Frontmatter

Editors
SIGAPP FY’17 Quarterly Report

Selected Research Articles

Detecting Anomalies in the Cytochrome C Oxidase I Amplicon Sequences Using Minimum Scoring Segments
M. Belcaid and G. Poisson

Automated SMT-based Consistency Checking of Industrial Critical Requirements
P. Filipovikj, G. Rodriguez-Navas, M. Nyberg, and C. Seceleanu

Contextual Understanding of Microservice Architecture: Current and Future Directions
T. Cerny, M. Donahoo, and M. Trnka

Ontology-based Framework for Internal-External Quality Trade-offs and Tenant Management in Multi-tenant Applications
S. Kalra and P. T.V.
SIGAPP FY’17 Quarterly Report

October 2017 – December 2017
Jiman Hong

Mission

To further the interests of the computing professionals engaged in the development of new computing applications and to transfer the capabilities of computing technology to new problem domains.

Officers

<table>
<thead>
<tr>
<th>Role</th>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chair</td>
<td>Jiman Hong</td>
<td>Soongsil University, South Korea</td>
</tr>
<tr>
<td>Vice Chair</td>
<td>Tei-Wei Kuo</td>
<td>National Taiwan University, Taiwan</td>
</tr>
<tr>
<td>Secretary</td>
<td>Maria Lencastre</td>
<td>University of Pernambuco, Brazil</td>
</tr>
<tr>
<td>Treasurer</td>
<td>John Kim</td>
<td>Utica College, USA</td>
</tr>
<tr>
<td>Webmaster</td>
<td>Hisham Haddad</td>
<td>Kennesaw State University, USA</td>
</tr>
<tr>
<td>Program Coordinator</td>
<td>Irene Frawley</td>
<td>ACM HQ, USA</td>
</tr>
</tbody>
</table>

Notice to Contributing Authors

By submitting your article for distribution in this Special Interest Group publication, you hereby grant to ACM the following non-exclusive, perpetual, worldwide rights:

- to publish in print on condition of acceptance by the editor
- to digitize and post your article in the electronic version of this publication
- to include the article in the ACM Digital Library and in any Digital Library related services
- to allow users to make a personal copy of the article for noncommercial, educational or research purposes

However, as a contributing author, you retain copyright to your article and ACM will refer requests for republication directly to you.

Next Issue

The planned release for the next issue of ACR is March 2018.
MAKE THINGS POSSIBLE
with a degree in applied computer science from the University of Wisconsin.

Organizations need IT professionals who can excel technically
and drive business results. With the UNIVERSITY OF WISCONSIN
BACHELOR OF SCIENCE IN APPLIED COMPUTING, YOU CAN
MAKE THINGS POSSIBLE—for your company and your career.

LEARN MORE

FUTURE-FOCUSED LEARNING
Solve today’s IT issues AND learn to develop tomorrow’s technology

TECHNOLOGY AND BUSINESS COMBINED
A computer science curriculum with a sharp focus on business skills

REAL-WORLD APPLICATION
Courses that teach applicable compound skills, plus valuable workplace experience in the capstone course
Detecting Anomalies in the Cytochrome C Oxidase I Amplicon Sequences Using Minimum Scoring Segments

Mahdi Belcaid
Hawaii Institute of Marine Biology
University of Hawaii at Manoa
Honolulu, Hawaii
mahdi@hawaii.edu

Guylaine Poisson
Information and Computer Sciences
University of Hawaii at Manoa
Honolulu, Hawaii
guylaine@hawaii.edu

ABSTRACT
The Cytochrome C Oxidase 1 (COI) gene is among the most popular markers for molecular biodiversity estimation. In essence, COI-based approaches for taxonomic identification rely on comprehensive reference databases to assign unknown sequences to known species and to enhance the identification of new species. As such, for COI-based methods to be effective, the accuracy and integrity of reference databases are critical. However, as COI repositories grow, it becomes challenging to curate and validate user-contributed data manually. This, in turn, propagates prediction errors, leading to future erroneous taxonomic assignments. Here, we propose a new computationally efficient approach for identifying anomalies which are either due to systematic biases (indels and chimeras) or to user error (mistranslation and misclassification). Our approach uses multiple sequence alignments to model insertions, deletions, and substitutions and flag sequences with incongruous fit to the model. Our analysis of a complete set of curated Insecta COI reference sequences identifies the presence of numerous anomalous sequences, which makes a strong case for the validity of our approach and for the importance of new strategies to screen publicly available COI references.

CCS Concepts
• Applied computing → Molecular sequence analysis; Sequencing and genotyping technologies; Bioinformatics;

Keywords
COI; Diversity estimation; Sequencing anomalies

1. INTRODUCTION
Accurate species identification is critical for estimating biodiversity and for better understanding the effects of various anthropogenic and natural disturbances on the underlying biological landscape. For taxa that are well studied, visual cataloging and identification of morphological features by an expert taxonomist is the gold standard for species identification [2]. However, expert-led taxonomic identification is time-consuming and requires high sample integrity [10] as well as intimate knowledge of phenotypic differences, including those resulting from changes in life stages or gender. These constraints render this approach inadequate in highly biodiverse or complex environments from which organisms cannot be extracted intact. Increasingly large biomonitoring efforts have resorted to molecular approaches where DNA-based markers are brought to bear on evaluating species diversity [10]. The mitochondrial gene Cytochrome C Oxidase I (COI) is a popular marker for the identification of members in the kingdom Animalia, due to its ease of sequencing in most, if not all, animal phyla [10, 9]. Additionally, the primary COI sequence, which consists of a 648-bp coding region, offers greater range of phylogenetic signal, particularly at the highly diverse third-position nucleotides, compared to other markers such as 12S or 18S ribosomal markers [13]. High-throughput sequencing projects rely on similarity and divergence patterns within and across taxonomic levels to estimate the number and the nature of species in a sample [18]. For these estimations to be accurate, the assembly of a comprehensive reference database is critical. The set of available COI reference sequences is collated in specialized archives such as the Barcode of Life Database (BOLD: http://www.boldsystems.org), a library of barcodes for eukaryotic life [18]; and the Moorea BioCode Project (http://mooreabiocode.org/), a comprehensive inventory of all non-microbial life in a complex tropical ecosystem. COI markers are also commonly submitted to public sequence databases, such as the NCBI’s GenBank. While the quality of the sequences deposited to public databases is traditionally screened for trivial discrepancies – such as the occurrence of a stop codon in the open reading frame (ORF) — ultimate responsibility for quality and taxonomic accuracy of the data lies with the project participants [18]. As such, as the number of COI sequences submitted to specialized databases increases, the likelihood of introducing erroneous references also increases in more than trivial ways. For instance, library preparation and sequencing error rates can exceed 4% in state-of-the-art protocols for amplicon sequencing [19]. As such, inaccuracies in reference databases are inevitable.

In preliminary simulations on a subset of 1,000 sequences downloaded from the BOLD database (data not included), random insertions or deletions (indels) did not produce a stop codon in more than 15% of the sequences. This shows that relying solely on the detection of premature stops in ORFs is not sufficient. In addition to indels, chimeras; or
hybrid sequences from multiple parents, are a significant source of bias in specialized marker databases. Chimeric sequences arise predominantly during the PCR amplification stage when an incomplete PCR fragment serves as a primer by binding the template DNA of different species [21]. According to Porazinska et al. [17], chimeras can account for as much as 46% of the sequence data. Chimeras can also arise, albeit to a lesser extent, from the fusion of unbound sequences [20]. Popular bioinformatics tools such as uchime [4] or ChimaraSlayer [6] are often used for detecting COI chimeras. These tools can achieve great sensitivity when the chimera fragments originate from evolutionarily distant sequences. However, due to their computationally intensive nature, chimera detection tools are often omitted from pipelines and replaced by cross-sample abundance checks, under the assumption that sequences that only occur in low-abundance in a single sample are erroneous. This approach perhaps partially explains why chimeras are still routinely identified in curated 16S databases despite the abundance of tools for detecting them [15].

Exogenous contamination is yet another source of bias in public databases. By exogenous contamination, we refer to kingdom-level COI sequences mismatches, such as the recovery of bacterial sequence in a project focused on animals. A common practice for handling exogenous contamination is by screening against known, exogenous references [18]. This approach is ideal for filtering out known contamination, but cannot detect sequences not available in the contamination database.

Given the crucial role that specialized reference databases play in diversity estimation, rigorous inspection and detection of erroneous sequences they may contain is critical. Indeed, systematic errors not only lead to the significant overestimation of diversity but can also bias measures of intra- and inter-taxonomic level sequence similarity, which, in turn, leads to future erroneous taxonomic assignments.

In this work, we present a COI sequence anomaly detection model which leverages the functional constraints on a COI amino acid sequence to identify potentially anomalous sequences that deviate significantly from the profile(s) compiled at a given taxonomic level. We hypothesize that given a large enough sample of correct sequences, anomalous sequences will appear as statistical outliers. Our approach works in three distinct steps: 1- sequence filtering and dereplication, 2- score profile inference and modeling and 3- identification of sequence anomalies. During the first step, we construct a taxonomic group’s multiple sequence alignment from a preprocessed dataset. In the second step, we compute the similarity for each sequence in the alignment from a preprocessed dataset. In the second step, we construct a taxonomic group’s multiple sequence alignment from a preprocessed dataset. In the second step, we compute the similarity for each sequence in the alignment from a preprocessed dataset. In the second step, we construct a taxonomic group’s multiple sequence alignment from a preprocessed dataset. In the second step, we construct a taxonomic group’s multiple sequence alignment from a preprocessed dataset. In the second step, we construct a taxonomic group’s multiple sequence alignment from a preprocessed dataset. In the second step, we construct a taxonomic group’s multiple sequence alignment from a preprocessed dataset.

By working at the amino acid level, we can detect artifacts are not challenging to identify using the nucleotide sequences. Furthermore, by choosing to model amino acid differences in each column of the multiple sequence alignment, we account for localized variability that results from functional constraints specific to different regions of the COI protein [14].

Our results show that this approach is effective for detecting insertions, deletions, chimeras, mis-translations and contaminations in Insecta sequences from the BOLD database. Furthermore, this method is computationally efficient and is a good addition to existing strategies for monitoring the integrity of specialized databases.

2. METHODS

All the COI amino acid sequences annotated to the Insecta class were downloaded from the BOLD database (BOLD System v3. Dec 2015). These sequences were preprocessed and analyzed as described below.

2.1 Sequence Filtering and Reference MSA Generation

Sequences with more than 3% of ambiguous amino acids (represented as X in the sequences) or shorter than 180 a.a. were discarded. The remaining sequences were clustered with cd-hit [5] using 99.5% similarity threshold. Representative sequences from each cluster were divided by order and aligned using MAFFT’s fast approximate alignment mode [11, 12]. Each MAFFT alignment was subsequently parsed to identify seed sequences and partial replicates. The seeds are references against which other sequences aligned with no insertions and with at least 93% similarity, whereas the partial replicates are the sequences that align against the seeds with no insertions and with at least 93% similarity. The seeds were subsequently aligned using MAFFT’s sensitive mode, yielding what we refer to in what follows as the reference MSA.

\[
\delta(i, j) = \begin{cases} 
0 & \text{if } i = j \\
1 & \text{otherwise}
\end{cases}
\]

where \(\delta(i, j)\) is the similarity score between amino acids at positions \(i\) and \(j\).

\[
S_{ij} = \frac{\sum_{d=1}^{20} \delta(i, d) \times \text{# of amino acids d at position j}}{n}
\]

where \(\delta\) is a function for scoring a pair of amino acids based the BLOSUM80 substitution matrix.

The score \(S_{ij}\) represents the overall similarity of sequence \(i\)’s \(j^{th}\) amino acid, to the remaining amino acids observed in that column [3]. For example, in Figure 1, the score \(S_{ij}\) for sequence 6 at column 12 (highlighted in red in the figure) is:

\[
S_{ij} = \frac{3/6 \cdot \delta(M, M) + 2/6 \cdot \delta(M, I) + 1/6 \cdot \delta(M, G)}{2/6 \cdot 6 + 2/6 \cdot 1 + 1/6 \cdot 4} = 1.66
\]

Figure 1: Sample multiple sequence alignment with \(n = 6\) and \(l = 12\).
To account for similarity biases such as when the amino acid at position \( j \) in sequence \( i \) is rare or comes from an undersampled species in the dataset, we smooth, \( S_{i,j} \) using a moving average approach with a window of size \( m \) [22]. Thus, for a sequence \( i \), the smoothed score \( SS_{i,k} \) at position \( j \) is computed as:

\[
SS_{i,k} = \sum_{j=k-m/2}^{k+m/2} S_{ij} \quad \text{for} \quad k < (l - m/2)
\]

We refer in what follows to the set of smoothed scores for sequence \( i \), \( SS_i = [SS_{i1}, SS_{i2}, \ldots, SS_{il}] \) as the score profile (See Figure 2A).

### 2.3 Score Profiles and Sequence Anomalies

At each column \( k \) of the reference MSA, we model the distribution of score profiles \( SS_{i,k} = (SS_{i1,k}, SS_{i2,k}, \ldots) \) using a univariate Kernel Density Estimation (KDE) with a Gaussian Kernel [16]. KDE is a non-parametric method for estimating the probability density function of a random variable from a set of observations. Conceptually, a KDE is similar to a histogram. However, instead of discretizing the data, a KDE is smooth and continuous.

We chose to model \( SS_{i,k} \) using a non-parametric KDE since this approach does not make any assumptions about the structure of underlying distributions of scores at each column. This makes the method particularly resilient to multimodal distributions of scores which could arise due to taxonomic subclustering of the sequences. This consideration is essential when working at higher taxonomic levels, particularly in old lineages, where sequence diversification is potentially non-negligible and where sequences are most likely to cluster tightly according to the underlying taxonomic levels. For example, due to the considerable phylogenetic distance between the Mecoptera and Diplura orders of the Insecta class is likely to translate for some columns as a multimodal distribution of the smoothed similarity scores.

Using a column’s KDE, we can quickly estimate the probability of observing an \( SS \) value at a particular column. Figure 2 (Bottom) gives an example of two KDEs fitted using the scores observed at columns \( k = 330 \) and \( k = 350 \) respectively. The distribution at column 330 shows less variance than that at 350, which reflects the distribution of scores seen over both columns. The probability density of the red curve quantifies our confidence in the quality of the alignment of the red sequences over that at 350.

Given that we model the scores independently at each column, the resulting KDEs capture the differences in amino-acid diversification rates across functionally distinct sites in the COI protein.

For each sequence, we were interested in identifying sub-sequences exhibiting poor fit against the reference MSA. This amounts to finding \( SS \) values with low probability, as computed by KDE distributions of profile scores. The occurrence of a single low-probability position is not \textit{per se} indicative of an anomalous mutation. In an MSA with a large number of sequences, a mutation in a highly conserved amino-acid, such as in Tryptophan (W) is assigned a low probability density. However, the occurrence of multiple, consecutive low-probability density positions is unlikely to occur in a normal sequence. To identify the longest stretches of low-probability density windows in each sequence, we used Kadane’s algorithm, as described in [1]. Given a sequence of positive and negative values, Kadane’s algorithm finds the continuous subsequence that maximizes the sum of its elements, i.e., no other subsequence can add up to a larger value. Given that Kadane’s algorithm requires positive and negative values as input, we mapped our probability densities into scores using the following equations:

\[
\sigma(l, \epsilon, k) = k(\sqrt{x} - \sqrt{\epsilon})
\]

(1)

![Figure 2](image_url)

**Figure 2:** Top: The black curve represents the profiles of smoothed scores for 20 sequences from the BOLD COI dataset. The red curve represents a profile which substantially deviates from the cloud of black curves over a substring of length 22 amino acids. Bottom: KDE distributions derived from windows of size \( m=11 \) and centered around columns \( k = 330 \) (solid line) and \( k = 350 \) (dashed line).

![Figure 3](image_url)

**Figure 3:** Increasingly large probability densities above threshold \( \epsilon \) (0.03 in this example) are assigned increasingly large, positive scores whereas increasingly small probability densities below \( \epsilon \) are assigned increasingly small, negative scores. The function is shown here over the 0-1 interval.

Equation 1 allows us to convert the probability densities, \( p \), such that values of \( p \) where \( p \leq \epsilon \) are converted into
Table 1: Breakdown of the number of sequences per order at each stage of the analysis.

<table>
<thead>
<tr>
<th>Order</th>
<th>Raw</th>
<th>Cleaned</th>
<th>Unique</th>
<th>Seeds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archaeognatha</td>
<td>216</td>
<td>204</td>
<td>86</td>
<td>6</td>
</tr>
<tr>
<td>Blattodea</td>
<td>2,024</td>
<td>1,853</td>
<td>431</td>
<td>33</td>
</tr>
<tr>
<td>Coleoptera</td>
<td>72,731</td>
<td>68,440</td>
<td>25,395</td>
<td>140</td>
</tr>
<tr>
<td>Dermaptera</td>
<td>175</td>
<td>153</td>
<td>71</td>
<td>7</td>
</tr>
<tr>
<td>Diplura</td>
<td>39</td>
<td>34</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>Diptera</td>
<td>105,850</td>
<td>93,659</td>
<td>42,262</td>
<td>336</td>
</tr>
<tr>
<td>Embiopetera</td>
<td>201</td>
<td>197</td>
<td>92</td>
<td>4</td>
</tr>
<tr>
<td>Ephemeropetera</td>
<td>3,350</td>
<td>3,312</td>
<td>933</td>
<td>18</td>
</tr>
<tr>
<td>Gryllloblattodea</td>
<td>53</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Hemiptera</td>
<td>42,674</td>
<td>37,887</td>
<td>15,433</td>
<td>180</td>
</tr>
<tr>
<td>Hymenoptera</td>
<td>192,672</td>
<td>170,616</td>
<td>68,998</td>
<td>353</td>
</tr>
<tr>
<td>Isoptera</td>
<td>2,218</td>
<td>2,138</td>
<td>259</td>
<td>19</td>
</tr>
<tr>
<td>Lepidoptera</td>
<td>31,627</td>
<td>28,609</td>
<td>22,856</td>
<td>226</td>
</tr>
<tr>
<td>Mantodea</td>
<td>947</td>
<td>925</td>
<td>418</td>
<td>9</td>
</tr>
<tr>
<td>Mantophasmatodea</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Megaloptera</td>
<td>165</td>
<td>157</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>Megalocoptera</td>
<td>141</td>
<td>138</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>515,133</td>
<td>466,227</td>
<td>177,833</td>
<td>5,729</td>
</tr>
</tbody>
</table>

The SS scores and their probability densities were computed for all sequences using a window size $m = 11$. This step took less than two minutes to complete on a laptop with 16GB of RAM and 2.8 GHz CPU.

Under the naïve assumption that the observed probability densities at consecutive positions of a sequence are independent, various approaches can be used to decide whether sequence $i$ is anomalous. For example, one can set either a minimum threshold on the lowest allowed probability density in a sequence, or a maximum allowable number of consecutive low-probability densities above which we deem a sequence anomalous. Our approach consists of identifying in each sequence the longest stretch with the lowest probability densities – this is analogous to finding the longest substring with the weakest fit against the reference MSA.

To achieve this, we convert the probability density in each position into a positive or negative score in a manner that is proportional to its probability densities, i.e., increasingly large probability densities are assigned increasingly large, positive scores and increasingly small probability densities are assigned increasingly large, negative scores. We then use the scores as input to Kadane’s algorithm.

In essence, any monotonically increasing function that assigns increasingly positive scores to increasingly probable events and increasingly negative scores to increasingly unlikely events is a potential candidate for converting probability densities to scores. A simple and intuitive example is a linear function with a positive slope which crosses the $x = -a$ at a given threshold $\epsilon$ (e.g., $y = 400x - 20$ crosses at $x = \epsilon = 0.05$). However, a linear function cannot increase sufficiently fast to assign a large range of values to the low-confidence interval $[0, \epsilon]$ without assigning extreme values to the high-confidence interval where $x > \epsilon$. By contrast, the function described in Equation 1 can allow sufficient control over the range of values assigned to the low- and high-confidence intervals.

We used the converted probability densities to compute the MSA values for each sequence in reference MSA. A total of 1,622 sequences had an MSA value of at least 1. The largest MSA value was 196 and the smallest MSA value was 0. An MSA value of 0 indicates an overall good quality fit to the reference MSA and, overall, no poorly aligned regions. On the other hand, an MSA value of 196 indicates a low-quality alignment against the reference MSA over a window of at least 196 amino acids. Except for trivial cases, confirming whether a sequence is anomalous requires manual review and supporting evidence. As such, the MSA value should be viewed as an indicator of the length of the misaligned region, rather than an absolute predictor of whether a sequence is anomalous. For our reference MSA, gap-free substrings are, on average, 11 amino acids long. As such, we use an MSA value of 25, which arises most frequently in our data from the class Insecta, spread across 16 taxonomic orders (See Table 1 for a breakdown of the number of sequences per order).

Due to the high similarity among COI sequences at the amino-acid level, the clustering- and alignment-based dereplication steps allowed us to reduce the dataset by over 1,000-fold, and resulted in 5,729 seed sequences (See Table 1). The seeds were aligned into a reference MSA using MAFFT’s sensitive mode. The reference MSA consisted of 515,133 sequences from the class Insecta, spread across 16 taxonomic orders (See Table 1 for a breakdown of the number of sequences per order).

The resulting reference alignment would most likely be different if it included the partial replicate sequences. However, we conjecture that since the partial replicates do not contain insertions, their addition would have introduced minor, confined substitutions, which would not impact our approach, particularly since we average the score profiles across a sliding window.

3. RESULTS AND DISCUSSION

The dataset downloaded from the BOLD database consisted of 515,133 sequences from the class Insecta, spread across 16 taxonomic orders (See Table 1 for a breakdown of the number of sequences per order).

The dataset downloaded from the BOLD database consisted of 515,133 sequences from the class Insecta, spread across 16 taxonomic orders (See Table 1 for a breakdown of the number of sequences per order).
quences (See supplementary files). For sequences with MSS values below 25, the mean MSS was 3.69 with a standard deviation of 5.

We randomly selected and manually inspected a subset of 70 sequences with an MSS higher than 25. A sequence was confirmed as anomalous if we could classify it unequivocally into one of following four error classes: 1- mistranslated amino acid sequence, 2- containing an insertion or a deletion (indel), 3- chimera, and 4- misclassification. We considered a sequence suspicious if its MSS aligned with low-quality to the reference MSA, but its assignment to any of the four error classes was not evident.

3.1.1 Mistranslated Sequences
The MSS values and the score profile curves are intuitive representations of anomalies. For instance, mistranslated sequences are trivial to identify visually using score profile curves; their score profile is consistently low, and their MSS value is close to their sequence length. These sequences are also trivial to validate, as they have an alternative, full ORF which produces a better overall alignment and substantially reduces the MSS value. One such sequence, CNKOC293-14, has a consistently negative profile score using the submitted translation (See Figure 4). When using the first ORF on the reverse strand, the same sequence yields a consistently positive score.

3.1.2 Indel Containing Sequences
The score profile curve of an indel-containing sequence is different. The DNA sequences pre- and post-indel either produce well or poorly aligned regions. The subsequence that aligns well has a higher profile score, which leads to a notable change in the score profile curve either from high to low or the inverse. In sequences with indels, the MSS length is a function of the location of the indel in the sequence. Indels represent the most abundant anomalies in our dataset. Sequence LEPIN057-14 is a good example as it shows an evident change in the sequence’s score profile near $k = 220$ (See Figure 4). Translation of this sequence into all reading frames shows that the second ORF produces a better fit for the first 220 amino acids of the reference MSA, whereas the first ORF results in a better fit for amino acids located after position 220 in the reference MSA. An early occurrence of an indel can yield a predominantly non-homologous amino-acid sequence with a large MSS and results in a score profile that is most similar to that of a mistranslated sequence.

3.1.3 Out of Class Contamination
Pipelines used for analyzing COI data, such as the one proposed in [18] and utilized in BOLD, suggest screening the COI sequences against a custom-compiled contamination database. This approach does not work when the sources of contamination are not cataloged. For instance, despite the screening carried out by BOLD on a small suite of possible contaminants, we were able to identify sequences with substantially divergent profile scores. For instance, sequence MANT163-13, was submitted to the BOLD database as originating from a Hymenoptera but showed a score profile curve that was consistently lower than the median score profile. This sequence of 217 amino acids has an MSS value of 196, suggesting a poor alignment over the complete sequence (See Figure 4). A Blast search of this sequence against the NR database (using the NCBI’s web BLASTP with default parameters) showed that most returned hits were of proteobacterial origin. More specifically, 49 of the top 50 BLAST hits were against Legionellaceae bacteria, with the best hit aligning with 98% similarity over the complete length of the sequence. Further investigation showed that Legionella can reside within insect hemocytes and that some insects are commonly used as models for studying the immune response to Legionella infections [7, 8]. Under the light of such evidence, we hypothesized that this sequence is in fact from a Legionellaceae pathogen, rather than from the Hymenoptera host. Other sequences were also predicted using BLASTP to be of bacterial origin. For instance, the top 50 best hits of sequence CNEIE0680-12 (MSS= 59) were against bacterial sequences in the Oxalobacteraceae family. For sequence SSWLE10476-13 (MSS = 47) which also shows a consistently low score profile that matched closely with that of MANT163-13, the three most significant Blast hits are to arthropods, two annotated at the class level (Insecta) and one annotated at the family level (Staphylinidae). The fourth hit, however, was against a bacterium Rickettsiella grphil. Interestingly, while we couldn’t initially neither confirm nor deny that this sequence represents bacterial contamination, a comment posted by a BOLD database curator in November 2015 identified this sequence as contamination. As such, we hypothesize that this sequence’s top Arthropoda hits are likely artifacts resulting from the propagation of erroneous annotation. These examples of contamination highlight the limitations in using a reference database for contaminant screening.

Indel containing sequences or chimeras are challenging to detect using alignment at the nucleotide level in the absence of references that are similar to it. However, the previous examples clearly illustrate that a switch in a score profile can be a reasonable tell-tale of a sequence anomaly, regardless of whether or not similar references are present in the database.

3.2 Identifying Anomalies in Out of Sample Data
We evaluated the performance of our method on out of sample sequences for which the status (anomalous vs. non-anomalous) is known. First, we generated two datasets from the sequences which we excluded from the reference MSA. The first dataset consisted of 100 randomly picked partial replicate sequence with a seed containing an MSS value less than 15, i.e., those are random sequences we believe are not anomalous. The second dataset consists of 50 randomly selected partial replicate sequences which aligned with a seed among the 70 we manually confirmed as anomalous. Sequences from both datasets were incorporated into the reference MSA using transitive alignments. Specifically, given a pairwise alignment of the partial replicate $a$ and its seed $b$, and given a reference MSA describing the alignment of the seeds $b$ and $c$, we can transitively align $a$ and $c$ by introducing gaps in $a$ so that its alignment in the reference MSA matches its first alignment on $b$ except where both $a$ and $b$ have gaps (See Figure 5 for an example). This transitive operation is unambiguous given that $a$’s alignment against
Figure 4: Score profile curves for three anomalous sequences (shown in red) plotted against typical score profile curves. The top and the bottom plots show entirely misaligned sequences representing an erroneously translated sequence and a misclassified sequence respectively. The middle plot shows a sequence with an upshift in the score profile occurring at the position of an insertion.

*b* does not include any gaps.

\[
\begin{align*}
1 & \quad a: \text{MK–AP–M–} \\
2 & \quad b: \text{MGDAPMKMA} \\
3 & \quad c: \text{–VDAVAPM–K–ARV}
\end{align*}
\]

Figure 5: The transitive alignment of partial replicate, *a*, against the seed *c*. 1) The Initial alignment used to derePLICATE *a* and keep *b* as a seed. 2) The alignment of seeds *b* and *c* in the reference MSA. 3) In addition to the original gaps introduced in *b* during the initial alignment (1), gaps from *b*’s alignment with *c* are also transitively added to *a*. The transitive gaps are shown in red.

For the first dataset, out the 100 sequences inspected, a single sequence (ASWAX567-08) had a MSS value above 15 (MSS value of 33). This suggested that this partial replicate was anomalous despite the fact that its seed (BBHEC816-10) had a MSS of 0. Upon manual inspection, we identified an insertion of an adenosine base at position 628, the removal of which produced a MSS of 0. This confirmed that sequenceASWAX567-08 was indeed anomalous.

For the second dataset, only 26 of the 50 sequences randomly generated from anomalous seeds had MSS values above 25. All the partial replicates with MSS values greater than 25 had the same irregular patterns observed in their seeds. However, the 21 remaining sequences had MSS values between 1 and 23, with an average MSS value of 5.1. After manually inspecting these sequences, we observed that these MSS-scoring sequences were shorter than their seeds and, in most cases, did not or did only partially cover the regions in their seeds where we observed the irregular patterns.

The outcomes form both out of sample datasets above show that partial replicates can have a different classification from that of their assigned seed. Therefore, one cannot screen dereplicated datasets and propagate the status of seeds onto the partial replicates they substitute. This applies whether or not we can confirm the seeds as anomalous. However, although our analyses did not extend to partial replicates – beyond the out of sample tests above – the computational tractability of our method facilitates the subsequent incorporation of partial replicates into the reference MSA and provides an efficient approach to screen additional sequences at a linear computation cost.

The seed sequences, the reference MSA a detailed analysis of the 70 sequence anomalies and all supplementary files can be found at the following url: [https://figshare.com/s/99bc4f716da30bba9327](https://figshare.com/s/99bc4f716da30bba9327).

### 3.3 Impact of the Window Size on the Identification of Anomaly Boundaries

The window size, *m*, plays a critical role in identifying the boundaries of an artifact causing a region of low-quality alignment in a sequence – particularly in the presence of large, conserved gaps. In essence, smaller values of *m* are ideal for identifying short regions that do not align well with the reference MSA, whereas larger values of *m* can smooth out short anomalies. For example, using *m* = 2, the *m*-smoothed scores for *S* = [5.5, 1.0, −2.1, 5.9, 5.5, −2.5]
are \( SS = [nan, 3.2, -0.5, 1.9, 5.7, 1.4] \), whereas the smoothed scores with \( m = 3 \) are \( SS = [nan, nan, 2.5, 2.5, 1.6, 2.2] \) (nan are assigned to position where the SS score cannot be computed to window constraints). As the windows size grows, the influence of distant, highly similar positions – including the gaps-rich columns introduced to compensate for rare artifacts – softens the contribution of misaligned columns and either shrink resulting MSS values or lead to completely missing the poorly aligned region. However, the influence of distant columns is desirable for merging windows of low conservation, when those are interspersed with gap rich column. Figure 6 illustrates this issue for a subsequence of the mistranslated seed (CNKOC293-14). Although this sequence has little similarity with the remaining seeds, the gap-rich columns of the reference MSA cause a significant upshift in the score profile curve. With a short window (ex. \( m = 5 \)), the two island of spurious similarity can interrupt the MMS extension, and therefore underestimate the extent of the anomaly.

![Figure 6: Alignment of mistranslated sequence along gaps-rich columns. Despite low similarity with the remaining seeds, the sequence shows score profile over the gap-rich columns.](image)

In the absence of a consistent window size for detecting anomalies, a reasonable strategy consists of employing a large window size to weed out conspicuous anomalies and to decrease the window size gradually to capture shorter anomalies. For instance, by using a window \( k = 15 \), we were able to identify 45 sequences with MSS values higher than 100. The removal of these sequences leads to more compact alignments (number of columns only 67% columns in the reference MSA contained gaps in 5,000 or more sequences compared to 79% before). Furthermore, removing 45 sequences identified with \( m = 15 \) leads to an increase in the observed MSS when reanalyzing the data using \( m = 11 \)

4. CONCLUSION

The COI marker has proved effective for metabarcoding projects and its use has made significant contributions in the field of ecology. Thus far, the process of cataloging new species in reference libraries has mostly relied on the use of targeted sequencing. This low-throughput activity has served the data curation process by keeping a handle on the number of errors in the COI reference databases. However, targeted sequencing is painstakingly slow and cannot be used to fill the large gap in sequence diversity. Thus, as researchers turn to high-throughput methods for cataloging diversity – notably for sequencing environmental DNA or bulk biodiversity samples –, automatic methods for scaling the validation of submitted COI sequences will become critical.

Here we propose a new approach that leverages the coding nature of the COI marker to probabilistically identify experimental errors that are challenging to detect at the nucleotide level. Our tests were able to identify numerous sequence anomalies in the well-curated BOLD database. Our results highlight the usefulness of this approach, both due to its computational tractability and its intuitive interpretability, and make a strong case for its use as a complement to existing tools to identify anomalous sequences in COI reference databases.

5. ACKNOWLEDGMENTS

We would like to thank Drs. Emma Ransome and Christopher Meyer for their helpful clarifications on sequencing biases in COI amplicon data. This work was supported by NSF-OCE grant number 1260169 and NIH-NIGMS grant number P30GM114737.

6. REFERENCES


ABOUT THE AUTHORS:

Mahdi Belcaid received a Ph.D in Computer Science from the University of Hawaii’s Information and Computer Sciences Department in 2012. He is currently an assistant research professor at the Hawaii Institute of Marine Biology. His current research focuses on the application of computational techniques for the analysis of large marine biology datasets.

Guylaine Poisson received a B.S. degree and a M.S. degree in Biological Sciences from the Univeristé de Montréal, Québec, Canada in 1994 and 1997, and a Ph.D. degree in Cognitive Computer Science from the Université du Québec à Montréal, Québec, Canada in 2005. She is currently an associate professor in the Department of Information and Computer Sciences at the University of Hawai‘i at Mānoa. Her research interests include bioinformatics and computational biology.
Automated SMT-based Consistency Checking of Industrial Critical Requirements

Predrag Filipovikj
Mälardalen University
Västerås, Sweden
predrag.filipovikj@mdh.se

Guillermo Rodriguez-Navas
Mälardalen University
Västerås, Sweden
guillermo.rodriguez-navas@mdh.se

Mattias Nyberg
Scania AB CV
Södertälje, Sweden
mattias.nyberg@scania.com

Cristina Seceleanu
Mälardalen University
Västerås, Sweden
cristina.seceleanu@mdh.se

ABSTRACT
With the ever-increasing size, complexity and intricacy of system requirements specifications, it becomes difficult to ensure their correctness with respect to certain criteria such as consistency. Automated formal techniques for consistency checking of requirements, mostly by means of model checking, have been proposed in academia. Sometimes such techniques incur a high modeling cost or analysis time, or are not applicable. To address such problems, in this paper we propose an automated consistency analysis technique of requirements that are formalized based on patterns, and checked using state-of-the-art Satisfiability Modulo Theories solvers. Our method assumes several transformation steps, from textual requirements to formal logic, and next into the format suited for the SMT tool. To automate such steps, we propose a tool, called PROPAS, that does not require any user intervention during the transformation and analysis phases, thus making the consistency analysis usable by non-expert practitioners. For validation, we apply our method on a set of timed computation tree logic requirements of an industrial automotive system called the Fuel Level Display.

CCS Concepts
\begin{itemize}
  \item Computing methodologies → Model verification and validation; Modeling and simulation; Model development and analysis;
\end{itemize}

Keywords
Requirements Consistency Analysis, Formal Methods, SMT, Z3

1. INTRODUCTION
Late detection of errors in the requirements specifications of industrial systems often results in a redesign or reimplementation of certain parts of the system, which leads to a considerable increase of costs. For these reasons, industry has high demands for techniques that enable early debugging of system specifications. This paper addresses the problem of detecting inconsistencies within system specifications, which occurs whenever the set of requirements is not realizable as such, due to internal contradictions. For illustration, let us look at the following example: assume a system $S$ that at any time operates in either of the two mutually exclusive operational modes $M_1$ and $M_2$. The mode change in the system is event triggered and is described by the following requirements: $R_1$: “If the event $P$ is observed, the system enters the $M_1$ operational mode and remains in the same for the next 5 time units”; $R_2$: “Whenever event $R$ occurs, it is immediately followed by an event $Q$”; and $R_3$: “If event $Q$ occurs, the system must switch to $M_2$ operational mode within 2 time units”. The system specification does not impose any restriction on the occurrence of the events $P$, $Q$ and $R$, meaning that there are two ways in which the system specification can be satisfied: trivially and non-trivially. Trivial satisfaction means that one constructs a system $S$ where events $P$, $Q$ and $R$ never occur. For such system it is impossible to violate the above system specification. However, we are interested in the following case: can the system specification be satisfied for a system in which events $P$, $Q$, $R$ do occur, and moreover they occur simultaneously? Intuitively, the answer to this question is negative, because the satisfaction of the specification would require the system to be simultaneously in modes $M_1$ and $M_2$ somewhere within 2 time units after the occurrence events of $P$ and $Q$.

For simple scenarios such as the one shown above, expert human-based debugging is usually enough to detect possible inconsistencies. However, the problem arises when one needs to deal with large system specifications composed of several tens or even hundreds of requirements. For such cases, a tool-supported approach, for instance based on formal methods, is needed. Most of the existing consistency analysis approaches [12, 3, 24] are based on model checking. Despite the methods being systematic and exhaustive, some of their characteristics, such as the complexity of constructing the analysis model or the analysis time, which in some cases has been reported to take days, limit the potential of such approaches to be adopted by industry. A less exhaustive, yet systematic and lightweight, approach could...
be more suitable for debugging the system specifications at early stages of system development, prior to the existence of any behavioral or structural model of the system.

As a potential solution to this need, in this paper, we propose a completely automated methodology based on Satisfiability Modulo Theories (SMT) for the consistency check of requirements specifications, starting from their description in natural language. We apply specification patterns to formalize the textual requirements into temporal logic, which are later transformed into a set of assertions encoded as a Satisfiability Modulo Theories Library (SMT-LIB) script that can then be fed to an SMT solver of interest. For performing the consistency analysis in this paper we use the Z3 SMT solver \cite{Z3}. Our idea of SMT-based methodology has already been discussed in our previous work \cite{Z3}, which in this paper is extended in two ways. First, the extension addresses the problem of automation. For that purpose, we propose a tool, called PROPAS, which automates the transformation of temporal logic formulas into SMT-LIB assertions suitable for analysis. Rather than encoding the SMT version of the system specification directly in a tool’s particular language, such as Z3 Python script, PROPAS generates a tool-independent encoding in the SMT-LIB language, which can be used as input to most modern SMT solvers that would fit the purpose. This constitutes the second extension of our work, as compared to our previous paper \cite{Z3}.

The paper continues as follows. In Section 2 we introduce the needed preliminaries, such as (timed) computational tree logic (TCTL), specification patterns, the formal definition of consistency and the satisfiability modulo theories together with the Z3 tool. Next, in Section 3 we describe the Fuel Level Display (FLD) system, which is an operational industrial system used as a working example for validating the proposed methodology, described in Section 3. In Section 5 we present our tool, PROPAS. Section 6 shows the application of PROPAS and our method on checking the consistency of the FLD example, using Z3, followed by discussion on the strengths and limitations in Section 7. We compare to related work in Section 8 and conclude the paper in Section 9 where we also outline future research directions.

2. PRELIMINARIES

In this section we introduce the concepts that are used in the rest of the paper. First, in Section 2.1 we introduce the computational tree logic (CTL) and its timed extension (TCTL), suitable for the specification of real-time systems, followed by an overview of the specification patterns that represent a user-friendly way to formally specify system requirements for non-experts in formal methods in Section 2.2. Next, in Section 2.3 we introduce a formal definition for consistency of a set of requirements encoded as temporal formulas, and finally in Section 2.4 we give an overview of the Satisfiability Modulo Theories (SMT) method and the Z3 tool used in our work.

2.1 (Timed) Computational Tree Logic

Computation tree logic (CTL) is a temporal logic used for the formal specification of finite-state systems. The interpretation of a CTL formula is defined over a branching model M that consists of a non-empty set of states S, a successor relation R that assigns a set of successor states to each state and a labeling function Label that assigns a set of atomic propositions to each state. Timed CTL (TCTL) is a timed extension of CTL suitable for specifying timed system properties. The concept of time is modeled through a set of non-negative real-valued variables called clocks, manipulated by clock formulas expressing constraints over the clocks. The clocks are incorporated into the notion of state, which includes the model’s location and clock valuation that determines the validity of clock constraints.

The syntax of CTL consists of path quantifiers (All, Exists), and path-specific temporal operators. The universal path quantifier “A” stands for “all paths”, while the existential quantifier “E” denotes that “there exists a path” from the set of all future paths P(s) starting from a given state s. A valid CTL formula is of type $\varphi \ U \psi$, where U (“until”) is the basic path operator, which is combined with either of the two path quantifiers. The rest of the path-specific temporal operators are defined based on the U operator, as follows: the F (Future) operator, which denotes that a formula eventually becomes true ($F \varphi \iff true \ U \varphi$) and the G (Globally) operator which denotes that a given formula is valid in all states along a given path ($G \varphi \iff \neg F \neg \varphi$)\cite{Z3}.

There exists also a weaker version of the U operator called “weak-until” denoted as W, with the following semantics: $\varphi W \psi \equiv (\varphi U \psi) \lor G\varphi$. It basically denotes a formula where the $\psi$ might hold, thus $\varphi$ must hold in all future states.

Timed CTL defines a timed version for each of the path-specific temporal operators based on clock constraints. In this paper, we use the following notation for timed path-specific operators: $Op_{\leq}^T$, where: Op$\in\{U, W, F, G\}$; $\leq \in \{=, <, \leq\}$ and T being a positive integer bound on clock variables. For instance, the formula $AF_{\leq T} \varphi$ denotes that on all execution paths starting from some initial state $s_0$, $\varphi$ eventually becomes true within T time units. For more details we refer the reader to previous work \cite{Z3}.

2.2 Specification Patterns

The Specification Pattern System (SPS) was introduced by Dwyer et al. \cite{Dwyer} to aid practitioners not skilled in formal methods to formally specify system properties. The proposed approach is based on the assumption that systems’ specifications are framed within reoccurring solutions, from which a set of patterns can be extracted and saved for future reuse.

The initial SPS catalog \cite{Dwyer} is compiled by analyzing more than 500 requirements from various domains. It contains 13 qualitative patterns, divided into two categories: order and occurrence, expressed in various temporal logics. Each pattern is characterized by two main parts: a behavior that it captures, and a scope that denotes the extent of program execution in which the behavior must hold. According to Dwyer et al. \cite{Dwyer}, there are five scopes defined as follows: "Globally" - the entire program execution, \ldots
the first occurrence of R, After Q - after the first occurrence of Q, Between Q and R - any part of program execution between events Q and R, After Q until R - similar to the previous, except that the occurrence of R is not mandatory.

Each pattern is expressed as a combination of literal and non-literal terminals. The non-literal terminals can be either boolean expressions that describe system properties or integer values that capture timing aspects. The rest of the pattern consists of literal terminals, which are fixed and cannot be changed.

In subsequent research endeavors, the initial SPS catalog has been extended in different ways. In one of the early extensions proposed by Konrad and Cheng [21], the catalog is enriched with real-time specification patterns intended to support the specification of real-time systems. The same extension introduces the constrained natural language (CNL) view of the patterns in addition to the various formalisms, such that properties become accessible to a broader set of users. There are other extensions of the specification patterns, such as the one by Grunske [17] which introduces probabilistic patterns. In this paper, we use only the patterns provided in the initial SPS catalog [9, 10] and the real-time extension by Konrad and Cheng [21].

2.3 Formal Definition of Consistency

Assuming that the system requirements specification has been encoded as a set of logical formulas, we can consider the following definition to check its consistency:

Definition 1 (Inconsistent specification). Let \( \Phi = \{ \varphi_1, \varphi_2, ..., \varphi_n \} \) denote the system requirements specification, where each of the formulas \( \varphi_1, \varphi_2, ..., \varphi_n \) encodes a requirement, respectively. We say that the set \( \Phi \) is inconsistent if the following implication is satisfied: \( \varphi_1 \land \varphi_2 \land ... \land \varphi_n \Rightarrow False \).

From the definition above, it follows that a system requirements specification is inconsistent if there does not exist a truth valuation of the conjunction of all the formulas in the specification. To disprove the inconsistency, it is enough to provide a witness set of valuations of variables that satisfies the conjunction of all the formulas \( \varphi_1 \land \varphi_2 \land ... \land \varphi_n \).

2.4 Satisfiability Modulo Theories, SMT-LIB and Z3

The problem of determining whether a Boolean formula can be made true by assigning true/false values to the constituent Boolean variables is called the Boolean satisfiability problem (SAT). If a given Boolean formula is satisfiable, the decision procedure generates a model that contains the valuation of the variables such that the formula is true. In the opposite case, that is, when the formula is not satisfiable, there exists no valuation for the constituent variables that will make the formula true. Satisfiability Modulo Theories (SMT) are an extension of SAT, in which some of the symbols are interpreted by a background theory [8]. One such example is the theory of arithmetic that restricts the interpretation of symbols to: \{ +, -, \leq, 0, 1 \}.

For SMT-based consistency analysis, in this paper we use the Z3 tool [7] from Microsoft Research, which is a state-of-the-art SMT solver and theorem prover. The input to the tool is a script composed of assertions that can be either declarations or formulas. The assertions are specified using the SMT-LIB language [5], which represents a standard input supported by most of the modern SMT solvers. Declarations can be either constants or functions. Constants are represented as uninterpreted functions with no inputs, whereas the functions are represented as uninterpreted functions that have one or more inputs. The data types in Z3 are called sorts, and the set of predefined ones consists of: Int, Real, Bool and Function. The set of sorts can be additionally extended by user-defined data types. The assertions in SMT-LIB language are specified in a Prolog-like manner. For example, an uninterpreted function \( \text{fun}_1 \) that accepts one parameter \( \text{param}_1 \) is specified as follows: \( (\text{fun}_1, \text{param}_1) \). The formulas express constraints over the declared variables, which are added to the internal stack using the \( \text{assert} \) command. There are two types of quantifiers: a universal (denoted as \( \forall \)) and existential (denoted \( \exists \)) one. For optimizing the decision procedure, there is a number of tactics provided by Z3.

The command \( \text{check-sat} \) determines whether the current formulas on the Z3 stack are satisfiable or not. If the set of formulas is satisfiable Z3 returns SAT, that in our case proves the analyzed consistency. If the set of formulas is not satisfiable, Z3 returns UNSAT, thus proving that the set of requirements is inconsistent. In cases when the Z3 cannot determine if the set of formulas is satisfiable or not, it returns UNKNOWN. When the command \( \text{check-sat} \) returns SAT, an additional command \( \text{get-model} \) can be used to retrieve an interpretation that makes all formulas on the Z3 internal stack true. In case of an UNSAT, the minimal inconsistent set of formulas is retrieved by calling the \( \text{unsat-core} \) command.

3. MOTIVATING EXAMPLE

In this section we describe the Fuel Level Display (FLD) system, whose requirements we want to analyze for consistency.

FLD is an operational system installed in all heavy-load vehicles produced by Scania, Sweden. The main functionality of the system is to estimate the remaining fuel in the fuel tank and display the correct value to the driver. It is realized through a cooperation between a number of computational components, sensors and actuators.

The estimation of the remaining fuel in the tank is implemented as a software function installed inside the Coordinator (COO) Electronic Control Unit (ECU). The value is calculated based on the following inputs: remaining fuel in the fuel tank (FT), provided by the fuel sensor (fuelSensor), and the current fuel consumption provided by the Engine Management (EMS) ECU. The system is classified as safety critical, meaning that its correct operation must be assured at all times. The system’s failure may lead to hazardous situations that potentially can cause severe material damage to the environment or even endanger human lives.

The simplified architectural breakdown of the FLD functionality is given in Figure [4]. The design is based on the concept of element, which is an extension over Heterogeneous Rich
Components [26]. All the entities in the system design, be they physical or logical, are represented via elements. In the architectural design given in Figure 1, the elements are denoted using rectangles (e.g., Fuel, PBS, ICL, etc.). The communication between an element and its environment occurs via its interface, represented as a collection of ports, denoted with gray rectangles. The behavior of the components is expressed via a set of constraints over their ports, specified according to the contract-based approach by using the assumption-guarantee type assertions [26]. For illustration, we list a subset of FLD requirements that we will model and analyze for consistency.

**SG** If actualParkingBrake (aPB) is false, then indicatedFuelVolume (iFV), shown by the fuel gauge, is less than or equal to actualFuelVolume (aFV).

**FSR** If it has not passed more than 1s since the last time CAN message DashDisplay (DD) appeared on CAN2 CAN bus, and the DD message is valid, then the iFV, shown by the fuel gauge, corresponds to FuelLevel (FL) signal value from the DD message.

**SSR** The Direct Memory Access (DMA) channel that corresponds to the input value of dmacCH when Dmac_enableCh() function is called, is enabled when Dmac_enableCh() function finishes its execution.

**SSR** The DMA channel that corresponds to the input value of dmacCH when Dmac_disableCh() function is called, is disabled when Dmac_disableCh() function finishes its execution.

Figure 1: Simplified version of the high-level architecture of the Fuel Level Display system.

Figure 2: PROPAS: Automated SMT-based consistency checking of requirements specifications.

4. SMT-BASED METHODOLOGY FOR CONSISTENCY ANALYSIS OF REQUIREMENTS

Our methodology for consistency checking of requirements is illustrated in Figure 2. It consists of four steps as follows: in Step 1, the system requirements are specified in constrained natural language (CNL) via the Specification Pattern System (SPS) [10] [21]. The formalized behavior encoded in the specification patterns and the user input are automatically combined to produce the temporal formulas, expressed in TCTL [1]. Since the SMT solvers operate over first-order logic (FOL) formulas, in Step 2, the TCTL patterns are transformed into FOL formulas by instantiating the semantics of path-specific temporal operators and path quantifiers. The FOL formulas are then encoded into SMT-LIB assertions in Step 3. In order to facilitate the analysis, the SMT-LIB assertions are additionally optimized by using a number of abstraction rules. Finally, in Step 4, we perform the consistency analysis using a state-of-the-art SMT solver (e.g., Z3), which returns the consistency verdict for the considered set of requirements.

In case the conjunction of the requirements is consistent (SAT verdict), the tool returns a model that contains a valuation of the system variables satisfying the analyzed requirements specification; in the opposite case (UNSAT), the tool generates the minimal inconsistent set (unsat-core command) containing the conflicting requirements. The traceability of the requirements starting from their natural language form until the SMT-LIB assertions used for analysis is assured by assigning a unique identifier which is preserved during all steps. To make the methodology potentially useful in industrial settings, we propose a tool called PROPAS that automates all the steps from the proposed method. The automation allows one to perform consistency analysis of their system specification with no intervention during the transformation and analysis steps, making PROPAS a suitable candidate for industrial adoption.
4.1 Step 1: Text to TCTL

The set of FLD requirements used in this study is originally specified in unrestricted natural language using a general-purpose text editor. Such free-text specifications in natural language are readable and expressive, yet sometimes ambiguous and definitely not amenable to automated analysis. In this section, we describe Step 1 of the method proposed in Figure 2, during which the free-text specification is converted into a more disciplined format with fixed structure and precisely defined semantics.

The formal system specification often represents a bottleneck as the industrial practitioners lack expertise in formal methods required for properly formalizing system specifications. To alleviate the problem, we adopt the specification patterns approach, which is considered a user-friendly formal specification approach that is expressive enough to capture requirements in the automotive domain. In addition to this, we use our in-house tool called SeSAMM Specifier that provides a user-friendly interface for requirements specification based on the specification patterns. The tool also offers mechanisms for validation of formalized requirements specification based on the specification patterns. The rest of the patterns are transformed in a similar fashion.

We formalize the complete set of FLD requirements using only five specification patterns, as shown in the list below. The frequency with which the patterns occur, that is the percent of the total FLD requirements specified using a specific pattern is given in Figure 3. The results are aligned with the earlier formalization attempts, revealing that, in principle, a small subset of SPS patterns suffices to express the majority of automotive systems’ requirements.

P1: Globally, Universally: $AG(\varphi)$

P2: Timed Globally, Universally: $AG(AG_{\leq T}(\varphi) \Rightarrow \psi)$

P3: Globally, Response: $AG(\varphi \Rightarrow A F_{\leq T}\psi)$

P4: After $\varphi$ Weak-until $\theta$ Universally: $AG(\varphi \Rightarrow A(\psi W_{\leq T}\theta))$

P5: Timed After $\varphi$ Weak-until $\theta$ Universally: $AG(AG_{\leq T}(\varphi) \Rightarrow A(\psi W_{\leq T}\theta))$

4.2 Step 2: TCTL to FOL

In this section, we present Step 2 of the methodology, during which the TCTL patterns are transformed into FOL formulas. The importance of this transformation is twofold: i) it bridges the semantic gap between TCTL formulas and SMT-LIB assertions, and ii) ensures the conservation of information between the two. This step is performed on a pattern level, meaning that once a pattern is transformed, all the requirements instances from that pattern recall the result from the structured derivation of the pattern. Due to page limitation and similarity of proofs, in this section we present only one lemma that shows the structured derivation of two of the TCTL patterns into equivalent FOL formulas. The rest of the patterns are transformed in a similar fashion.

The TCTL to FOL transformation is carried out by instantiating the semantics of the TCTL operators according to the definitions given by Katoen [20], assuming a timed transition system as the underlying semantic model of our system. The semantics of a TCTL formula is based on the following concepts: $\sigma$ denotes a single path from the set of all paths $P_M(s)$ starting from a given position $s$. Each path $\sigma$ is represented as a set of positions, denoted as $Pos(\sigma)$. A position in the path is a pair $(i, d_i)$, where $i$ is the location number, whereas $d_i$ is the time distance. The location determines the set of atomic propositions that are valid for that position, whereas the time distance is a real number that corresponds to the time elapsed during the delay transitions as compared...
from the initial state in the path; a set of such points characterizes the states traversed along \( \sigma \) while going from state \( s_i \) to the successor \( s_{i+1} \) for any \( i \in \mathbb{N} \). The time elapsed on a path relative to the initial state \( s_0 \) to any state \( s_i \) is defined as:

\[
\Delta(\sigma, 0) = 0,
\]

\[
\Delta(\sigma, i + 1) = \Delta(\sigma, i) + \begin{cases} 0, & \text{for discrete transition,} \\ d_i, & \text{for delay transition.} \end{cases}
\]

The elapsed time on a path is measured using real-valued variables called clocks, which increase with rate one. We denote a clock valuation by \( v \). The value of the clocks can be manipulated only through a reset action \( \text{reset} z \in v \), which sets a set of clocks to zero. The definition of the reset function is given as follows:

\[
\text{reset} z \in v(y) = \begin{cases} v(y), & \text{if } y \neq z, \\ 0, & \text{if } y = z. \end{cases}
\]

Lemma 1 below proves the conjectured equivalent FOL form of pattern P4 in TCTL, as a structured derivation that uses the FOL counterpart of a P1, which is also stated by Lemma 1. The proof for (1) has been omitted due to space limitation and the fact that similar proof already exists [20].

**Lemma 1 (P1, P4 into FOL).** Given a transition system \( M \), predicates \( \varphi, \psi, \theta \), a state \( s \) of \( M \), and \( \omega \) a clock valuation formula, the following two equivalences hold:

\[
(1) \quad s, \omega \models AG_{\geq 0}(\varphi) \\
\Leftrightarrow \forall \sigma \in P_M(s). (\forall i, d) \in Pos(\sigma). (\sigma(i,d), (z = \Delta(\sigma, i)) \models \varphi))
\]

\[
(2) \quad s, \omega \models AG_{\geq 0}(\varphi \rightarrow A(\psi W_{\leq T} \theta)) \\
\Leftrightarrow \forall \sigma \in P_M(s). (\forall i, d) \in Pos(\sigma). (\sigma(i,d), (z = \Delta(\sigma, i)) \models (\neg \varphi \lor (\forall \sigma' \in P_M(s,i,d), (\exists j,d'), (i < j \lor (i \land d \leq d')) \in Pos(\sigma'). \sigma'(j,d'),(z = \Delta(\sigma', j)) \models \theta \land z \leq T)) \land (\forall k,d \leq d' \leq T \in Pos(\sigma'). \sigma'(k,d'),(z = \Delta(\sigma', k)) \models (\psi \land z < \Delta(\sigma', j))) \lor (\forall \sigma' \in P_M(s,i,d). (\exists j,d'). (i < j \land d \leq d') \in Pos(\sigma'). \sigma'(j,d'),(z = \Delta(\sigma', j)) \models (\psi \land z \leq T))))
\]

Proof:

\[
(2) \quad s, \omega \models AG(\varphi \rightarrow A(\psi W_{\leq T} \theta)) \\
\Leftrightarrow \{ AG \equiv AG_{\geq 0}; \text{Rule: } \varphi \rightarrow \psi \rightarrow \neg \varphi \lor \psi, \text{definition of } W_{\leq T} \} \\
\]

\[
(2) \quad s, \omega \models AG_{\geq 0}(\neg \varphi \lor A(\psi U_{\leq T} \theta) \lor A(\psi U_{\geq T} \theta)) \\
\Leftrightarrow \{ \text{Definition of } U_{\geq 0}; \text{let } z \text{ be a ‘fresh clock'} \} \\
\]

Q.E.D.

The FOL formulas obtained by similar derivations, which correspond to the rest of the patterns are given below.

P2 \( \forall \sigma \in P_M(s). (\forall i, d) \in Pos(\sigma). (\sigma(i,d), (z = \Delta(\sigma, i)) \models (\exists \sigma' \in P_M(s,i,d), (\exists j,d'). (i < j \lor (i \land d \leq d')) \in Pos(\sigma'). \sigma'(j,d'),(z = \Delta(\sigma', j)) \models \theta \land z \leq T)) \land (\forall k,d \leq d' \leq T \in Pos(\sigma'). \sigma'(k,d'),(z = \Delta(\sigma', k)) \models (\psi \land z < \Delta(\sigma', j))) \lor (\forall \sigma' \in P_M(s,i,d). (\exists j,d'). (i < j \land d \leq d') \in Pos(\sigma'). \sigma'(j,d'),(z = \Delta(\sigma', j)) \models (\psi \land z \leq T))))

P3 \( \forall \sigma \in P_M(s). (\forall i, d) \in Pos(\sigma). (\sigma(i,d), (z = \Delta(\sigma, i)) \models (\neg \varphi \lor (\forall \sigma' \in P_M(s,i,d). (\exists j,d'). (i < j \land d \leq d') \in Pos(\sigma'). \sigma'(j,d'),(z = \Delta(\sigma', j)) \models (\theta \land z \leq T)) \land (\forall k,d \leq d' \leq T \in Pos(\sigma'). \sigma'(k,d'),(z = \Delta(\sigma', k)) \models (\psi \land z < \Delta(\sigma', j))) \lor (\forall \sigma' \in P_M(s,i,d). (\exists j,d'). (i < j \land d \leq d') \in Pos(\sigma'). \sigma'(j,d'),(z = \Delta(\sigma', j)) \models (\psi \land z \leq T))))

P5 \( \forall \sigma \in P_M(s). (\forall i, d) \in Pos(\sigma). (\sigma(i,d), (z = \Delta(\sigma, i)) \models (\neg \varphi \lor (\forall \sigma' \in P_M(s,i,d). (\exists j,d'). (i < j \land d \leq d') \in Pos(\sigma'). \sigma'(j,d'),(z = \Delta(\sigma', j)) \models (\theta \land z \leq T)) \land (\forall k,d \leq d' \leq T \in Pos(\sigma'). \sigma'(k,d'),(z = \Delta(\sigma', k)) \models (\psi \land z < \Delta(\sigma', j))) \lor (\forall \sigma' \in P_M(s,i,d). (\exists j,d'). (i < j \land d \leq d') \in Pos(\sigma'). \sigma'(j,d'),(z = \Delta(\sigma', j)) \models (\psi \land z \leq T))))

Returning to Definition [1] the conjunction of all requirements is obtained by instantiating the above patterns in FOL. Next, the conjunction is encoded as SMT-LIB assertions, which can then be used as an input into an SMT solver of choice, Z3 in our case, for checking the consistency.

### 4.3 Step 3: Encoding in SMT-LIB Language

In this section, we present the third step of our methodology, in which FOL formulas are encoded as SMT-LIB assertions,
which can be analyzed by state-of-the-art SMT solvers. In this step, we apply the following three encoding rules:

R1: Directly map the FOL constructs into SMT-LIB syntax elements. For instance, mapping the quantifiers (∀ into forall, ∃ into exists, etc.), modeling port values as functions of time, etc.

R2: Reduce complexity by abstraction: (a) eliminate path (σ) universal quantifiers, and (b) collect location (i) and time in location (d) into a tuple position (pos).

R3: Abstract the universally quantified pos = (i, d) to the universally quantified pos.d.

The process of applying rules R1, R2 and R3 on the set of patterns from Section 4.2 can be illustrated as follows:

P1 \xrightarrow{R_1,R_2} P'_1 \xrightarrow{R_3} P_{SMT}, i \in [1, 5]

The application of rule R1 results in an SMT-LIB script where each assertion corresponds to an individual requirement, with quantifiers and Boolean expressions encoded using SMT-LIB-specific constructs. Due to the underlying branching model over which the TCTL formulas are interpreted (Section 2.1), the assertions are quantified over three variables: execution path (s-paths), locations and clock values. However, only the clock quantifiers are bounded due to the timed-constrained nature of the system specification.

The number of quantified variables has a negative impact on the decidability of the SMT procedure [23]. To remedy this, we propose an abstraction technique (rules R2 and R3) that reduces the number of quantified variables in the assertions, abstracting only the information related to variables that cannot be sources of requirements inconsistency (e.g., σ, and i). This is possible because all formulas are universally quantified over the branches, and there is no fixed labeling function as the model of the system is not available. Further, we collect location and time variables into a tuple position, denoted by pos.d. To access the location component of the position we write pos.i. Similarly, time valuation in that position is obtained by pos.d.

For the FLD requirements, the path component of all FLD properties is always universally quantified because all the requirements are safety requirements, meaning that no inconsistency can occur due to path quantifiers, as existentially quantified path properties do not exist in our case study. Therefore, proving consistency on an arbitrary path of infinite length (chosen via the “select” operator) suffices. Consequently, the quantified path variable disappears in our SMT-LIB encoding.

All our patterns rely on semantic models in which progress is ensured by instantaneous discrete transitions in which location index pos.i increases, or via delay transitions, which model the passage of time while the system remains in the same location, causing an increase of the time distance compared to the initial position on the path, that is pos.d increases. The progress along the path is modeled by the binary operator “≪” that compares positions, defined as: pos ≪ pos' ⇔ (pos.i < pos'.i) ∨ (pos.i = pos'.i ∧ pos.d < pos'.d).

Possible inconsistencies can arise from contradicting formulas that should hold in each position, e.g. φ, ψ etc., or at from a certain time point on.

By applying the rules R1 and R2 explained above we obtain the following valid abstracted versions of patterns P1-P5:

P1′: select σ ∈ P_M(s). ∀pos ∈ Pos(σ).pos.i, (z = Δ(σ, pos.i) ≫ φ)

P2′: select σ ∈ P_M(s). ∀pos ∈ Pos(σ).pos.i, (z = Δ(σ, pos.i) ≫ (∼−(select σ ∈ P_M(pos).∃pos'.pos ≪ pos' ≪ pos + T) ∈ Pos(σ'), pos'.i, (z = Δ(σ', pos'.i)) ≫ (z ≤ T ∧ φ)) ≫ ψ))

P3′: select σ ∈ P_M(s). ∀pos ∈ Pos(σ).pos.i, (z = Δ(σ, pos.i) ≫ (∼φ ∨ (select σ ∈ P_M(pos).∃pos'.pos ≪ pos' ≪ pos + T).pos'.i, (z = Δ(σ', pos'.i)) ≫ (z ≤ T ∧ φ))))

P4′: select σ ∈ P_M(s). ∀pos ∈ Pos(σ).pos.i, (z = Δ(σ, pos.i) ≫ (∼φ ∨ (select σ ∈ P_M(pos).∃pos'.pos ≪ pos' ≪ pos + T).pos'.i, (z = Δ(σ', pos'.i)) ≫ (z ≤ T ∧ φ))))

P5′: select σ ∈ P_M(s). ∀pos ∈ Pos(σ).pos.i, (z = Δ(σ, pos.i) ≫ (∼φ ∨ (select σ ∈ P_M(pos).∃pos'.pos ≪ pos' ≪ pos + T).pos'.i, (z = Δ(σ', pos'.i)) ≫ (z ≤ T ∧ φ))))

Finally, we apply rule R3 on P1′, ..., P5′ to abstract ∀pos and ∃pos into ∀pos.d and ∃pos.d as it is the only component of the tuple that counts for the inconsistency checking. The abstracted patterns can be encoded into SMT-LIB assertions that contain only one quantified component, pos.d modeled via the real-valued variable “time”. In addition, the predicates (φ, ψ, θ) in the FOL patterns are substituted with Boolean expressions over the system variables represented as functions of time denoted as (φ time), (ψ time) and (θ time), respectively. The complete set of patterns encoded in SMT-LIB is presented in the list below:

P1_{SMT} (forall((time Real))(= (var1 time) val1))

P2_{SMT} (forall((time Real))(=> (and(= (var1 time) val1))

(not(exists ((time1 Real)and (not (= (var1 time) val1))

(= (time1 time))(= time1 (+ time T))))))

(= (var2 time) val2))

P3_{SMT} (forall((time Real))(=> (= (var1 time) val1))

(exists ((time1 Real)and (var1 time)(< time1 (+ time T))))

(= (var2 time) val2))

P4_{SMT} (forall ((time Real))(=> (= (var1 time) val1)

(or (exists ((time2 Real)and (var2 time)(< time2 (+ time T))(= (var2 time1) val2))not(exists ((time2 Real)

(=> (var2 time2) val2))(not (= (var2 time3) val3))))))not(exists ((time3

val3)))))

APPLIED COMPUTING REVIEW DEC. 2017, VOL. 17, NO. 4 21
The original set of requirements expressed via patterns $P_1$, ..., $P_5$ are stronger than their counterparts encoded in SMT-LIB using the abstraction rules. Consequently, proving the inconsistency of the encoded versions means proving the inconsistency of the original ones, however the similar inference does not hold for the positive case in which the encoded versions are proven consistent. However, practically, we can still infer the consistency of the original requirements if their abstract encoding is satisfiable, based on our argumentation around the only possible sources of inconsistency.

In the next section, we show how the complete procedure of formalization of natural language requirements and their transformation into a format suitable for SMT-based consistency analysis is automated via our PROPAS tool. The automation is a very important feature of the approach that makes it accessible for users not skilled in formal methods who want to check the consistency of their specifications and contributes to its potential adoption in industrial settings.

5. TOOL SUPPORT: PROPAS

In this section, we present our tool called PROPAS (PROperty PAtten Specification and analysis), illustrated in Figure 2 which automates the procedure of transforming the TCTL formulas into an SMT-LIB script suitable for analysis with a state-of-the-art SMT solvers. The core of the tool is a library called SMTLibReq, which provides the underlying functionalities for transforming the temporal formulas into SMT-LIB, by automating Steps 2 and 3 (described in Sections 4.2 and 4.3 respectively) of our methodology. It is an open source project, with the source code available freely [13].

5.1 The SMTLibReq Library

The high-level design of the SMTLibReq library is given in Figure 3. The overall functionality of the library is realized by two components, depicted as squares: i) a parsing engine, called ExpressionParser, that transforms logical expressions given as strings into binary tree structures encoding both the semantics and the syntax of each part of the original expression (the Formula, represented with a circle in Figure 3); and ii) a transformation engine, called FormulaTransformer, that transforms formulas into a formalism of interest, in this case, an SMT-LIB standard script.

The SMTLibReq library is designed modularly. There are numerous advantages of such design, including reduced complexity, separation of concerns, higher testability and maintainability. An additional benefit is the fact that it can be used independently of the PROPAS tool, meaning that it can be used as an external library in any other project or tool developed using the C# programming language within the .NET framework.

5.1.1 ExpressionParser

The main functionality of the ExpressionParser is to parse logical formulas encoded in TCTL provided as arrays of characters (strings) into a tree-structure that encodes both the syntax and the semantics of the input formula. In the following, the input formula provided as string will be referred to as expression, whereas the tree structure will be referred to as formula. Concretely, the produced formula is of type binary tree, which is suitable for encoding unary and binary operators. Considering the fact that the primary focus of our work are system properties expressed in TCTL, the expressiveness of binary tree suffices. In the binary-tree structure there are two types of nodes: internal and leaf nodes. Each internal node represents an operator, be it unary or binary, whereas the leaf nodes represent the atomic propositions of the expression.

The parsing of the expressions into formulas is performed in two steps, called flattening and parsing, respectively. The flattening procedure represents a preprocessing routine, during which complex formulas composed of a number of sub-formulas are transformed into flat ones. Such flat structures are then used as an input for the parsing procedure that generates the binary tree that encodes the original formula. Both the flattening and parsing procedures are generic and applicable to most of the formalisms, whereas the operators and their ordering must be defined for a specific one. Currently, the ExpressionParser supports the transformation of TCTL and FOL properties composed of binary operators only into SMT-LIB assertions. For the next releases, we plan to extend the library to support more formalisms. In the following, we illustrate the transformation process of a logical expression into a formula.

Let us consider the following TCTL formula, encoded as string: $AG(p \Rightarrow AF_{\leq 7}(q <> r))$, which captures the following behavior: it is always the case that whenever the proposition $p$ becomes true, then eventually within $T$ time units $q <> r$ has to become true as well. One should note that in our work the “$<>$ ” operator denotes boolean inequality and not the future path-specific temporal operator, which is denoted
During the parsing procedure, the original formula is analyzed for existence of sub-formulas. In this case, the expression contains one sub-formula: $AF_{\leq T}(q <> r)$. In order to create a flat expression, the sub-formula is replaced by a temporary atomic proposition, with the sub-formula placed in hashtable. After the substitution is applied, we obtain the following flat formula: $AG(p \Rightarrow exp1)$ (Figure 5[a]). A hashtable entry is created, with $exp1$ being the key and the sub-formula being the value. This procedure is recursive and starts from the atomic sub-formulas and is build upwards. For TCTL, an atomic sub-formula is a TCTL formula that does not contain nested path-specific operators or quantifiers over paths. The flattened expression is then passed to the parser engine together with the hashtable that contains the mappings.

For parsing the TCTL expressions into formulas, the parsing engine assumes the following order of operators ($AG$, $AF$, $A$, $U$, $W$, $\Rightarrow$, $||$, $&&$, $!$, $==$, $<>$, $\oplus$, $\ominus$), from the weakest to the strongest binding. This means that arithmetic operators have the strongest binding, followed by logical operators, with path-specific temporal operators and the path quantifiers having the weakest binding.

A flattened TCTL formula has the following properties: it contains at most one path-specific temporal operator, at most one path quantifier and one or more logical operators. In some cases, path-specific temporal operators and the path quantifiers are considered as one operator in the binary tree (ex: $AG$, $AF$), whereas in the case of $U$ and $W$ operators they are always treated separately. Once the temporal operators have been identified and parsed, the logical formula is then transformed according to the predefined ordered list of logical operators and arithmetic ones. Everything else is considered to be an atomic proposition.

When the parser encounters an atomic proposition, it looks in the hashtable to check whether it is an original one, or it corresponds to a mapping introduced by the flattening procedure. In the given example, $p$ is an atomic proposition from the original formula, whereas the proposition $exp1$ corresponds to a mapping. Once the mapping is identified, the original sub-formula is retrieved from the hash-table and the procedure continues with the sub-formula being treated as a new property. The sub-tree that the sub-formula generates is then simply appended to the original tree by replacing the mapping leaf node (Figure 5[b]). The procedure terminates once all the mappings from the hashtable have been transformed.

### 5.1.2 FormulaTransformer

The second component in the SMTLibReq library is the FormulaTransformer that parses the binary tree formula into a format that can be used as a direct input into an SMT solver. Similar to the ExpressionParser, the FormulaTransformer component has its basic functionality customizable for particular formalisms. The customization depends on two factors: first, the semantics and the syntax of the input formula (the parsing engine operates on a predefined set of operators and their semantics), and second, the format of the output, which can for instance be the general SMT-LIB, a Z3 Python script, or something else. For the sake of simplicity, in this section, we describe a transformation procedure of a TCTL binary-tree formula into SMT-LIB, which is the same as any other transformation procedure except that it “understands” the semantics of the TCTL and FOL operators only, thus is capable of generating an SMT-LIB script that can be used as an input to state-of-the-art SMT solvers.

The main procedure of transforming the binary-tree formulas into an SMT-LIB script involves two main activities: i) creating assertions that describe the constraints encoded in the formula, and ii) creating the declarations that define the variables that are used within the model.

The parsing of the binary-tree formula is performed top-to-bottom starting from the root node of the tree. The transformation for both the operators and the atomic propositions is quite similar. For processing each node, be it internal or leaf, the procedure analyzes the expression and loads an appropriate template used to instantiate the expression into an SMT-LIB construct. Each template is composed of literal and non-literal parts, with the literal parts being fixed, and the non-literal parts being replaced by the formula-specific expressions.

Once the template has been loaded, the non-literal parts of the template are instantiated by the template-instantiation function. In order to illustrate the parsing procedure, we recall the same example as previously, i.e. we apply the transformation on the binary-tree given in Figure 5[b].

The transformation starts from the root node, which in this case contains the $AG$ operator. The template selection engine analyzes the operator expression and determines that it should be transformed using the quantifier template, given as:

```plaintext
(#quantifier# (#quantifiedVariable#) (#expression#)).
```

The expressions surrounded by hash symbols represent the non-literal terms of the template. The loaded template then represents the basis for the template instantiation functionality that determines the non-literal terms and substitutes them in the template. For the $AG$ expression, the template instantiation engine determines that the $#quantifier#$ will be replaced with a universal quantifier over a real-valued
variable that models the notion of time. To be able to determine the formula specific value of the #expression# part, the parsing calls the parsing procedure for the child nodes of the current one. All the subsequent operator nodes are transformed in a similar fashion. The recursive transformation stops at the leaf nodes. Since the leaf nodes contain the atomic propositions, when such a node is parsed, a declaration is additionally created for the variables defining that particular atomic expression. For that purpose, the engine determines the type of the variable (which in SMT-LIB can be either a constant or a function) and its inputs and output and creates an appropriate declaration for it. Once the complete formula structure has been traversed, the assertion is added to the set of assertions and the set of declarations produced by the formula is added to the set of declarations of the script.

After the complete formula from Figure 5b has been completed, we obtain the following assertion:

\[
(\text{forall } ((\text{time } \text{Real})) \implies (\text{implies } (\text{p time } 1.0) (\text{exists } ((t1 \text{Real})) (\text{and } (> t1 \text{time}) (\langle t1 (+ \text{time} T)) (\not= (q t1) (r t1))))) )))
\]

accompanied by the following declarations:

\[
(\text{declare-const } T \text{ Real})\\(\text{declare-fun } p (\text{Real}) \text{ Real})\\(\text{declare-fun } q (\text{Real}) \text{ Real})\\(\text{declare-r fun } r (\text{Real}) \text{ Real})
\]

6. CONSISTENCY ANALYSIS OF FLD REQUIREMENTS USING Z3

In this section, we describe the process of consistency analysis of the SMT-LIB assertions generated during the previous step.

In order to be able to analyze the FLD requirements, we integrate the SMTLibReq library into the SeSAMM Specifier, which is possible due to the modular design of the PROPAS (see Section 4.4). Then, one can automatically generate an SMT-LIB script that corresponds to the complete set of requirements for the FLD system. An excerpt of the script that corresponds to the requirements presented in Section 4.1 is given as follows:

\[
(\text{assert (! (forall } ((\text{time } \text{Real})) \implies (\langle (\text{aPB time} 0.0) \not= (\text{iFV time} \text{aFV time}))))) ) :\text{named SG})
\]

\[
(\text{assert (! (forall } ((\text{time } \text{Real})) \implies (\langle (\text{CAN2 time} \text{DD time}) (\not= (\text{DD time} \text{ERR})) (or (exists (t1 Real)) (\langle t1 \text{time} \langle t1 (+ \text{time} 100)) (\not= (\text{CAN2 t1} \text{DD t1})) (not (exists (t1 Real)) (\langle t1 \text{time} \langle t1 (+ \text{time} 100)) (\not= (\text{iFV t1} \text{DD t1) 1.0}))))))) ) :\text{named FSRICL})
\]

\[
(\text{assert (! (forall } ((\text{time } \text{Real})) \implies (\langle (\text{DmacDis-ableCh time chU32} 1.0) (\not= (\text{DmacCH time chID} 1.0)))) ) :\text{named SSR2DMAC})
\]

By analyzing the assertions generated based on the requirements, we notice that the analysis process can be additionally optimized by encoding the domain knowledge not explicitly captured by the requirements. An example of such information is the fact that the fuel level cannot be less than zero or greater than the tank size. Below, we show one such assertion, actualFuelBound, which bounds the value of the actualFuelVolume parameter to a range of allowed values used in the FSR_ICL requirement of Section 4.

\[
(\text{assert (! (forall } ((\text{time } \text{Real})) (\langle (\text{aFV time} 0) (\langle (\text{aFV time} \text{TANK_SIZE}))) ) :\text{named actual- FuelBound}))
\]

Four out of five TCTL patterns include implication, so they can be trivially satisfied if the antecedent evaluates to false. For example, FSR_ICL is trivially satisfied if CAN2(time) = DD(time) never evaluates to true. To prevent the trivial satisfaction of the requirements, we explicitly instruct the solver to check for satisfiability when all of the antecedents hold. The hidden problem of using this technique may arise from requirements that model complementary behavior, so enabling of the antecedents must be performed carefully in order to avoid false positive inconsistencies.

The SMT-LIB script containing 36 assertions that correspond to the FLD requirements has been analyzed using the Z3 tool on a Linux machine with 2.4 GHz Dual Core processor and 4GB RAM. Using the unbounded model-based quantifier instantiation (mbqi) the procedure does not terminate within 48 hours, whereas bounding the procedure to a maximum of 1000 runs for generating the model yields the verdict UNKNOWN. To determine the cause of non-termination of the SMT analysis, we incrementally insert the requirements one by one, into the solver, and perform the consistency analysis on every step. In this way, we are able to isolate the requirements for which the SMT procedure cannot terminate. By applying this strategy, we discover two classes of requirements: the ones for which the SMT procedure terminates (solvable) and the ones for which it does not, called non-solvable. In the following, we discuss the characteristics of both classes and the mitigation strategy used for the non-solvable ones.

**Solvable Requirements.** The requirements formalized by instantiating the patterns P1, P2 and P3, which represent 73% of the total requirements (see Figure 3) do not hinder termination of the SMT analysis process; an input script constructed exclusively from such requirements is analyzed within seconds. This shows that the tool can handle pattern instances with a maximum of two nested quantifiers without difficulty. Pattern P1 contains only one universal quantifier
In its current state, the SMTLibReq library provides the necessary means for transforming the temporal formulas encoded in TCTL into SMT-LIB constraints suitable for consistency analysis using any state-of-the-art SMT solver. Despite the fact that the TCTL specifications are generated using patterns, the implemented transformation procedure is more general and can be applied on any valid TCTL formula, be it part of the original set of patterns or not. This feature enables us to expand the set of patterns used for formalizing the requirements, while not jeopardizing the correctness of the final result. However, one should note that the current version was tested over the patterns from the FLD specification only, so evaluation with other patterns or arbitrary TCTL formulas may lead to situations that the tool cannot handle.

Mitigating Non-solvable Requirements. In order to tackle the requirements formalized using patterns P4 and P5, one of the nested quantifiers must be eliminated. By analyzing the semantics of the patterns, we find that an additional existential quantifier is added to model the sporadic events. These quantifiers can be eliminated by converting the sporadic events into periodic, that is, by providing a witness valuation from the set of allowed values. For illustration, we apply this technique on the FSR_{ICL} requirement (see Section 4.4). The original requirement captures the sporadic occurrence of the event CAN2 = DD. By applying our mitigation strategy, we modify the requirement such that the given event occurs every 100 time units after the antecedent is satisfied. In the TCTL form, we replace the W_{≤100} with U_{100}, which results in the following formula:

\[ AG(CAN2 = DD \land DD \neq ERR) \Rightarrow iFV = DD U_{100} CAN2 \neq DD. \]

This model is pessimistic but still valid, since once a witness is found, the satisfaction of the original formula follows.

After applying the mitigation technique, the SMT analysis over the complete set of FLD system requirements returns SAT accompanied by a valid model within. To validate that our approach can detect temporal inconsistencies, we perform controlled faulty assertions injection. Examples of such assertions include: enabling requirements expressing mutually exclusive behaviors (SSR_{DMAC} and SSR_{DMAC}) at the same time, or assertions that violate existing ones. All of the injected faults have been detected by Z3, and the conflicting assertions (requirements) contained in the minimal inconsistent set have been generated by the solver using the unsat-core command.

8. RELATED WORK

Various approaches for checking requirements consistency, based on different definitions of consistency and different analysis techniques, have been proposed in the literature.

A consistency checking procedure similar to ours has been proposed by Barnat et al. [3]. The authors define a model-free sanity-checking procedure including consistency for system requirements specifications in Linear Temporal Logic (LTL) by means of model checking. The notion of consistency is reduced to checking whether an automaton obtained as a conjunction of all the formulas in the specification has a non-empty accepting language. The same has later been extended [2] to be able to generate a minimal inconsistent set of requirements. The approach relies on tool support similar to our PROPAS, which uses specification patterns for formal system specification and a parallel LTL model-checker called DiVinE [4]. Despite the exhaustiveness, the approach suffers
from the inherent complexity of transforming the LTL formulas into automata, especially for large systems, thus potentially making its application in industrial contexts challenging. A similar approach for consistency checking of requirements specified in LTL is proposed by Ellen et al. [12]. The paper presents a so-called existential definition, that is, the existence of at least one run of the system that satisfies the complete set of requirements - which is an approach close to ours. For performing the consistency checking, the authors use bounded model checking using the isAT SMT solver [11]. The proposed technique is capable of generating a maximal set of consistent requirements, as well as a minimal inconsistent subset of requirements. Similar to our approach, the tool has been integrated into the existing tool called BTC Embedded Specifier [6], which is a proprietary software that relies on their tool-specific specification patterns.

The work by Post et al. [24] defines the notion of rt-(in)consistency of real-time requirements. The notion covers cases where the requirements in the system’s requirements specification can be inconsistent due to timing constraints. The checking for rt-inconsistency is reduced to model checking. Compared to our approach, the generation of the formalism that is subjected for consistency analysis is performed manually.

The notion of consistency is also checked for requirements specified in domain-specific notations. Heimdhal and Leveson [18] provide an approach for consistency analysis for requirements specified in RSML (Requirements State Machine Language). The proposed definition for consistency is suitable only for requirements specified in RSML and is not applicable for requirements expressed in any other notation. Real-time embedded systems can also be specified using the Software Cost Reduction (SCR) method. The SCR method is suitable for specifying both functional and extra-functional system requirements. A complete suite for analyzing system specifications in SCR has been developed by Heitmeyer et al. [19]. The suite provides tools for requirements specification, symbolic execution and formal analysis.

Despite the fact that the approaches above [2, 3, 24] can exhaustively check for the consistency of requirements specifications, all of them suffer from one major limitation, which is the verification time that grows exponentially with the number of requirements. In the early phases of system requirements specification, a more lightweight and considerably faster procedure as proposed in this paper might be more suited. Hence, our method can be used as a complementary approach to the above listed methods for consistency checking. Another important aspect of our approach is the fact that the complete procedure is completely automated and the formal system specification is performed in a user-friendly manner using the specification patterns.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an automated solution for SMT-based consistency analysis of formal requirements specifications encoded as Timed CTL formulas. The proposed solution is supported by our tool called PROPAS, which completely automates the requirements transformation and consistency analysis procedures.

The implementation of the PROPAS tool provides a “push-button-analysis” approach, meaning that the complete process of transformation and ultimately the analysis of the requirements is completely hidden from the users. The generation of the analyzable format is performed according to well-justified abstraction rules that simplify the original formulas to ensure their analyzability while assuring the preservation of information that could contribute to potential inconsistencies. Our initial validation of the tool on an industrial case, the Fuel Level Display from Scania, shows its potential for consistency checking of running industrial systems. However, the full potential of the tool needs to be tested on more complex and larger system specifications.

Given the current status of the underlying SMT-based consistency analysis methodology and its automation though the PROPAS tool, there are several directions for future research. The SMT-based methodology can be improved in several ways, including: proposing a more general definition of consistency, improving the encoding in order to speed up the analysis procedure, which could possibly lead to the tool being able to analyze other classes of formal properties also. Moreover, we have to address the missing features and the limitation of the current version of the PROPAS tool, discussed in Section [7]. The fact that the transformation process is completely automated opens up the possibility to validate the tool at a more extensive scale, on larger industrial systems. To meet such a goal, we have already started preparing future case studies, based on running industrial examples to further investigate the boundaries of applicability of our tool and method.

Acknowledgments

This work has been funded by the Swedish Governmental Agency for Innovation Systems (VINNOVA) under the VeriSpec project 2013-01299.

10. REFERENCES


ABOUT THE AUTHORS:

Predrag Filipovikj is a PhD student at Mälardalen University, Sweden since January 2014. He holds MSc degrees in Software Engineering from Mälardalen University, Sweden, and Computer Networks and e-Technologies from Ss. Cyril and Methodius University, Macedonia, respectively, and a Licentiate degree in Technology from Mälardalen University, Sweden. His research is focused on the application of formal methods for increasing the quality of cyber-physical systems, with particular focus on the automotive domain. His active topics of research include: engineer-friendly formal system specification and analysis, and formal verification of design-time models mostly by model checking. He performs research in cooperation with Scania AB CV and Volvo Groups Truck Technology, both from Sweden. He is a student member of ACM and IEEE, and a member of the Swedish Requirements Engineering Research Network.

Guillermo Rodriguez-Navas received the telecommunication engineer degree from the University of Vigo, Spain, in 2001, and the doctorate in informatics from the University of the Balearic Islands (UIB), Spain, in 2010. At present, he is senior lecturer at the School of Innovation, Design and Engineering of the Mälardalen University, Sweden. He has authored or co-authored +60 peer-reviewed scientific papers and is currently co-supervising three doctoral dissertations. His research interests include dependability and safety aspects of complex embedded systems, fault tolerance for distributed embedded systems, clock synchronization, scheduling algorithms for real-time time-triggered networks, and application of formal verification techniques to requirements engineering and system specification, particularly in the domains of automotive and cyber-physical systems.

Mattias Nyberg is an adjunct (part-time) professor at Royal Institute of Technology (KTH) in the department of Mechatronics. His main affiliation is Scania CV AB, a leading global heavy-truck manufacturer. He received a PhD in Electrical Engineering from Linköping University in 1999 specializing in vehicular systems. After dissertation he has worked mainly in industry; first for Daimler in Stuttgart, Germany, with diesel engine diagnosis, and later at Scania with diagnosis and functional safety. In parallel with his industrial career, he is very active in academic research. He has supervised six PhD students in the area of diagnosis and functional safety. He is also an author of more than 100 scientific publications, and has received the SAE Vincent Bendix award for the best paper of year 2015 in the area of automotive electronics engineering.

Docent Cristina Seceleanu is Associate Professor at Mälardalen University, Sweden, Embedded Systems Division, and leader of the Formal Modeling and Analysis of Embedded Systems research group. She has a MSc. in Electronics (Polytechnic University of Bucharest, Romania) and a Ph.D. in Computer Science (Åbo Akademi, Finland). Her research focuses on developing formal models and verification techniques for predictable real-time and adaptive embedded systems. She has been involved in several national and European research projects (FP7, ARTEMISIA, AAL), out of which she is currently leading three. She has served in the editorial boards of several journals such as Frontiers in ICT: Formal Methods, and International Journal of Embedded and Real-Time Communication Systems (IJERTCS), as well as in the PC of about 100 conferences in the field.
Contextual Understanding of Microservice Architecture: Current and Future Directions

Tomas Cerny  
Computer Science  
Baylor University  
Waco, TX, USA  
tomas_cerny@baylor.edu

Michael J. Donahoo  
Computer Science  
Baylor University  
Waco, TX, USA  
jeff_donahoo@baylor.edu

Michal Trnka  
Computer Science, FEE  
Czech Technical University  
Prague, Czech Republic  
trnkam11@fel.cvut.cz

ABSTRACT

Current industry trends in enterprise architectures indicate movement from Service-Oriented Architecture (SOA) to Microservices. By understanding the key differences between these two approaches and their features, we can design a more effective Microservice architecture by avoiding SOA pitfalls. To do this, we must know why this shift is happening and how key SOA functionality is addressed by key features of the Microservice-based system. Unfortunately, Microservices do not address all SOA shortcomings. In addition, Microservices introduce new challenges. This work provides a detailed analysis of the differences between these two architectures and their features. Next, we describe both research and industry perspectives on the strengths and weaknesses of both architectural directions. Finally, we perform a systematic mapping study related to Microservice research, identifying interest and challenges in multiple categories from a range of recent research.

Keywords

SOA; Microservices; Architectures; Self-contained Systems; Systematic mapping study; Survey

CCS Concepts

•Information systems → Web services; •Applied computing → Enterprise architectures; Service-oriented architectures; •Computer systems organization → Distributed architectures;

1. INTRODUCTION

Over the last decades, industry demands have pushed software design and architectures in various directions. The ever-growing complexity of enterprise applications, along with change and evolution management ushered in the rise of architectures such as Common Object Request Broker Architecture (CORBA), Java RMI, and Enterprise Service Bus. Service-Oriented Architecture (SOA), became the answer to multiple industrial demands for large enterprises, replacing its predecessors; however, Microservice Architecture (µService) appears poised to replace SOA as the dominant industry architecture.

Both SOA and µServices suggest decomposition of systems into services available over a network and integratable across heterogeneous platforms. In both approaches, services cooperate to provide functionality for the overall system and thus share the same goal; however, the path to achieving the goal is different. SOA focuses on design of system decomposition into simple services, emphasizing service integration with smart routing mechanisms for the entire company’s IT. The smart routing mechanism provides a global governance or so-called centralized management and is capable of enforcing business processes on top of services, message processing, and service monitoring / control. SOA services are uncoupled, reacting without knowing the event trigger. A new service can be easily integrated by reacting to such event. For instance, one service can write an invoice and another can initiate delivery. A new logging system can just listen to events, without impacting other services.

µServices, on the contrary, suggest decomposition preferring smart services while considering simple routing mechanisms [12], without the global governance notable in SOA. This naturally leads to higher service autonomy and decoupling, since services do not need to agree on contracts on the global level. However, services become responsible for business processes management as well as for interaction with other services.

There are, however, other perspectives to consider. SOA’s difficulty comes with the complex stack of web service protocols necessary for transactions, security, etc., spanning through all the interoperable services. Moreover, since SOA enables building business processes on top of the services on the integration level, it brings flexibility to change the processes, but at the same time binds all services to a single general context. As a consequence, service contracts expressing the service operation expose its data types leading into dependencies regarding deployments [115], in an extreme case leading into one large monolith deploy.

In µServices, it is possible to involve light and heterogeneous protocols for service interaction. Each service only maintains its context and its own perspective over particular data, possibly leading into duplicities across services. If deployment dependencies exist among services, they are on a much lower scale since no general context and no centralized governance exists. The primary goal of µService is to enable independent service deployments and evolution.

The above features lead to multiple consequences. For instance, it is fairly easy to selectively deploy overloaded µSer-
service in order to scale it; however, it is not easy in SOA [116]. The SOA integration mechanism and centralized governance predetermine a bottleneck when the system needs to scale up. When a scaling issue arises in some SOA feature, it is hard to determine where the bottleneck is and whether it is the service itself, the integration, or in a shared database. Self-contained µServices are more efficient when it comes to elasticity, scalability, automated, and continuous deploy with fast demand response. The above characteristics make µServices more cloud-friendly [62].

The price of such flexibility is that µServices must restate and redefine data definitions or even business rules across services, introducing replications in databases, lacking a centralized view on the overall system processing, rules, constraints, etc.

Industry seems to be in the shift towards µServices, leaving SOA behind. However, µServices are not a superset of SOA and many of its challenges do not exist in SOA. Various interpretations of these architectures [37] put part of the community on the side considering µServices to be a subset of SOA, although many others [93] see them as distinct architectures.

In this paper, we aim to describe the differences between SOA and µServices so that reader gets a clear picture what to expect from one or the other. We also point out strengths and weaknesses of each approach. Moreover, we each approach to disambiguate terminology and give the reader a solid understanding of benefits of the particular approach. Since µServices seem to be the future direction for the industry, we emphasize our focus on challenges this architecture has to face. Section 2 provides the background. The next two sections introduce SOA and µServices in details. Section 5 provides an example that enlightens the differences. The architecture comparison is the subject of Section 6. Open research challenges in service integration are discussed in Section 7. Section 8 provides a mapping study on µServices introducing challenges addressed in existing work from over 100 papers. The last section concludes the paper.

2. BACKGROUND

SOA and µServices are two major architectures that are being used for decomposing systems into services. The question is how to coordinate services to achieve particular use cases. In general, there are two well-accepted approaches: centrally orchestrated and independent or distributed. Centrally orchestrated approaches are the common pattern for SOA, and the decentralized is the dominant pattern for µServices. In this section, we provide definitions of terms that help us to compare and contrast these two architectures.

Service is a reusable software functionality usable by various clients for different purposes enforcing control rules. It implements a particular element of the domain, defines its interface, and can be used independently over the network. While involving well-known interfaces and communication protocols, it brings platform-independence. In SOA, services are registered in a directory or registry to easily locate them. In order to reduce coupling, services are composed to produce an outcome.

The interaction patterns for centralization and decentralization are called orchestration and choreography, respectively. These indicate how services collaborate, how the sequence of activities look like, or how the business process is built. Service orchestration expects a centralized business process, coordinating activities over different services and combining the outcomes. Fig. 1 depicts a service orchestration.

Choreography does not have a centralized element for service composition. Service choreography describes message exchange and rules of interactions as well as agreements among interacting services. The control logic is not in a single location. Fig. 2. depicts service choreography.

When involving orchestration through a mediation layer, we often introduce a canonical data model. In such a mode, various system parties agree or standardize their data models on the business objects they exchange. However, often [115] the entire system ends up with having just one kind of business object. For instance, there is a single Person, Order, Entry, Invoice, etc., with matching attributes and associations, since everyone agrees on them. It is easy to introduce such a model with orchestration; however, later changes to the model are very difficult since all parties have to agree on them and the individual system has limits on evolution.

An alternative approach arising from Domain-Driven development [108] is called Bounded context. Here each service aims to operate with particular business objects in a specific context, and thus it may make sense for some service to consider certain attributes, and ignore others. For instance, we may consider a User’s address to process shipments and ignore others; however, for price calculations, it is sufficient to only have User customer-rank. Thus a large model is divided into small contexts, allowing the modeling of business objects differently based on particular needs. Not all services have the same needs and thus should have the independence to design their needs.
3. SERVICE-ORIENTED ARCHITECTURE

The main reason for a software architect to use SOA or µServices is to modularize a system into services. SOA, however, requires significant upfront commitments, since the entire company IT must distribute into separate services. It is fairly easy to introduce a new SOA service; one can take a legacy application and define a new network accessible interface for it. A more advanced design divides an application onto multiple services, opening for broader service reuse and service orchestration. The challenging task when introducing SOA is the setup of centralized governance, the component responsible integration of services and their communication. Usually, the solution for integration is Enterprise Service Bus (ESB) that forms the backbone for SOA. As mentioned earlier, it enables orchestration; moreover, services can interact through messages or events where the trigger is unknown while multiple services react on the particular event. In addition, business processes may be defined at the integration level, allowing flexible reconfiguration. However, this solution inclines toward the introduction of a canonical data model. The key point here is that the integration platform is smart, but also complex. SOA is sometimes referred as “simple services and smart pipes” [62] for this reason. On top of the system is usually a separate component of the user interface, such as a portal or a dedicated web system. Fig. 3a shows a sample SOA deployment with two systems, A and B, exposing services for the integration platform above them. The user interfaces part then communicates with the services through the integration component.

The main advantages of SOA manifest when enough services are available. The business processes implement service orchestration with control over the company processes. It becomes easy to compose services, introduce alternative ways to deal with processes, and build new functionality. The services can be even open to third-parties. However, the layout is usually that various system parts (such as A and B in Fig. 3a) are maintained by different teams. Separate teams usually exist for the integration component or user interface. When changing a particular service that introduces an interface modification, the update most likely promotes to the integration level, as well as to the user interface demanding redeploy of multiple components. In such a manner, SOA deployment happens as a monolith involving multiple services; one defected service may prevent the entire application deployment with a complex rollback.

The situation becomes worse since processes span across services dedicated to different teams, exacerbating communication overhead and requiring coherence in the development and deploy. On top of this, companies tend to have a centralized administration unit managing all changes in the SOA service infrastructure, which leads to conservatism and limitation on individual service evolution.

One of the main issues in SOA is system versioning since we do not know the service users. There are even cases when a company maintains over 20 different versions of the same service with a slightly modified interface to accept different data [23]. This significantly impacts the operations involvement demanding monitoring and maintenance.

According to Red Hat [21], SOA community considers the transition to µServices because the common SOA practices services to complex protocols stacks, such as SOAP, a protocol for web service communication, and WSDL, to describe a service [54]. While this is not an SOA requirement, in practical usage it degrades to solely SOAP and web services.

To summarize, SOA makes it easy to change business processes, although, changing a service requires deploy of the component providing the service. This may cascade to the whole SOA monolith, not to mention the need to reflect the changes in the user interface. The integration platform is usually very complex when it comes to the first deployment, and since it is the centralizing particle, it can become the system bottleneck that has to deal with communication overhead or distributed transactions. The integration unit is usually an ESB, that serves the purpose of integration, orchestration, routing, event processing, correlation, and business activity monitoring. From the communication perspective, SOA is about orchestrating large services.

4. MICROSERVICE ARCHITECTURE

µServices base on three Unix ideas [115]:

- A program should fulfill only one task, and do it well.
- Programs should be able to work together.
- Programs should use a universal interface.

These ideas lead to a reusable component design, supporting modularization. The major point is that services are brought to production independently of each other, which is one of the main differences with most SOA solutions. It does not only impact deployment but also evolution and modification efforts. µService followers often cite Conway’s law [29], stating that “Organizations, which design systems are constrained to produce designs which are copies of the communication structures of these organizations.”

µServices emphasize lightweight virtual machines. They are implemented as containers (e.g., a Docker) or individual processes. This unbinds dependency on a certain technology, enabling usage of a service-specific infrastructure. µServices usually do share the same database schema as it would predict determine a bottleneck as well as coupling. Each µService is in charge of its own data model, which possibly leads to replication. In the Background section, we mentioned Bounded context, which is the direction for µServices. Later in this section, we elaborate more details on Bounded context.

Unlike SOA, µServices do not have integration component responsible for service orchestration and prefer choreography. Business processes are embedded in services, and there is no logic in the integration. Thus µServices themselves are responsible for interaction with others. This gives limited flexibility to design or adjust business processes over the company IT’s. It is a payoff for µService independent service management and deploys. Of course, one can still utilize orchestration; however, this is not a typical approach.

It can be noted that µServices emphasize isolation in a way that a particular process and user interaction operate in the scope of a particular service. A service is usually managed by

1 https://github.com/Netflix/conductor
a single team; however, changes to μServices require user interface propagation. For this reason, both the user interface and μService should be under control of a single team. This gives the team flexibility to manage changes while avoiding bureaucratic negotiation on interface changes. Having μServices independent regarding development and deployment brings individual scalability and continuous delivery. This provides resilience to failure [12] since a request may be balanced among several service instances.

In [11], authors discuss a DevOps practice, which goes hand-in-hand with μServices. DevOps is a set of practices that aim to decrease the time between changing the system and deploying the change to production while maintaining software quality. A technique that enables these goals is a DevOps practice [13]. One prevalent DevOps practice is continuous delivery, enabling on-demand automated deployments supporting system elasticity to request load. Similarly, continuous monitoring provides early feedback to detect operational anomalies.

The difference from SOA can be seen in Fig. 3b that shows System A mentioned in Fig. 3a. Each μService is an individually-deployable unit maintained by a separate (or the same) team. Note in the figure that no complex integration technology exists over the enterprise; the integration part can be the user interface part or services can interact directly.

Compare to SOA, the user interfaces part may be integrated into the μService, which avoids communication overhead. The communication among μServices does not require REST or messaging, and the user interface integration may talk to other services and involve data replication instead; however, both REST and messaging are commonly used.

μService should be comprehensible by individual developers and not developed by multiple teams [62]. At the same time, it cannot become a nanoservice since it would demand high network communication, which is expensive compared to local communication. Transactions spanning multiple μServices become complex. For this reason, the design should target transaction spanning a single μService or involve messaging queue. However, recently a no-ACID transaction type has been proposed for this context, known as compensation transactions

μService should be enough large to ensure data consistency. For μServices, it is a critical decision when it comes to fragmenting the data model; we mentioned this when introducing the Bounded context. A single service cannot capture the whole context, but there must have a certain boundary. There are strategies [62] to determine how two systems interact, determining the boundary.

1. The shared kernel strategy suggests that each domain model shares common elements, but in the specialization areas, they differ.
2. Customer/supplier suggests that the subsystem provides a domain model that is determined by the caller.
3. In conformist, the caller uses the same model as provided by the subsystem and reuses its knowledge.
4. The anti-corruption layer provides a translation mechanism to keep two systems decoupled. This is often used for legacy systems.
5. Separate ways suggests no integration among systems.
6. Open host service expects a system to provide special services for everybody’s use to simplify integration.
7. Published language has unchangeable linguistic elements (contracts, events, etc.) visible to the outside with a meaning in multiple subdomains.

From the data model perspective, the above strategies (4-6) provide a lot of independence, while (1-3,7) tie the domain models together. From the communication efforts perspective among teams, (5) requires least efforts followed by (3), (4), (6), (7), (2) and (1) with most efforts.

4.1 Self-Contained Systems

The last architectural variation we mention is a Self-Contained System (SCS) [2]. In this approach, a particular system breaks into multiple SCS components that consist of two parts, a user interface and separately-deployable μServices sharing the same code-base. Various SCSs communicate asynchronously if necessary. Each functional is ideally under
a particular SCS component. To draw a comparison, an e-commerce shop system would contain 100 µServices or would only consist of 5-25 SCSs. SCS suggests that a µService has around 100 lines of code. For instance, in practice, in Java EE the project setup would have a multi-module maven project sharing code between the UI and µServices where each part is separately deployable WAR file. SCS can be seen as a specialization of µServices. The approach is promoted by various authors [115].

5. ENLIGHTENING EXAMPLE

To clearly understand the differences, we examine an example application designed in SOA, µServices, and SCS. The aim is to emphasize the characteristics of each particular approach. The example system is an event ticketing system assisting users with ticket selection and purchase. For the purpose of demonstration, it consists of various components. We highlight three: Event Management (EMgm), User Mgmt (UMgm) and Billing Mgmt (BMgm).

In SOA design, we consider the canonical data model, which is enforced by contract agreement on the integration level implemented by ESB, which handles business processes and orchestrates management components. The User Interface (UI) part is a portal also handling user authentication across the system. Fig. 4a shows possible design decomposition in SOA. Each system is a separate application, exposing its own web services. While such applications are independent, the contract agreement and centralization pushes towards deployments as a monolith. Service orchestration goes through the ESB, and the portal sits on top of the ESB. Each management component has a separate code space; however, all component agree on data model in order to design processes in the ESB. We highlight user data model with multiple attributes managed by UMgm. A change to particular management component must be well discussed and promoted to the ESB by maintaining teams; the change most likely promotes to the portal as well impacting another teams. However, the dependencies may impact services in different components, e.g. user data model changes may impact other management components and their internals.

The µServices approach no longer considers the centralized integration via ESB; instead, it delegates business processes to services. Services may share the same code-space, avoiding repetition denoted by the dashed box, while still allowing extraction of various deployables as separate µServices. The original UMgm component partially dissolves into the Single Sign-On module (SSO) for authentication, which is implemented by the portal in SOA. Moreover, notice the bounded contexts of user data model in each code space and the SSO. Fig. 4b shows the example system. Note the different teams responsible particular components. Moreover, note the location of business processes definition and shared code space. The UI is maintained by a separate team responsible for the service integration. Moreover, a service from EMgm could call a service from BMgm. Changes impacting a particular µService can be deployed individually, and the data model extension can be service specific. However, these may impact the UI and another team, which is an argument for SCS that we consider next as variant of µServices.

The SCS design is a specialization of µServices, where the same team maintaining a particular service is now responsible for the UI. However, it is not a separate UI application above the services, but a separately deployable application with direct code access. This has multiple advantages. The team is familiar with the knowledge, it fully controls the changes and change-propagation, and there is less overhead since no web services need to be used for given UI scope. However, the team deals with the whole development stack, including UI, middleware, and database development. SCSs should ideally not communicate with each other, while this is fine for µServices. Moreover, SCSs should favor integration at the UI layer. Fig. 5 shows the transition from µServices into SCS. It highlights the integration in the Billing UI involving the SSO and selected EMgm µServices. The distinct UIs for Billing and Events must correlate with the look-and-feel in order to confirm unity of a single system. When a small change is needed in one SCS, the UI and a particular µServices are redeployed, independent of other µServices in or out of the code base or other SCSs.

For a deeper understanding, consider a possible component interaction in a use case when a user purchases selected tick-
ets. We consider that the user finds tickets through the UI, and next aims to process the purchase of his/her selection. First, consider SOA processing in our example in Fig. 4a. It would look similar to this:

1. The user sends a request via the portal to buy tickets
2. It propagates to ESB that triggers the business process orchestrated by the ESB
3. EMgm checks whether the tickets are still available, and it locks the selected tickets for 30 minutes
4. BMgm makes sure that user paid previous orders
5. UMgm loads user address and level for possible discount
6. BMgm resolves the discounted price and processes payment
7. EMgm removes tickets locks and makes them not available
8. BMgm issues an invoice for user address
9. UMgm recalculates user-level based on recent purchase

We can see that ESB has a rather complex responsibility to orchestrate a lot of services. However, it is easy to reroute the process to different order or consider alternatives on top of existing services.

Next, we consider the interaction in µServices to highlight the differences. The aim is to give the responsibility of a particular process handling to a given µService - in the example we call it payService under the BMgm. If possible we try to avoid distributed transactions; however services may act in a choreography. When considering Fig. 4b, the interaction may look like:

1. User selects tickets in EMgm UI and starts the purchase by locking the tickets for 30 minutes through EMgm UI; next, it routes and posts the user selection to the BMgm UI
2. The request triggers the business process in the BMgm UI
3. BMgm UI checks user’s pre-existing due payments
4. To determine possible discount BMgm UI loads user-level from the bounded context
5. Next, BMgm UI processes the purchase payment
6. To remove the lock and set availability on selected tickets in EMgm, the UI BMgm contacts its µService, which emits an event to a message broker, to which an EMgm µService reacts - updating ticket availability and locks
7. BMgm UI issues invoice for an address from bounded ctx.
8. Finally, it updates user-level in bounded context

We can see that the interaction is delegated to a particular SCS, while it emits events to get outside of the SCS. Changes to the process involve SCS modification and new deployment, however, there is higher autonomy in performing such a change, possibly involving a single team.

In a second use case we consider data analytics. It aims to send an advertisement email to past customers offering a discount to events matching user history and details. We start with SOA and next consider µService approach:

1. EMgm is the initiator of the action over ESB
2. It polls an aggregate user list with details from UMgm
3. In batch for each user, it issues a business process via ESB
4. It fetches the last user event from BMgm
5. For no history it skips the user; for existing history, it determines the event category in the EMgm
6. In EMgm it finds a matching event by category, the earliest date nearby user address
7. BMgm determines the price basing on user-level
8. EMgm sends the advertisement content via user email

In µServices, a particular service performs the process:

1. EMgm µService is the initiator
2. It uses BMgm to fetch user filtered list with all details from UMgm
3. It initiates a batch business process for each user
4. It determines the event category, based on users last event
5. Next, it finds the first matching event by category, earliest date, and location near to user address
6. It determines the ticket price using BMgm µService, considering the particular user.
7. Finally, it sends the advertisement through EMgm µService

Similarly, even in this case, we see SOA’s flexibility to reorder or extend the process using existing services on the ESB level. In SCS we would need to involve decoupling; however, it may introduce a redundancy in user history in both BMgm and EMgm to simplify the processing, which is, unfortunately, a common approach for preserving autonomy.

When a third-party service appears providing the distance from user location to the event venue, ESB has a wide pallet of communication adapters to contact it, while most likely introducing a new service as a facade. In µServices with protocol independence, we may contact the service directly, but only if our language provides an implementation of the protocol, which may lead again to the introduction of a new µService. It seems the best solution is to combine µServices with an integration framework providing a collection of adapters.
Table 1: Comparing μServices and SOA

<table>
<thead>
<tr>
<th>Concern</th>
<th>μServices</th>
<th>SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy</td>
<td>Individual service deploy</td>
<td>Monolithic deploy, all at once</td>
</tr>
<tr>
<td>Teams</td>
<td>μServices managed by individual teams</td>
<td>Services, integration and user interface managed by individual teams</td>
</tr>
<tr>
<td>User interface</td>
<td>Part of μService</td>
<td>Portal for all the services</td>
</tr>
<tr>
<td>Architecture scope</td>
<td>One project</td>
<td>The whole company/enterprise</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Fast independent service deploy</td>
<td>Business process adjustments on top of services</td>
</tr>
<tr>
<td>Integration mechanism</td>
<td>Simple and primitive integration</td>
<td>Smart and complex integration mechanism</td>
</tr>
<tr>
<td>Integration technology</td>
<td>Heterogeneous if any</td>
<td>Homogeneous/Single vendor</td>
</tr>
<tr>
<td>Cloud-native</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Management/governance</td>
<td>Distributed</td>
<td>Centralized</td>
</tr>
<tr>
<td>Data storage</td>
<td>Per Unit</td>
<td>Shared</td>
</tr>
<tr>
<td>Scalability</td>
<td>Horizontally better scalable. Elastic</td>
<td>Limited compared to μServices. Bottleneck in the integration unit or a message parsing overhead. Limited elasticity.</td>
</tr>
<tr>
<td>Unit</td>
<td>Autonomous, un-coupled, own container, independently scalable</td>
<td>Shared Database, units linked to serve business processes. Loosely coupled.</td>
</tr>
<tr>
<td>Mainstream communication</td>
<td>Choreography^1</td>
<td>Orchestration</td>
</tr>
<tr>
<td>Fit</td>
<td>Medium-sized infrastructure</td>
<td>Large infrastructure</td>
</tr>
<tr>
<td>Service size</td>
<td>Fine-grained, small</td>
<td>Fine or coarse-grained</td>
</tr>
<tr>
<td>Versioning</td>
<td>Should be part of architecture, more open to changes</td>
<td>Maintaining multiple same services of different version</td>
</tr>
<tr>
<td>Administration level</td>
<td>Anarchy</td>
<td>Centralized</td>
</tr>
<tr>
<td>Business rules location</td>
<td>Particular service</td>
<td>Integration component</td>
</tr>
</tbody>
</table>

6. COMPARISON

Both μServices and SOA divide systems into services, but in different ways. SOA can still be seen as a monolith from the deployment perspective [115], while μServices lead towards independent deployments. Industry relates μServices to container technologies simplifying automated deployment [62]. Containers can be given the credit for building such self-contained μService deployment units. Moreover, μService go towards design autonomy with plenty of teams resulting in heterogeneity of components, which some may criticize.

SOA has a wide enterprise scope, while the intention of μServices is to do “one thing well” [79]. SOA gains it flexibility from centralized management while μServices inclines for distribution. μServices fit well to the context of cloud computing. Researchers refer to μServices as cloud-native, while SOA is rarely referenced that way in the literature [62]. The key features here are the individual and automated service deployment. In general, service-based approaches are vital for cloud-native approaches [37].

Considering industry demands and recent research directions [62], μServices seems to be the future direction. However, there are counterexamples. In [76] author argues that SOA and μServices are not the same as nothing in μServices build on SOA and multiple pieces are missing for μServices. [93] points out the fundamental concepts:

μServices architecture is a share-as-little-as-possible architecture pattern that places a heavy emphasis on the concept of a bounded context, whereas SOA is a share-as-much-as-possible architecture pattern that places heavy emphasis on abstraction and business functionality reuse.

SOA is a better fit to a large, complex, enterprise-wide infrastructure than μServices [93]. SOA suits well to the situation with many shared components across the enterprise. μServices do not usually have messaging middleware and fit better to smaller, well-partitioned web based applications. Moreover, SOA better enables services and service consumers to evolve separately, while still maintaining a contract. μServices fail to support contract decoupling, which is a primary capability of SOA. Finally, SOA is still better when it comes to integrating heterogeneous systems and services. μServices reduce the choices for service integration.

Table 1 provides a summary comparison between SOA and μServices, which we described in this paper.

7. RESEARCH CHALLENGES

IN SERVICE INTEGRATION

Both architectures come with drawbacks and features that are complex or cause difficulties. These are challenges to address in research. Clearly, SOA has the issue with monolithic deploy, centralization, and bound data model leading into canonical data model or complex protocol stack. On the other hand, it is flexible with business process changes, giving a centralized view on system processes. μServices bring higher autonomy to services reducing data structure dependencies, relocating business processes to particular services. This together with the connection of virtual boxes brings the benefits of individual service deploy enabling elastic service scalability.

Multiple issues can be found in service composition involving both SOA and μServices. In [67], authors consider such issues. They note cross-cutting concerns that repeat across services, such as exceptions, transactions, security, and ser-
service-level agreements. One of the main issues they point out is knowledge reuse. We may want to reuse a component, a data transformation rule, a process fragment, or template. To some extent, SCS accomplishes this in a limited scope. One possible reuse approach is to describe the certain rules in machine-readable or queryable format, e.g., it is easy to make a query to the database to find expected data constraints. The most common approach is to manually evaluate the knowledge and copy/paste it to a secondary location. However, this only exacerbates the complexity of evolution management, since once the knowledge changes, it has multiple locations to maintain. Next, there exist recommender systems [67] to facilitate the composition process. For instance, they perform on-the-fly similarity search over a knowledge base of reusable patterns.

Paths addressing the above issues in service integration use automation. A synthesis [67] reuses knowledge of integrated services. For instance, symbolic model checking may generate an executable business process for the integrating component [88]. Artificial intelligence can be applied involving semantic service with machine-readable descriptions of service properties and capabilities with reasoning mechanisms to select and aggregate services [92]. Naturally, the problem with this approach is the extensive development effort to provide and maintain the semantic information in correlation to the system internal knowledge. Finally, involving Model-Driven Development approach on service design allows transformation of knowledge across various services. However, because of the high-level of abstraction used in the approach demanding complex generalization and design of transformation rules, the approach is rarely used. Developers prefer to focus on specific problem description rather than its abstraction.

Specifically for µServices, the issue can raise from the service-specific data model or business processes hidden from others, which facilitates the service autonomy. On the other hand, this leads to replication of knowledge among services with service integration. Moreover, µServices sacrifice the centralized view on business processes, or generally the knowledge, that is now distributed and hidden across services. For example, consider two communicating Services A and B. These services exchange information while both maintain information in distinct data formats, conforming business rules or processes, or even security restrictions. Once the Service A changes its internal knowledge, it must still correlate to Service B. While the correlation applies only to some extent, it could cause system failure if not properly maintained. Let's consider that Service B could base its reasoning on a Service A internal knowledge in computation. This could leverage the maintenance efforts in service composition. A great example where this happens often is the user interface.

When low-level data format changes, the user interface forms, tables, and reports must reflect such change. However, this is very difficult to preserve [22] since components are maintained by distinct teams and limited type safety exist in user interfaces. In [22] the author suggests utilizing metaprogramming and aspect-oriented programming to let the source service stream meta-information to the integration component. For example, a web browser JavaScript library weaves the together provided information and templates at runtime to assemble the user interface fragments reflecting the actual information from the service. Such approach minimizes maintenance efforts on the user interface part since any change in the service is immediately reflected in the user interface fragment. The approach complies well with [24] contemporary client-based frameworks such as Angular2 or React. Moreover, having the meta-information exposed allows rendering the user interface fragments on various platforms using native components, while all adapting to the changes in services or even context.

While [22] provides an approach for minimizing change impacts of structural data manipulation across integrating services, there is still a dependency on business rules and security. In [20], the authors suggest that services capture their constraints, business rules, and security untangled from the service source code, e.g., in domain specific language description or involving annotation descriptors. This not only improves readability of the above but also allows their automated inspection and transformation into a machine-readable format that can be shared across services. [20] shows a system with its internal constraints and business rules captured in Java EE and Drools being automatically inspected and transformed into a JavaScript description applied across user interface enforcing the constraints and rules already at the client-side.

There are multiple open challenges left to address, beyond the scope of this paper. We highlight the main areas:

1. How should be changes and evolution communicated across different teams working in distinct services?
2. How to detect/test incompatibilities in service integration?
3. How to determine the scope/boundary of a particular µService?
4. How to effectively mitigate failures in service integration?
5. Distributed transactions across services, e.g. compensating transactions, and no-ACID transactions.
6. Cross-cutting issues with code replication across services.
7. Security across services must correlate when one service allows half of the process the second can deny it.
8. How to migrate existing systems onto µServices?
9. How to deal with replicated info. in service specific databases?

8. MICROSERVICE MAPPING STUDY

This chapter discusses research trends and analyzes research challenges addressed by existing related work. The analysis is broad and considers one hundred papers that deal with µServices. Our strategy to harvest evidence considering the approach of a mapping study [1]. Through multiple research indexing sites and portals, we download an evidence to analyze and classify.

8.1 Harvesting the Evidence

To receive systematic evidence in the area of µServices, we involve the main indexing sites and portals (indexers) including IEEE Xplore, ACM Digital Library (DL), SpringerLink and ScienceDirect. These indexers provide a mechanism to search on existing conference and journal papers published in the past years using full text terms, utilizing logical com-
There are similar works addressing mapping study of SOA

8.2 Similar Studies

Table 2 shows in the first column the total number of pa-

8.3 Research Challenge Categorization

We identify the following challenges and keywords from the

combination of these terms. While the syntax and capabilities
differ, all provide the ability to search within the title, ab-

permanent indexer.

Table 2: Resources and filtering using indexers

Indexer & Downloaded Abstract filter First mention Full text filter
IEEE Xplore 55 2014 53
ACM DL 29 2015 22
SpringerLink 24 2015 13
ScienceDirect 11 2015 9
Other sources 4 - 3
Sum 124 - 100

Table 2: Resources and filtering using indexers

index. For this reason, we filter papers outside of the
target scope, papers with no specific output or presenting
an opinion.

Table 2 shows in the first column the total number of pa-
ners we downloaded from a specific indexer. However, the
total number of examined papers was even higher. For in-
stance, a search with IEEE Xplore resulted in 218 papers
to consider, which we further filtered to 55 papers. In the
next stage, we filtered papers based on the full-text ana-
lysis. In the next stage, we eliminated papers focused on
case studies, experience, and opinion papers as they do not
fit to the scope of research challenges and directions. From
the reverse perspective, we consider related work and re-
ferences from selected papers to extend the considered pool
of papers. Besides the four, above-mentioned indexers, we
obtained work from other sources. The final numbers of
papers we work with are represented by the last column of
Table 2. The table also shows the year of first mention in
the particular index.

8.2 Similar Studies

There are similar works addressing mapping study of SOA
[62] and µServices [6] [82], providing a roadmap. [6] gives
a reasonable format of categorization of research challenges,
while considering credible sources. In this study, we consider
a similar categorization structure and thus build on [6]. We
include the same 33 sources referenced from their work; how-
ever, our approach is different. First we consider 107 papers.
Next, we extend the categorization set. Finally, in their
work, they employ a process where they define keywords for
which they search the full texts papers to draw categoriza-
tion. In our case, we utilize keyword extraction since a
search for a text occurrence does mean it fits to a keyword.
To accomplish this, we utilize RAKE algorithm [95] that ex-
tacts reasonable keywords that we consequently categorize
similarly to [6]. However, we manually verify and correct
the categorization.

brella of these terms. While the syntax and capabilities
differ, all provide the ability to search within the title, ab-

abstract, keywords and some even through full text.

Our search query is rather primitive and we search for oc-

our case, we utilize keyword extraction since a
text occurrence does mean it fits to a keyword.
To accomplish this, we utilize RAKE algorithm [95] that ex-
tacts reasonable keywords that we consequently categorize
similarly to [6]. However, we manually verify and correct
the categorization.
bottleneck and since μServices are distributed this is more challenging than in monoliths. The identifying keywords are: runtime metrics, call graph, tracing, logging, debugging, profiling, monitoring, application monitoring, health monitoring

**Modeling** - provides analytic methods, related to models, visualization and formal specification to build a better high-level understanding across the services. The identifying keywords are: modeling, UML, chart, visualization, formal specification, schema

**Security** - is a general topic and μServices have their specifics, various work consider usage of security protocols with respect to service interactions and trust, authentication or global identity management. The identifying keywords are: security, secure, authentication, authorization, OAuth, OAuth2, encryption, vulnerability, attack, access control, OpenID, privacy

**Mapping study/Survey** - is related to literature review. The identifying keywords are: survey, mapping study, literature, literature review, systematic mapping

**Context-awareness** - is the context-awareness provided by service. These works include adaptability or personalization. Some existing work also considers the usage of neural networks, which we include in this category. The identifying keywords are: personalized, personalization, recognition, contextual, context aware, context-aware, neural network

### 8.4 Mapping to Categorization

Next, we apply the categorization to the aggregated evidence. Since the mapping is better to read in a structured format, we show the mapping in Table 3. The table shows the particular category, the volume of work and particular papers we considered. Multiple papers, however, fit into multiple categories, and we do not enforce exclusive designation. Fig. 6 then shows the relative volume of work per category. The top keywords when compared to [6] are much lower. The top keyword is API gateway with 9 occurrences, testing with 7, security with 5, and service discovery with 5. Other keyword occurrences showed 4 or fewer hits using the RAKE algorithm.
This mapping shows an up-to-date research road map for µServices. Next, it depicts existing interest in particular categories. When considering the major benefits of µServices, such as alternative to SOA, independent development and deploy, automation for deployment, good scalability, we may notice that these reflect and correlate with the main research interest. However, important topics including monitoring, tracing, security, or service discovery do require research interests. Modeling and context-awareness usually apply to mature approaches. Significant attention is payed to case studies, but their actual merit on production-level approach is questionable.

8.5 Threats to Validity
A primary threat to the validity of survey studies is inadequate coverage. This is addressed by the breadth of our coverage and the design of our research parameters. Similar to the guidelines for conducting systematic mapping studies [1], we performed the evidence elimination. While, 100 percent coverage of related papers cannot be guaranteed, we believe we have selected all relevant papers, within the scope of this study. We addressed this threat by selecting and examining several search strings that fit our paper control set. Next potential threat is data extraction bias based on the human factor. We addressed this threat by using a RAKE keyword extraction algorithm [95] that we matched with manually-extracted ones.

9. CONCLUSION
This paper attempts to demystify ambiguous usage of term and definitions of SOA and µServices. It specifies characteristics and differences of both architectures, pointing out their strengths, weaknesses, and differences. While both architectures address system integration, the industry seems to move toward µServices, leaving SOA as legacy. The main credit for this tendency can be given to the ability of independent service deploy and elastic scalability.

For instance, in the market of integrated systems, Gartner\textsuperscript{4} predicts a boom where hyperconverged systems reach 24% of the overall market by 2019. Moreover, Gartner states that in this segment recently started new era bringing the continuous application and µServices delivery.

While µServices seems to be the winner of the two, there are still multiple challenges that come with the architecture that leaves developers with restatements and complex processing. These are addressed in future research; we show multiple of these issues are actively addressed by researchers. In this paper, we introduced a mapping study from around 100 papers related to µServices.

Readers may also wonder about the symbiosis with Internet of Things, Cloud computing, Platform as a Service (PaaS), and systems for handling Big Data. All these can be addressed using µService architecture; however, these are beyond the scope of this paper and left for future work.

In the end, we raise a philosophical question. Since the evolution path went from heterogeneous system integration, through SOA into µServices, can we expect in near future another step back towards SOA?

Acknowledgments
This research was supported by the ACM-ICPC/Cisco Grant 0374340.

10. REFERENCES

\textsuperscript{4}http://www.gartner.com/newsroom/id/3308017


D. Richter, M. Konrad, K. Utecht, and A. Polze.


ABOUT THE AUTHORS:

Tomas Cerny received his Bachelor's, Master's, and Ph.D. degrees from the Faculty of Electrical Engineering at the Czech Technical University in Prague, and M.S. degree from Baylor University. He is a Professor of Computer Science at Baylor University. His area of research is software engineering, security, aspect-oriented programming and design, user interface engineering, enterprise application design and networking.

Michael “Jeff” Donahoo received his B.S. and M.S. degrees from Baylor University, and Ph.D. in the Computer Science from Georgia Institute of Technology. Jeff is currently a Professor of Computer Science at Baylor University where he conducts research on networking, security, and enterprise application development.

Michal Trnka obtained master degree in computer science from Faculty of Electrical Engineering of Czech Technical University in Prague and he is currently Ph.D. student in Czech Technical University in Prague. He received Fulbright scholarship for academic year 2017/2018. His area of interests is software engineering and especially in context aware applications security. Lately he focused on using context information to enhance security rules as well as methods to obtain context from trusted sources, like Internet of Things devices.
Ontology-based Framework for Internal-External Quality Trade-offs and Tenant Management in Multi-tenant Applications

Sumit Kalra  
Department of Computer Science and Engineering  
IIT Kanpur  
Kanpur, India  
sumitk@cse.iitk.ac.in

Prabhakar T.V.  
Department of Computer Science and Engineering  
IIT Kanpur  
Kanpur, India  
tvp@iitk.ac.in

ABSTRACT
Software Quality Attributes (QAs) can be categorized as either internal to the system as experienced by the developers or external to the system perceived by the end users. These QA categories have trade-off among them - an emphasis on internal QA may result in a compromise of an external QA. For example, there is a trade-off between maintainability and performance. Model-driven development approaches manage this trade-off and increase the degree of internal QA maintainability. In this work, we propose an ontology-based communication mechanism among software components to handle the trade-off. The approach increases the degree of internal QAs such as modifiability, maintainability, testability during the design and development phases without compromising the external QAs for the end users during the operation phase. We also evaluate a prototype system to validate the proposed approach using Software Architecture Analysis Method (SAAM). It is also easier to integrate into the software development lifecycle as compared to existing model-driven approaches. The internal quality attributes become more significant in a multi-tenant scenario than conventional software. It requires managing dynamic requirements of tenants continuously. The proposed approach also useful in such scenario to reduce the maintenance overhead without compromising the degree of multi-tenancy.

CCS Concepts
● Software and its engineering → Software design engineering; Software design tradeoffs; Integrated and visual development environments; Rapid application development; Software design engineering;

Keywords
Software Product Quality Attributes; Internal quality attributes; External quality attributes; Multi-tenant; Quality attributes trade-off;

1. INTRODUCTION
A software system goes through various phases during its lifetimes such as design, implementation, testing, operation, and maintenance. These phases can be followed in a sequential order such as in waterfall model or iterative order such as in prototyping model and agile methodology. The end users interact with the system only during the operation phase and sometimes during the beta testing phase. Rest of the time developers deal with the system. The development team includes designers, architects, programmers, testers, and maintainers. Each stakeholder desires different degree of Quality Attributes (QAs) during the various phases of Software Development Life Cycle (SDLC). These software product quality attributes are defined and categorized in ISO/IEC 25010 standard [19]. The model classifies all these QAs into two broader classes - internal QAs and external QAs.

Internal QAs are based on static attributes of a software product which require design documents, code, and test procedures. For example, modifiability is an internal quality attribute because it deals with the source code of the system. A software product has a high degree of modifiability if it is effectively modifiable without introducing defects or degrading existing product quality. Other than modifiability, flexibility, portability, reusability, readability, testability, and understandability are also internal quality attributes.

External QAs are the measurement of dynamic attributes which require executing the software product in intended system environment to measure system-dependent properties. For example, time efficiency is an external quality attribute. It is measured using response-time. External software quality is the measurement of system functionality and its ability to provide the expected business values. The set of external quality attributes also includes correctness, usability, performance efficiency, reliability, integrity, adaptability, accuracy, and robustness.

We follow this quality model as it categorizes quality attributes as internal and external from the perspective of developers and end-users respectively. The relationships between the internal and external quality attributes are not well studied. There has been little work to identify the impact of these two categories on each other [38]. We mention some of the conflicting pairs of internal and external quality attributes in the Table 1. Definitions of all the qual-
ity attributes discussed in this paper are adapted from the ISO/IEC 25010 standard.

At development time, developers modify the system frequently like changing the workflow in the system, adding or removing components, trying different components with various interfaces from third-party vendors, etc. [7]. In a typical software development process, it requires a lot of time and effort to try different off-the-shelf (OTS) components with different interfaces and quality-of-service (QoS). Every such change requires modifications in the code, and a compile step [37]. According to [25], the cost of locating and debugging appropriate OTS components is very high. It requires developers to integrate multiple components to find out the most appropriate component corresponding to the required quality and functionality. A system with high degree of modifiability, testability, and maintainability reduces the developers' efforts significantly to extend and maintain the system. On the other side, the end-user expects better performance, accuracy and, usability from the system during the operation phase [8].

The trade-off between external and internal QAs suggests that increase in the degree of one type of QA may reduce the degree of another type of QA(s). For example, Model Driven Architecture (MDA) based design approaches produce code from the abstract and human understandable modelling diagrams [35] [36]. It focuses on developing source code from the modeling diagrams with the help of Architectural Description Languages (ADL). Once the developers modify the source code, the model is no longer referred during rest of the SDLC process [16]. It results in the higher degree of internal QAs and lower degree of external QAs. Similarly, Aspect-oriented programming (AOP) increases the modularity by separating the concerns of business specific logic and non-functional logic. It mainly focuses on cross-cutting concerns of the system [23]. Although MDA and AOP are popular, the lack of seamless, continuous integration with SDLC phases hinders the widespread adoption in the industry [11]. This leads us to identify an approach, which gives the data and control among them. Software connectors bind the functional components of a software system and transfer the data and control among them. These connectors are

<table>
<thead>
<tr>
<th>Internal Attribute</th>
<th>External Quality Attribute</th>
<th>Conflict Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testability</td>
<td>Adaptability</td>
<td>Adaptability requires more generic interfaces and makes it less friendly to write test suits for complete coverage [31].</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Performance Efficiency</td>
<td>Loosely coupled system is more maintainable and modifiable [30] whereas tightly coupled system is more performance efficient [1].</td>
</tr>
<tr>
<td>Portability</td>
<td>Correctness</td>
<td>Portable systems requires more effort, time to resolve the platform specific issues and maintain functional correctness [18].</td>
</tr>
</tbody>
</table>

### Table 1: Conflicts among Internal and External Quality attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Quality</th>
<th>Conflict Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testability</td>
<td>Adaptability</td>
<td>Adaptability requires more generic interfaces and makes it less friendly to write test suits for complete coverage [31].</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Performance Efficiency</td>
<td>Loosely coupled system is more maintainable and modifiable [30] whereas tightly coupled system is more performance efficient [1].</td>
</tr>
<tr>
<td>Portability</td>
<td>Correctness</td>
<td>Portable systems requires more effort, time to resolve the platform specific issues and maintain functional correctness [18].</td>
</tr>
</tbody>
</table>

At development time, developers modify the system frequently like changing the workflow in the system, adding or removing components, trying different components with various interfaces from third-party vendors, etc. [7]. In a typical software development process, it requires a lot of time and effort to try different off-the-shelf (OTS) components with different interfaces and quality-of-service (QoS). Every such change requires modifications in the code, and a compile step [37]. According to [25], the cost of locating and debugging appropriate OTS components is very high. It requires developers to integrate multiple components to find out the most appropriate component corresponding to the required quality and functionality. A system with high degree of modifiability, testability, and maintainability reduces the developers' efforts significantly to extend and maintain the system. On the other side, the end-user expects better performance, accuracy and, usability from the system during the operation phase [8].

The trade-off between external and internal QAs suggests that increase in the degree of one type of QA may reduce the degree of another type of QA(s). For example, Model Driven Architecture (MDA) based design approaches produce code from the abstract and human understandable modelling diagrams [35] [36]. It focuses on developing source code from the modeling diagrams with the help of Architectural Description Languages (ADL). Once the developers modify the source code, the model is no longer referred during rest of the SDLC process [16]. It results in the higher degree of internal QAs and lower degree of external QAs. Similarly, Aspect-oriented programming (AOP) increases the modularity by separating the concerns of business specific logic and non-functional logic. It mainly focuses on cross-cutting concerns of the system [23]. Although MDA and AOP are popular, the lack of seamless, continuous integration with SDLC phases hinders the widespread adoption in the industry [11]. This leads us to identify an approach, which gives the data and control among them. Software connectors bind the functional components of a software system and transfer the data and control among them. These connectors are

### Problem Statement:
The focus of this work is to address the issues of accommodating conflicting internal and external software product quality attributes during various phases of software development lifecycle. How to design and build a software system such that it can emphasize different conflicting sets of quality attributes at development and operation time.

### Contribution:
The key contributions of this work is: An ontology-based software development approach which facilitates the automatic transformation from a high degree of internal and low degree of external quality attributes at development time to a low degree of internal and a high degree of external quality attributes at production time without significant overhead.

Rest of the paper is organized as follows. In the section 2, we discuss the proposed approach in detail. The section 3 and 4 present the real world use case scenarios. The section 5 includes the application of the proposed framework to manage tenants in a multi-tenant scenario. Then we compare our work with the existing techniques in the section 6. In the end, we discuss and conclude the paper in the section 7 with future research directions.

### 2. PROPOSED APPROACH

In this section, we discuss the software development approach to achieve a higher degree of internal quality attributes during various phases of software development without affecting the degree of external quality during the operation phase. A software product consists of multiple modules, and these modules are part of a single or multiple components. A module is a design-time entity whereas a component is runtime entity. These components interact with each other via connectors. Software connectors bind the functional components of a software system and transfer the data and control among them. These connectors are
Figure 1: A software development approach - using an ontology to implement communication logic during the development phase and convert it into the source code for production phase.

responsible for communication and coordination among the functional components. They are first class entities in software architecture and well studied in the literature. There are various types of such connectors such as Procedure call, Event, Data Access, Linkage, Stream, etc. as described in [40].

The model-driven approach allows defining the connectors and communication-related logic such as interfaces, protocols, etc. It makes the system highly modifiable and flexible. We extend the idea of separately defining the connectors and communication logic to achieve a higher degree of internal quality attributes during the development phases. Communication logic of a software system can be seen as a semantic call graph of connectors among different components. Nodes of this call graph are components, and each edge between nodes represent connector between the two components. The edges can be uni-directional or bi-directional. Apart from all the interacting pairs of components, the semantic call graph also contains following information about each edge:

- Protocol: It identifies the type of protocol such as synchronous, asynchronous, https, soap, etc.
- Parameter information: It defines the number and type of parameters.
- Accessibility: Defines which component or event can use the connector.
- Ordering: In case of multiple connector calls, it decides the ordering and preference of all the connectors.
- Cardinality: It defines the number of instances of the connector can exist simultaneously.

We use ontology [42] to represent the semantic call graph. Some of the reasons for using ontology over existing representation formats adapted in MDA and AOP approaches are:

- It can be modified at runtime without recompiling since it is a non-compile executable entity.
- It is a specification of a conceptualization. In this approach, it uses connector specific concepts which make it application independent and reusable across multiple applications.
- Semantic inference capability of the ontology facilitates to encode complex communication logic containing the sequence, priorities, protocol conversion, etc. into the communication ontology.
- With the help of a Reasoner[4], one can quickly find out conflicting logic encoded in the ontology.
- One does not require core programming skills to deal with ontology. A designer, architect, project manager and other stakeholders with not-so-good programming skills can also easily use it.
Ontology has support from existing tools [24] to generate corresponding source code. It allows us to replace ontology with efficient implementation in the form of native programming language for the operation phase and maintain a higher degree of external QAs.

All these features make it easy to integrate with all the development phases - design, implementation, testing, and maintenance. In the design phase, ontology helps to create visual models to satisfy functional requirements and verification of the models through ontology reasoners [42]. During the implementation and maintenance phases, ontology-based implementation of communication-related logic increases the degree of modifiability. It is easier to try out different OTS components using ontology. Once the connectors specifications for a component is edited into the communication ontology, it can be reused across all the software products which use the component. Using different versions of ontology also allows to build and manage multiple software products using a set of components. It also takes care of workflow management. During the testing phase, it is relatively easier to find a bug and repair since the functional logic and communication logic are implemented separately. Ontology Reasoners also makes it possible to perform testing on the communication logic. We validate every claim here with our case study. The proposed approach uses the semantic inference power of ontology to empower the developers. Ontology fits in our solution to represent complex communication logic in a simple and elegant way.

2.1 Structure

Now we elaborate structural component of our approach. It has following key constituents.

- Functional logic components: These contain the functionality of the application.
- Communication Agents (CA): Each functional logic component has one associated communication agent. It is an ontology parser [14] and retrieves the specifications from the communication ontology and establishes the communication.
- Communication Ontology (CO): It contains the complete semantic call graph of the application.
- Ontology compiler: It compiles and generates the source code from the communication ontology [39].

One of the contributions of this work is to build a reusable communication ontology component. We describe this component in detail here.

Communication ontology: It creates dynamic connectors at runtime to establish communication among the components. The key idea of the proposed framework is to use ontological implementation of connectors to achieve modifiability and flexibility during development phases. Mostly the efficient implementation of these connectors is left for the underlying networks[2]. Ontological implementation of the connectors makes these connectors dynamic and configurable at runtime and increases the degree of flexibility and maintainability.

We design communication ontology for various types of connectors. As a proof of concept, we present Procedure call and Event connectors ontology in the figure 2. We briefly describe these two connectors here for quick reference.

- Procedure Call Connector: This type of connector is one of the most widely used connectors. Procedure call connectors provide communication and coordination services to the functional components. They can transfer data using parameters and return values. They can also transfer control by invoking procedures. There are different dimensions, in which procedure call connectors can differ from each other such as parameters, the number of entry points, invocation type, synchronicity, cardinality (fan-in, fan-out) and accessibility [40].
- Event Connector: These connectors are similar to procedure call connectors. It also provides communication and coordination services to the functional components. As defined in [32], events occur instantaneously when an invocation of an operation on the object terminates. As soon as an event connector learns about the occurrence of an event, it notifies to all the subscribed components and transfers the program control. Event connectors differ in delivery mechanism, synchronicity, etc. These event connectors can be implemented in hardware or software layers[40].

We call the ontological implementation of these connectors as dynamic connectors since these connectors can be modified at runtime. Ontology does not need any compilation step and allows to incorporate modifications at runtime. With the help of dynamic connectors, developers can modify the application workflow, and replace, add or remove the components without recompiling the application code again and again. It allows developers to try out different third-party components without much effort. It results in savings of time and cost. Multiple versions of dynamic connector ontology can also empower developers to manage multiple versions and variations of the software product efficiently.

2.2 Dynamics

In this section, we explain the inner dynamics of our approach. The figure 1 shows the constituents and interaction among them during the different phases of SDLC. In this sub-section, we discuss how it achieves the goal of maintaining different sets of quality attributes during the development and operation phases. The proposed approach operates in two phases as described below.

- Development Phases: All the components interact via communication ontology with the help of communication agents. Communication ontology is a runtime non-compile entity. It implements dynamic connectors. Dynamic connectors are editable at runtime and allow to modify the behavior of the application. Other application components are compiled entities which require compilation step before execution. Figure 3 depicts the graphical view of component interaction via CAs and CO. In this phase, a developer can also leverage ontology-based reasoners to debug the communication logic [42].
Figure 2: Communication ontology to implement the Procedure call (dark gray) and Event (light gray) connectors dynamically. Some of the concepts (sky blue) are common between these two connectors.

Figure 3: Communication Agents (CA) are associated with Functional Components. CA interacts with the Communication Ontology (CO) to retrieve the specifications for establishing communication among components.

- **Operation Phase**: Communication ontology component is converted into source code and compiled along with the rest of the application components. No communication agent is required in this phase for establishing communication among application components. Now, all the dynamic connectors are static in nature and can not be modified without a compile step. It is embedded into the application along with functional logic.

During the development phase, **CL** and **CA** are responsible for executing the component interaction. Ontological implementation of communication logic gives the developers flexibility of adding or removing components, changing the workflow or connector specifications in a declarative manner using ontology editors such as protege[17]. Developers can modify the **CL** while running the system for the purpose of debugging and testing. On the other hand, the overhead of fetching specifications and then generating the connectors on-the-fly results in lower runtime performance. To enhance the speed of interaction, it allows generating native source code from the communication ontology [43]. This code is compiled with **FL** to reduce the runtime overhead of generating connectors and make the execution faster. In summary, the approach uses an ontology to encode communication logic at development time to let developers modify the call graph quickly and convert the ontology-based communication into compiled version to speed up the performance in operation phase. The transformation from the development version to operation version is a uni-directional task and requires generating source code from the ontology and compiling it with the functional logic.

3. **CASE STUDY**

In this section, we use the Software Architecture Analysis Method (SAAM) [22] to evaluate our approach. SAAM approach requires to identify different scenarios and evaluate each scenario for the candidate architectures of a particular system. We use a typical e-commerce system with basic functionality to validate our approach. The prototype system **Arch1** along with semantic call graph of its component
Figure 4: A typical architecture of an e-commerce system with semantic call graph information as edge labels.

Figure 5: The e-commerce system architecture having communication ontology for semantic call graph is depicted in the Figure 4. It supports the following workflow:

- Customer adds product(s) to the cart
- At checkout from cart server, customer logs in if not already done
- Order server handles different payment gateways
- Delivery order process performs order placement and tracking

Another architecture Arch2 of the same system as shown in the Figure 5 using the proposed approach. We describe few scenarios in the Table 2 to evaluate these two architectures using the SAAM method. These scenarios are carefully chosen to represent the most of the generic requirements during the development phases. We evaluate each scenario against two parameters - 1) the number of components need to be modified, and 2) effort in person-month to do the required modifications. We show the findings of the evaluation process in the result Table 3. The values in the result table are estimated after brainstorming with five software engineers having industry experience ranging from 6 to 20 years. The analysis shows that using the communication ontology makes it easy to apply those modifications which are related to the communication logic such as changing the workflow, replacing the component and modifying the communication protocol specifications.

4. USE CASE SCENARIOS

In this section, we present some use case scenarios in which the framework benefits the developers as well as the end-users. We divide these use cases into six categories; each one covers the different aspect of the framework.
Table 2: Scenarios

<table>
<thead>
<tr>
<th>QA Scenario</th>
<th>Source</th>
<th>Stimulus</th>
<th>Artifacts</th>
<th>Environment</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifiability</td>
<td>Developer</td>
<td>Instead of only &quot;adding item into cart&quot;, now inventory status of the</td>
<td>Code, components and interfaces</td>
<td>Design and build</td>
<td>Make and test modifications</td>
</tr>
<tr>
<td>Scenario (S1)</td>
<td></td>
<td>product is updated if the &quot;adding item into cart&quot; is followed by</td>
<td></td>
<td>time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;successful checkout&quot; action</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modifiability</td>
<td>Developer</td>
<td>A requirement to handle more load during sale period by making the</td>
<td>Code, components and interfaces</td>
<td>Design time</td>
<td>Make and test modification</td>
</tr>
<tr>
<td>Scenario (S2)</td>
<td></td>
<td>communication asynchronous with 3rd party payment gateways.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testability</td>
<td>Integration Testers</td>
<td>Replacement of an older version of component with newer which supports</td>
<td>The portion of the system being tested</td>
<td>Integration time</td>
<td>Execute test suite and capture the results</td>
</tr>
<tr>
<td>Scenario (S3)</td>
<td></td>
<td>secure socket layer over http protocol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portability</td>
<td>Developer</td>
<td>Running the core components of the system on other platform with different</td>
<td>Targeted platform and libraries</td>
<td>Build time</td>
<td>Build and execute the system on new platform</td>
</tr>
<tr>
<td>Scenario (S4)</td>
<td></td>
<td>set of dynamic link libraries</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reusability</td>
<td>Designer</td>
<td>New business requirement is to build an integrated social networking</td>
<td>components, interfaces, code</td>
<td>Design time</td>
<td>Check if some off-the-shelf component is available</td>
</tr>
<tr>
<td>Scenario (S5)</td>
<td></td>
<td>module for the e-commerce platform</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: SAAM Analysis result table (lower values are better)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Number of components</th>
<th>Efforts (person month)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arch1</td>
<td>Arch2</td>
</tr>
<tr>
<td>S1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

4.1 Multiple workflows for product line management

To achieve cross-platform compatibility, often a development team writes a generic code and compile it with platform-specific libraries. While writing a generic code, to handle different name aliases of methods across different header files and libraries, the compiler uses mapping from generic function call to platform-specific function call. For example, a generic code uses `Read` to receive the input from the user. While compiling this code using C libraries, it is converted to `scanf` statement. Similarly, it is converted to `cout` and `readLine` statement for C++ and Java respectively. Conversion of `Read` into `scan` and `cout` are comparatively straightforward than `readLine` due to instantiating `InputStreamReader` and `BufferedReader` classes before invoking the `readLine` method. For Java, it requires different workflow to receive user input. The communication ontology can encode this complex logic of mapping in a declarative manner easily. The task of adding support for the new platform and creating a new product from the existing product line is reduced to editing ontology to add new connector specifications.

Another scenario of modifying workflow is adding LDAP authentication module in the login and authentication work-
flow. Initially, Login component was interacting with User Database component to verify user details as shown in the Figure 6. Now Login component works with LDAP service, which internally uses User Database component as presented in the Figure 7. Modifying connector specification changes the application logic. Simple editing in the communication ontology can incorporate these kinds of changes for including off-the-shelf (OTS) components in the application on-the-fly at runtime. With these two versions of ontology connectors, we have two products from the product line.

4.2 Connectors invocation sequence
Java interceptor framework provides a way to insert processing logic before or after method calls and handle exceptions to perform alternate processing at runtime [33]. Some of the interceptors come with the utility such as alias which allow parameters to have different name aliases across requests, createSession to create an HTTP session on-the-fly, execAndWait to block the sender and execute the receiver in the background, exception to redirect the action call, etc. This interceptor framework allows inserting these interceptors at compile time or inside bytecode using javassist [15]. Our framework provides the way to realize all these interceptors in a declarative manner using communication ontology. It also allows to modify the mapping dynamically and call sequence without writing code for interceptor classes and annotations in the original Java source code. A custom interceptor can also be defined by editing ontology instead of writing new interceptor classes implementing the Interceptor interface.

4.3 Prioritized connectors
This scenario we faced during the development of a platform for hosting online courses called mookit[26]. This platform has a unique feature of content delivery to the users. If the user has sufficient internet bandwidth to access the mookit portal, then the content delivery happens in the form of videos. In the case of lower bandwidth, it delivers the content as static slides with audio transcripts. In the case of very low bandwidth, the mookit platform makes a call on the registered mobile number and delivers the content using interactive voice response (IVR) system over the telephony network. Communication ontology handles this scenario by using prioritized connectors calls. Based on the network bandwidth parameter, ontology infers the appropriate connector call specific to a content delivery mode.

4.4 Adding and Removing Components
A new e-commerce organization, decide to add a 3rd party component for payment gateway which works with only blocking call. For the efficiency purpose, the e-commerce company has implemented all its modules as non-blocking, and they interact with each other in asynchronous mode. The CA of Checkout component adapt this quickly by fetching specifications from the CO and sending a blocking call request to the 3rd party payment gateway with necessary parameters such as order id, request id, credit card details, and the amount. Now the organization decides to switch to its in-house payment gateway instead of using a 3rd party component, which requires a different set of parameters such as order id, card details, and the amount with non-blocking calls. The company decides to drop the request id, as it is not needed explicitly in the new scenario because the payment process is not going outside the system boundary to access some 3rd party component and order id is unique within the system boundary. Figure 8 depicts this scenario. The CO is updated with the new call graph of components with different parameters and connector specifications.

![Figure 8: CheckOut component has been redirected to in-house payment gateway at runtime with a non-blocking call. Initially, it was connected to 3rd party payment gateway with blocking call and the different set of parameters. Dotted line shows the old connectors among components.](image)

4.5 Accessibility of the components
Continuing from the previous use case of the e-commerce organization, it is desirable from the security perspective that any other component except Checkout component must not be able to call the Payment process and read credit card details. Ontology can also specify this kind of restrictions by defining accessibility property of the connectors.

4.6 Handling parameters
In case, where a component requires say n parameters and call to this component contains m (< n) number of parameters. Then, communication ontology can infer these extra parameters from the logic defined in the ontology itself. This approach allows the developers to deal with different components without modifying the functional logic components. This makes the ontological component complex and requires an ontology expert.

5. TENANT MANAGEMENT IN A MULTI-TENANT APPLICATION
In this section, we will look at the usage of the proposed framework in designing and implementing a multi-tenant application. A multi-tenant application capable of hosting multiple tenants having heterogeneous requirements on a single application instance [27]. It makes the tenants
with different requirements share hardware and software resources. All co-hosted tenants use single database and software services. A tenant is defined as an organization which consists of a group of users. As some tenants grow, it becomes difficult to handle multiple tenants. The maintenance cost may go up for a service provider of multi-tenant application. At one end, multi-tenancy increases the degree of resource sharing among tenants and on the other end, it also increases the design complexity of the application [9].

The most appropriate approach is to design a multi-tenant application with lower design complexity. Some of the desirable features of a multi-tenant applications are:

- **Workflow orchestration**: A tenant in a multi-tenant application can have customized workflow requirement. Ability to orchestrate workflow for each tenant at runtime allows co-hosting tenants with different workflows on a single application instance.

- **Defer Binding**: Tenant joins and leaves the multi-tenant instance at runtime. With the help of defer binding, a new tenant can be added to the system, and out-going tenant can be terminated from the system.

- **Dynamic resource allocation**: A tenant is a group of users. The number of users may vary with time. Within a lifetime of a tenant, tenant workload may also vary. Dynamic resource allocation makes sure that the sharing of the resources among tenants is optimal and adapt to the dynamic workload of the tenants.

- **Tenant aware logging**: Information about tenant is available at the application level. It should be propagated to generate the system level logs. It makes it possible to for developers to trace the activity of each tenant individually.

- **Tenant as a State**: Tenant should be represented as a state to make the addition, modification, and deletion of tenants easier. A state representation can be modified at runtime with less overhead. It also reduces the need for recompiling application component and re-load the executables.

Increase in the resource sharing among heterogeneous tenants results in reduced number of application instances a service provider requires to manage. It is in contrast with the multi-instance approach, where each tenant with different requirements is given a dedicated application instance. Multi-instance approach leads to a large number of service instances to maintain. It increases the maintenance cost for the service provider. On the other hand, although multi-tenant decrease the number of application instances but it may still incur higher maintenance cost due to its complex design.

In the section, we will look at how the proposed framework of using an ontology for communication logic can be useful to manage multiple tenants and implement a multi-tenant application. We have also developed a prototype system for multi-tenant microblogging [20].

Figure 9 shows the partial view of ontology to manage ten-
ants in a micro-blogging application. The application has multiple users; each user belongs to a Tenant group. Each user optionally relates to other users as following and/or follower. Based on which tenant users belong to, we demonstrate simple variations in the display of followers and background color using the depicted ontology scheme. The web app is accessible at the link given in the reference [20]. At the time of sign up, a user can specify the tenant id. It can also be updated later in the user profile section. A user can also visit other users page. Few users have been randomly allocated tenant id (either 1 or 2). Based on the tenant id, order of followers display and background color information is inferred from the ontology. The following ruby on rails code snippet demonstrates the parsing of the ntriples data generated using the ontology scheme shown in the Figure 9. The ontology is read using it’s IRI (Internationalized Resource Identifier). Then it loads the data into the graph in memory from the ntriples file. The code here shows the iteration through the complete dataset and parsing of various attributes such as username, tenant id, background color and sorting order. The advantage of using ontology over any other data format is ontology reasoners and inference capabilities.

```ruby
require 'rdf'
require 'rdf/vocab'
require 'rdf/ntriples'
require 'rdf/vocab'

graph = RDF::Graph.load("multitenant.nt")
query = RDF::Query.new({
  :person => {
    RDF::URI("http://www.semanticweb.org/sirius/ontologies/2017/11/mtu#User")
    => :name,
    RDF::URI("http://www.semanticweb.org/sirius/ontologies/2017/11/mtu#hasTenantId")
    => :tenantid,
    RDF::URI("http://www.semanticweb.org/sirius/ontologies/2017/11/mtu#hasBGColor")
    => :bgcolor
    RDF::URI("http://www.semanticweb.org/sirius/ontologies/2017/11/mtu#has_ordering")
    => :ordering
  }
})

results = query.execute(graph)

results.each do |r|
  if r[:name] == "User Tenant1"
    puts "Name: #{r[:name]},
    Tenant id: #{r[:tenantid]},
    Background Color: #{r[:bgcolor]},
    Sorting order: #{r[:ordering]}"
  end
end
```

6. RELATED WORK

In this section, we compare our work with existing approaches. Ontology-based communication approach has been explored in the context of micro-service only for dynamic service discovery [24] instead of communication among components. Authors in the work [12] proved that use of ontology as a mediator between two components reduce the cyclomatic complexity and circular dependency, but this work is limited to the design patterns.

Several researchers proposed methods to deal with the trade-offs among quality attributes. To accommodate conflicting quality attributes in a software product, attributed feature model is proposed in [28], which accounts the concern of multiple end-users in software product lines. A self-tuning method to adapt the dynamic environment to meet quality requirements optimally is also given by researchers [29]. Authors in [34] proposed dynamic reconfiguration management approach for component-based systems to achieve the trade-off among quality attributes on-the-fly.

All these work concerned towards improving either external or internal quality attributes of the software product. These approaches force to choose among quality attributes to adopt. None of the work focuses on varying the degree of these attributes to meet conflicting quality requirements during development and production phases. Apart from this, in [3] authors advocate the better ways to provide support at run-time while demolishing the rigid boundary between development and production as future research directions in software engineering research. Our work also enhances the internal and external quality attribute requirements during these two phases and fills the gap mentioned by the authors to some extent.

There have been a few efforts to use ontology in a various manner for multi-tenant applications. Authors have elaborated the advantages of using ontology as an architectural tool for decreasing maintenance overhead [13]. It is possible to leverage the micro-service architectural approach to implement multi-tenancy with the help of software product lines [41]. Usage of ontology along with service-oriented architecture has been explored also [10]. It does not build the multi-tenancy into the system. Recently authors have argued to combine microservice based approach with ontology to deliver variability for multi-tenancy [21]. None of the work deals with managing tenants to reduce the maintenance overhead in a multi-tenant scenario.

7. CONCLUSION

In this section, we conclude our work with future research directions. We present an ontology-based software development approach for accommodating conflicting quality attributes for developers and end-users. This is done by using an ontology-based connectivity at the development time and ‘hard-wiring’ it through a compile step for the operation phase. Ontology has been proven to be useful in scenarios where informed decision making is done dynamically at runtime. In scenarios where business rules are fast changing, it may be more appropriate to use ontology version of the software also in the operation phase. It allows business managers to handle the modifications to some extent without much technical support on the fly. It is also possible to partially compile the ontology and use partial ontology during the operation phase. When components are short-
lived or frequently replaced by other components with varying specifications, this approach increases the internal quality attributes of the software product such as maintainability, flexibility, modifiability, and portability. The ontology-based communication logic approach for software development can be integrated smoothly in all the phases of SDLC. Instead of using different tools during different phases, the ontology provides a one-stop solution with benefits of all the approaches. It paves the way to increase the degree of external quality attributes such as performance, efficiency by converting the ontology connectors into the ontology-free native source code. Apart from procedure call and event connectors discussed in this paper, the approach is extendable to other types of connectors - stream, data access, etc.

A proof of concept for developing a multi-tenant application using proposed framework is presented. It reduces the overhead of tenant management significantly in a multi-tenant scenario. It makes it easy to define new tenants with its configurations and modify them without the need of compilation steps. The expressive power of ontology in the form of semantic inference also helps in increasing the degree of multi-tenancy.

It will be interesting to analyze the compatibility of different kinds of connectors and how they can be converted from one type to another dynamically. Another useful direction for future research is to design a generic communication ontology which encompasses all the types of connectors. Creating complex connectors dynamically, which consists of multiple different types is also a challenging area to explore.

8. ACKNOWLEDGEMENT
The authors gratefully acknowledge the financial support from MHRD, Govt. of India for this work.

9. REFERENCES
multi-tenant saas applications. 2018.


ABOUT THE AUTHORS:

Sumit Kalra is a PhD scholar at Department of Computer Science and Engineering at Indian Institute of Technology Kanpur, Uttar Pradesh, India. He received a Bachelor of Engineering degree in Computer Science from UIET, Panjab University, Chandigarh, India and a Masters of Technology in Computer Science from IIT Kanpur, India. He has worked with Infosys Research Lab and IBM India Research Lab. He has worked in Healthcare domain, Human Skill Learning and Cloud Computing. His research interests are Software Architecture, Multi-tenant Cloud Applications, and applications of large scale data processing on contemporary computing platforms.

Prabhakar T. V. is a Professor in the Department of Computer Science and Engineering at Indian Institute of Technology Kanpur, Uttar Pradesh, India. He is interested in building systems and is still excited about science and technology. He has worked in multiple areas like Databases, Logic Programming, User Interface Design, Internet Technologies, Software Architecture, Knowledge Modelling, Indian Language Technologies. He has worked with CMC R&D, Satyam Computer Services and has been a consultant to a large number of IT organizations and e-governance initiatives. His research interests include Software Architecture, Knowledge Modelling, Web 2.0 and Indian Language Content.