# Semi-Formal, not Semi-Realistic: A New Approach to Describing Software Components

E James Montgomery, Rune Meling, Daniela Mehandjiska

School of Information Technology
Bond University
Queensland, Australia
Email: jmontgom@bond.edu.au, rune.meling@morecom.no, dmehandj@bond.edu.au

**Abstract.** A new semi-formal method for describing and retrieving components has been devised, implemented and validated through the development of a component description manager. A new classification framework which allows component providers to describe components in a semi-formal but consistent way is proposed. The component descriptions are stored in a repository, and component consumers can use the same framework to define their requirements when searching for components. The framework is semi-formal and focuses on ease of use for the component providers and consumers. It aims to provide a level of accuracy and consistency close to that achieved by formal methods, without the same level of complexity.

## 1 Introduction

The appearance of patterns, frameworks and component-based software engineering in the last five years was significant and signalled a new phase in the development of software [4,5,9–11]. To facilitate the re-use of components across many applications, it is crucial that component providers are capable of describing their components in such a way that a component consumer can locate required components.

Often the effort required to create a generic component, find, adapt and integrate it into a specific application is greater than the effort needed to create the component from scratch. Therefore, the ability to create, locate and integrate software components is a critical factor in ensuring the success of this new paradigm of software development.

Extensive research has been conducted in the area of component management to help software engineers to track and reuse software components in the component repository. According to Mili *et al.* [8], retrieval methods can be divided into three major families:

- text-based encoding and retrieval;
- lexical descriptor-based encoding and retrieval;
- formal specifications-based encoding and retrieval.

As the size of software libraries and the complexity of components increases, and components' semantic differences become finer and finer, formal specification-based encoding and retrieval methods become more important [8].

### 1.1 The Formal Approach to Specifying Components

Interface specification is used in component-based software engineering to describe components. The interface separates the specification of a component from its design and implementation. Interface specification uses a formal specification language, which has its own syntax and semantics to formally describe components. It has been recognised as the best way of describing the semantics of a component abstractly and formally [1]. Use of a technique based on semantic descriptions of components offers the possibility of more precise retrieval results [7,8].

Zaremski and Wing [12] propose a component retrieval method based on matching the signatures of the operations. This method describes the behaviour of an operation by a formal specification language Larch/ML. However, this method only describes the behaviour based on the terms appearing in the operation's signature. This method does not allow the semantics of a component to be described completely. In fact, research on retrieval methods has always focused on theories for verifying the match between a specification and a query. Retrieval methods rarely address the question of how to describe the semantics of a component completely.

Catalysis [2–4] and this is CBD provide a mechanism to define the semantics of components completely. This approach applies a set of attributes as a vocabulary to specify the behaviour of operations. It also defines the effects of operations precisely through pre-conditions and post-conditions.

Although a formally described component would provide opportunities for automatic component matching and other software engineering automation, there are some obvious disadvantages with formal description languages:

- All formal description languages are very complex. To describe all aspects of a component's functionality and interfaces can be nearly as difficult as re-implementing exactly the same component in another language;
- The process of describing a component formally is time-consuming and will therefore increase the time-to-market;
- As specifications become more complex, it becomes difficult to match the whole interface specification;
- Developers may specify identical components in different ways;
- The description languages are usually tied to one specific technology or programming language.

### 1.2 The Informal Approach to Specifying Components

Most retrieval systems apply text-based encoding and retrieval methods [6]. With text-based encoding, the functionality of components in the repository is described in natural language. The retrieval is based on the words and strings appearing in the description. As indicated in [7], there are advantages and disadvantages to this

approach. The main advantage of the text-based method is that it is inexpensive and easy to use. The disadvantage is that it does not take into account the context. This method needs to be used in conjunction with other search methods in order to achieve retrieval results with higher precision. Despite its deficiencies, the informal approach is used by the majority of component marketplaces available on the Web.

### 1.3 An Alternative Approach to Specifying Components

The use of formal methods to describe components is not a realistic approach for most component vendors. Formal methods require an extensive amount of work by developers to describe their components. Yet informal methods are inadequate for the efficient retrieval of components. A new semi-formal approach is proposed in this paper that combines some of the rigour of formal approaches with the ease of use of informal description techniques.

## 2 A Semi-formal Framework for Component Description

A new semi-formal method for describing and retrieving components has been devised, implemented and validated through the development of a component description manager (CDM). CDM provides a classification framework for effective component management. The framework consists of two parts:

- A Classification Tree that provides a vocabulary to be used when describing components. The tree also provides a mechanism to reference all other existing components and standards.
- A Component Description that consists of several key-value pairs, where the key is a node from the Classification Tree, and the value is either a string or another node from the Classification Tree.

### 2.1 Classification Tree

The Classification Tree provides a vocabulary for describing components, which combines domain knowledge, ontological information, and semantics. The Classification Tree offers a semi-formal alternative to the highly informal techniques used to describe components at most component marketplaces as well as to the highly formal specification languages that are difficult to use. Component Descriptions can be created using the descriptive terms provided by the Classification Tree. The tree consists of four main sub-trees: characteristics, grammar, components and standards. Figure 1 shows the top-level branches of the Classification Tree.

**Characteristics Sub-tree.** This sub-tree provides the user with a vocabulary for describing components, component libraries and other software artifacts in a systematic and consistent way. The words stored in this sub-tree are typical words

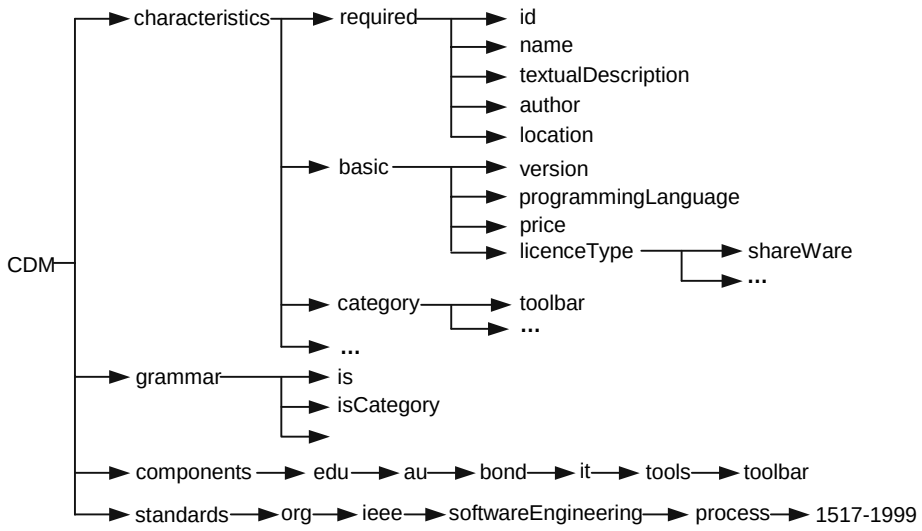


**Fig. 1.** Top-level branches of the Classification Tree

used for describing any component. To keep the sub-tree as general and static as possible, all words are at a fairly abstract level. For example, a branch called “sound” under the branch “dataFormat” is acceptable, but instances of this dataFormat like “mp3” and “wav” are very specific and likely to change. Therefore they should be stored in the standards sub-tree.

It is not intended to store all possible characteristics of components in this sub-tree. It should be kept at a size which makes it easy for component providers to find the desired characteristics when describing a component, and for component consumers to describe the characteristics of the component they are looking for.

**Grammar Sub-tree.** The grammar specified in this sub-tree is used both when describing and searching for components. The grammar words are verbs that specify how the characteristics from the characteristic sub-tree are related to the component being described.

**Components Sub-tree.** The components sub-tree allows any component to be uniquely identified. Each component stored in CDM has its ID stored in this sub-tree. The structure of the tree is built up in a way similar to the suggested universal naming convention for Java components, in which the largest domains appear towards the root of the tree while more specific domains appear further away from the root. This method of unique identification enables component descriptions to refer to other components whose descriptions are stored in the repository.

This sub-tree is not a repository for component descriptions, it is only a mechanism for identifying existing components. For example, classes in the class libraries of all Java versions can be listed under the components.com.sun.java branch. This does not mean that all these classes are described using this framework, it only means that

other descriptions can refer to them. For example, using the key-value pair {"grammar.extends", "components.com.sun.java.jdk1_2.java.swing.jbutton"} as part of a component description provides important information about the component being described. Each new component whose description is added to the repository receives an ID in this sub-tree.

**Standards Sub-tree.** This sub-tree has a similar structure to the component sub-tree, but instead of storing components it stores standards. Official standards accepted by large standards organizations such as the ISO are included in this tree. In addition, de facto standards, Requests for Comment (RFCs) and draft standards can be stored in this tree to enable component developers to describe what standards their components adhere to. This is a very powerful mechanism, since it allows precise description of components through the standards that they support and thus, for the effective location (by component consumers) of components.

**Properties of the Classification Tree.** All separate installations of CDM use the same characteristics and grammar sub-trees, but do not contain all components and standards that exist. It is envisaged that instances will specialise in one type of component, and incorporate only those standards that relate to components of this type. This tree structure has a number of features:

- Each node in the tree has a single attribute, a string representing its name. Because each node's address in the tree is unique, names can recur. This is important as some terms have different meanings in different contexts.
- Each node has a unique and constant address, given by the address of its parent followed by a period (.) and the name of the node. For example, a node with the name "basic," located just below the "characteristics" sub-tree's root node, has the address "characteristics.basic."
- It is possible for the tree to grow as new components are added to the tree.
- The first two branches, which contain the characteristics and grammar should as static as possible.
- The last two branches that contain actual components and standards can grow to incorporate the newest standards and components.

### 2.2 Component Description

The description of a component contains a set of features that differentiate it from other components in terms of its area of use, business domain, visibility to the user and other features. Each Component Description (CD) is made up of a set of ordered pairs of strings. This makes the CD simple, yet flexible. Generally, these strings are addresses of nodes in the Classification Tree. The first string in the pair is an address taken from the Grammar sub-tree. The second string is generally an address taken from the Characteristics sub-tree, although for some Grammar terms such as "extends," it should be a component ID taken from the Component sub-tree. When describing mandatory information about a component, the first string is the address of a node on the "required" branch of the characteristics sub-tree. This branch contains

basic information that must be supplied with each component. The second string is user specified, such as the author's name or component name. An example component description for a fictional toolbar component is presented in Table 1.

**Syntax Diagrams.** The syntax diagrams in Figure 2 specify the valid strings for values that Component Descriptions can contain. Single values are those strings that represent the value associated with a node in node-value pairs that make up a Component Description.

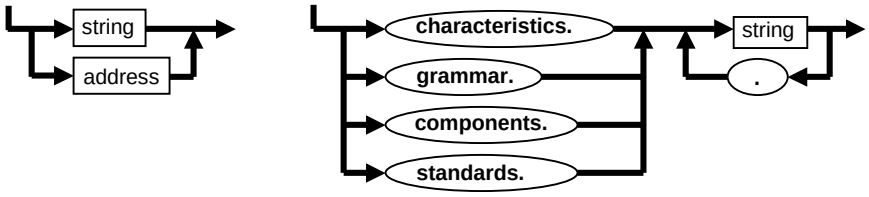


**Fig. 2.** Syntax diagrams

### 2.3 Storing and Retrieving Component Descriptions

The component descriptions are stored separately from the Classification Tree. Each instance of CDM can choose the way to store the component descriptions. Typically, different repositories use the same tree to describe components, but they do not contain the same component descriptions.

When a component consumer is searching for a component, he uses exactly the same mechanism to describe the component as that used by component providers. A special-purpose branch of the grammar tree could be constructed that is used solely for searching. Except for this, consumers build up a component description in the same way as a provider. For example, if a consumer constructs a component description that contains two key-value pairs

{"grammar.is", "charateristics.basic.licenceType.shareware"}
{"grammar.isCategory", "characteristics.category.toolbar"},

this very simple component description is submitted to the repository, and used to locate the description of all registered shareware toolbars.

## 3 A Component Description Manager

A prototype of the system has been built with the following functionality:

- Descriptions of components are constructed using the classification framework;
- Users are able to submit component descriptions to the system;

**Table 1.** An example component description

| | |
|---|---|
| characteristics.required.id | components.edu.au.bond.it.tools.toolbar |
| characteristics.required.name | "The glowing toolbar" |
| characteristics.required.textualDescription | "A toolbar which highlights the button the mouse is currently over. The toolbar can contain images on the buttons..." |
| characteristics.required.author | "John Dow, Bond University" |
| characteristics.required.location | "http://www.bond.edu.au/componets/glowbar.html" |
| characteristics.basic.version | "2.1" |
| characteristics.basic.programmingLanguage | "Java 2.0" |
| characteristics.basic.price | "AUD 30" |
| grammar.is | characteristics.basic.licenceType.shareware |
| grammar.is | characteristics.userInterface.control.toolbar |
| grammar.uses.show | characteristics.dataFormat.binary.picture |
| grammar.is | characteristics.componentType.component |
| grammar.isCategory | characteristics.category.toolbar |

- Users are able to search for components, based on the descriptions stored in the system;
- The system indicates the similarity between two components (degree to which one can replace the other) and/or compatibility (degree to which they can work correctly together).

### 3.1 Use Case Models

Users of the system fall into four categories, component providers, component consumers, validators and administrators. Component providers are able to construct descriptions of their components and submit these to the system validators for review and possible inclusion in the repository. Component consumers are able to search for components that meet their requirements. Validators are responsible for reviewing newly submitted component descriptions. Administrators are responsible for managing existing component descriptions as well as for maintaining the web server that contains the repository.

**Actors.** Four actors have been identified.

- **Component provider:** This actor uses CDM to register components.
- **Component consumer:** This actor uses CDM to find components already registered in CDM's repository.
- **Administrator:** This actor is responsible for the correct operation of CDM. The administrator is a super user who is authorized to access all functionality provided by CDM including editing of the Classification Tree (see Section 2.1) and performing backups of the system.
- **Validator:** The validator is responsible for reviewing and processing CD's submitted by describers.

The use case model of the system is shown in Figure 3. Three of these use cases are described Tables 2, 3 and 4.

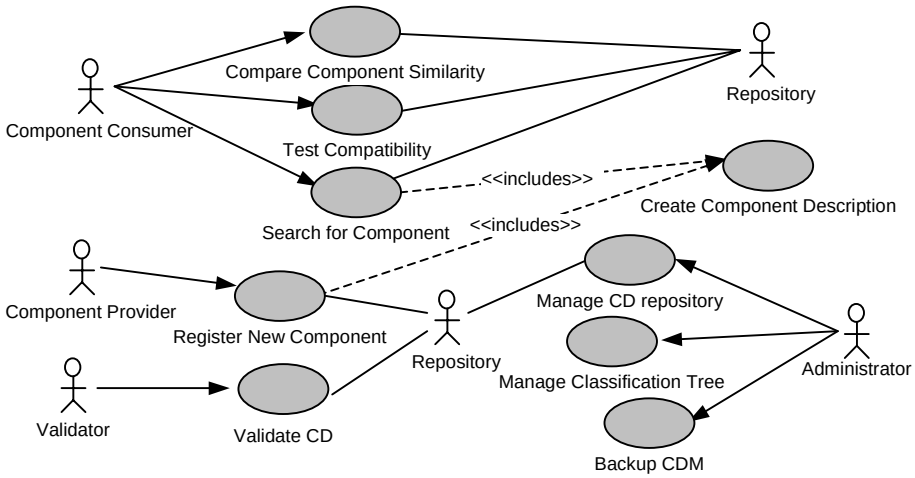


**Fig. 3.** Use case model

**Table 2.** Register new component use case

| | |
|---|---|
| **Responsibility** | Let the actor register a component with the CDM system. |
| **Relation to other use cases** | Uses the "Create Component Description" use-case. |
| **Actors** | Component Provider |
| **Pre-condition** | None |
| **Description** | • Actor opens the CDM web site in a browser<br>• Actor selects link labeled "Register your own component"<br>• Actor is asked to fill out personal contact information<br>• Use case "Create Component Description" is invoked<br>• Actor presses <Submit><br>• CDM sends a "Confirmation of submission" back to actor |
| **Post-condition** | A component description is waiting for validation by the Validator. |

### 3.2 Architectural Design

CDM consists of three packages as shown in Figure 4. The User Interface package contains those parts of the system that interact with the user. It consists of two user interfaces, one for component developers and searchers, and another for the administrators of the system. The Component Description Manager package is responsible for managing the repository of component descriptions. This includes typical management tasks such as add and delete as well as the important tasks of searching for components, comparing two components' similarity and compatibility.

The Tree package is responsible for managing the Classification Tree. This involves checks on the consistency of the Tree as well as facilities for updating the Tree.

**Table 3.** Create component description use case

| | |
|---|---|
| **Responsibility** | Create a CD |
| **Relation to other use cases** | Used by the "Register New Component" and "Search for Component" use cases. |
| **Actors** | Component Provider, Component Consumer |
| **Pre-condition** | This use case can only be accessed as part of the "Register New Component" and "Search for Component" use cases. |
| **Description** | • CDM creates an empty component description<br>• Actor is asked to fill out any required fields<br>• Actor is presented with a window containing newly created CD with all required fields completed<br>• CDM provides tools to the user to add further lines to the CD, based on information from the Component Description Tree<br>• Actor presses <OK><br>• CDM validates description based on syntax rules |
| **Post-condition** | A valid CD is created, ready to be used by other parts of the system. |

**Table 4.** Search for component use case

| | |
|---|---|
| **Responsibility** | Let the actor search for components registered in the CDM. |
| **Relation to other use cases** | Uses the "Create Component Description" use case. |
| **Actors** | Component Consumer |
| **Pre-condition** | None |
| **Description** | • Actor opens the CDM web site in a browser<br>• Actor selects the link labeled "Search for a component"<br>• Use case "Create Component Description" is performed<br>• Actor presses <Search><br>• CD built above is used as a search expression, and sent to CDM<br>• CDM searches the repository for CDs matching the required CD<br>• CDM returns result of search as a list of closest matching CDs |
| **Post-condition** | None |

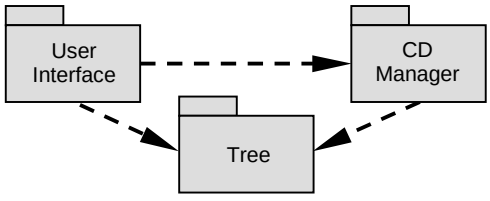


**Fig. 4.** Packages in CDM

## 4 Validation

To validate the effectiveness of the proposed concept over existing formal and non-formal solutions:

- Several components have been described using the classification framework;
- A prototype has been developed.

Components like word processors, toolbars, and spreadsheets have been described by using the classification framework, and compared with formal and non-formal descriptions of similar components.

The focus of the proposed semi-formal description technique has been on ease of use, and the level of precision achieved. The time and effort spent describing components using the new classification framework have been significantly reduced. The proposed approach is semi-formal, so using it to describe components is consequently far less complicated than if a formal language were used. Although the level of precision achieved by the proposed approach is not as high as that of a formal language, the reduction in effort needed to describe components makes this approach quite cost effective.

A prototype has been developed to demonstrate how these component descriptions can be used for searching, comparing similarity (if two components can replace each other), and checking compatibility (if two components can work together).

## 5 Future Extensions

The similarity and compatibility checks implemented in the prototype of CDM are not as rigorous as would be necessary to prove that one component could replace another or that two components will function correctly together.

The CDM allows the mechanism for searching, and for checking compatibility and similarity to be replaced by plug-ins. This allows intelligence to be added in these processes. An example of use is a CDM repository which specialises in JavaBeans. This repository can allow the whole Java-interface be included as part of the description, and have search/compatibility objects plugged in. These objects will understand the syntax of the Java interface, and use this when checking the compatibility of two components.

## 6 Conclusions

The proposed semi-formal approach has proven to be nearly as easy to use as a non-formal textual description of components. The descriptions are not as precise as if a formal language were used, but their ease of use, combined with the high level of precision achieved demonstrates the power of this concept.

If components are going to be used and re-used when developing software, it is crucial that they are described properly so they can be located and retrieved. Although some of the larger component developers might have the resources to

describe their components formally, it is just as important that the consumers of the components understand the descriptions to make sure the component is what they are looking for. This means that both the creators and users would have to understand the same formal language.

It is unlikely that an average developer will be capable of understanding such a description, and much less likely that he or she will take the time to describe self-developed components in such a way. This, combined with the need for something more formal than pure textual descriptions, identifies the need for a semi-formal solution like CDM.

## References


1. Brown, A.: From Component Infrastructure to Component-Based Development. Proceedings of the 20th ICSE International Workshop on Component-Based Software Engineering, Kyoto, Japan (1998)
2. D'Souza, D., Wills, A.: Catalysis – Practical Rigor and Refinement: Extending OMT, Fusion, and Objectory. http://iconcomp.com/papers/catalysis/catalysis.frm.html (1995)
3. D'Souza, D., Wills, A.: Types, Behaviors, Collaborations, Refinement, and Frameworks: Input for OMG OOA&D Submission. http://iconcomp.com/papers/omg-ooad/omg-rfp-2.pdf (1996)
4. D' Souza, D., Wills, A.: The Catalysis Book. http://www.trireme.com/catalysis/book/ (1998)
5. Fowler, M.: Analysis Patterns: Reusable Object Models. Addison-Wesley, Reading, MA (1997)
6. Frakes, W. B., Pole, T. P.: Proteus: A Reuse Library System that Supports Multiple Representation Methods. ACM SIGIR Forum **24** (1990) 43–55
7. Mili, H., Mili, F., Mili, A.: Reusing Software: Issues and Research Directions. IEEE Transactions on Software Engineering **21** (1995) 528–562
8. Mili, R., Mili, A., Mittermeir, R.: Storing and Retrieving Software Components: A Refinement Based System. IEEE Transactions on Software Engineering **23** (1997) 445–460.
9. Ning, J. Q.: CBSE Research at Andersen Consulting. Proceedings of the 20th ICSE International Workshop on Component-Based Software Engineering, Kyoto, Japan (1998)
10. Seacord, R. C., Hissam, S. A., Wallnau, K. C.: Agora: A Search Engine for Software Components. Technical Report CMU/SEI-98-TR-011, Software Engineering Institute, Carnegie Mellon University (1998)
11. Wills, A., D'Souza, D.: Rigorous Component-Based Development. Trireme Object Technology & ICON Computing (1997)
12. Zaremski, A., Wing, J. M.: Specification Matching of Software Components. Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering (1995) 6–17