Abstract

Microservices are considered a hot trend, with every company that implements it having an exclusive blog detailing every change in its codebase. Its functional decomposition and scaling capabilities coupled with autonomous development makes it highly valuable in business comparing to a traditional monolithic approach. This thesis researches the scope regarding the incremental migration from a monolithic to a microservices architecture, merging all background and principles necessary for the development of tools that assist developers on this process while addressing software re-engineering concepts, such as domain-driven design, code analysis and graph clustering.

The proposed method starts with the retrieval of information regarding the relationship between the classes and controllers of an application, subsequently applying hierarchical clustering with a similarity measure based on controller access. This allows us to explore transactional contexts while addressing the migration issue of context separation. The results of this approach, paired with an user interface with visualization capabilities, allows the filling of a gap in terms of support tools that exist for software architecture migrations.

The comparison of our approach with software architecture formal tools and expert decompositions resulted in high precision values, that reflect that accurate service candidates are produced. The approach proposed in this thesis asserts that from the transactional contexts of an application, modular conclusions can be drawn, while also showing that it can better inform developers in the process of an architecture migration to microservices.

Keywords: Monolithic, Microservices, Domain-Driven Design, Static Analysis, Hierarchical Clustering

1. Introduction

A microservices architecture is an architectural pattern composed by a set of services originated from the decomposition of a monolithic software application, being that these services ideally encapsulate different functionalities of the system. The advantages of using this type of architecture are the following:

- **Increase in quality and speed** on the development of applications with a large and complex domain due to the existence of autonomous teams developing different services that integrate into the same system. The presence of small teams allows an agile software development approach and requires only point-to-point collaboration, driven by the interaction between microservices.

- **Ease of implementation** using new technologies, since every service has its own solid boundary, different services can be written in different programming languages and organized around its business capabilities.

- Enables independent and elastic scalability, as each service can be replicated autonomously depending on the workload it is experiencing.

Alternatively, there are monolithic architectures, which are present mainly in enterprise applications and based on a 3-tier approach with a client-side UI, server-side application and database. The monolith is present in the server-side application, as every change in this system will imply the rebuilding and deployment of the entire server. As reinforced by Chris Richardson,

---

while also only allowing horizontal scaling. However, there are authors that defend the monolithic architecture\footnote{https://martinfowler.com/bliki/MonolithFirst.html} in the sense that defining early boundaries for microservices can hinder the development process, while a monolith is change tolerant. When a monolithic system requires the qualities provided by a microservices architecture, there is a need to migrate to this type of architecture. This paper researches the scope of this migration and how to do it incrementally and in a controlled way, defining a workflow to identify boundaries of services.

This stepwise migration should be supported by tools that help the developers in this task. In our case, we developed a tool that tackles the problem of context separation by validating whether the transactional contexts of an application can translate into its modularization, this is, whether the information regarding the relationship between its controllers and classes can group domain classes together as services.

The subsequent sections are going to be summarized as follows. Section 2 presents the state of the art concepts behind architecture migrations and existing proposed solutions, Section 3 describes the implementation of solution and Section 4 the interpretation of the results obtained. Section 5 presents the concluding remarks.

2. Related Work
Industry experts and researchers have tackled the topic of architecture migration, revealing fundamental concepts behind it, as described in the subsequent sections:

2.1. Domain-Driven Design
A large and complex system needs to follow strategic design, so it is able to be easily comprehended and manipulated by developers. Domain-Driven Design exposes a crucial term called bounded context, which delimits a context that a model can be applied, only worrying about the applicability inside its bounds. This concept is directly related to a microservice migration [3] [15], as the boundaries created by bounded contexts are the ones intended for services of this architecture, being small and independent while communicating through its APIs.

2.2. Static Analysis
To capture the relationships between the already existing modules of a monolithic architecture, some type of code analysis is needed, being static analysis can be used for this purpose.

This type of analysis is used in software applications, for example, in the identification of design patterns [20] [2]. In the previous approaches, a graph representation of the application is created where classes are presented by nodes and its relationships by the edges, being that the identification of a design pattern requires a certain number of relationships between specific nodes.

For statically analyzing the source code of an application researchers mostly used three tools:

1. WALA: Executes given a .jar file as an argument and allows the creation of configurations with distinct types of interprocedural analysis, including class hierarchy, call graphs, intermediate representations and system dependence graphs.

2. SOOT: Executes over bytecode or source code and is capable of generating three kinds of intermediate representations, Baf, Jimple and Grimp. Presents functionalities similar to WALA but has a weaker customization potential.

3. Java-Callgraph: Given a .jar file with the corresponding bytecode files, this tool generates a text representation of a call graph, which presents every method call and their related classes and arguments.

On the context of this paper, these tools can be used for the initial information retrieval between the controllers and classes of the software application.

2.3. Dynamic Analysis
Monitoring the execution of a program can help on capturing specific interactions between elements of a software application, which can be done through profiling or software texts (with maximum coverage). Approaches using this type of analysis for design patterns [17] [10] reach the conclusion that a dynamic approach provides an increase in the precision of the results and is usually used in complement with static analysis. The effort here is the generation and selection of test suites.

For dynamic analysis there are tools such as Evosuite [7] [8], with the functionality of automatic test case generations for Java classes and providing several code coverage criteria, including line, branch, exception, exception, weak mutation, output and top-level method coverage.

2.4. Modularity Patterns in code
After analyzing the code of an application using either static or dynamic analysis, the identification of patterns of modularity in the information retrieved can be done. R. Schwanke [19] presents a tool that addresses this modularization, showing that to do so, a clustering method and a similarity measure must be chosen. Reviews of data clustering methods [11] show that algorithms responsible for natural group discovery fall into two categories:
1. **Partitional:** The separation of data into clusters without the knowledge of an hierarchical structure. K-means [9] is an example of a partitional algorithm.

2. **Hierarchical:** Searches for nested clusters, establishing a parallelism with an hierarchical structure. This exploration can be agglomerative, separating each object into its individual cluster and joining those who have a higher similarity, or divisive, grouping all points on a single cluster and iteratively partitioning it. This type of clustering has three types of linkages:
   (a) **Single Linkage:** The similarity between two clusters is the similarity between the two most similar points in that cluster (shortest distance).
   (b) **Complete Linkage:** The similarity between two clusters is the similarity between the two less similar points in that cluster (biggest distance).
   (c) **Average Linkage:** The similarity between two clusters corresponds to the average similarity between each member of one cluster and each member of the other.

From the previous categories, information retrieval using clustering approaches for a better understanding of software architecture or modules is usually done through the use of Hierarchical Clustering [12] [22] [5]. The referred authors use this type of clustering when there is no capability to know the number of clusters that the software system will be divided into. Moreover, these types of systems have a natural hierarchical structure, being that the existence of modules and submodules is a common software practice. Another challenge is defining the similarity measure by which the software components are coupled with and that will be used in the clustering algorithm. On our approach, the class information retrieved from the analysis of the monolithic code needs to be grouped and the number of resulting clusters will be unknown. This research shows that hierarchical clustering is the optimal method to structure the resulting data.

2.5. **Domain-specific languages**

As the software applications that our approach is based on have domain-specific languages there is a need to explain its context and functionality. Arie Van Deursen et al. [21] shows that it is a short language that is applied to a particular domain and offers a high amount of expression by the use of abstractions. On our examined web applications, the domain-specific language present is called DML (Domain Modeling Language), and as presented by J. Cachopo et al. [4], it is used to solve issues that are present in web applications, such as the persistence of states, the concurrent execution of processes and code domain verifications. It allows integration with a Java web application and also the benefit for the developers to focus their attention of designing the system in an object-oriented fashion. A relation keyword defines a relationship between two domain entities, its existing roles and subsequent multiplicity. This code is then compiled, generating two class types:

1. **Base Class:** Represents the structural information of the class, having the correspondent getters, setters and relationship management methods, responsible for keeping these relationships up to date.

2. **Controllable base class extension:** Extends the base class with no predefined implementation, allowing complete manipulation by the developer to add any functionality/methods desired.

2.6. **Existing migration architectures**

Recent studies show that there is a lack of support tools[16] for the automation of microservice migration, however, there are some proposed solutions to deal with the issue of this architecture migration. Genc Mazlami et al. [14] present a suggestion of ideal microservice candidates following an algorithmic approach subsequent to its extraction from a version control system (e.g., GitHub) of a monolithic application. It starts with the definition of a monolith M as a tuple (Cm, Hm, Dm), being that Cm represents the class files associated with the monolith, Hm the change history that contains the sequence of change events that happened in the code base and Dm the developers that participated in the development of the application. The history Hm will allow the authors to have access to the change in class files with its respective timestamp and developer ownership. The monolith in question is represented as a graph G = (E, V), being that every vertex v ∈ V is a class and every edge e ∈ E is a weight associated to the coupling level between classes. The graph G is subsequently partitioned into microservice candidates using graph clustering. However, not all existing migration models are automatic, being that the process of moving from a monolithic to a microservices architecture can be seen as a manual process of decomposition with the use of several techniques, as written by Zhamak Dehghani and Christian Posta. These

4https://github.com/
5https://martinfowler.com/articles/break-monolith-into-microservices.html
6http://blog.christianposta.com/microservices/the-hardest-
are the steps taken with the objective of creating an ideal microservices architecture:

1. Separate the system with significantly decoupled capabilities that have no need for a data repository.
2. Decouple capabilities that have no dependencies.
3. Harder concepts are reconstructed along with their domain ideas, joining them accordingly into services.
4. Decouple the data, creating a data repository for each service that is driven by its domain, creating persistent data tables needed for only that service.
5. Develop APIs that are easily used by developers and redirect all the front-end applications to them.

Additional to this workflow, tools to identify the dependencies to the monolith and the overall code structure should be used, in order to get the ideal services with the highest level of coupling and maximum capability encapsulation. An example of a tool with these functionalities is Structure101\footnote{https://structure101.com/}.

3. Methodology
The main objective of this paper is building a set of tools that supports developers on the process of migration from a monolithic to a microservices architecture. Our solution relies on the identification of contexts from the priorly presented domain-driven design and applying clustering techniques in retrieved static analysis data from applications modeled by DML.

First, we pretend to identify the bounded contexts of the applications, as defining these boundaries helps us separate the contexts of a monolithic application. We use bounded contexts as modular boundaries identified by groups of classes that have high cohesion and low coupling. This solution aims to improve the process of manual decomposition by experts, providing more information to the expert about the context decomposition of the application, supporting an informed decision-making process.

The monolithic systems analyzed in our approach are Java-based applications that use the Fénix Framework, implying that the structure of the domain is represented by DML and the mapping of objects to the database is not done directly but through an object-relational mapper. The typical architecture of web-application is divided into controllers, interface, domain and data access, being that for the context separation we are interested in knowing the relationship between the controllers and the domain. As a controller reaches different domain objects through the invocation of several methods, this flow has a parallelism with the flow found in a service belonging to a microservices architecture. Therefore, the writes and reads from a controller to the domain objects correspond to a transaction context. We explore this transactional context by the way those objects are associated with each other and their respective controllers, creating a modular context. This is going to be done through the use of static analysis on the code of the application under analysis.

To identify a cluster subsequent to the information retrieval, we are using the following specifications:

1. **Similarity Measure**: Our similarity is calculated through an attribution of a weight to the relationship between two classes, defined by the subsequent formula:

   \[
   W_{C1C2} = \frac{N_{Ctrl}(C1C2)}{N_{Ctrl}(C1)} \quad (1)
   \]

   Given two domain classes, \(C1\) and \(C2\), the weight from class \(C1\) to class \(C2\) is the quotient between the number of controllers in which the invocation tree has \(C1\) and \(C2\) as nodes \((N_{Ctrl}(C1C2))\) and the total number of controllers in which the invocation tree has \(C1\) as a node \((N_{Ctrl}(C1))\). When applying this measure to a clustering approach, in an ideal decomposition, the classes in the same cluster are accessed by the same controllers. One characteristic of the similarity measure we are defining is that it is naturally asymmetric. This means that the value of this measure from class \(C1\) to \(C2\) is not the same as from \(C2\) to \(C1\), as we are computing the ratio according to the total number of controllers of only one of the classes.

2. **Clustering Method**: To cluster the classes, **Hierarchical Clustering** is used, computing the results for its methods of single, complete and average linkage.

The overview of the process behind the examination of the monolithic application can be seen in Fig.1, having the following workflow:

1. Running static analysis on the code regarding the controllers of the web application, using the tool **Java-Caligraph** formerly described. As the data flow patterns identified are part of the Fénix Framework, the generation of a call graph associated with the controllers and
all its called methods will represent the invocation of the domain objects generated by the DML at the end the method call stack (through get and set methods), being able to not only capture the domain classes accessed by each controller but also to divide and manipulate the output of this tool into two different contexts:

(a) **Transactional Contexts Partial Graphs:** Partial graphs that represent the method invocations from the controller to the domain objects. Each graph will represent the transactional boundary associated to the respective controller analyzed. This graph has as a main node the controller under analysis, being that the nodes linked to them are the classes that it accesses. Its edges represent an invocation of such class, being able to portrait that there is an access to the object of such class, but not discriminating if it is a read or a write. This information is retrieved using a regular expression implementation on the text representation given by the java-callgraph tool, outputting a serialized python dictionary that contains all the classes that the controller under analysis accesses.

(b) **Modular Context Dendrograms:** The modular context of the application is derived from its transactional context. This is done by executing hierarchical clustering on the classes obtained from the concatenation of all the partial graphs into a system-level transactional graph. The result of the clustering will be presented as a dendrogram, as the several possible cuts can define alternative modular contexts depending on the desired cohesion between clusters, also building an exploratory approach.

2. Through the creation of a GUI, the user is prompted to select a cut on the dendrogram, enabling the creation of several combinations of cluster formations.

3. In order to provide such freedom to the user, a web browser in-depth visualization of the clusters with system information is presented in parallel to the interface.

We applied this exact workflow to the web applications LdoD\(^8\) and Blended Workflow\(^9\), where the behavior analysis of the controllers was made using Python, mainly the libraries NetworkX\(^10\) and Scipy\(^11\).

Two iterations of this implementation for each application were done:

1. **Whole Controllers:** The simpler way to analyze the behavior of the controllers is running Java-callgraph on them as-is. This approach will not make any alterations to the already existing code.

2. **Split Controllers:** The execution of our workflow on controllers that have not been changed has one main disadvantage. If each controller does not represent each functionality of the system then the domain classes access information can be inaccurate. An example is if all controllers have a cleanup method that accesses all domain classes, being that all controllers will be seen, with this workflow, as reaching all those classes when that is not true. Our objective is to separate functionality and therefore separate the domain classes accurately. An approach to avoid or mitigate this problem is separating each controller so that each copy of it has only one method.

4. **Results & discussion**

The results shown for both analyzed applications refer to the second iteration with the split controllers. Moreover, the dendrogram linkage assessment is the average linkage type.

4.1. **LdoD**

The LdoD archive\(^12\) is a collaborative digital archive that contains the Book of Disquiet, origi-
nally written by Fernando Pessoa. As this book is fragmented, several interpretations of it can be done by reading it in different sequences. This application, in addition to allowing the reading and comparison of four editions of the book, it also enables users to create their own virtual editions based on the existing fragments. LdoD originally contains 152 controllers and 56 domain classes, being that 37 of the controllers do not make contact with the domain (24% of the systems controllers). The dendrogram of this application, originated from the hierarchical clustering approach, can be seen in Fig.2.

Figure 2: Average linkage dendrogram of applying hierarchical clustering to the LdoD application with split controllers.

4.2. Blended Workflow
The Blended Workflow application has the intention of fusing an activity-based approach with a goal-based approach by allowing its end users to follow a predefined behavior on the work they are developing. Blended Workflow originally has 94 controllers and 46 domain classes. 1 of the 94 controllers has no domain access as it handles the exceptions of the system. The execution of the hierarchical clustering algorithm generates the dendrogram seen in Fig.3.

Figure 3: Average linkage dendrogram of applying hierarchical clustering to the Blended Workflow application with split controllers.

two clusters.

4.4. Metric evaluation
As supported by the evaluation of other approaches for software architecture recovery [13] [1], an internal and external assessment of the clusters is made.

4.4.1 Internal evaluation
To perform an intrinsic evaluation of the clustering results for our applications, we perform an ad hoc analysis with metrics proposed by us (except for the silhouette score), being those:

1. **Number of Singleton clusters (NSC)**, being that having more than 2 singleton clusters is considered negative. Considering a final microservice architecture with clear functional boundaries established, it is likely that there are not two services in which its content is only one class.

2. **Average cluster size (ACS)**, should be between 2 and half of the size of the system. Even with a cluster size inside this range, there is also a dependency regarding the number of instances inside them, due to data consistency and concurrency. Establishing transactional locks on the operations that will occur in these clusters can make us change certain clusters regarding their size if they have too many instances and negatively impact the system’s performance.

3. **Maximum cluster size (MCS)**, should not be bigger than half of the size of the system. As an application does not usually have just one functionality, there should be at least two clusters, representing this maximum size boundary.

---

13https://wiki.python.org/moin/Tkinter
14https://d3js.org/
4. **Silhouette score (SS)**, given by Equation 4, where \(a\) represents the mean intra-cluster distance (Equation 2: distance between object \(o_i\) and the remaining objects in the cluster) and \(b\) the mean nearest-cluster distance (Equation 3: distance between object \(o_i\) and the objects of the neighbor cluster). This score ranges its values from -1 to 1, representing incorrect clustering (samples on wrong clusters) and highly dense clustering respectively. For every object in a cluster, when this score is high (closer to 1) the mean intra-cluster distance is going to be smaller than the mean nearest-cluster distance, implying that the object is well classified. This metric creates a parallelism with the overall coupling of the clusters of the system, as our objective was to obtain a high intra-cluster similarity and a low inter-cluster similarity, so the partition between clusters is well defined. The silhouette value evaluates exactly this information. In the scope of our problem we demonstrate the silhouette score for the entire cluster data of the presented cut, meaning that we have to calculate the silhouette of each cluster by averaging the score of all the object inside them and then average the score of all the clusters, reaching a value for the entire dataset.

\[
a(o_i) = \frac{1}{|C_A| - 1} \sum_{o_j \in \bigcup C_A, o_j \neq o_i} d(o_i, o_j) \tag{2}
\]

\[
b(o_i) = \min_{C_B \neq C_A} \frac{1}{|C_B|} \sum_{o_j \in C_B} d(o_i, o_j) \tag{3}
\]

\[
\text{Silhouette}(o_i) = \frac{(b(o_i) - a(o_i))}{\max(a(o_i), b(o_i))} \tag{4}
\]

We consider several cuts (NRC = Number of retrieved clusters):

**Table 1:** Internal evaluation results for the second iteration of the solution on LdoD.

<table>
<thead>
<tr>
<th>Cut(0.01)</th>
<th>Cut(1.5)</th>
<th>Cut(2.5)</th>
<th>Cut(3.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSC</td>
<td>34</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>MCS</td>
<td>5</td>
<td>18</td>
<td>26</td>
</tr>
<tr>
<td>ACS</td>
<td>1.31</td>
<td>5.0</td>
<td>18.33</td>
</tr>
<tr>
<td>NRC</td>
<td>40</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>SS</td>
<td>0.38</td>
<td>0.48</td>
<td>0.55</td>
</tr>
</tbody>
</table>

In both tables, four cuts are presented for the scope of both applications:

1. The maximization of intra-cluster similarity, given by a cut with the lowest value possible.
2. A cut at an intermediate height, establishing an attempt to make a trade-off between granularity cluster similarity.

3. Two high valued cuts that try to split the system into its main components, usually with a size of 2-4 clusters.

Assessing first our ad-hoc metrics, for both applications, when increasing the value of the height of the cut on the dendrogram, the NSC and NRC decrease while the MSC and ACS increase, which is logic as the bigger the height, less clusters are formed, being that those contain more domain classes. Also, the silhouette score increases with height to a maximum, showing that at that point are formed the ideal clusters according to this metric.

4.4.2 External evaluation

In this type of evaluation we are comparing the results of our approach with an expert decomposition and a software architecture analysis tool, Structure101\(^\text{15}\). Usually, in the computation of evaluation metrics following the use of clustering, these are done in a pairwise fashion, as presented by S.Walde[18] in her PhD thesis. The most appropriate metrics for our approach are the subsequently calculated pairwise precision, recall and f-score, given by Equations 5, 6 and 7 respectively:

\[
\text{Precision} = \frac{\text{Number of common pairs}}{\text{Number of clustering pairs}} \tag{5}
\]

\[
\text{Recall} = \frac{\text{Number of common pairs}}{\text{Number of expert decomposition pairs}} \tag{6}
\]

\[
F\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}
\]

The pairwise assessment of these metrics for system-level cuts is made on the Tables 3, 4 and 5.

**Table 2:** Internal evaluation results for the second iteration of the solution on Blended Workflow.

<table>
<thead>
<tr>
<th>Cut(0.01)</th>
<th>Cut(1.1)</th>
<th>Cut(2.0)</th>
<th>Cut(3.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSC</td>
<td>30</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>MCS</td>
<td>4</td>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td>ACS</td>
<td>1.31</td>
<td>4.18</td>
<td>11.5</td>
</tr>
<tr>
<td>NRC</td>
<td>35</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>SS</td>
<td>0.35</td>
<td>0.47</td>
<td>0.57</td>
</tr>
</tbody>
</table>

**Table 3:** External evaluation result of the pairwise precision metric.

<table>
<thead>
<tr>
<th>LdoD</th>
<th>Blended</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactional C.</td>
<td>LdoD - 2.5 Cut</td>
</tr>
<tr>
<td></td>
<td>Blended - 2.0 Cut</td>
</tr>
<tr>
<td>Structure101</td>
<td>58% ((\frac{106}{180}))</td>
</tr>
</tbody>
</table>

\(^{15}\text{https://structure101.com/}\)
We can see that the results of our approach for all the presented metrics are higher than Structure101.

Analyzing the three cluster formations of the 2.5 LdoD cut, we reached the conclusions that the first is a sub-cluster of a cluster of the expert, capturing all the text related classes and other classes that directly manage them. The second is a mix of all the different functionalities and is used by 100% of the controllers of the system, existing the need to split this cluster into smaller parts. The final one shows a group of 5 classes that are responsible for deleting and loading fragments, that by only using a specific set of controllers are highly coupled together. Structure101 originates 10 clusters from which 6 are singletons. From the remaining 4, three are sub-clusters of the expert and one is a join of all functionalities. For Blended workflow at the cut of 2.0, our approach retrieves three cluster formations. The first one contains the Models cluster of the expert and a sub-cluster of the Design cluster. The second cluster is a perfect match to the Execution cluster of the expert and the final cluster is a subset of the Design clusters that deals with particular conditions. Structure101 retrieves 6 clusters, from which 1 is a singleton, 3 are sub-clusters of the expert and the remaining 2 are mixed. A system-level cut can provide us important information regarding the decomposition of a monolith, but an exploration phase is also needed, where we cut at intermediate levels and subsequently compare the smaller clusters to the expert decomposition. This can also help solve ambiguity in the bigger clusters of the system cuts.

The chosen intermediate cuts of the system and its evaluation are shown in the subsequent tables:

**Table 4:** External evaluation result of the pairwise recall metric.

<table>
<thead>
<tr>
<th>Transactional C.</th>
<th>LdoD - 2.5 Cut</th>
<th>Blended - 2.0 Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>LdoD</td>
<td>48% (115/238)</td>
<td>48% (120/250)</td>
</tr>
<tr>
<td>Structure101</td>
<td>18% (106/592)</td>
<td>31% (111/356)</td>
</tr>
</tbody>
</table>

**Table 5:** External evaluation result of the F-Score metric.

<table>
<thead>
<tr>
<th>Transactional C.</th>
<th>LdoD - 2.5 Cut</th>
<th>Blended - 2.0 Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>LdoD</td>
<td>0.58</td>
<td>0.56</td>
</tr>
<tr>
<td>Structure101</td>
<td>0.27</td>
<td>0.40</td>
</tr>
</tbody>
</table>

Note that for the intermediate cuts, our precision is much higher than at system-level, this happens as these smaller clusters formed by a lower cut are almost all subsets of the clusters of the expert decomposition. Moreover, the recall values are low, as the singletons are properly penalized by this metric.

Starting by LdoD and its cluster formations, the only false positive (in 1/11 retrieved clusters) resides in the cluster with the classes LdoD, LdoDUser and VirtualEdition. LdoDUser and VirtualEdition are classified as being used in two different scopes, Authentication and Virtual Edition management respectively. LdoD is irrelevant as it is an immutable singleton, being able to be part of any cluster. Our tool classified these classes as being part of the same cluster as they appear together transactionally so, we are going to analyze these cases by assessing the D3.JS visualization. First, counting that the VirtualEdition class is out of the cluster of Virtual Edition management and LdoDUser is out of the Authentication cluster, the remaining elements of those clusters for the expert decomposition are perfectly represented in our cluster divisions. Moreover, this cluster is accessed by 105/115 (91%) controllers of the system, indicating that it could be a candidate for a critical microservice. On a microservices architecture, its components are built around the business capabilities of the system, and as this cluster captures the main classes of two functionalities, it is useful for understanding system functionality and the classes that are susceptible to being overloaded (critical microservice). If each of these classes represents the core class of a functionality, this means that they can be scaled in a different way than the rest of the functionality if we introduce them as an entry point service. As the controller access is 91%, every execution of the service can overload the service access-wise, existing the need to apply replication techniques. The expert added to this matter that the entities involved in these operations are created once and rarely changed. We can interpret that in a service of its functionality, another service is defined as its entry point that can
be replicated in a different way than the rest. We can also establish a parallelism between this cluster and an architectural pattern of a shared kernel, as every use of the system has to go through these classes, but not as a domain-sharing pattern, but as API-sharing, as the core classes we are analyzing constitute the first entry-points to the different functionalities that exist in this application. Analyzing shared controllers through the use of our tool, we can see that all the controllers of the Authentication are shared (9/9) and from the Virtual Edition management, most of them are also shared (47/49), concluding that LdoDUser and VirtualEdition are present in the context of these two functionalities but, are responsible for much more, as to perform most of the capabilities of the web application, the LdoD instance and the corresponding authenticated user and virtual edition are required, retrieving exactly these domain classes. Concluding, to solve this issue, we should either create a separate service with these 3 classes that constitute the entry-points of the system as an entry-level service, or simply adding LdoDUser to the Virtual Edition management functionality, as they are grouped together in many functionalities. Comparing the clusters originated from this cut with the system-level cut at 2.5, we can see that the cluster that had mixed functionalities and that was hard to reason is now split into 6 clusters which allows us to explore the controllers that access each of these individual clusters instead of one big cluster.

For the Blended Workflow application and its cut at 1.1, we can see that 2/11 retrieved clusters have false positives. Examining these cases with our tool we conclude that for both cases, the expert should consider moving a class in its decomposition, as the assessment of the controllers of that class indicate a different functionality than the one of the cluster it was assigned to. The assessment of the true positives also revealed a candidate for a critical microservice, as the classes Blended-Workflow and Specification are accessed by 90/93 (97%) of the application's controllers. This is confirmed by analyzing the related code, as to execute any functionality in this application, the specification that we are working on must be retrieved. And to retrieve such specification, the specification identifier is fetched from the corresponding blended workflow. Therefore, these two classes can also be part of an entry-point service, as we know that most of the accesses to the system will have to go through these classes.

For both applications, we can assess the relevant controllers of the remaining cluster formations through the use of our tool and determine the major functionality of the cluster.

4.5. Solution limitations
With our presented approach we pretend to tackle the problem of microservice migration in terms of context separation, however, some limitations are present:

1. The manual inspection of the static analysis approach creates a system which is hardly replicated, due to the difficulties associated with manual code analysis.
2. The applications we are examining are part of a codebase that has not yet been migrated to microservices, being that the expert's decomposition does not bring a total ground-truth.
3. We are only dealing with the problem of context separation, this is, a subset of a migration to a microservices architecture. However, as this work is planned to be continued, the objectives of the first step of the migration were met.

5. Conclusions
In summary, this document introduces a novel approach that uses static analysis on the source code of a monolithic business system to discover significant information that is crucial in the division of the monolithic code into contexts. The main objective of this approach is constructing an informed decision-making process for the migration from a monolithic architecture to a microservices architecture by presenting the cluster formations that mimic service behaviors and enabling the user to customize them. Due to the values presented in our evaluation being higher than an already existing formal tool (Structure101), we consider that our hypothesis is validated and that there is the capability of retrieving modularity from the transactional contexts of an application, as the relationships between the controllers and domain classes can be used to group domain classes together in contexts. The outcomes presented will also be useful as a benchmark to be used in further examination of these systems or other iterations of our approach.

5.1. Future Work
Several directions can be taken for advancing our approach. First, capturing system information regarding the direction, frequency and type of interaction(read or write) can lead to a better interpretation of the available data and also the detection of cyclic dependencies. Second, improving or re-implementing the tool that statically analyzes, as a manual inspections is needed presently. Third, integrating the functionality with an IDE (Eclipse) so workable projects are accompanied with the tool. Finally, we can also test the existing tool for existing applications that have available their monolithic...
and microservices codebase, enabling the comparison of the results with a ground-truth.

References


