in 2010.

References