Analysis of Approaches to Constructing Component-based Systems from Natural Language Requirements

Azlin Nordin*
* Department Of Computer Science, Kulliyyah Of Information And Communication Technology, International Islamic University Malaysia, 50728 Gombak, Kuala Lumpur, Malaysia

Article Info

Article history:

Received September , 2012
Accepted November, 2012

Keyword:

natural language requirements
mapping
construction
component-based systems

ABSTRACT

System development generally starts with Requirements Engineering (RE) process. Based on requirements, system analysts produce the requirements documents and analyze them in order to produce design documents. These artefacts will be inputs to the later stages in the system development. It is argued that during the transition between these stages, considering many other various factors, information might be missed out or misinterpreted along the way. Hence, a better transition in the development cycle is required. This paper outlines and provides an analysis of the existing approaches in the literature on constructing systems from natural language requirements (NLR) as to provide the motivation of a new approach to constructing component-based system from NLR.

Corresponding Author:

None

1. INTRODUCTION

System development generally starts with Requirements Engineering (RE) process. The outcomes of this process are typically regarded as input to the succeeding processes. In these processes, intermediate requirements models (e.g. object-oriented systems analysis and design (e.g. use case, class, state chart diagrams), structured systems analysis and design (e.g. context, data flow diagrams)), are derived. These intermediate requirements models are manually[1] constructed as a result of some degree of the abstraction process. Such abstraction models, at best, only approximate the raw requirements. Therefore, the same is true of the requirements specification that results from the RE process.

Such a scenario can generally be visualised in Fig. 1. Based on an input containing a set of raw requirements, asystems analyst abstracts the required information and models it according to any appropriate models in the light of thechosen software development methodology. For instance, in object-oriented software development, an analyst deciphers a set of requirements and represents the abstracted information in a series of diagrams (e.g. use case, class, sequence diagram etc.) according to the Unified Modelling Language (UML) notation[2]. This whole process is entirely dependent on the expertise of the systems analysts and the domain experts involved. Eventually, the software programmer implements the system based on these abstracted diagrams.

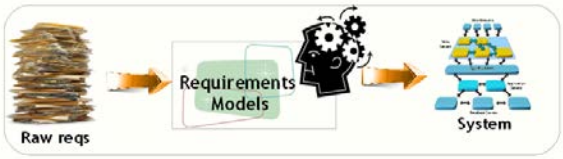

Fig. 1: Current scenario

[1] Manual here means derived from scratch or semi-automated.
[2] UML is the de-facto standard of object-oriented modeling notation.

During the modeling process, crucial information that influences the design decisions might have been overlooked or disregarded. In practice, the modeling process may involve more than one systems analyst, not to mention a number of domain experts. The complexity of this process is further increased by the number of diagrams involved. While producing and maintaining the evolution of this set of diagrams, traceabilities between each modeling element and the original requirements have to be fully addressed. These scenarios are part of the common issues in RE processes and they are considered sufficient to demonstrate the complexity of the processes. Regardless of these issues, however, it is important to ensure that crucial information is not lost during the early stages as it is the basis of information for design decisions in the later stages.

This raises the questions of (1) how and to what degree of accuracy can these abstraction processes fully elicit, capture and satisfy the entire set of requirements; (2) how the risk of information lost during such an abstraction process can be tackled especially when the models are being transformed from one another in each phase of the development life cycle, and further being handled by different stakeholders in the development team; and (3) how these processes can address scalability when dealing with many stages of transformations while having to keep abreast of the relationships among the software artefacts.

A considerable number of existing approaches that deal with handling requirements with regards to architecture have been investigated. Through an analysis of these approaches, a category of mapping approaches is defined. Fig. 2 depicts the idea of the defined category. The category is establishedbased on (1) how these approaches deal with requirements; and (2) how these approaches model information from the requirements into systems architecture.

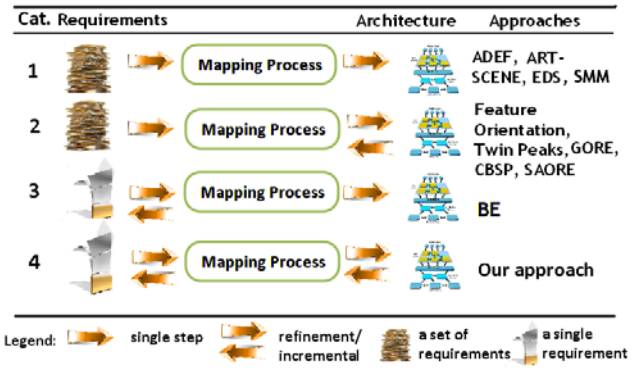

Fig. 2: Summary of Existing Mapping Approaches

For the first criterion, approaches that deal with each single requirement or deal with an entire set of requirements are being investigated. As presented in Section I, an approach that can systematically deal with each single requirement can at least reduce the possibility of disregarding any useful information stated in the requirement. The current state of requirements elicitation and analysis processes are solely subjected to the abstraction made based on the judgment and expertise of the systems analyst. This abstraction process may also run the risk of losing useful details from the requirements. In order to overcome such an issue, various types of traceability techniques are introduced in the literature [1], [2], [3], [4], [5], [6]. Accordingly, more cost and effort are required to produce traceability links and to maintain them.

For the second criterion, approaches that build systems architecture in a single step and approaches that build systems in incremental steps were compared. Building systems architecture in a single step is an approach that analyzes the relevant information (derived from the mapping process, see Fig. 2) and can directly derive a system architecture. On the contrary, the other approaches build the system architecture incrementally or by refinement processes. Through the refinement process, abstracted analysis and design models will be refined into detailed design. An essential strategy in the object-oriented analysis and structured analysis is the decomposition of a problem into smaller and manageable units[3]. On the other hand, an incremental approach allows us to deal with any number of requirements, and therefore it should scale up to arbitrarily large requirements documents.

[3] This strategy is known as the *diide-and-conquer* technique.

These mapping approaches will be discussed as based on the defined category shown in Fig. 2. The category column represents the following:(1) approaches that deal with all requirements and build the system architecture in a single step; (2) approaches that deal with all requirements and build the system architecture incrementally (or by refinement); and (3) approaches that deal with a single requirement at each incremental step and build the system architecture in a single step.

It is worth noting that the details of these approaches will be sufficiently discussed in the context of how these categories were defined i.e. how these approaches deal with functional requirements, and how these pproaches handle and derive the systems architectures.

It is useful to distinguish between the terms 'refinement' and 'incremental' used in this category. By refinement, it is used to describe the process of detailing the design specification. In literature, such a refinement process is also known as forward engineering [7]. This process is normally applicable in a top-down approach where an abstraction concept is identified, and details are added to the corresponding models. For instance, in a feature-orientation approach, each feature is an abstraction of the generic properties of a system being modeled. This feature needs to be refined to achieve the designated design artefact. On the other hand, by 'incremental', it is used to acknowledge the process of adding increments to the architecture or system but at the same time preserving the formerly incremented behaviour without compromising its correctness. A refinement can be, but is not necessarily, incremental. Typically, a refinement process is a top-down activity, while an incremental process supports bottom-up activities.

A. Category 1: Deal with all requirements and build the system architecture in a single step

In this category, the entire set of requirements is analyzed and relevant information based on the corresponding selected domain models is extracted. In these tasks, the expertise of the systems analysts and the domain experts is obviously required. The produced set of domain models will be used as a basis for later software development stages. All the required information is elicited, analyzed and modeled (in addition to the existing domain models from the earlier stages), systems architecture will also be derived in the same manner.

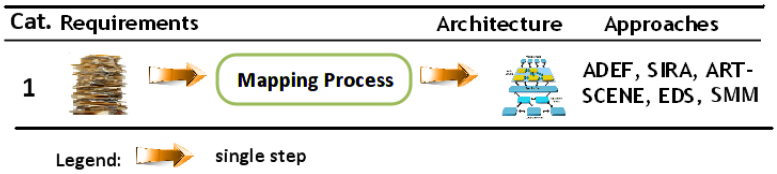

Figure 3: Category 1

Among the examples are the Architectural Decision Elicitation Framework (ADEF) [8], the Analysing RequirementsTrade-offs - Scenario Evaluations (ART-SCENE) [9], the Extended Design Spaces (EDS) [10] and the Scenario and Meta Model-Based Approach (SMM) [11].

The following sections include an introduction to each of the approaches in this category, and continuing with an analysis of each of the approaches based on the theme of the defined category.

1. *Architectural Decision Elicitation Framework:* The Architectural Decision Elicitation Framework (ADEF) [8] approach provides not only design guidance in capturing architectural decisions from the requirements using a rule-based implementation, but also contains automatic reasoning capability for making architectural decisions and resolving conflicting decisions. The ADEF approach comprises two main modules, namely (1) *reasoning* and (2) *presentation* modules, as depicted in Fig. 4.

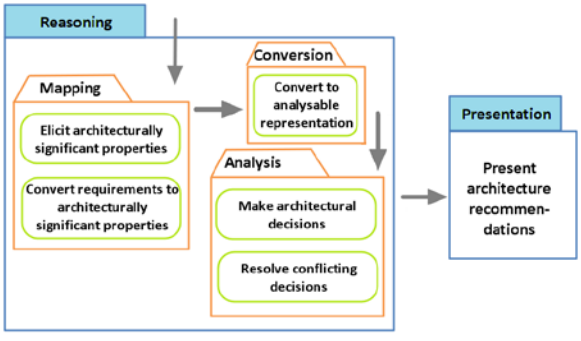

Fig. 4: ADEF[8]

The *Reasoning* module encapsulates the decision making knowledge and justifications on the requirements elicitation to be mapped to any relevant architectural decisions. This module contains three sub modules which are: (1) Mapping; This sub-module uses built-in decision trees represented in directed acyclic graphs to provide guidance to the user in manually mapping each requirements specification to one or more architecturally significant properties; (2) Conversion; During the conversion process, the decision units will be converted to a form which can be interpreted by the analysis of the sub module. These decisions are stored as facts; (3) Analysis; In analysis, an automated reasoning is provided in order to make the architectural decisions and to resolve conflicting decisions. Finally, the *Presentation* module acts as the presentation layer to the user and also update any changes made to the preceding results.

Analysis.: The ADEF approach adopts straightforward transitions by analysing the entire set of requirements, and relates the mapped architectural significant properties into architectural units (e.g. component, connector or bus, system). Clearly, the construction of an architecture of a system is performed in a single step. For this reason, this approach is classified into the first category.

2. *Analysing Requirements Trade-offs - Scenario Evaluations (ART-SCENE):* The ART-SCENE approach is designed with the intention of utilising an integration of the existing tools and techniques to relate requirements and architecture [9]. By integrating the chosen RE techniques and tools, the ART-SCENE approach claims to provide a comprehensive *plug-and-play* approach in exploring the requirements and architecture trade-offs with different scenarios.

This approach advocates scenario generation, architectural model and agent-based simulation to explore architectural models for various scenarios [9]. The researchers claimed that scenarios are essential in integrating requirements and architectures. Scenarios, in this approach, are used to realize requirements and to simulate the choice of architectural design, which are then validated for compliance with the requirements as shown in Fig. 5.

Analysis.: In the ART-SCENE approach, the whole set of requirements is analyzed and modeled into a group of scenarios. While abstracting the requirements into scenarios and modeling them into use cases, important requirements or information might be lost. This design decision is based on the expert judgment and intuition of the involved systems engineer or the systems analyst. Based on this abstracted information, the engineer or analyst models the system requirements using the *i*notation*[12]. Architecture, is then designed (without refinement) and simulated using the *AgentSheets*[13], [14] tool.

3. *Extended Design Spaces (EDS):* The EDS is a semi- formal technique, which aims to achieve reusability in CBSE by providing a set of processes from requirements capturing to configuration [10]. The authors argue that to foster component reuse, software development processes should be geared to consider reusability aspects during the processes.

The EDS process model starts by mapping requirements to components. See Fig. 6. In many of the existing RE approaches, requirements are stated without any consideration of the architectural designs. To some extent, this is the desirable RE feature, because RE is expected to cover the what part of the problem, and not the how part of the solution. However, the drawback of this is that from the reusability aspect, the developers need to manually decide how to bridge these gaps [10]. In order to take advantage of this situation, the EDS provides guidance in the form of a checklist, which contains suggested types of requirement according to the chosen type of architecture.

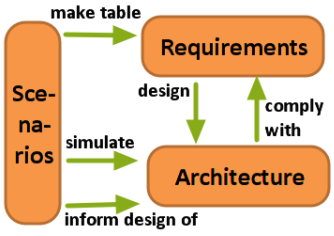

Figure 5: Relationships between scenarios, requirements and architectures [9]

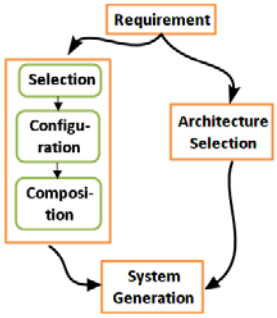

Fig. 6: Simplified EDS Process Model [10]

Firstly, a DS (Design Space) profile is created. The outcome of this is the Requirement DS (RDS), which comprises a list of functional and non-functional aspects of the system. The application DS (ADS) captures the properties of the application, while the RDS describes the requirements in the perspective of the run-time platform. The subsequent step is to map the ADS to the RDS.Some of the ADSs are not directly mapped to

the RDS. Hence, some transformation rules are required. These rules are derived through the correlation between the two spaces.

Secondly, the DS profile will be used to select the architecture. An architecture, in this context, specifies the design decisions and component compatibility. Components that are built for a specific type of architecture share the same assumptions about the design decisions. There is no doubt that the decision about the most appropriate architecture largely depends on the expertise of the software architect. Nonetheless, by using the correlation results from ADS and RDS, the choice of architecture can be justified.

Thirdly, components are selected, configured and composed. Component selection for reuse purposes is done based on the component profiles. The component selection process will then compare the desired DS profile with the provided DS profile from the repository. If there is no match, then a new component will need to be developed from scratch. The component configuration provides the means to determine the parameter settings of the components which are retrieved from the repository. Complex parameters that cannot be handled using the DS technique will need to be manually resolved (e.g. informal hints).

Finally, when all the desired components have been retrieved and configured, a generator uses the components together with their parameter settings and translates them into implementations [10].

Analysis.: The EDS approach deals with the whole set of requirements in an attempt to map functional and non- functional requirements into an architecture. On the other hand, the construction of the architecture is done in a single step once the desired components have been discovered. With this justification, we assign the EDS approach to the first category.

4. *Scenario and Meta-Model-Based Approach (SMM):* The SMM approach defines the gap between requirements and architecture as a structural gap [11]. The existence of this gap is claimed because of the large conceptual distance between the terms used in both of the stages. A requirement may be modelled into one or more components, whereas one component may relate to one or more requirements. Based on the claim that scenarios can be used as a means to elicit the inter-relationships between requirements and architecture, in relating both of the stages, the SMM approach adopts scenario integration in these stages. Nonetheless, the main aims of this approach are to establish traceability from requirements to implementation and to support change integration.

Meta-models contain a set of descriptions to capture the modeling elements used in a model-driven environment. In this approach, the meta-models are represented using a UML class diagram. These meta-models can be specified into domain-specific meta-models by adding specialised domain concepts. Altogether, six meta-models have been defined to capture the modeling elements with regard to relating the requirements and architecture. These meta-models are illustrated in Fig. 7.

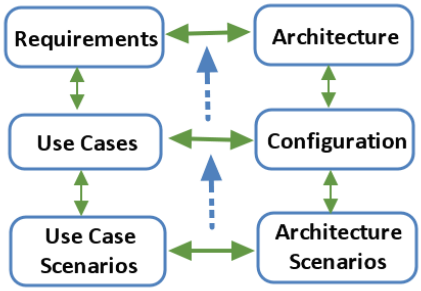

Fig. 7: Scenario Meta Model [11]

During the requirements stage a requirements meta-model identifies FRs and NFRs; a use case meta-model defines goals, external actors, triggers, pre-conditions, post-conditions, results and use case scenarios; a use case scenario meta- model includes the satisfy use case scenario (i.e. primary and alternative) and exception use case scenario, and scenario steps (i.e. action and communication).

During the architecture stage, architectural meta-model specifies internal components and connectors. An architecture consists of a collection of components and connectors. An architectural configuration meta-model identifies the configurations of the components and connectors. An architectural scenario meta-model represents the dynamic behaviour of the architecture. It is a realisation of some use cases, in which the architecture scenario steps (i.e. action and communication) are defined.

Analysis.: The SMM approach deals with the whole set of requirements in order to specify scenarios. These scenarios are used to identify the elements of the defined meta-models in order to look for architectural elements. The architecture of the system is derived once all the architectural elements are discovered. Hence, we classify this work in the first category. In addition, we believe that the abstraction process of going from requirements to scenarios, and the process of identifying elements of the meta-models are based onn the

intuition and expert judgment of the analyst(or the designer), instead of relying on the methods that this approach has to offer.

B. Category 2: Deal with all requirements and build the system architecture incrementally or through refinement

The approaches in the second category deal with the entire set of requirements, which has the same input as the first category (see Fig. 8). Nonetheless, in contrast to the first category, the construction of the architectre is performed by means of refinement or incremental process. In this section, how these approaches handle requirements, and how the architecture is derived from requirements, will be elaborated.

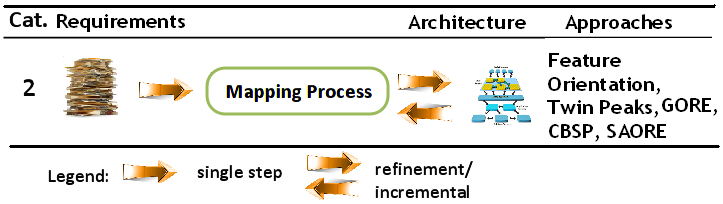

Fig. 8: Category 2

The ability to derive architecture incrementally while at the same time ensuring the incremented behaviours are maintained without compromising its correctness has a significant and promising impact. The essence of incremental is the fact that it can address the issue of scalability. When progressing with small increments, the architecture can be built up and eventually the final architecture can be derived. Each increment has already been validated and hence, less or ideally no rework effort has to be invested whenever additions are made.

On the other hand, construction of an architecture by means of refinement can also build the architecture hand-in-hand with the requirements. As stated earlier, refinement does not necessarily support incremental. This in turn reflects that preservation of behaviours in the former refinement steps may not be guaranteed. The examples of approaches that are considered in this category are the Feature Orientation (FO) [15], the Twin Peaks (TP) [16], the Component-Bus-System-Property (CBSP) [17], [18], the Goal-Oriented Requirements Engineer- ing (GORE) [19], [20], [21], and the Software Architecture- Oriented Requirement Engineering (SAORE) [22].

1. *Feature Orientation (FO):* The FO-based approaches intended to develop a family of applications for a specific domain instead of for a single application [15], [23], [24], [25], [26], [27], [28]. A feature is defined as “a logical unit of behavior that is specified by a set of functional and quality requirements” [29]. The core technique in FO is the analysis of variability and commonality of an application domain (i.e. the feature analysis) [27], [28]. These features are then transformed into feature models (see Fig. 9), defined in the Feature Oriented Domain Analysis (FODA) technique and used as an input to a series of refinement processes in order to derive an architecture.

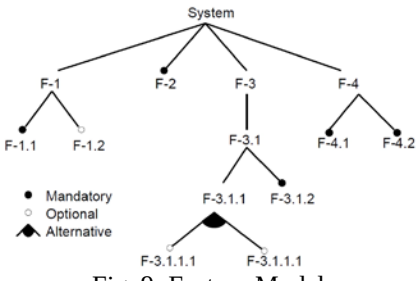

Fig. 9: Feature Model

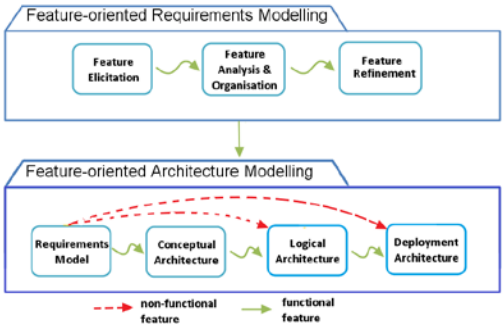

Fig. 10: Feature Mapping [15]

However, the FODA technique does not provide explicit interaction between features and its corresponding architectural elements [23]. Both the FODA and its modeling technique, namely the Feature-Oriented Requirements Modeling (FORM), remain vague on the matter of mapping feature models to

architectural elements [23]. In order to overcome this issue, the feature-oriented mapping process is proposed [15]. The process comprises two main stages as shown in Fig. 10.

The stages are (1) Feature-Oriented Requirements Modelling (FORM) and (2) Feature-Oriented Architectural Modelling (FOAM). The aims of the FORM stage are (1) to capture the problem domain requirements and transform them in the context of features and their relationships (i.e. during the elicitation process); (2) to analyse and transform the features into a tree form (i.e. during the organisation and analysis process (see Fig. 9)); and (3) to refine the features into a detailed set of functional requirements (i.e. during feature refinement).

Moreover, in the second stage, namely the FOAM stage, the outcome of the FORM stage serves as an input to this stage in order to derive an architecture. A functional feature can be mapped to a subsystem or a component. In FOAM, mapping is addressed in three layers, which are conceptual, logical and deployment architecture as illustrated in Fig. 10. In the conceptual architecture, only functional features are considered. By taking implementation and non-functional features into consideration, these features are detailed for the logical architecture. Finally, the deployment architecture represents how the features are distributed and communicated among computational nodes in the deployment environment.

Analysis.: These FO approaches deal with an entire set of requirements and abstract them as features. This abstraction of features together with their relationships is represented as a feature model. Moreover, the construction of an architecture is refined into a series of conceptual, logical and deployment architectures. Clearly the FO-based approaches work on the basis of dealing with the entire set of requirements and transforming the abstracted information into an architecture through the refinement process. For this reason, we assign the FO approach into the second category.

2. *Twin Peaks*: The Twin Peaks (TP) model adopted the Problem Frames (PF) [30] approach. Thus, some foundations of the PF approach will be briefly introduced. The PF approach attempts to decompose problems into a collection of interacting sub-problems and to manage the separation of the real-world problems into the problem and solution spaces [30]. By having such a separation, a developer can focus on the problem structure while attempting to solve the real-world problem rather than trying to find a solution to the problem. Nevertheless, coming up with the problem domain without considering the solution domain is, if not impossible, rather a difficult task.

In order to reduce the gaps between problem and solution spaces, and to allow requirements and design opportunities to be explored, current software development should exploit the relationships that exist between these spaces [31]. This approach defines frames to a particular problem in which each frame has an associated frame concern, and does not address the concerns of the software architects [16]. This frame concern identifies the correctness argument of the frame. By excluding these concerns, the development process is less likely to be shortened [16]. Hence, the PF approach is extended to consider the solution spaces as a part of the problem domain [16].

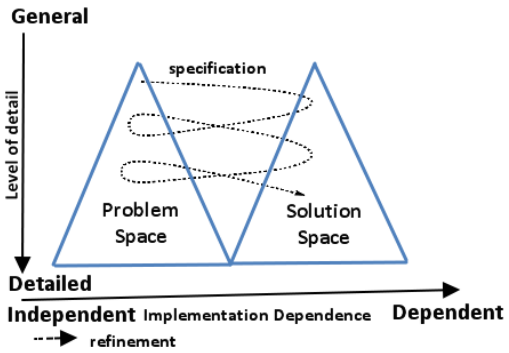

Fig.11: The Twin Peaks Model[31]

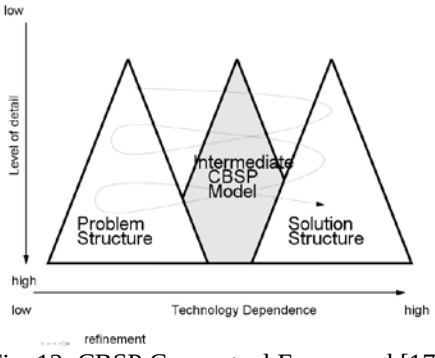

Fig. 12: CBSP Conceptual Framework[17]

In relating both of the spaces, the iterative nature in software development is also taken into consideration, from the general to a more detailed design solution. During this process, some architectural support decisions are used in the problem space. In order to address the issue, the TP model highlights the equal importance of requirements and architectures by separating the problem space and the specification from the solution space [31], [16].

The specification comprises detailed requirements and design specifications which are produced by means of refinement. Here, the refinement is confined to the process of detailing the specification, hand-in-hand with the requirements. This concept is illustrated in Fig. 11. The model mainly addresses the problems dealing with evolving user requirements, applying and managing COTS software, and handling rapid

changes [31]. This approach adopts a causal logic event model [32] to express the event notation, and also applies an informal imperative pseudo code to describe behaviours.

The first TP mapping step is to decompose the problem into independent sub-problems i.e. not having any relationship with any other problem or sub-problems. Each sub-problem will be represented according to the chosen notation. Once all of the sub-problems have been modelled into the commanded behaviour frame notation, the requirements will be represented in the form of causal logic event model expressions.

Analysis.: The TP approach deals with the entire set of requirements, and refines the specification by attempting to reduce gaps between the problem and solution spaces. Approaches in the second category handle requirements by processing the whole set of requirements. The fact that the requirements specification is refined in TP indicates that the mapping process itself addresses such a refinement process. Nonetheless, the requirement must be analyzed as a whole, rather than dealing with each requirement at a single point in time. The second consideration is regarding the construction of the architecture. In TP, architectures are derived through the refinement process, hand-in-hand with the refinement of requirements.

3. *Component-Bus-System-Property (CBSP)*: The CBSP is defined as "a lightweight approach which provides a systematic way of reconciling requirements and architecture using intermediate models" [17], [18]. This generic approach adopts the Twin Peaks Model [31], [16] to exhibit the iterative process between requirements and architecture. In addition to this, an intermediate model is introduced between these stages, as shown in Fig. 12. This model helps to decide on a suitable architectural style to be adopted as a basis of transforming the draft architecture to an actual software architecture implementation [17].

The quintessential concept in CBSP is that each requirement will implicitly or explicitly relate to the software architecture elements. The CBSP defines dimensions which consist of a set of architectural elements (i.e. components, connectors (buses), topology and their properties). These dimensions are used to classify and refine the requirements in order to capture architectural-related issues. As depicted in Fig. 13, the mapping process begins with requirements selection. The selection is a voting-based process where requirements with low priorities will be eliminated. This voting process is conducted by stakeholders of the software project team.

Accordingly, the main criteria for voting are the importance and feasibility of the requirements. The second step is to classify the requirements into architectural constructs. This step is performed by experts. In order to classify the constructs, the CBSP dimensions are introduced. The CBSP approach specifies six dimensions to be applied to the basic architectural constructs. These dimensions represent elements that imply (1) processing or data components; (2) a connector or bus; (3) features of a subset of the components and connectors; (4) NFR aspects of a component; (5) NFR aspects of a connector; and (6) system (sub-system) properties. These dimensions are used to refine the requirements in natural language into CBSP model elements. The experts will then examine each of the requirements and vote on the relevance based on the CBSP dimensions.

The third step is to identify and resolve mismatch classifica- tion issues. This is where any conflicting perception during the classification in the previous steps will be handled. Again, a voting process is adopted. If there is still no consensus among the experts, a discussion is needed so as to avoid the conflicts and to achieve the final decision.

In the fourth step, each of the requirements will be refined based on overlapping CBSP properties and concerns. Finally, at this step it is assumed that all the conflicts have been resolved. Based on the CBSP model elements a proto- architecture will be derived. This is also the point where architectural styles (e.g. client-server, event-based, layered and pipe-and-filter architecture) [33] will be used to achieve the desired system qualities. This process is heuristically performed, with the justification that there are also possibilities that more than one architectural style is needed or preferable. By using the elements in the selected architectural style(s), the CBSP model elements will be transformed into components, connectors, configurations and data.

Analysis.: The CBSP mapping process is an iterative process that is used to refine the draft architecture model derived from a given set of requirements. In the solution structure, the architecture is matched with the existing architectural styles. This architecture is refined iteratively until the completion of the model. The CBSP is considered under this category because although each requirement is voted, and the CBSP elements are identified, the architecture is not created based on the behaviours that directly originated from requirements. Instead, the chosen architectural style is refined, together with the original requirements as illustrated by the spiral dotted line in Fig. 12.

4. *Goal-Oriented Requirements Engineering (GORE):* A goal is "a prescriptive statement of intent about some system whose satisfaction in general requires the cooperation of some of the agents that form the system" [34]. These goals cover both functional and non-functional goals (NFG). In brief, this approach starts by providing a process to produce software specification based on a set of requirements. From the

functional specification, an initial abstract architecture is drafted. Following this, the architecture is refined to meet the domain-specific architectural constraints.

The GORE approach adopts Knowledge Acquisition in Automated Specification (KAOS) methodology [19], [20], [21] in addressing the architectural design based on the requirements. The KAOS methodology specifies an ontology for capturing requirements [21]. The relevant part of this approach is the architecture derivation process, which happens after the requirements are modelled, specified and analyzed. Such a derivation process (1) provides a systematic guidance to software architects; (2) is incremental; (3) leads to an architecture that satisfies the functional and NFR; and (4) allows different architectural views (e.g. performance view, security view, etc.) [34].

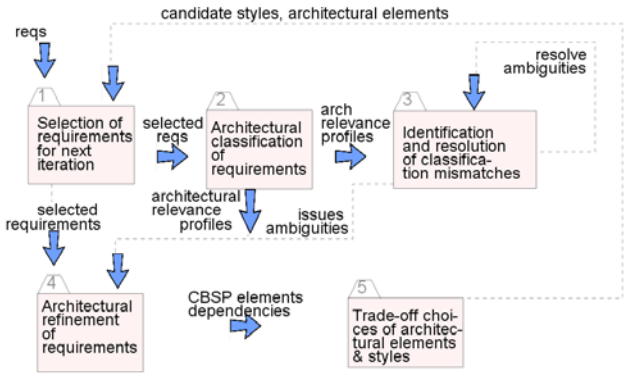

Fig. 13: CBSP Process [17]

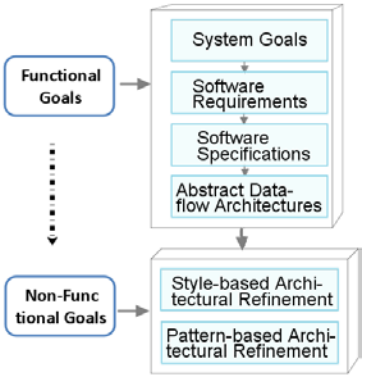

Fig. 14: GORE Process

The GORE process can be abstractly viewed according to the functional and NFG derivation processes. As shown in Fig. 14, for the functional goals, all the relevant documents are referred to in order to identify system goals. Based on these system goals, software requirements are identified. During the following step, the whole set of requirements is analyzed and software specifications are further derived. Finally, the abstract data flow architectures are derived, based on the software specification.

Analysis.: In the GORE approach, the architecture refinement process is specifically required to include the NFG. In the context of functional properties or behavioural properties, the refinement process is not required. However, without the NFG, the derivation process might not be complete. For this reason, this approach is classified in the second category.

5. *Software Architecture-Oriented Requirement Engineering (SAORE):* The SAORE approach attempts to relate the requirements and architecture stages[22]. In essence, the approach views the architecture model as the reference model in the software development process. The main idea of this approach is to introduce elements of software architecture (SA) in requirements analysis and specification. The SAORE approach identifies two main constituents: components and connectors. A component is defined as an element which encapsulates a set of coherent functionalities, which is expected to perform the desired computational units. A connector, on the other hand, defines the interaction between components. The connector is treated as a first-class entity not only in the problem space, but also in the solution space. Examples of the connectors are procedure calls, file Input/Output (I/O), remote procedure calls (RMC) and client-server.

The SAORE process [22] comprises a series of iterative activities, as illustrated in Fig. 15. The starting point of the process is to determine the boundary of problem spaces i.e. the scope of the system to be built. Following this, stakeholders of the system and their roles are identified. The information is useful to determine the external relationships of the system and also to partition the system into sub-systems or components. In the next step, a top level system model will be produced using a Use Case model. The challenging step here is how to identify the components and connectors from the elicited information.

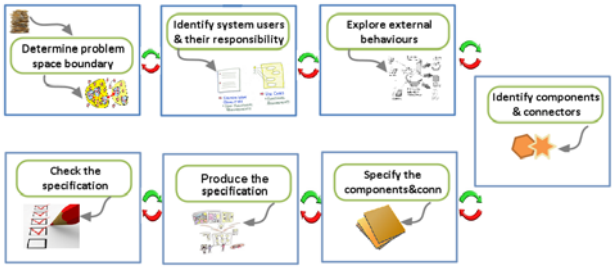

Fig. 15: SAORE[22]

Analysis.: The SAORE approach deals with the entire set of requirements and analyzes them. The components are extracted from use cases or any element that contributes system behaviours, whilst the connectors encapsulate relationships between components. Consequently, the requirement models are derived through a series of refinement steps and this is why the approach is addressed as in the second category.

C. Category 3: Deal with a single requirement at each step and build the system architecture in a single step

In contrast to dealing with the entire set of requirements, the approach in this category deals with each requirement and processes the requirements incrementally. As a result, the architecture is constructed once the complete design specification is derived. This process is therefore done in a single step.

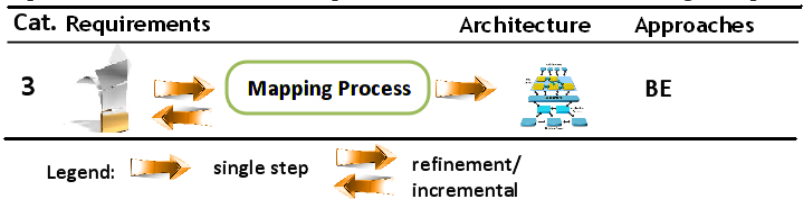

Fig. 16: Category 3

To the best of our knowledge, as shown in Fig. 16, the only work that fits into this category is the Behaviour Engineering (BE) approach.

1. *Behaviour Engineering*: Behaviour Engineering (BE) establishes an approach for building large-scale systems from natural language requirements. The principle of the BE approach lies in its concept of building design out of requirements instead of satisfying the requirement [35]. It defines the Behaviour Modeling Process (BMP) which comprises a set of processes to support Behaviour Modeling Language (BML). The BML consists of a series of behavioural diagrams including Behaviour Trees (BT), Integrated (IBT), Model (MBT) and Design (DBT) [36], [37], [38], [39], [35], [40], [41]. A BT translates individual (raw) requirements into behaviour trees.

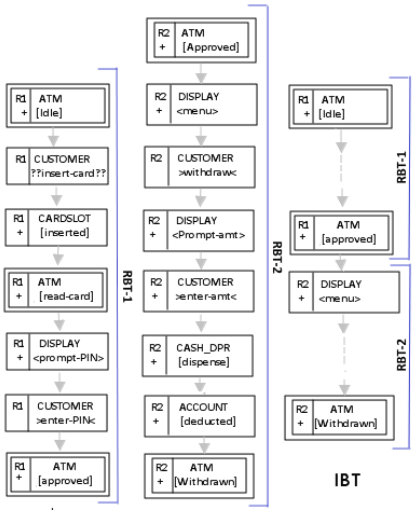

Fig. 17: Behaviour Trees

A BT is "a tree-like graph that represents the behaviour of a set of entities which realise or change states, make decisions, respond to or cause events, and interact by exchanging information and/or passing control" [35].

Behaviour trees for all the individual requirements (called requirements behaviour trees (RBTs)) are merged into an IBT that describes the required behaviours of the whole system. Fig. 17 shows examples of the RBTs for two requirements R1 and R2 of an ATM Example and their merged IBT. The IBT is used to integrate all the RBTs. The following diagram, that is the MBT, is used to discover and fix defects on issues such as inconsistencies, ambiguities, etc. As a result of the defect correction, a design behaviour tree (DBT) is produced.

The DBT is again refined using the provided design decisions. This refinement process clarifies the system's boundaries. From the DBT of the whole system, a component diagram [37], [36] is derived, together with the behaviours of individual components.

Analysis.: According to the approaches category, BE clearly deals with each single requirement by producing these RBTs. Nevertheless, when the complete DBT is produced through refinement, only then is the component diagram constructed. This process is performed in a single step. In such a case, iteration or refinement from design to the system architecture diagram is not required. For these justifications, the BE is considered as in the third category.

D. Category 4: Deal with a single requirement and build the system architecture in each incremental step

Based on the analysis so far, there is a gap in the way that we can handle requirements and at the same time utilize the benefit of incremental approach towards building system architecture. In this new category, each requirement is analyzed one-by-one and the result of the analysis is used to aid in creating an architecture incrementally (in contrast to in a single step or through a series of refinement processes). To the best of the author's knowledge, none of the existing approaches in the literature offer such a strategy. As a consequence of dealing with each requirement and incrementally building the architecture, this kind of approach can utilise the benefits from both strategies. Firstly, the use of intuition and abstraction can be reduced by dealing with each requirement. In addition, the issue of scalability can be addressed as the approach is designed to accommodate any number of requirements. Secondly, the incremental construction of architecture/system also supports scalability with the ability to support the preservation of behaviour of the incremented architecture.

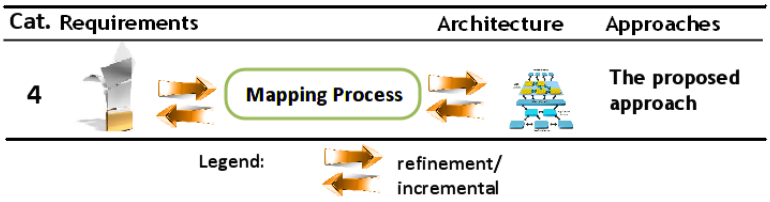

Fig. 18: Category 4

2. THE PROPOSED APPROACH

The proposed approach is to extract elements of component-based systems from raw requirements, map them directly into the elements of the desired system, and thereby construct the system. The aim of this approach is to maximize the chances of achieving a better match between the final system and the raw requirements. It is believed that by having support from a suitable component model, such an aim could be achieved.

It is necessary to consider if we deal directly with each requirement, analyse and model them without having to deal with the whole set of requirements as conceptualised in Fig. 18. Such a strategy assists in relatively reducing the complexity that the systems analysts or the domain experts need to handle. Nonetheless, we are not arguing that dealing with each requirement can directly lead to simplicity. The challenge with this kind of approach is how we can effectively compose the mapped architecture for each requirement in each increment. As the main theme of this paper is to provide the motivation of the proposed approach, the details of the approach will not be included. Nonetheless, further information can be accessed from these papers [42], [43].

Reflecting on the category of approaches to construct architecture from requirements, the significant benefit of having these two strategies in an approach is mainly to address scalability. In principle, we may handle a larger set of requirements when we deal with a single requirement at a time and incrementally build up the system architecture.

3. CONCLUSION

The problem with the existing approaches in the literature in order to solve the gap between problem and solution, be it using structured or object-oriented approach, is that they provide insufficient emphasis on software architecture elements [15]. As a result of this, transformation from requirements to design is arbitrarily performed based on the intuition or judgment of the involved software engineers.

These gaps between requirements and architecture exist because of the large conceptual distance between the terms used in both stages [11]. In many of the existing RE approaches, requirements are stated short of consideration of the architectural designs. To some extent, this is the desirable RE feature because RE is expected to treat the *what* part of the problem, and not the *how* part of the solution. However, the drawback of this feature is that from the reusability aspect, the developers need to manually decide how to bridge these gaps [10].

In addition to scalability, by adopting a component-based approach, we could reuse pre-existing components from component repositories; in contrast to building new parts of the system for each software development project. This factor leads to reducing effort, time and cost as compared to building new parts of the system from scratch. We argue that by having an underlying component model that supports incremental

composition, such an objective can be accomplished. In order to realise the objective, an approach founded on a specific component model should provide a plausible transition from requirements to system architecture for a component-based system.

ACKNOWLEDGEMENTS

The analysis has partly contributed to the author's PhD work at the Department of Computer Science, University of Manchester. The author would like to thank Dr. Kung-Kiu Lau for his comments and efforts in supervising the work. The author would also like to acknowledge the Malaysian Ministry of Higher Education (MOHE) and International University of Malaysia (IIUM) for sponsorship during the study.

REFERENCES

[1] Gotel OCZ. Contribution Structures for Requirements Traceability. University of London; 1995.
[2] Ramesh B, Jarke M. Towards reference models for requirements traceability. IEEE Transactions on Software Engineering. 2001;27(1):5.
[3] Antoniol G, Canfora G, De Lucia A. Maintaining Traceability During Object-Oriented Software Evolution: A Case Study. In: ICSM '99: Proc. of the IEEE International Conference on Software Maintenance; 1999. p. 211–219.
[4] El Emam K, Madhavji NH. A field study of requirements engineering practices in information systems development. In: RE '95: Proc. of the 2nd IEEE International Symposium on Requirements Engineering. York, UK: IEEE Computer Society Press; 1995. p. 68–80.
[5] Hayes JH, Dekhtyar A, Sundaram SK. Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. IEEE Transactions on Software Engineering. 2006;32(1):4–19.
[6] Vasilecas O, Caplinskas A, Wojtkowski G, Wojtkowski W, Zupancic J, Wrycza S, editors. Customizing Traceability in a Software Development Process. Information Systems Development: Advances in Theory, Practice, and Education. USA: Springer; 2005.
[7] Egyed A, Medvidovic N. Round-Trip Software Engineering Using UML: From Architecture to Design and Back. In: Proc. of the 2nd Workshop on Object-Oriented Reengineering WOOR; 1999. p. 1–8.
[8] Liu W, Easterbrook S. Eliciting Architectural Decisions from Requirements using a Rule-based Framework. In: STRAW '03: Proc. of the 2nd International Workshop From Software Requirements to Architectures. Portland, Oregon; 2003. .
[9] Zhu X, Maiden N, Pavan P. Scenarios: Bringing requirements and architectures together. In: SCESM '03 : The 2nd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools. Portland, Oregon, USA; 2003. .
[10] Baum L, Becker M, Geyer L, Molter G. Mapping requirements to reusable components using Design Spaces. In: ICRE '00 : The 4th IEEE International Conference on Requirements Engineering. Schaumburg, Illinois; 2000. p. 159–167.
[11] Pohl K, Brandenburg M, Gülich A. Integrating Requirement and Architecture Information: A Scenario and Meta-Model Based Approach. In: REFSQ '01: Proc. of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality. Interlaken, Switzerland; 2001.
[12] Maiden N, Pavan P, Gizikis A, Clause O, Kim H, , et al. Making decisions with requirements: Integrating I* Goal modelling and AHP. In: REFSQ'02: Proc. of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality. University of Essen, Essen, Germany; 2002. p. 24–35.
[13] Huber BSa. AgentSheets Reference Manual. AgentSheets, Inc.; 2001. Available from: http://www.agentsheets.com/Documentation/windows/ Reference-Manual.pdf.
[14] Repenning A, Ioannidou A, Zola J. AgentSheets: End-User Programmable Simulations. J Artificial Societies and Social Simulation. 2000;3(3). Available from: http://dblp.uni-trier.de/db/journals/jasss/jasss3.html#RepenningIZ00.
[15] Liu D, Mei H. Mapping Requirements to Software Architecture by Feature-Orientation. In: STRAW' 03 : Proc. of the 2nd International Software Requirements to Architectures Workshop. Portland, Oregon; 2003. p. 69–76.
[16] Hall JG, Jackson M, Laney RC, Nuseibeh B, Rapanotti L. Relating software requirements and architectures using problem frames. In: RE '02: Proc. of the 10th IEEE Joint International Conference on Requirements Engineering. Essen, Germany; 2002. p. 137–144.
[17] Grünbacher P, Egyed A, Medvidovic N. Reconciling software requirEments and architectures with intermediate models. Software and System Modelling. 2004;3(3):235–253.
[18] Grünbacher P, Egyed A, Medvidovic N. Reconciling Software Requirements and Architectures: The CBSP Approach. In: RE '01: Proc. of the 5th IEEE International Symposium on Requirements Engineering. Toronto, Canada: IEEE Computer Society; 2001. p. 202–211.
[19] van Lamsweerde A. Goal-oriented requirements enginering: A roundtrip from research to practice [enginering read engineering]. In: RE '04: Proc. of the 12th IEEE International Requirements Engineering Conference. Kyoto, Japan; 2004. p. 4–7.
[20] van Lamsweerde A. Goal-Oriented Requirements Engineering: A Guided Tour. In: RE '01: Proc. of the 5th IEEE International Symposium on Requirements Engineering. Toronto, Canada: IEEE Computer Society; 2001. p. 249.

[21] Darimont R, Delor E, Massonet P, van de A. GRAIL/KAOS: An environment for goal-driven requirements engineering. In: ICSE '97: Proc. of the 19th International Conference on Software Engineering. New York, NY, USA: ACM; 1997. p. 612–613.
[22] Mei H. A complementary approach to requirements engineering Software architecture orientation. SIGSOFT Software Engineering Notes. 2000;25(2):40–45.
[23] Sochos P, Philippow I, Riebisch M. Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the Architecture. In: Weske M, Liggesmeyer P, editors. Object-Oriented and Internet-Based Technologies. vol. 3263 of Lecture Notes in Computer Science. Springer Berlin/Heidelberg; 2004. p. 23–42. Available from: http://dx.doi.org/10.1007/978-3-540-30196-7_11.
[24] Kang KC, Lee J, Donohoe P. Feature-Oriented Project Line Engineering. IEEE Softw. 2002;19(4):58–65.
[25] Dao T, Kang K. Mapping Features to Reusable Components: A Problem Frames-Based Approach. In: Bosch J, Lee J, editors. Software Product Lines: Going Beyond. Lecture Notes in Computer Science. Springer Berlin / Heidelberg;. p. 377–392.
[26] Sochos P, Riebisch M, Philippow I. The feature-architecture mapping (FArM) method for feature-oriented development of software product lines. In: ECBS '06: Proc. of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems; 2006. p. 308–318.
[27] Lee K, Kang KC, Chae W, Choi BW. Feature-based approach to object-oriented engineering of applications for reuse. Softw, Pract Exper. 2000;p. 1025–1046.
[28] Lee K, Kang KC, Lee J. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: ICSR-7: Proc. of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools; 2002. p. 62–77.
[29] Bosch J. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Addison-Wesley Professional; 2000.
[30] Jackson M. Software Requirements and Specifications: A Lexicon of Practice, Principles, and Prejudices. Addison-Wesley; 1995.
[31] Bashar N. Weaving together requirements and architectures. IEEE Computer. 2001;34(3):115–119.
[32] Moffett JD, Hall JG, Coombes A, McDermid JA. A Model for a Causal Logic for Requirements Engineering. Requirements Engineering. 1996;1(1):27–46.
[33] Fielding RT. Architectural Styles and the Design of Network-based Software Architectures. University of California; 2004.
[34] van Lamsweerde A. Formal Methods for Software Architectures. In: From System Goals to Software Architecture. Springer-Verlag; 2003. p. 25–43.
[35] Dromey RG. Architecture as an Emergent Property of Requirements Integration. In: STRAW '03: Proc. of the 2nd International Software Requirements to Architectures Workshop; 2003. p. 77–84.
[36] Myers T. The Foundations for a Scaleable Methodology for Systems Design. School of Computer and Information Technology, Griffith University, Australia; 2010.
[37] Myers T, Dromey RG, Fritzson P. Comodeling: From Requirements to an Integrated Software/Hardware Model. IEEE Computer. 2011;44(4):62–70.
[38] Myers T, Dromey R. From Requirements to Embedded Software — Formalising the Key Steps. In: Software Engineering Conference, 2009. ASWEC '09. Australian; 2009. p. 23–33.
[39] Dromey RG. From requirements to design: Formalizing the key steps. In: SEFM '03: Proc. of the 1st International Conference on Software Engineering and Formal Methods; 2003. p. 2–11.
[40] Gonzalez-Perez C, Henderson-Sellers B, Dromey G. A metamodel for the behavior trees modelling technique. In: Henderson-Sellers B, editor. ICITA '05: Proc. of the 3rd International Conference on Information Technology and Applications. vol. 1; 2005. p. 35–39.
[41] Dromey RG. Engineering Large-Scale Software-Intensive Systems. In: ASWEC '07: Proc. of the 18th Australian Software Engineering Conference; 2007. p. 4–6.
[42] Lau K, Nordin A, Rana T, Taweel F. Constructing Component-based Systems Directly from Requirements using Incremental Composition. In: EUROMICRO '10: Proc. of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications. Lille, France; 2010. p. 85–93.
[43] Lau K, Nordin A, Ng K. Extracting Elements of Component-based Systems from Natural Language Requirements. In: EUROMICRO '11: Proc. of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications. Oulu, Finland; 2011. .