Framework for component model selection

H. Aris¹  S.S. Salim²

¹Software Engineering Department, College of Information Technology, Universiti Tenaga Nasional, Kajang, Selangor 43000, Malaysia
²Software Engineering Department, Faculty of Computer Science and Information Technology, University of Malaya, Lembah Pantai, Kuala Lumpur 50603, Malaysia
E-mail: hazleen@uniten.edu.my; salwa@um.edu.my

Abstract: Advances in component-oriented software development research have led, among others, to the birth of a large number of component models, which has caused difficulties to the software developers in selecting the suitable component models to be used. To overcome the difficulties, a framework for component model selection is proposed in this study. The framework allows software developer to specify the criteria of the component model that they would like to use and recommends the component model(s) that fits most of the criteria. An application of the framework shows that it has the potential to be used by the software developers in determining the suitable component models.

1 Introduction

Component-oriented software development (COSD) is an approach in software development where (software) applications are being developed using (software) components. COSD promises the benefits of increased reuse, reduced development cost and shorter time to market the applications. In order to successfully fulfilling the promises, COSD relies heavily on component model, which is a specification of standards that governs the construction and composition of (software) components. As such, component model is regarded as the essence of COSD [1], and component developers and component users have to conform to the component model when developing and using the components, respectively, to ensure the success of COSD.

Being the essence that determines the success of COSD, it does not come as a surprise when component model receives enormous attention from researchers in this area. Research advances in component model construction have witnessed the birth of a large number of component models. At least 32 component models have been proposed to date, ranging from domain specific to generic component models [2]. The presence of such a large number of component models has posed a new challenge to the software developers who want to adopt the COSD approach, that is, the difficulty in determining the suitable component model to be used [3]. The difficulty is mainly caused by the differences that exist between the component models, as each component model focuses on the different aspects, such as formal component modelling and quality prediction [4]. Thus, it becomes very difficult for the component models’ comparison to be performed, let alone to determine the suitable ones.

Realising the difficulty, efforts have been made to determine the common features shared by the existing component models, despite the differences that exist between them, with the purpose of understanding the current state of component models and hence, looking for the possible room of improvement of the existing models. Examples of such work include the taxonomy of component models [5], the component model classification framework [3] and the refinement of component model standards and conventions [6]. These works tried to identify the common properties that different component models may have and use them (the common properties) as a basis to classify existing component models.

However, the common properties alone are not sufficient to overcome the difficulty faced by the software developers in determining suitable component models to be used, but they certainly pave the way towards that direction. To enable software developers to make such determination, a process that shows how the common properties are being used that eventually leads to the software developers being able to identify and hence, select suitable component models to be used needs to be defined. Therefore we introduce the term component model selection, to mean the process of identifying and selecting suitable component models to be used by matching the requirements of the software developers with the common properties supported by the component models.

This article therefore describes about our proposed component model selection (CMS) framework, which aims at offering a solution to the software developers in determining the suitable component model to be used in their software development projects. The necessity to have the framework had been justified by Aris and Salim in [2]. The justification was made from a questionnaire survey that was performed to identify the current state of COSD application, and from a literature survey to determine the usage level of component models among the researchers.
Results from both surveys indicated that a CMS framework will be able to give the following contributions:

- exposes the availability of existing component models,
- increases the reusability of existing component models and
- enables a more systematic component model selection.

The CMS framework uses the common properties of component model proposed by other researchers to determine the suitable component model, in particular, the properties derived in [9]. It offers the functionalities that

- allow software developers to specify the requirements of their ‘desired’ component model,
- perform comparison between the existing component models and the desired component model,
- rank the existing component models according to the degree of similarity with the desired component model and
- recommend the existing component models at the highest rank to the software developers as the suitable component models to be used.

The rest of this article is organised as follows. Section 2 describes the related works and their contributions towards our research. Section 3 lists down the issues and challenges that need to be addressed in constructing the CMS framework. Section 4 provides an overview of the framework. Section 5 explains how the selection process is performed using the framework. In Section 6 an instance of the CMS framework is created to show a working example. Finally, Section 7 concludes the article.

2 Related work

CMS is a relatively new research area in COSD, and at the time of writing, there is not a single CMS framework, mechanism or method that is known to have existed. However, quite an extensive number of its counterparts in the field of commercial-off-the-shelf (COTS) have been proposed. Land et al. [7] gathered a total of 17 COTS component selection methods that have been published between 1995 and 2006. These methods can be generally described in terms of the following four processes:

1. preparation,
2. evaluation,
3. selection and
4. (only in some methods) supporting processes.

In the preparation process, potential component candidates are identified, and evaluation criteria as well as comparison method are defined. In the evaluation process, actual data that answer the evaluation criteria are collected and used to perform comparison of the candidates. In the selection process, a decision is made based on the comparison result. Supporting process includes activities such as team formation and documentation.

A more recent work on COTS component selection method, which is the development of a component selection framework for COTS libraries by George et al. [8] that describes each of the above processes in greater detail, was also found. In the framework, a target component, which is a virtual component that is modelled after the evaluation criteria specified by the developer, is first created. It represents the ideal component that fulfils all the evaluation criteria, that is, the functional and non-functional properties specified by the developer. The target component is then compared against each of the existing components in the COTS libraries. Each comparison yields a value called the satisfaction index (SI) value that represents the similarity between the target component and the real component. The framework then selects an existing component that scores the highest SI value. In the case of a tie between the real components, the developer is allowed to specify additional selection criteria to refine the selection.

Coming back to the component model selection, even though a framework for CMS does not exist yet, we found at least two published works that are leading into that direction. These are the work by Lau and Wang [5] on the taxonomy of component models, and the work by Crnkovic et al. [3] on the classification framework for component models.

The work on component models taxonomy was an analysis of existing component models, with the aim to clarify and/or unify the terminology in component-based software engineering. The component models were analysed by categorising them according to the three proposed taxonomies; taxonomy based on components semantics, taxonomy based on components syntax and taxonomy based on components composition. However, it was discovered that only taxonomy based on components composition was meaningful and relevant in achieving the objective of the analysis. Categories in the taxonomy based on components composition was derived from an idealised component life cycle, consisting of three phases; design, deployment and run-time. The categories are

- design without repository,
- design with deposit-only repository,
- deployment with repository and
- design with repository.

Analysis result shows that the existing component models attempted to fulfil the characteristics of an idealised component life cycle, but with varying degrees of success, which opened up rooms for improvement of existing component models. An extended version of the work by the same authors can be found in [1].

In the second work, Crnkovic et al. [3] made an attempt to construct a classification framework for component models. The aim was to identify and quantify the basic principles of a component model. The classification framework is made up of 15 characteristic points that are divided into four dimensions of a component model; life cycle, constructs, extra-functional properties and domains. The classification framework was used to classify a selection of component models. From the classification exercise, it was observed that

1. general component models utilise the client–server style,
2. specialised component models mainly use the pipe and filter style and
3. support for extra-functional properties is rather scarce.

In addition to the above work on component model taxonomy and classification, our earlier work on the refinement of component model standards and conventions [6] and its extended version on the component model properties [9] also aim at determining the common characteristics of component model that can be used as a basis to understand the current state of existing component models. In our work, the common characteristics called component model properties are derived by reviewing a
number of component model definitions and extracting their commonalities, which are regarded as the ‘essence’ of a component model. The properties are categorised under interface, contract and composition. A comparative study performed on a selection of existing component models using the properties reveals that some properties are well supported by the existing component models and some are less supported, which indicates the extent to which they adhere to the component model definition.

Land et al.’s work on the COTS selection best practices above provides a strong foundation in determining the processes that constitute a CMS framework. As it reviews quite an extensive number of COTS selection methods and approaches, we are convinced that the common processes that they have identified are representative of any selection method. Therefore the four processes that are commonly present in the existing COTS component selection methods are also applicable to the CMS selection framework that we are constructing.

On the details of how each process should be carried out, the selection framework for COTS libraries developed by George et al. makes a good source of information that provides sufficient description of each process, which enables them to be modified accordingly and applied in the CMS framework construction. Furthermore, the authors of the framework have already considered the current selection approaches and their limits in proposing the framework. Therefore it serves as our main reference in constructing the CMS framework. However, certain details under each process, such as the criteria to be used in evaluating the component models, have to be customised to suit the characteristics of a component model. For this purpose, the characteristics of an idealised component life cycle introduced by Lau and Wang, the characteristics points of a component model proposed by Crnkovic et al., and the list of component model properties from our previous work are the possible candidates for the evaluation criteria for the component models.

Land et al. and George et al. are working on the selection process at the component level, while ours is at one level higher, the component model level. Lau and Wang, Crnkovic et al. and Aris and Salim are working on the derivation of the common characteristics of component models that can serve as a basis for their comparison. In this article, we extend these work further by incorporating the characteristics into a framework that actually performs the comparison. We do so by addressing the issues and challenges identified in constructing the CMS framework as described in Section 3.

### 3 Issues and challenges in component model selection

As discussed in the previous section, the four processes presented by Land et al. [7] that are found to be commonly present in existing COTS selection methods are expected to also become the processes in a CMS framework. However, since the fourth process does not present in all methods and it is only a ‘supporting’ rather than a core process, the discussion that follows are only focusing on the first three processes; preparation, evaluation and selection. The purpose of the discussion is to explore each process in order to identify the issues that need to be addressed in the construction of the CMS framework.

Preparation process identifies potential component candidates, defines the evaluation criteria and defines the comparison method. The evaluation criteria are related to the requirements of the system to be developed and defined with evaluation attributes that are used as metrics. The evaluation process collects actual data (from the component candidates) that answers the evaluation criteria, which are used to perform the comparisons. In the selection process, a decision is made depending on the result obtained from the comparison.

In a related work on a literature review and classification of enterprise software selection approaches, Sen et al. [10] had also listed the following as the major phases that are common across the 43 software selection approaches under their study:

1. determination of selection criteria,
2. prioritisation of the selection criteria and
3. evaluation of candidate software products and final purchase decision.

Similarities can be seen in the two works described above, with regard to the processes that constitute a selection approach and the items that need to be addressed under each process. In the above paragraphs, these items have been italicised. It is believed that they represent the core elements of a selection approach. Therefore we summarise them as the issues that need to be addressed in the construction of a CMS framework, as follows.

#### 3.1 Selection criteria

Selection criteria or evaluation criteria are the characteristics or features or a component model from the software developers’ point of view. These criteria determine which component model will be selected. To enable a selection to be made, the CMS framework should provide a means for the software developers to specify the selection criteria of interest and prioritise them if necessary. The questions to be answered here, that is, the challenges, are

1. What would be the selection criteria and how to determine them?
2. What is the means for software developer to specify the desired criteria?
3. What is the mechanism to be used to prioritise the selected criteria?

#### 3.2 Metric

For certainty and reliability, each component models comparison performed by the CMS framework has to produce a result that is measurable. In order for the result to be measurable, metric needs to be defined for each of the selection criteria of the component model. The use of metrics that are associated to the selection criteria is able to indicate the extent to which a chosen selection criteria is fulfilled by a component model. The challenge here is

1. What is the metric to be used to measure each selection criteria?

#### 3.3 Representation of component models

Component models that exist today originate from various different sources. As a result, formal description used for
each component model differs from one another. To enable comparison between the existing component models, a standard description format has to be used where all component models are ‘rerepresented’ using the same format. Therefore the challenge here is

1. What is the format to be used to describe the component models?

3.4 Comparison method

Comparison is the process of inspecting each available candidate component model to determine the ones that fulfil the specified selection criteria. If more than one component models fulfil the criteria, the component models have to be ranked to indicate the ones that best fulfil the criteria. As the criteria selected by the software developers may be more than one, the use of multi-criteria decision making (MCDM) method is envisaged. Furthermore, if there is a tie between two component models or more, mechanism has to be devised that allows for further specification of selection criteria to refine the comparison. Therefore the challenge here is

1. What is the MCDM method to be used in performing the comparison?

4 Framework overview

An overview of CMS framework is illustrated in Fig. 1. Rectangles in the figure represent the entities in the framework and ovals represent the processes performed by the framework. The arrows indicate the flow. Software developers specify the selection criteria and in return, the CMS framework presents the recommended component model that is deemed suitable based on the specified criteria. Also shown in Fig. 1 are the issues to be addressed in the construction of the framework. Each issue is written next to the entity or process of concern. In Sections 5 and 6, each issue is elaborated in detail and the approach taken to answer each challenge is explained.

5 Framework construction

In this section, we describe the process of constructing the CMS framework. We do so by first, elaborating on each issue in the CMS framework construction that was highlighted in Section 3 above and then explaining our approach in overcoming the challenges posed by each issue.

5.1 Selection criteria

As can be seen from Fig. 1 above, selection criteria act as the ‘interface’ between the software developers and the CMS framework. The processes in the CMS framework are triggered by the selection made by the software developer.

In COTS component selection framework, functional requirements of an application to be developed, such as ‘download’ function or ‘login’ function, are usually directly taken as the selection criteria to determine suitable components [8, 11, 12]. In addition, non-functional requirements are also included as part of the selection criteria that serve as indicators to the quality of the chosen components.

The role of the selection criteria in the CMS framework is the same as the role of the selection criteria in a COTS component selection framework, in the sense that, when selected, they trigger the processes in the framework. They are also the factors that determine which component model (resp. component) will be chosen. However, in the CMS framework, functional and non-functional requirements of the application to be developed are not suitable to be used as the selection criteria, because a component model is of higher abstraction level, that is, meta-component level, and therefore it does not contain such information.

To determine the list of selection criteria for component model, a combination of the following research methods was used.

1. Literature review and analysis.
2. Survey through questionnaire distribution.

5.1.1 Literature review and analysis: An exhaustive literature search was performed to find researches that work on component model quality and measurement. Nevertheless, as
The closest quality model that can be used to assess the component model is the one presented in Mahmood et al. [13], which are shown in Table 1.

Since the metrics are meant for the measurement of component-based systems, their applicability for measuring component model requires further scrutiny. To do so, each metric is traced back to their original resources in [14, 15] to obtain more details, because there was no elaboration of each metric in Mahmood et al. However, the explanation on each metric provided in [15] is so brief that their suitability to be used for component model cannot be determined. On the other hand, the metrics defined in [14] were found to be COTS-component oriented and therefore are not applicable for component model owing to the different abstraction levels.

A subsequent publication by Gill and Grover [16], was found that elaborates on interface complexity, which emphasises on the need to characterise the interface of a component to measure its complexity. The suggested characteristics are signature, constraints, packaging and non-functional properties. Interface characterisation is relevant and applicable to component model. As also explained in [6], component model interface can also be characterised into signature and behaviour, enabling its interface complexity to be measured. Therefore interface complexity is included as one of the selection criteria of the CMS framework. The rest of the metrics in the table are not quite relevant to component models and are not considered further.

From the discussion on component model properties presented in [9], it is learned that different component models support different types of composition. There are three types of components composition; component–component, component subassembly and component-framework compositions. Component–component composition is the composition between two components of the same level. Component subassembly is the composition between a component (subcomponent) and its parent component (super component). Component–framework composition is the composition between a component and its framework. Some component models support all of the three types of composition, some support two and some support only one type of composition. As such, the number of types of composition supported by a component model might be of interest to the software developer in deciding which component model to be used. It is therefore a valid factor in component model selection, and hence, being included as another possible selection criterion in the CMS framework.

5.1.2 Questionnaire survey: In our previous work that surveyed about the component models usage in the research and industry fields [2], it was found that software developers from both fields tend to use component models that are used most often by others, although many other component models also exist. From the findings, it can be seen that the history of a component model also contributes to the decision made by the software developers. Therefore we include the track record of a component model as one of the selection criteria in the CMS framework.

Finally, we also use the other results from the same survey to determine the remaining possible selection criteria for the CMS framework. The survey, among others, gathered information about the factors that influence the software developers in deciding whether to apply COSD approach or not in their software development projects. Among the factors were found to be influencing are tools support and cost, which are included as the selection criteria for the CMS framework.

5.2 Metric

In order to determine whether a chosen selection criterion is fulfilled by a particular candidate component model or not, or to determine the extent to which the selection criterion is fulfilled, metric is associated to each selection criterion. A metric defines two things [17]

1. the measurement method and
2. the measurement scale

of the selection criterion. A selection criterion can have one or more metrics associated with it. Measuring a selection criterion is therefore a process of assigning a value to the criterion according to the type of metric that is associated with it.

To the best of our knowledge, for component model, neither metrics nor quality model has been found published so far, even though research work in this direction can be seen [1, 3, 9]. The closest quality model that can be used to

<table>
<thead>
<tr>
<th>Quality attribute</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>complexity</td>
<td>interface</td>
<td>an estimate of the complexity of interfaces</td>
</tr>
<tr>
<td></td>
<td>integration</td>
<td>measure of effort required in the integration process of the component-based system</td>
</tr>
<tr>
<td>portability</td>
<td>semantic</td>
<td>measure of complexity of the relationship of components to a given language</td>
</tr>
<tr>
<td></td>
<td>adaptability</td>
<td>opportunity to adoption to different environment</td>
</tr>
<tr>
<td></td>
<td>replaceability</td>
<td>opportunity and effort of using components in place of other components</td>
</tr>
<tr>
<td></td>
<td>portability compliance</td>
<td>adherence to application of portability related standard</td>
</tr>
<tr>
<td>reliability</td>
<td>maturity</td>
<td>capacity to avoid failure as a results of faults in the component-based system</td>
</tr>
<tr>
<td></td>
<td>fault tolerance</td>
<td>ability to maintain a specified level of performance in case of faults</td>
</tr>
<tr>
<td>functionallity</td>
<td>recoverability</td>
<td>capacity to re-establish level of performance after the faults</td>
</tr>
<tr>
<td>maintainability</td>
<td>suitability</td>
<td>presence of set of functions for specified tasks</td>
</tr>
<tr>
<td></td>
<td>accuracy</td>
<td>provision of agreed results or effects</td>
</tr>
<tr>
<td></td>
<td>interoperability</td>
<td>capability of the component-based system to interact with specified system</td>
</tr>
<tr>
<td></td>
<td>analysability</td>
<td>identification of deficiencies, failure causes, components to be modified</td>
</tr>
<tr>
<td></td>
<td>changeability</td>
<td>capability to enable a specified evolution to be implemented</td>
</tr>
<tr>
<td></td>
<td>stability</td>
<td>capability to avoid unexpected effects from evolution</td>
</tr>
<tr>
<td></td>
<td>testability</td>
<td>capability to enable validating the evolving component-based system</td>
</tr>
</tbody>
</table>
measure the selection criteria of the CMS framework is the Software Component Quality Model (CQM) defined in Alvaro et al. [17]. The CQM is modified from the ISO/IEC 9126 generic software quality model by adding, renaming and removing some of the subcharacteristics of the generic quality model. Table 2 shows an excerpt of the CQM with only relevant characteristics and subcharacteristics included. The selection criteria derived in Subsection 5.1 above are equivalent to the (quality) attributes of the CQM.

In the following five subsections, the metric used for each selection criterion of the CMS framework are described. The order of the description, however, does not imply the order of importance or priority of the selection criteria.

5.2.1 Metric for interface complexity: Interface complexity measures the level of complexity of the component model interface. This selection criterion allows software developer to specify the level of complexity of a component model interface that they would like to have. In principle, the less complicated an interface is, the better because complicated interface complicates testing, debugging and maintenance [15]. To measure the complexity of the interface of a component model, the component model properties for interface derived in [9] are used. These properties are input specifications, output specifications, precondition, postcondition and invariant. The presence of a property is represented by a Boolean ‘1’ and the absence is represented by a Boolean value ‘0’.

5.2.2 Metric for composition type: Composition type refers to the types of composition supported by a component model. In general, components are being composed to produce software applications. However, the types of composition supported by the component model differ. Some component models allow composition between components at the same abstraction level only while others allow components to have subcomponents. As explained in [9], there are three possible types of composition that a component model can support, ranging from simple composition to a more complicated component-framework composition. We use a scale of 1–3 to indicate the types of composition supported by a component model with 1 being the simplest, that is, only simple composition is supported.

5.2.3 Metric for tool support: Tool support refers to the availability of computer-aided software engineering tools to support the development of software applications using the component model. The tools may be produced by the component model developers themselves or by a third party who is using the component model. Availability of tools that can be used to develop and compose components, for example, may greatly affect the software developer in selecting a particular component model to be used. The measurement method chosen to measure tool support is the availability of tool to support the development of software applications using the component model. Since only the presence of tool is measured, the scale used is the same as the scale used for interface complexity.

5.2.4 Metric for track record: Past performance of a component model is also a factor that influences the chances of a particular component model to be selected. Track record is referring to the usage history of a component model. As discovered by Aris and Salim [18], some component models are being used frequently by (third-party) software developers whereas others are hardly used at all once introduced. Basically (but not necessarily), a more frequently used component model will be more preferred compared to a less frequently used component model. One of the measurement methods that can be associated with the track record of a component model is how often it is being applied in software development, that is, frequency of usage. The scale takes the actual frequency value of the component model.

5.2.5 Metric for cost: Cost is also found to be one of the influential factors for the software developers in deciding whether or not they will use COSD in their software development [9]. Existing component models come in various forms; freeware, shareware or sold in packages. Software developer would want to specify the kind of component models that they can afford depending on their financial constraints.

Cost can be measured in terms of the amount of money required to obtain a component model and use it. Some component models are freely available for download while others have price tags. For the measurement method, three possible types are identified for to measure the cost; freeware, shareware and fully paid. Therefore the scale used ranges from 1 to 3 with 1 to mean fully paid component model. Table 3 is the extended version of Table 2 above, which includes the metric used to measure the selection criteria.

The measurement methods and scales defined for the metrics above may not be concrete enough to measure the achievability of their respective selection criteria. As

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute (selection criteria)</th>
</tr>
</thead>
<tbody>
<tr>
<td>functionality</td>
<td>suitability</td>
<td>interface complexity</td>
</tr>
<tr>
<td>usability</td>
<td>learnability</td>
<td>tool support</td>
</tr>
<tr>
<td>reliability</td>
<td>maturity</td>
<td>track record</td>
</tr>
<tr>
<td>marketability</td>
<td>affordability</td>
<td>cost</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Subcharacteristic</th>
<th>Attribute</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>interface complexity properties implemented</td>
<td>0, 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>types of composition supported</td>
<td>1–3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>frequency</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>availability</td>
<td>0, 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>price</td>
<td>1–3</td>
</tr>
</tbody>
</table>

Table 2  Selection criteria as attributes of the CQM

Table 3  Metric (method and scale) for the selection criteria
mentioned before, it is not part of this research to rigorously define metrics or quality model for component model. The definitions of the metrics above are simplified as their purpose here is just to demonstrate their role in the CMS framework. Since the definitions of the metrics are simplified, shortcomings are therefore anticipated. Furthermore, quality model and metrics for component models is a broad research topic in itself.

5.3 Representation of component models

Subsections 5.1 and 5.2 discussed about the selection criteria and the metrics used to measure them. These are the information that need to be gathered from each candidate component model before any comparison can be performed. In this subsection, the description format, that is, how these information is represented, is explained. The need for a common description format is obvious, because component models that exist today originate from various different sources and each (component model) developer use their own format in representing the component models.

To enable component model comparison, and hence selection, to be performed by the CMS framework, component models that come in different formats have to be re-represented using a same description format. To achieve this, the unified modelling language (UML) class diagram is used where all the required information gathered from each candidate component model is represented by a UML class. Fig. 2 shows the description format used to represent the component model. Details of the description format are given in the following subsections.

5.3.1 Architectural artefact: In the middle of the diagram in Fig. 2 is the class Artefact, which is written in italic font to denote an abstract class. An abstract class is not instantiated and serves to encapsulate its subclasses. Three artefacts have been identified in the description format; component model, interface and contract. Each is represented by ComponentModel, Interface and Contract classes, respectively, that inherit from the Artefact class. The Artefact class has an attribute, weight, which carries the priority value of each artefact as assigned by the software developer.

An interface consists of signature and behaviour specifications, represented by Signature class and Contract class, respectively. A component model has at least one interface, and an interface may have a number of input and output specifications. Input specification is represented by Input class and output specification by Output class. A behaviour specification is made up of precondition, postcondition and invariant, represented, respectively, by PreCond, PostCond and Inv classes.

5.3.2 Information associated with artefact: As can be seen from Fig. 2, class Artefact is associated with CompositionType class. CompositionType class describes the types of composition that a component model supports. A component model supports at least one type of composition and at most three types of composition as discovered by [9]; simple composition, component sub-assembly and component–framework composition. Simple composition and component sub-assembly are represented by SimpType class and SubCompType class, respectively. Both classes inherit from CCType class that represents component–component composition. Component–framework composition is represented by CFType class. Any other information that can help in making the selection can be included in the description format and is represented by the Info class.

![Fig. 2 Description format of the component model](image-url)
5.3.3 Non-functional property: Subsection 5.3.1 describes the classes that are responsible for the selection criteria that fall under functional characteristic, which are interface complexity and composition type. The rest of the selection criteria fall under reliability, usability and marketability characteristics, which are collectively termed as non-functional property (NFP) characteristics. NFP is represented by the NFP class in Fig. 2. An NFP is associated to one or more evaluation attributes, represented by the class QA in Fig. 2.

Class Metric represents the metric associated with each evaluation attribute. Since a metric can be numeric or ordinal, two other classes that inherit from the class Metric are defined in the description format; the Ordinal class and Numeric class, respectively. The domain of an ordinal metric is a subset of positive real numbers including integers and per cent values. A numeric metric has a supplementary attribute called direction. Direction allows for the interpretation of a metric result. An increasing direction means that the higher the metric result value, the better the quality of the corresponding component model is. Conversely, a decreasing direction means that the lower the metric result value, the better the quality of the corresponding component model is. Time taken to execute an operation is an example of a numeric metric of decreasing direction.

The domain of an ordinal metric is a finite and totally ordered set. An ordinal metric also has one supplement attribute called hierarchy. It gathers and ranks all the metric possible values by associating a key (usually a number value) to each one of them. The key defines the total order relation on the metrics domain. If a value is better than another, its rank is strictly superior to the other one’s rank in the associated hierarchy. Likert scale is an example of an ordinal metric.

6 Comparison method

Comparison method, the last issue to be addressed, is concerned with the process of comparing each candidate component model in the repository with a target component model. The target component model is created based on the selection criteria determined by the software developers. Thus, it represents the ‘ideal’ component model that fulfils all of the criteria specified by the software developer. The CMS framework is then responsible to find the closest, if not exact, match from the component model repository and recommends it to the software developer.

Fig. 3 shows the first level algorithm for the comparison method. The target component model is created once the selection criteria are chosen by the software developer. The target component model is then compared with each of the available candidate component model that has been re-represented using the description format in Fig. 2. The comparison process repeats for as many times as the number of available candidate component models. Results from each comparison are temporarily stored. When all available candidate component models have been compared with, the results of the comparison are analysed to determine the candidate component model that is deemed as most suitable. This component model will be recommended to the software developer.

6.1 Component model’s information

To enable the comparisons, first of all, candidate component models in the repository have to be redescribed using a common format. For this purpose, the description format presented in Subsection 5.3 is used. Looking from the structural point of view, it can be seen that the description format is hierarchical in nature. Information that need to be retrieved from each candidate component model are those at the leaf nodes of the description format. These information, which are shown shaded in Fig. 4, are

1. specification of input (Input),
2. specification of output (Output),
3. artefact information (Info),
4. precondition (PreCond),
5. postcondition (PostCond),
6. invariant (Inv),
7. simple composition (SimpType),
8. component subassembly (SubCompType) and
9. component-framework composition (CFType).

On top of the information listed above, each component model’s information on track record, cost and support tool availability are also recorded. These information are examples of non-functional properties used to measure the quality of a component model. With respect to the description format above, they are the instances of the NFP class.

6.2 Satisfaction index

To determine the closest match, a form of MCDM technique called satisfaction index (SI) [8] is used. SI allows us to measure to what extent a candidate component model matches the target component model. To obtain the SI value, index function is used. Index($E_1$, $E_0$) compares values from two ‘comparable’ elements $E_1$ and $E_0$. Two elements are comparable if they are of the same type. For that reason, each node is assigned a type function, Type, to check element’s type. Examples of types are Interface, Contract and CompositionType. Index function is formally defined by (1) below [8].

$$
\text{Index}(E_n, E_0) = \begin{cases} 
0 & \text{if Type}(E_n) \neq \text{Type}(E_0) \\
-\text{Comp}(E_n, E_0) & \text{if } E_0 \text{ is a leaf node} \\
\sum((\text{Weight}(e_0) \cdot \text{MAX}(\text{Index}(e_0, e_n) | e_n \in E_0)) | e_0 \in E_0)) & \text{if } E_0 \text{ is an inner node} 
\end{cases}
$$

From the equation, it can be seen that only two elements of the same type can be compared. Otherwise, the SI between them will return 0. If the elements to be compared are inner nodes, that is, having subnodes, SI value is the cumulative sum of the SI values of their subnodes multiplied by their respective weight. If there are more than one subnode which are comparable to the subnode of the target element (belonging to the target component model), the maximum (MAX) SI value is chosen. The maximum value for the SI is 1. SI value equals to 1 means that the candidate component model completely matches the target component model.

6.3 Weight function

In (1), a weighting function is also applied. Weighting function enables weight to be assigned to each node. Weighting function Weight($E$) allows each node $E$ to be assigned a numeric value, which gives its importance relative to other nodes. Thus, Weight($E$) returns the weight value assigned to node $E$. For every type of node, ‘weighting by distribution’ is used, which means that the weight assigned to each node is distributed such that the total weight of all nodes at the same level is 1.

6.4 Comparison function

In (1), there is also a comparison function Comp($E_1$, $E_0$) that compares two elements, $E_1$ and $E_0$, where both $E_1$ and $E_0$ are leaf nodes. In this case, the SI value is actually the value returned by this comparison function. In the case where $E_1$ and $E_0$ are inner nodes, Index($E_1$, $E_0$) function is recursively called until the leaf nodes are found and the comparison function is performed on the leaf nodes. Comparison function is defined differently for different elements.
depending on their measurable unit, which are described below.

### 6.4.1 Comparison function between NFPs:

Comparison function $\text{Comp}(P_1, P_0)$ between an NFP, $P_1$, belonging to a candidate artefact $A_1$ and a NFP $P_0$ belonging to the target artefact $A_0$ is defined as follow [8] (see (2))

$$\text{Comp}(P_n, P_0) = \begin{cases} 
\text{MIN}(\text{result}_{P_n}, \text{result}_{P_0}) & \text{if } P_n \text{ and } P_0 \text{ do not measure the same property} \\
\text{Comp}_{\text{dec}}(P_n, P_0) & \text{if } M \text{ is numeric with an increasing direction} \\
\text{Comp}_{\text{inc}}(P_n, P_0) & \text{if } M \text{ is numeric with a decreasing direction} \\
\text{Comp}_{\text{bool}}(P_n, P_0) & \text{if } M \text{ is boolean} \\
\text{Comp}_{\text{ord}}(P_n, P_0) & \text{if } M \text{ is ordinal, and if rank(\text{result}_{P_0}) > 0} \\
1 & \text{if } M \text{ is ordinal, and if rank(\text{result}_{P_0}) = 0} 
\end{cases}$$

(2)

First of all, $P_1$ and $P_0$ are comparable only if they measure the same quality attribute. Otherwise, $\text{Comp}(P_1, P_0)$ will return 0. As described in Section 5.2, metrics used can be numeric or ordinal. If the metric $M$ used to measure an attribute is numeric with an increasing direction, the comparison function returns the smaller value between the similarity measure between the metric result of $P_1$ and the metric result of $P_0$ in the corresponding direction, that is, increasing or decreasing.

On the other hand, if the metric used is ordinal, the comparison function will return the smaller value between the ratio of $P_1$’s rank and $P_2$’s rank and 1. In this way, the largest possible value returned by the comparison function is 1, to mean that the candidate component model fits exactly the target component model with respect to the compared quality attributes $P_1$ and $P_0$.

### 6.4.2 Comparison function between interface types:

Interface type is used to denote the complexity of a component model interface. Usually, a component model offers two types of interfaces, requires interface and provides interface [19]. Therefore a component model is considered to have moderate interface complexity if it implements these two interface types. If the candidate component model $E_i$ implements at least all types required by the target component model $E_0$, $\text{Comp}(E_1, E_0)$ returns 1.

### 6.4.3 Comparison function between other elements:

Other than the NFPs and interface information, comparison function between pair of other elements (nodes) only detects the presence of those elements in the candidate component models. ‘True’ indicates that the element is present and ‘false’ indicates otherwise. Thus, $\text{Comp}(I_1, I_0)$ measures the similarity between two elements, $I_1$ and $I_0$, with regard to their presence, that is, whether or not element $I_1$ exists in the candidate component model in the same way element $I_0$ exists in the target component model. $\text{Comp}(I_1, I_0)$ function returns 1 if $I_1$ is also present and returns 0 otherwise.

### 7 Example of CMS framework application

In this section, we create a scenario that instantiates a description format for a target component model based on the general description format in Fig. 2.

#### 7.1 Target component model

To facilitate the process of determining the desired component model criteria, the CMS framework provides a list of component model criteria that the software developers can choose from. In this scenario, it is assumed that the software developer has chosen the selection criteria shown in Table 4 with their respective weight determined by the software developer. Fig. 5 is the instance of the format description in Fig. 2 that is produced based on the selected criteria in Table 4.

Instances of Interface, Signature, Input, Output, Contract, PreCond, PostCond and Inv classes are created as a result of the interface complexity being chosen as one of the selection criteria. NFP classes are instantiated as a result of the second criterion in Table 4. Interface and NFP are given the weight values of 0.4, that is, 40% and 0.6 according to the assignment made in Table 4. Instances of Signature and Contract classes are equally weighted at 0.5. Similarly, the weight for instances of Input, Output, PreCond, PostCond and Inv are equally distributed. Weight for frequency object is weighted at 1 since it is the only node in its level.

To indicate moderate interface complexity, instance of Inv class is set to ‘false’ and the value for frequency object, an instance of QA class is set to 3 to indicate good track record.

#### 7.2 Database of candidate component models

Before any comparisons can be made between the target component model in Fig. 5 and candidate component models, information from each component model has to be retrieved and modelled using the same description format. Fig. 6 shows the description for UML component model using the format in Fig. 2. For each candidate component model, complete description format is modelled, that is, all

<table>
<thead>
<tr>
<th>Selected criteria</th>
<th>Level</th>
<th>Assigned weight, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface complexity</td>
<td>moderate</td>
<td>40</td>
</tr>
<tr>
<td>track record</td>
<td>good</td>
<td>60</td>
</tr>
</tbody>
</table>

![Table 4 Example of selected criteria combination](image-url)
nodes are included. However, when comparisons are performed, only relevant nodes are being compared. Note that in Fig. 6, the weight of each node is omitted. Weight is not relevant here, as explained by (1). It is only meant for the target component model. As such, this is not an instantiation of the description format in Fig. 2, but is described in a similar manner to determine information that need to be captured and stored in the candidate component models database. Information on the leaf nodes are those that need to be stored in the database for comparison purpose. Information that need to be captured are shown shaded in Fig. 6.

Table 5 shows the information retrieved from six candidate component models in the repository. The component models are chosen from the top three and lowest three of the most frequently used component models presented in [2]. Thus, they may not sufficiently represent the existing component models to date. We include them here for the purpose of demonstrating how the framework works.

### 7.3 Comparing component models

Comparison process follows the formula explained in Section 6.2. Based on the target component model description in Fig. 5, only required information of the component models in the repository will be compared with. The leaf nodes involved in the comparison are Input, Output, Precondition, Postcondition, Invariant and Frequency. The comparison process consecutively compares each node in the target component model with their counterparts in candidate component model. Table 6 shows the result of the comparison.

From the table, it can be seen that SOFA and Fractal component models score the highest SI value with respect to the target component model specified in Fig. 5. Therefore both component models will be recommended to the software developer. On the other hand, in the case of a tie like this, software developer can be allowed to specify further selection criteria so that the new SI for both
Apart from demonstrating the application of the framework as a means to evaluate it, a focus group evaluation with software developers and practitioners as the participants had also been conducted, involving ten participants with the software development background. Among others, two quantitative data pertaining to the feasibility of the CMS framework were obtained in the form of questionnaire distribution towards the end of the focus group discussion session. The two questions are shown in Table 7 above. The participants were asked to rank the questions using the scale of 1–5 with 1 means strongly agree and 5 means strongly disagree.

From the results shown in Table 7, it can be seen that majority of the software developers rate the practicality of the CMS framework as moderate. Majority of them are also willing to adopt the CMS framework in their software development projects if needed. With standard deviation values of 0.632 and 0.707, respectively, it can be said that the results are convergent and unanimous.

Other than the feasibility, the focus group session also discussed other aspects of the CMS framework including the qualitative evaluation of the processes that constitute the framework. One of the issues brought up by a participant is concerning the process of extracting the relevant information of the component model to be included in the component model repository of the CMS framework. Firstly, on whether the process will be done manually or automatically and secondly on the software developers have to extract and enter the information themselves. Automation is one possible solution to this. However, automatic extraction of component models information still has a very long way to go as it requires the documentation to be presented in certain specified format before information extraction can take place. As mentioned earlier at the beginning of this paper, there are about 32 component models existing to date. Therefore manual extraction of information, although tedious and time consuming, is still possible.

Other constructive comments were also given and discussed during the session. However, detail discussion of the focus group evaluation session is beyond the intended scope of this article.

**8 Conclusion and future work**

This paper presents the outcome of our research on the construction of the CMS framework, which can be used to assist software developers in determining the suitable component models to be used in their software development projects. It first determines the processes that constitute the framework by reviewing a number of existing selection approaches and then identifies the issues that need to be addressed to realise each process. Each issue is then elaborated in detail and the approach taken to find the solution to each issue is explained. Combined together, the solutions produce the CMS framework. In this paper, we have also demonstrated how the CMS framework is being applied in determining component models that fulfil the desired criteria specified by the software developers. Result from the application shows that the framework is able to

---

**Table 5** Metric results for selected component models

<table>
<thead>
<tr>
<th>Component model</th>
<th>Metric</th>
<th>Input</th>
<th>Output</th>
<th>Precondition</th>
<th>Postcondition</th>
<th>Invariant</th>
<th>Frequency</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFA</td>
<td></td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Fractal</td>
<td></td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>KobrA</td>
<td></td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>UML</td>
<td></td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>ADL</td>
<td></td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>Palladio</td>
<td></td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

**Table 6** Comparison result of each candidate component model

<table>
<thead>
<tr>
<th>Component model</th>
<th>Input</th>
<th>Output</th>
<th>Precondition</th>
<th>Postcondition</th>
<th>Invariant</th>
<th>Frequency</th>
<th>Subtotal (A)</th>
<th>Subtotal (B)</th>
<th>SI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFA</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.27</td>
<td>0.6</td>
<td>0.87</td>
</tr>
<tr>
<td>Fractal</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.27</td>
<td>0.6</td>
<td>0.87</td>
</tr>
<tr>
<td>KobrA</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.2</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>UML</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1/3</td>
<td>0.4</td>
<td>0.2</td>
<td>0.6</td>
</tr>
<tr>
<td>ADL</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1/3</td>
<td>0.27</td>
<td>0.2</td>
<td>0.47</td>
</tr>
<tr>
<td>Palladio</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1/3</td>
<td>0.27</td>
<td>0.2</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Subtotal (A) for interface complexity, subtotal (B) for track record and SI for satisfaction index

**Table 7** Questionnaire results on the practicality of the framework

<table>
<thead>
<tr>
<th>Question</th>
<th>Mode</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The CMS framework has the potential to be implemented in practice.</td>
<td>3</td>
<td>2.8</td>
<td>0.632</td>
</tr>
<tr>
<td>2. I have no objection to implementing the CMS framework when the necessity arises.</td>
<td>2</td>
<td>2.5</td>
<td>0.707</td>
</tr>
</tbody>
</table>

Component models can be computed and a closer match can be determined.

Apart from demonstrating the application of the framework as a means to evaluate it, a focus group evaluation with software developers and practitioners as the participants had also been conducted, involving ten participants with the software development background. Among others, two quantitative data pertaining to the feasibility of the CMS framework were obtained in the form of questionnaire distribution towards the end of the focus group discussion session. The two questions are shown in Table 7 above. The participants were asked to rank the questions using the scale of 1–5 with 1 means strongly agree and 5 means strongly disagree.

From the results shown in Table 7, it can be seen that majority of the software developers rate the practicality of the CMS framework as moderate. Majority of them are also willing to adopt the CMS framework in their software development projects if needed. With standard deviation values of 0.632 and 0.707, respectively, it can be said that the results are convergent and unanimous.

Other than the feasibility, the focus group session also discussed other aspects of the CMS framework including the qualitative evaluation of the processes that constitute the framework. One of the issues brought up by a participant is concerning the process of extracting the relevant information of the component model to be included in the component model repository of the CMS framework. Firstly, on whether the process will be done manually or automatically and secondly on the software developers have to extract and enter the information themselves. Automation is one possible solution to this. However, automatic extraction of component models information still has a very long way to go as it requires the documentation to be presented in certain specified format before information extraction can take place. As mentioned earlier at the beginning of this paper, there are about 32 component models existing to date. Therefore manual extraction of information, although tedious and time consuming, is still possible.

Other constructive comments were also given and discussed during the session. However, detail discussion of the focus group evaluation session is beyond the intended scope of this article.

8 Conclusion and future work

This paper presents the outcome of our research on the construction of the CMS framework, which can be used to assist software developers in determining the suitable component models to be used in their software development projects. It first determines the processes that constitute the framework by reviewing a number of existing selection approaches and then identifies the issues that need to be addressed to realise each process. Each issue is then elaborated in detail and the approach taken to find the solution to each issue is explained. Combined together, the solutions produce the CMS framework. In this paper, we have also demonstrated how the CMS framework is being applied in determining component models that fulfil the desired criteria specified by the software developers. Result from the application shows that the framework is able to

---

IET Softw., 2011, Vol. 5, Iss. 5, pp. 474–486
assist software developers in determining the suitable component models for their use. Focus group evaluation performed to evaluate the feasibility of the CMS framework also showed that it has the potential to be applied in practice by the software developers. Further work includes extending the framework to incorporate more selection criteria and component model properties into the CMS framework. Having more selection criteria enables software developers to better describe their preferred component model and having more component model properties enables candidate component models to be represented more accurately in the framework.

9 Acknowledgments
This project was funded in part by the University of Malaya Research Grant (UMRG). We would like to also express our appreciation and thanks to the software developers who have participated in the focus group discussion session.

10 References