www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 7 Issue 2 February 2018, Page No. 23557-23566
Index Copernicus Value (2015): 58.10, 76.25 (2016) DOI: 10.18535/ijecs/v7i2.07

Building Software Component Architecture Directly from User Requirements

Badamasi Imam Ya'u[1] and Muhammed Nura Yusuf[2]

Mathematical Sciences Department, Abubakar Tafawa Balewa University, Nigeria.

Abstract:
Building software architectures from a set of requirements has been an area of research where programmers, architects and software engineers spend a lot of time using their expertise in resolving peculiar problems of mapping requirements to architectures. Some of these problems are directly associated with the ambiguity, incompleteness and inconsistency of requirements which draw a wide gap between the informal and formal specification of these requirements. The main objective here is to reconcile the mismatch in-between these domains by providing a systematic mapping technique. This paper presents a tool from which requirements are read from user in natural language or file and generated into words whereby the user makes some selections and maps the selected words directly to components architecture. Based on the design of this tool, human heuristic is used in the selection of the words. Unlike components, connectors are set as static. Partial architecture of requirements is drawn incrementally until complete system architecture is constructed**.**

Keywords: Component-based, Requirements Mapping, Architecture, Natural Language

1. Introduction

Component-based development is a branch of software engineering, which aims towards systematic way of reusing pre-built software components or sub-systems in a cost effective manner for building larger systems incrementally [1, 2]. Reusability of software units is very important especially in developing complex software systems [3, 4]. Unlike in the past when software systems were being developed from scratch through long processes of writing codes, the introduction and wide acceptance of component-based software development make a lot of activities in industries very easy and making productions very fast. The contribution of component-based development does not lie only on the reusability of the software units, but also plays a vital role of reducing software products time to markets, giving opportunities for system's upgrade and offering cost effective software development.

One of the challenges in of software development has been a mapping process between the two domains of software development, that is, the problem and the solution domains. This may be related to the wide gap between the requirements specification and the software architectures. Many approaches have been tried to make reconciliation in bridging the gap in between these two domains to achieve an effective and straightforward mapping [5]. However, there is still a clear gap to automate the process completely.

This paper presents and describes a tool that takes requirements presented in natural language as input and enables incremental construction (drawing) of these requirements to partial architecture. The tool reads requirements from file or directly from user, splits the words contained in the requirements and generates component automatically from these words according to selection made by a user. The mapping (in this case, the generation of components) and drawing processes require human intervention in which heuristics and guidelines are used. The paper adopts a component-based model [6-9] that possesses properties of encapsulation and compositionality; using exogenous connectors from all levels of compositions. As such, the selection of the words completely depends on the key semantics of the adopted component model which are "computation and control". The rationale here is to build a partial architecture from requirements using direct and systematic mapping. The architecture is built incrementally until it is complete and satisfactorily for the set of the requirements.

2. Mapping of Requirements to Architectures

In general, software development process constitutes a collaborative effort between requirement engineering and architectural design activities. To understand the relationship between requirements and architectures, we need to define what a mapping is all about.

According to web dictionary [10], mapping is defined as "function: (mathematics) a mathematical relation such that each element of a given set (the domain of the function) is associated with an element of another set (the range of the function)". From this definition, mapping of requirements to software architectures can be seen as the concrete relationship that transforms the elements of the requirement domain to an architecture view point of the requirements.

2.1 Related Approaches

The Researchers have been looking for an efficient and cost-effective ways of mapping requirements to appropriate architectures that perhaps accomplish the software development process. As a result, a lot of activities have been done right from traditional mapping approach, which is considered inadequate [11] due to its limited technique of mapping requirements directly to architectures. Hitherto, many approaches claim improvement in mapping requirements successfully to architectures. The following are some of the approaches.

A. *Behaviour Tree:* Behaviour tree is defined by Dromey in [12] as "a formal, tree-like graphical form that represents behaviour of individual or networks of entities which realize or change states, respond- to/ cause events, and interact by exchanging information and/ or passing control". The technique entails a straightforward translation of the requirements represented in natural language in a systematic sentence-to-sentence, phrase-to-phrase or word-to-word style. The mapping and incremental addition of requirements in this approach are related to the context of this paper. Figure 1 shows the component-state, a rule-kind for the design of the architecture that strictly depends on the expression found in a set of requirements. Because of these rules, the translation clearly exposes all actors and components involved, the interactions between them, constraints that control the behaviour and events and conditions that trigger change of state initially realized. The nature of the translation makes the relationship between the informal and formal specification so strong, flexible, clear, direct and hence traceable especially in the case of redundancy control, missing and update of requirements [13, 14]. In the translation process, individual functional requirements that are recognized as fragments of behaviour are represented in a form of tree i.e. the words in the natural language that show the behaviour, condition, constraints, state are extracted and put in a tree-like form with some links indicating the flow of the activities.

B. *Feature Oriented Mapping:* Feature oriented mapping is a software system development approach that claims mapping of requirements directly to architectures. This is a move towards improving the limitations of traditional, structured and objected oriented approaches in mapping requirements directly to software architectures. It is related to this paper in the sense that, the mapping process between the two domains (problem and solution) is natural and direct. A feature in this respect is defined as "a higher-level of abstraction of a set of relevant detailed software requirements, and is perceivable by users (or customers)" [11]. The goal of feature-oriented mapping is to carry out a direct and natural mapping between feature model and architecture model by establishing a concrete mapping relationship between the requirements and software architectures. In this attempt, significant roles of functional and non-functional features are observed and handled separately. As a result, the mapping process entails two phases: feature oriented requirements modelling and feature oriented architectural modelling. This is depicted in figure 2 [11].

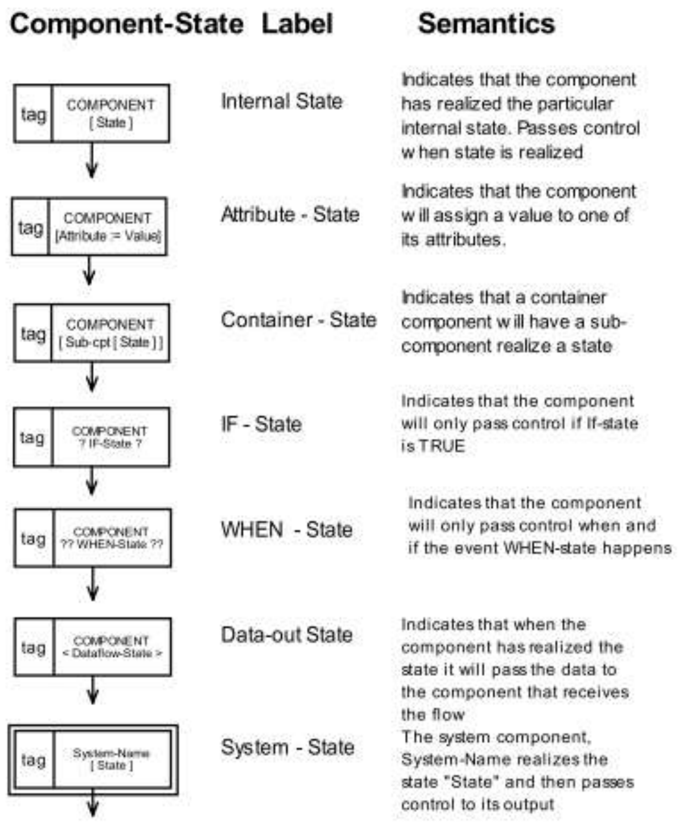

Component-State Label
Semantics
tag
COMPONENT [State]
Internal State
Indicates that the component has realized the particular internal state. Passes control when state is realized
COMPONENT [Attribute := Value]
Attribute - State
Indicates that the component will assign a value to one of its attributes.
COMPONENT [Sub-cpt [State]]
Container - State
Indicates that a container component will have a sub-component realize a state
COMPONENT ? IF-State ?
IF - State
Indicates that the component will only pass control if If-state is TRUE
COMPONENT ?? WHEN-State ??
WHEN - State
Indicates that the component will only pass control when and if the event WHEN-state happens
COMPONENT < Dataflow-State >
Data-out State
Indicates that when the component has realized the state it will pass the data to the component that receives the flow
System-Name [State]
System - State
The system component, System-Name realizes the state "State" and then passes control to its output

Figure 1: Behaviour tree notation, key elements from [15]

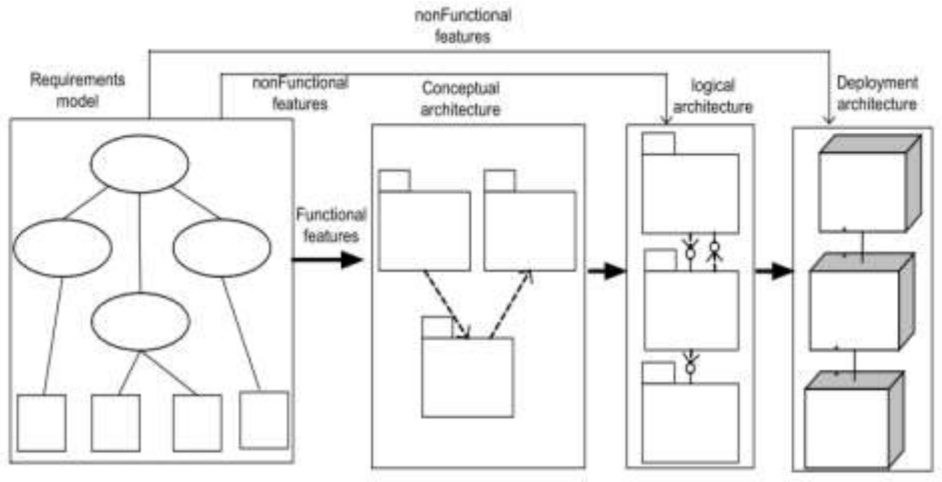


Figure 2: Mapping feature model to architecture model [11]

3. Methodology

3.1 Selection of Words for Conceptual Components

It is expedient to know how words are selected and mapped from a table for the generation of conceptual components. In this regard, the components are called conceptual components because they are not found in any repository. Rather, the components are derived directly from the words in the requirements. To get these components right, the following guidelines should be followed:

- Identification of parts of speech that express *computations*. In this paper two key semantics are considered: computation and control, which are associated directly with components and connectors respectively. Computation expresses a functionality a component renders in terms of actions such as data storage, verification, execution, calculation etc. Glaringly, in the context of parts of speech, verbs, nouns and phrases express actions performed in sentences [16]. Since we are dealing with sets of requirements written in natural language, we now go through each requirement scrupulously and select some of these words or phrases that express unique actions that give a manifest picture of those requirements. Here, the user should use heuristic to identify the right candidates so that even if another user is to use the same set of requirements, virtually, the same result is obtained. Some of these candidates that express computation from a sentence include:

a. Verbs. It is obvious that verbs always express actions. Some examples of these are: press, click, go, close, open, drag, drop, insert, print, display, delete, and calculate, among others.
b. Nouns. Although nouns are names of anything, some of them clearly express actions. E.g. authentication, movement, illumination, indicator, generation, sound, withdrawal, to mention a little.
c. Phrase. Some combination of words such as: saving accounts, next floor, get balance, add to, switch off, push button, dispense cash, increase speed, slow down, apply now, enter password, and the rest.

➢ Select or skip words. When such words with computation are identified, the user selects them and adds more information such as *caption* and *description* for generation of the conceptual components when drawing the architecture. When reading the requirements, some of the words are not used because they do not clearly show any *computation* or *control*, therefore the user is required to skip them and go with the right ones.

3.2 Selection of Composition Connectors

Having two or more components, it is necessary to find a connector that joins these components to form composite components. The connectors that accomplish this task are called composition connectors. These connectors come with different functionalities with respect to the nature of control flow between components. For this component model viewpoint, these connectors are termed exogenous connectors [17], which popularly include: selector connector, for branching between components; pipe connector, for forwarding data required in another component; sequencer connector, for making a serial execution among the components. Since exogenous connectors encapsulate control in this model, and the tool does not make automatic text analysis about which word from the requirement should act as connector when building the partial architecture, human talent is also used here to decide the type of the connector that best suits for the compositions. To be able to compose the architecture well, the following guidelines should be used:

➢ *Identification of control flow*: Before any decision is made with regarding to the choice of connectors, the user should critically observe how the components he intends to connect interrelate with each other in the requirements. In other words, he should identify the lexical flow that binds the expression of the components according to the requirement statement. To make the identification of the control flow easier, the following parts of speech are taken in to consideration:
a. Preposition. This is part of speech that connects a noun to another word in a sentence. Some of these include: to, at, after, on, before, etc.
b. Conjunction. This is the type of part of speech that joins clauses or sentences or words. E.g. and, but, when, or, etc.

➢ *Connector tree node*: Unlike the component tree node that is generated directly from the selected words, the connector tree node is generated anytime the application is run. The tree node provides two more connectors namely: guard connector, for filtering data movement from one component to another; and loop connector, for making an iterative execution. Depending upon set of requirements, the user should heuristically choose any of the connector from the tree node based on the identified control flow found in the requirements expression.

Since the nature of the control flow always varies, preposition and conjunction significantly help the user in identifying the interrelationship between conceptual components according to their appearance in the requirements.

3.3. Composition of partial Architecture

Assuming all components required for drawing the architecture are mapped into the component tree panel of the application, it is now time to compose the partial architecture in an incremental fashion in the drawing panel provided. The user follows the following steps to draw the component based partial architecture:

- **Drag and drop**. To be able to draw anything in the drawing panel, the user should drag and drop components in the drawing panel.
- **Original order of the requirements**. Normally, when building component architecture from set of requirements, a sequential technique is used. A sequential in the sense that, in bottom up design, components composition begins from the root upward. In this case, compositions absolutely depend according to the original presentation of requirements i.e. from requirements one to the last unless otherwise some specifications are given. Therefore, a user starts the composition with the first requirement, followed by second and continued till the last one is reached.
- **Connection of the shapes**. All component and connector shapes in the drawing panel are joined together by right click option from the mouse button. The user connects two or more shapes by selecting their names hence, connection is done

immediately. In the same way, the connection can be removed when it is not needed.

- **Deleting a Connector Node**. At any level of the drawing, any shape drawn in the panel can be deleted as well as how components are added in the components tree pane. However, connector tree node is not erasable. This is due to the fact that these connectors are designed to be static in the code. If user attempts to delete a connector node, a warning message is displayed, which notifies the user that, a connector cannot be deleted.

3.4 Complete Example

The following is a simple example that presents a detailed step by step process of mapping requirements to architecture. In this example, complete system architecture for counting words from a file is built incrementally from given set of requirements. The requirements are enumerated below:

R1. The system shall take a file name as an input.

R2. Each time a file is given, it must be validated. If the file name is invalid, an error message will be displayed.

R3. Once validated, the words in the file will be formatted according to the type of input file.

R4. The formatted word will then be counted.

R5. The result of the calculation that contains the number of words in the file will be displayed to the user.

Starting always from the first requirement, the phrase "take a file name" expresses a *computation* and the noun "input" defines the nature of the computation. From this expression, we need a component in the architecture for reading a file input. From R1, the word "input" is chosen to be the conceptual component and adding to it a suitable name, in this case "Input Reader". For the R1, only one component is required.

From R2, the phrase "must be validated" indicates that something must occur after the commencement of an event. For this reason, we require a component unit for this phrase, say "File Validation". To generate this component, the word "validated" is chosen from the sentence. From the phrase "each time a file" at the beginning of R2, it is indicated that the "File Validation" component always requires an input from the "Input Reader" component. In this situation, a *pipe connector* is required for composing these components. The two components along with the pipe connector are composed as shown figure 3.

The second part of R2 is a conditional statement for testing the validity of the file. Displaying error in this regard expresses *computation*. Another component therefore, for error message is required. The word "displayed" is added and a name "Error Massage" is given to the component. Since there is a condition, the conjunction "if" specifies a *selector connector* to branch between components.

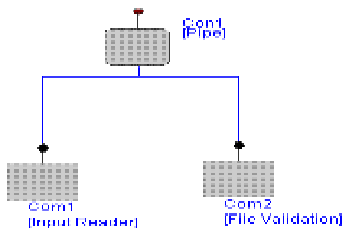


Figure 3: First composite component

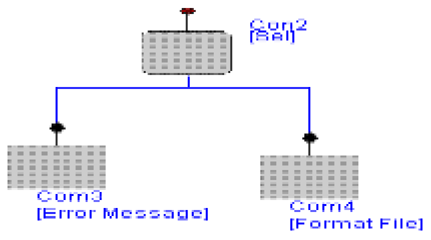


Figure 4: Second composite component

R3 completes the conditional expression about the validation process taking place in "File Validation" component. When the condition is true, the expression "will be formatted" suggests another component unit to carry out some computations. The word "formatted" is also selected from the sentence with a caption "Format File". Since the condition is to go one of the two ways, the *selector connector* initially identified is used to compose "Error Message" and "Format File" components as described in figure 4.

Requirement four is straighter forward. The verb "counted" expresses *computation* and for this reason, a component "Word Counter" is needed. This verb is now chosen as conceptual component. In R4, the expression "will then be" shows a transition of data from one place to another. This indicates a *pipe connector* is required for the partial composition of the component "Word Counter" with other composite components from R1, R2, and R3. To achieve this composition, two more *pipe connectors* at higher and lower levels are required respectively. This is because the levels of composition of the components are now different because of *selector connector* that branches down. This is shown in figure 5.

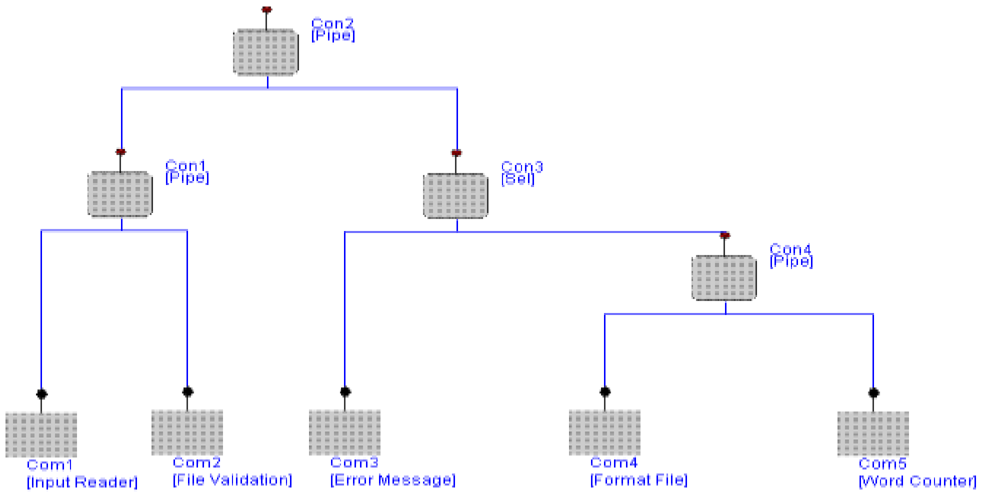


Figure 5: Third composite component

Similarly, in R5, the phrase "will be displayed" defines a *computation*. A component "Display Result" is required. The verb "displayed" is chosen as the conceptual component. Figure 6 shows the final system architecture of world count problem.

Implementation

This section explains in detail how the application in this paper is implemented. Much time was spent in the process of coding to ensure the application runs and works according to the aims of the paper. One of these aims is the automatic analysis of text from a set requirement in natural language which is not achieved hitherto this implementation. However, the application accepts and maps words from the requirements into components shapes. This allows user to draw component partial architectures that will satisfy the set of requirements. Since the selection of the words are done manually by the user, a heuristic approach for decision making upon which words to use as conceptual components is used as described in the previous section.

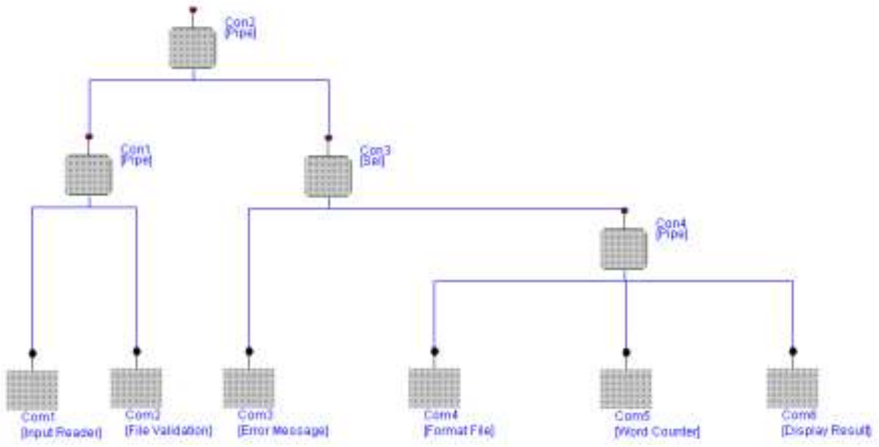


Figure 6: Final system architecture of word count

4.1 Description of the Tool

This paper entails two major parts: the mapping of requirements and transformation of these requirements into component-based architectures. The requirements are the description and specification of a system to be developed, which are written informally in natural language form. In the first part, requirements in English language are read from user through keyboard or loaded from existing file stored on disk or from any device connected to the computer and thereafter, the entire words that makeup of requirements are generated serially in a tabular form. When the generation is done, options are given in respect to which words to be selected and mapped as conceptual components. The chosen candidates are transformed and arranged into component tree node automatically. Component based partial architectures that represent and satisfy the set of requirements read are drawn incrementally in drawing panel of the application by employing a drag and drop technique. To facilitate the composition during the drawing, exogenous connectors in connector tree node are used. However, unlike components, these connectors are set as static during the runtime.

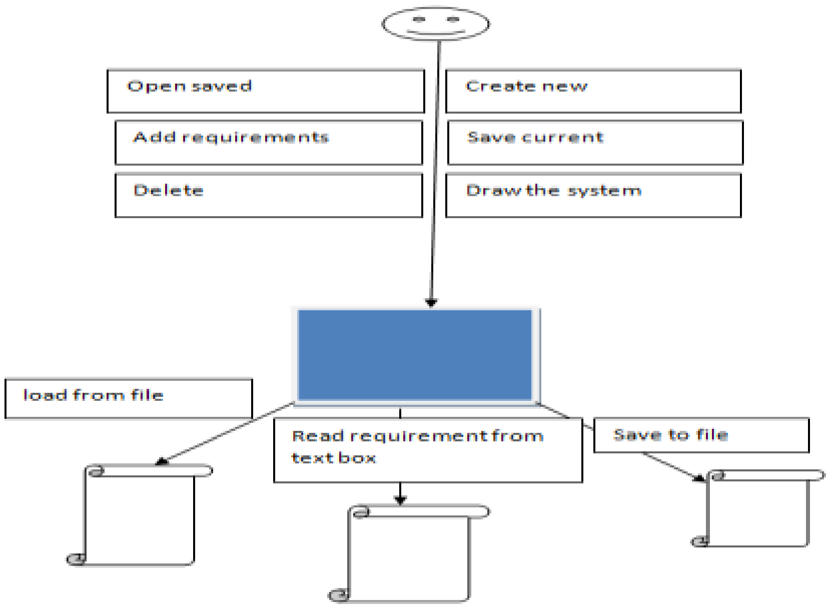
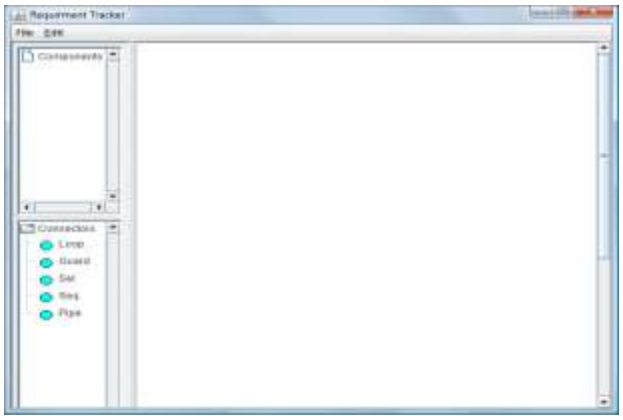


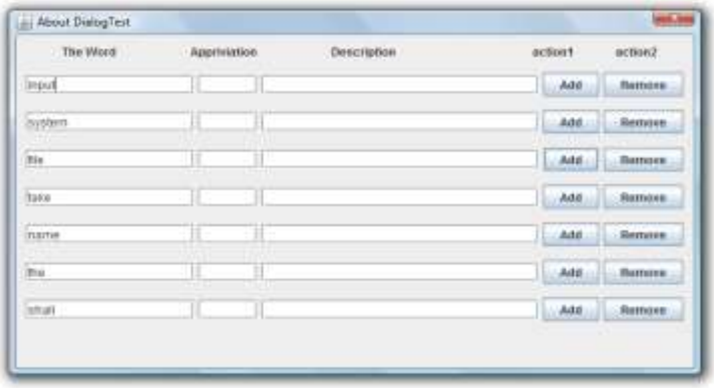
Figure 7: Structure of the system

When the application is launched, the first screen (screenshot 4.1) is displayed. From this, the user has options to choose a file or edit menus. The file menu provides more options such as *new*, for starting new architecture drawing; *open*, for browsing and opening saved work; *save*, for saving current work; and *exit*, for closing the application. While the edit menu offers three options: read requirements from file, enter new requirements and delete. At the left side of the screen, is a tree node panel that consists of upper and lower panes for components and connector respectively. The structure of the system is illustrated in figure 7.

Let us assume there is no any saved architecture design and would like to start by reading requirements from one of the sources (say file). In this case, *edit* menu and *requirements from file* options are selected. A dialog box that shows all folders stored in the system launching the application and devices attached to it is displayed. The same dialog box is displayed when user intends to open saved file from the file menu option. When this option is chosen, the tool reads all the requirements word by word in the file at the same time, leaving no any single token behind.



Screenshot 4.1: First screen of the application

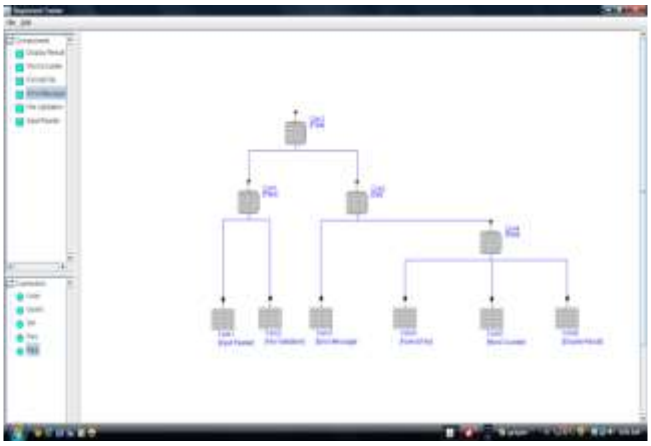


The Word	Appriviation	Description	action1	action2
input			Add	Remove
system			Add	Remove
file			Add	Remove
take			Add	Remove
name			Add	Remove
the			Add	Remove
shall			Add	Remove

Screenshot 4.2: A table generating strings of requirements

In a situation where the requirements are to be read directly from user via keyboard, *enter new requirements* option in the *edit* menu is selected and thereafter input dialog box is popped up. Unlike reading requirements from a file, in this option, a user can enter one or more requirements at time in the text box, and by clicking *OK*, any character that is typed, is displayed serially in a table as shown in screenshot 4.2. The same table generates words when requirements are read from file. With the aid of drag and drop, components and connectors are manipulated in the drawing panel until complete system architecture is built. Screenshot 4.3 shows the complete system architecture for the word count example illustrated in subsection 3.4. These online tutorials [18-20] were found helpful during implementation of this tool, in particular on the features that make every shape movable from one point to another.

The tool presented in this paper is not designed and implemented for a specific problem. The tool is generic and can be used to read more requirements of arbitrary size from variety of problems and hence construct their respective system architectures.


Screenshot 4.3: Final system architecture of word count example

5. Discussion and Conclusion

One of the most important and non-trivial areas of software engineering is component-based development which entails finding solution for mapping requirements to software architectures. Because of the ambiguity, inconsistency and incompleteness of requirements presented by stakeholders, transformation between these domains becomes a labor-intensive activity. However, many researchers come up with different approaches that address this issue.

In this paper, a tool for constructing component-based architectures was implemented. The purpose is to map user requirements presented in natural language directly to component-based architecture which satisfies these requirements. The tool reads requirements either directly from user or from a file, splits each requirement into words and automatically generates components from a selection made by a user. Although the mapping of the words to components is automatic, human heuristic is used throughout for decision making regarding which word of the requirements will be selected. The tool offers an incremental composition as the construction of the architecture is done manually one at a time based on the original order of the requirements.

From the literature survey, we found that some approaches that claim design solutions from requirements to architectures were proposed. Feature oriented mapping is chosen as related mapping approach in the sense that, requirements are organized in feature model from which architectures are derived using a direct and natural mapping. One of the aims of this paper is to offer this kind of mapping. Furthermore, the approach offers iterative and incremental activities during the development. Mapping process is done manually based on the features identified and refined from the requirements. However, this approach deviates from this paper, because it not developed on the background of a component-based model.

Behavior tree in the other hand is more closely related to this paper. The primary task of the paper is to construct component-based architecture incrementally from a set of requirements using a direct mapping that eventually satisfies these requirements. Behavior tree approach does this but, clearly in different way. Though it performs well in incremental composition, it however falls short in systematic definition of component development concept. This is due to the fact that, the development of the system is capitalized on the view of the constructed tree i.e. design behavior tree (DBT). In this approach, systems are developed from scratch to the end. The approach is cumbersome to use in a very large and complex systems and this militates against its adaptability. Some of the major things that incur problems to the usability of behavior tree are the nature of its semiformal notation and lack of tool support. Because of the prevailing nature of the notation, the semantic of behavior tree is not that precise. For instance, to have a component view of the design, individual components are papered out; ignoring all other component-states.

Despite the similarities mentioned earlier in the related approaches, this paper implements a tool that offers a technique of mapping requirements to partial architecture in a quite different way. One distinguishing feature is the way requirements are read, split into words and transformed into components. This enables user to systematically drag, drop and draw partial architecture incrementally according to the original requirements order.

Future Work

This paper is a step towards mapping requirements to components-based software architecture. However, the tool presented and implemented in this paper is currently limited to generation of components automatically from a selection of words made by user, while connectors are generated as static in the runtime. Any other activity is done manually using human heuristic. In future, a text analyzer will be designed and added to the tool. This will make the application more scalable and robust especially when dealing with large number of requirements.

Acknowledgment

This work is sponsored by Tertiary Education Trust Fund (TETFund) of Nigeria. We would also like to thank the Department of Mathematical Sciences, Faculty of Science and Abubakar Tafawa Balewa University Bauchi for the support and promotion of research activities.

References

[1] Sommerville, I., Software documentation. Software engineering, p. 143-154, 2001

[2] B.I. Ya'u, "Component-Based: The Right Candidate for Restructuring the Nature of Software Development in Organizations", International Journal of Engineering and Computer Science,**4**(8): pp. 8, 2015

[3] B.I. Ya'u, A. Nordin, and N. Salleh, "Investigation of Requirements Reuse (RR) Challenges and Existing RR Approaches", Advanced Science Letters, 2017

[4] B.I. Ya'u, A. Nordin, and N. Salleh, "Software Requirements Patterns and Meta model: A Strategy for Enhancing Requirements Reuse (RR)", in IEEE International Conference on Information & Communication Technology for the Muslim World (ICT4M 2016) Conference, Jakarta Indonesia. 2016

[5] P. Grünbacher, A. Egyed, and N. Medvidovic, "Reconciling software requirements and architectures with intermediate models", Software & Systems Modeling, **3**(3), pp. 235-253, 2004

[6] Z. Wang, and K.-K. Lau, "Software Component Model with Encapsulation and Compostionality", University of Manchester, 2007

[7] K.-K. Lau, A. Nordin, T. Rana, and F. Taweel "Constructing component-based systems directly from requirements using incremental composition", in 36th IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2010

i. Nordin, "Constructing Component-based Systems Directly from Requirements Using Incremental Composition", 2013

[8] K.-K Lau, and F.M. Taweel, "Data encapsulation in software components", in International Symposium on Component-Based Software Engineering, Springer, 2007

[9] WordNet. [cited 2016 25-02-16]; Available from:http://wordnetweb.princeton.edu/perl/webwn

[10] Liu, and H. Mei, "Mapping Requirements to Software Architecture by Feature-Orientation", in STRAW, 2003

[11] G.Dromey, "Formalizing the transition from requirements to design", 2006

[12] R.G Dromey, "Using behavior trees to model the autonomous shuttle system", in 3rd International Workshop on Scenarios and State Machines: Models, Algorithms and Tools,(SCESM04), IET, Edinburgh, 2004

[13] G. Dromey, "System Composition: Constructive Support for the Analysis and Design of Large Systems", 2005

[14] R.G Dromey, "From requirements to design: Formalizing the key steps", in Proceedings of IEEE First International Conference on Software Engineering and Formal Methods, 2003

[15] M. Saeki, H. Horai, and H. Enomoto, "Software development process from natural language specification", in IEEE 11th International Conference on Software Engineering, 1989

[16] K.-K. Lau, P.V. Elizondo, and Z. Wang "Exogenous connectors for software components" in Software Engineering International Symposium on Component-Based, Springer, 2005

[17] DnD (Drag and Drop) JTree code. [cited 2011 11-11-11]; Available from: http://www.java2s.com/Code/Java/SwingJFC/DnDdraganddropJTreecode.htm

[18] *Resizable Component in Java Swing*. [cited 2010 07-05-10]; Available from: http://zetcode.com/tutorials/javaswingtutorial/resizablecomponent/

[19] *Swing- An example of drag and drop in JTree [locked*. [cited 2010 13-05-10]; Available from: http://forums.sun.com/thread.jspa?threadID=296255&start=0